```
**********************************************************
    8733 Tue Jan 14 16:16:54 2014
new/usr/src/Makefile.lint
Bring back LX zones.
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

# Copyright (c) 2003, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright (c) 2012 by Delphix. All rights reserved.

# include global definitions
include Makefile.master

#
# As pieces are made lint-clean, add them here so the nightly build
# can be used to keep them that way.
#
COMMON_SUBDIRS = \
        cmd/acctadm \
        cmd/asa \
        cmd/amt \
        cmd/audio/audioctl \
        cmd/audio/audiotest \
        cmd/audit \
        cmd/auditconfig \
        cmd/auditd \
        cmd/auditreduce \
        cmd/auditstat \
        cmd/auths \
        cmd/autopush \
        cmd/availdevs \
        cmd/avs \
        cmd/awk \
        cmd/banner \
        cmd/bart \
        cmd/basename \
        cmd/bdiff \
        cmd/bfs \
        cmd/busstat \
        cmd/boot \
        cmd/cal \
        cmd/captoinfo \
        cmd/cat \
        cmd/cdrw \
        cmd/cfgadm \
        cmd/checkeq \
        cmd/checknr \
```



```
        cmd/chgrp \
        cmd/chmod \
        cmd/chown \
        cmd/chroot \
        cmd/clinfo \
        cmd/cmd-crypto \
        cmd/cmd-inet/lib \
        cmd/cmd-inet/lib/netcfgd \
        cmd/cmd-inet/lib/nwamd \
        cmd/cmd-inet/sbin \
        cmd/cmd-inet/usr.bin \
        cmd/cmd-inet/usr.lib/bridged \
        cmd/cmd-inet/usr.lib/dsvclockd \
        cmd/cmd-inet/usr.lib/ilbd \
        cmd/cmd-inet/usr.lib/in.dhcpd \
        cmd/cmd-inet/usr.lib/in.mpathd \
        cmd/cmd-inet/usr.lib/in.ndpd \
        cmd/cmd-inet/usr.lib/inetd \
        cmd/cmd-inet/usr.lib/pppoe \
        cmd/cmd-inet/usr.lib/slpd \
        cmd/cmd-inet/usr.lib/vrrpd \
        cmd/cmd-inet/usr.lib/wpad \
        cmd/cmd-inet/usr.lib/wanboot \
        cmd/cmd-inet/usr.sadm \
        cmd/cmd-inet/usr.sbin \
        cmd/cmd-inet/usr.sbin/ilbadm \
        cmd/cmd-inet/usr.sbin/nwamadm \
        cmd/cmd-inet/usr.sbin/nwamcfg \
        cmd/col \
        cmd/compress \
        cmd/consadm \
        cmd/coreadm \
        cmd/cpc \
        cmd/cpio \
        cmd/crypt \
        cmd/csplit \
        cmd/ctrun \
        cmd/ctstat \
        cmd/ctwatch \
        cmd/date \
        cmd/dd \
        cmd/deroff \
        cmd/devctl \
        cmd/devfsadm \
        cmd/devinfo \
        cmd/devmgmt \
        cmd/devprop \
        cmd/dfs.cmds \
        cmd/diff3 \
        cmd/dis \
        cmd/dirname \
        cmd/diskscan \
        cmd/dispadmin \
        cmd/dladm \
        cmd/dlmgmtd \
        cmd/dtrace \
        cmd/du \
        cmd/dumpadm \
        cmd/dumpcs \
        cmd/echo \
        cmd/eject \
        cmd/emul64ioctl \
        cmd/env \
        cmd/expand \
        cmd/fcinfo \
        cmd/fdetach \
```

```
128         cmd/fdformat \
129         cmd/fdisk \
130         cmd/fgrep \
131         cmd/file \
132         cmd/filebench \
133         cmd/find \
134         cmd/fmthard \
135         cmd/fmtmsg \
136         cmd/fold \
137         cmd/fm \
138         cmd/format \
139         cmd/fs.d/fd \
140         cmd/fs.d/lofs/mount \
141         cmd/fs.d/mntfs \
142         cmd/fs.d/pcfs/mount \
143         cmd/fs.d/proc \
144         cmd/fs.d/tmpfs \
145         cmd/fs.d/udfs/mount \
146         cmd/fs.d/ufs/mount \
147         cmd/fs.d/ufs/fsirand\
148         cmd/fs.d/zfs/fstyp \
149         cmd/fwflash \
150         cmd/fuser \
151         cmd/gcore \
152         cmd/genmsg \
153         cmd/getconf \
154         cmd/getdevpolicy \
155         cmd/getfacl \
156         cmd/getopt \
157         cmd/gettext \
158         cmd/grep \
159         cmd/grep_xpg4 \
160         cmd/groups \
161         cmd/halt \
162         cmd/head \
163         cmd/hostid \
164         cmd/hostname \
165         cmd/hotplug \
166         cmd/hotplugd \
167         cmd/idmap \
168         cmd/init \
169         cmd/intrstat \
170         cmd/ipcrm \
171         cmd/ipcs \
172         cmd/isaexec \
173         cmd/isalist \
174         cmd/iscsiadm \
175         cmd/iscsid \
176         cmd/iscsitsvc \
177         cmd/isns \
178         cmd/itadm \
179         cmd/kbd \
180         cmd/killall \
181         cmd/ldap \
182         cmd/last \
183         cmd/lastcomm \
184         cmd/ldapcachemgr \
185         cmd/line \
186         cmd/link \
187         cmd/locator \
188         cmd/localedef \
189         cmd/lockstat \
190         cmd/lofiadm \
191         cmd/logadm \
192         cmd/logger \
193         cmd/login \
```



```
194         cmd/logins \
195         cmd/ls \
196         cmd/luxadm \
197         cmd/lvm \
198         cmd/machid \
199         cmd/makekey \
200         cmd/mdb \
201         cmd/mesg \
202         cmd/mkdir \
203         cmd/mkfifo \
204         cmd/mkfile \
205         cmd/mkmsgs \
206         cmd/mknod \
207         cmd/mpathadm \
208         cmd/modload \
209         cmd/msgfmt \
210         cmd/msgid \
211         cmd/mt \
212         cmd/mv \
213         cmd/ndmpadm \
214         cmd/ndmpd \
215         cmd/ndmpstat \
216         cmd/newform \
217         cmd/newgrp \
218         cmd/newtask \
219         cmd/nice \
220         cmd/nl \
221         cmd/nohup \
222         cmd/nscd \
223         cmd/od \
224         cmd/pagesize \
225         cmd/passwd \
226         cmd/pathchk \
227         cmd/pbind \
228         cmd/pcidr \
229         cmd/pcitool \
230         cmd/pfexec \
231         cmd/pgrep \
232         cmd/picl/picld \
233         cmd/picl/prtpicl \
234         cmd/plockstat \
235         cmd/pools \
236         cmd/power \
237         cmd/powertop \
238         cmd/printf \
239         cmd/latencytop \
240         cmd/ppgsz \
241         cmd/praudit \
242         cmd/prctl \
243         cmd/priocntl \
244         cmd/profiles \
245         cmd/prstat \
246         cmd/prtconf \
247         cmd/prtdiag \
248         cmd/prtvtoc \
249         cmd/ps \
250         cmd/psradm \
251         cmd/psrinfo \
252         cmd/psrset \
253         cmd/ptools \
254         cmd/pwck \
255         cmd/pwconv \
256         cmd/ramdiskadm \
257         cmd/raidctl \
258         cmd/rcap \
259         cmd/rcm_daemon \
```

```
260         cmd/rctladm \
261         cmd/renice \
262         cmd/rm \
263         cmd/rmdir \
264         cmd/rmformat \
265         cmd/rmt \
266         cmd/roles \
267         cmd/rpcgen \
268         cmd/rpcsvc/rpc.bootparamd \
269         cmd/runat \
270         cmd/savecore \
271         cmd/sbdadm \
272         cmd/sdpadm \
273         cmd/sed \
274         cmd/setpgrp \
275         cmd/smbios \
276         cmd/sgs \
277         cmd/smbsrv \
278         cmd/smserverd \
279         cmd/sort \
280         cmd/split \
281         cmd/srptadm \
282         cmd/srptsvc \
283         cmd/ssh \
284         cmd/stat \
285         cmd/stmfadm \
286         cmd/stmfsvc \
287         cmd/stmsboot \
288         cmd/streams/strcmd \
289         cmd/strings \
290         cmd/su \
291         cmd/sulogin \
292         cmd/svc \
293         cmd/swap \
294         cmd/sync \
295         cmd/syseventadm \
296         cmd/syseventd \
297         cmd/syslogd \
298         cmd/tabs \
299         cmd/tail \
300         cmd/th_tools \
301         cmd/tip \
302         cmd/touch \
303         cmd/tr \
304         cmd/truss \
305         cmd/tty \
306         cmd/tzreload \
307         cmd/uadmin \
308         cmd/ul \
309         cmd/userattr \
310         cmd/users \
311         cmd/utmp_update \
312         cmd/utmpd \
313         cmd/valtools \
314         cmd/vrrpadm \
315         cmd/vt \
316         cmd/wall \
317         cmd/who \
318         cmd/whodo \
319         cmd/wracct \
320         cmd/wusbadm \
321         cmd/xargs \
322         cmd/xstr \
323         cmd/yes \
324         cmd/yppasswd \
325         cmd/zdb \
```



```
326         cmd/zdump \
327         cmd/zfs \
328         cmd/zhack \
329         cmd/zinject \
330         cmd/zlogin \
331         cmd/zoneadm \
332         cmd/zoneadmd \
333         cmd/zonecfg \
334         cmd/zonename \
335         cmd/zpool \
336         cmd/zlook \
337         cmd/ztest \
338         lib/abi \
339         lib/auditd_plugins \
340         lib/libbe \
341         lib/pylibbe \
342         lib/brand/sn1 \
343         lib/brand/solaris10 \
344         lib/crypt_modules \
345         lib/extendedFILE \
346         lib/libadm \
347         lib/libadutils \
348         lib/libadt_jni \
349         lib/libaio \
350         lib/libavl \
351         lib/libbrand \
352         lib/libbsdmalloc \
353         lib/libbsm \
354         lib/libc \
355         lib/libc_db \
356         lib/libcfgadm \
357         lib/libcmdutils \
358         lib/libcommputil \
359         lib/libcontract \
360         lib/libcryptoutil \
361         lib/libctf \
362         lib/libdevice \
363         lib/libdevid \
364         lib/libdevinfo \
365         lib/libdhcpagent \
366         lib/libdhcpdu \
367         lib/libdhcpsvc \
368         lib/libdhcputil \
369         lib/libdisasm \
370         lib/libdiskmgt \
371         lib/libdladm \
372         lib/libdlpi \
373         lib/libdoor \
374         lib/libdscfg \
375         lib/libdtrace \
376         lib/libefi \
377         lib/libelfsign \
378         lib/libexacct \
379         lib/libfcoe \
380         lib/libgen \
381         lib/libgrubmgmt \
382         lib/libgss \
383         lib/libhotplug \
384         lib/libidmap \
385         lib/libilb \
386         lib/libinetsvc \
387         lib/libinetutil \
388         lib/libinstzones \
389         lib/libipadm \
390         lib/libipmi \
391         lib/libipmp \
```

```
392         lib/libipp \
393         lib/libipsecutil \
394         lib/libiscsit \
395         lib/libkmf \
396         lib/libkstat \
397         lib/liblgrp \
398         lib/liblm \
399         lib/libmalloc \
400         lib/libmapmalloc \
401         lib/libmapid \
402         lib/libmd \
403         lib/libmp \
404         lib/libmtmalloc \
405         lib/libndmp \
406         lib/libnsctl \
407         lib/libnsl \
408         lib/libnvpair \
409         lib/libnwam \
410         lib/libpam \
411         lib/libpctx \
412         lib/libpicl \
413         lib/libpicltree \
414         lib/libpkg \
415         lib/libpool \
416         lib/libproc \
417         lib/libpthread \
418         lib/libraidcfg \
419         lib/librcm \
420         lib/librdc \
421         lib/libreparse \
422         lib/librestart \
423         lib/librstp \
424         lib/librt \
425         lib/libscf \
426         lib/libsec \
427         lib/libsecdb \
428         lib/libsendfile \
429         lib/libsip \
430         lib/libshare \
431         lib/libsldap \
432         lib/libslp \
433         lib/libsmbfs \
434         lib/libsmbios \
435         lib/libsmedia \
436         lib/libsrpt \
437         lib/libstmf \
438         lib/libsun_ima \
439         lib/libsysevent \
440         lib/libthread \
441         lib/libtsnet \
442         lib/libtsol \
443         lib/libumem \
444         lib/libunistat \
445         lib/libuuid \
446         lib/libuutil \
447         lib/libvrrpadm \
448         lib/libwanboot \
449         lib/libwanbootutil \
450         lib/libxnet \
451         lib/libzfs \
452         lib/libzfs_jni \
453         lib/libzonecfg \
454         lib/libzoneinfo \
455         lib/lvm \
456         lib/madv \
457         lib/mpss \
```



```
458         lib/nametoaddr \
459         lib/ncad_addr \
460         lib/nsswitch \
461         lib/pam_modules \
462         lib/passwdutil \
463         lib/pkcs11 \
464         lib/print \
465         lib/raidcfg_plugins \
466         lib/scsi \
467         lib/smbsrv \
468         lib/fm \
469         lib/udapl \
470         lib/watchmalloc \
471         psm \
472         test \
473         ucbcmd/basename \
474         ucbcmd/biff \
475         ucbcmd/echo \
476         ucbcmd/groups \
477         ucbcmd/mkstr \
478         ucbcmd/printenv \
479         ucbcmd/sum \
480         ucbcmd/test \
481         ucbcmd/users \
482         ucbcmd/whoami

484 i386_SUBDIRS= \
485         cmd/acpihpd \
486         cmd/biosdev \
487         cmd/rtc \
488         cmd/ucodeadm \
489         lib/brand/lx \
490 #endif /* ! codereview */
491         lib/cfgadm_plugins/sata \
492         lib/cfgadm_plugins/sbd \
493         lib/libfdisk

495 sparc_SUBDIRS= \
496         cmd/datadm \
497         cmd/dcs \
498         cmd/drd \
499         cmd/fruadm \
500         cmd/ldmad \
501         cmd/prtdscp \
502         cmd/prtfru \
503         cmd/sckmd \
504         cmd/virtinfo \
505         cmd/vntsd \
506         lib/libds \
507         lib/libdscp \
508         lib/libpri \
509         lib/libpcp \
510         lib/libtsalarm \
511         lib/libv12n \
512         lib/storage \
513         stand

515 LINTSUBDIRS= $(COMMON_SUBDIRS) $($(MACH)_SUBDIRS)

517 .PARALLEL:      $(LINTSUBDIRS)

519 lint:           uts .WAIT subdirs

521 subdirs:        $(LINTSUBDIRS)

523 uts $(LINTSUBDIRS):     FRC
```

```
524         @cd $@; pwd; $(MAKE) lint

526 FRC:
```

```
**********************************************************
   70200 Tue Jan 14 16:16:54 2014
new/usr/src/Targetdirs
Bring back LX zones.
**********************************************************
```

```
   1 # CDDL HEADER START
   2 #
   3 # The contents of this file are subject to the terms of the
   4 # Common Development and Distribution License (the "License").
   5 # You may not use this file except in compliance with the License.
   6 #
   7 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   8 # or http://www.opensolaris.org/os/licensing.
   9 # See the License for the specific language governing permissions
  10 # and limitations under the License.
  11 #
  12 # When distributing Covered Code, include this CDDL HEADER in each
  13 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  14 # If applicable, add the following below this CDDL HEADER, with the
  15 # fields enclosed by brackets "[]" replaced with your own identifying
  16 # information: Portions Copyright [yyyy] [name of copyright owner]
  17 #
  18 # CDDL HEADER END
  19 #

  21 #
  22 # Copyright (c) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
  23 # Copyright 2011, Richard Lowe
  24 # Copyright (c) 2012 by Delphix. All rights reserved.
  25 # Copyright 2012 OmniTI Computer Consulting, Inc.  All rights reserved.
  26 # Copyright (c) 2013 RackTop Systems.
  27 # Copyright 2013 Nexenta Systems, Inc.  All rights reserved.
  28 #

  30 #
  31 # It is easier to think in terms of directory names without the ROOT macro
  32 # prefix.  ROOTDIRS is TARGETDIRS with ROOT prefixes.  It is necessary
  33 # to work with ROOT prefixes when controlling conditional assignments.
  34 #

  36 DIRLINKS=       $(SYM.DIRS)
  37 $(BUILD64)      DIRLINKS += $(SYM.DIRS64)

  39 FILELINKS= $(SYM.USRCCSLIB) $(SYM.USRLIB)
  40 $(BUILD64)      FILELINKS += $(SYM.USRCCSLIB64) $(SYM.USRLIB64)

  42 TARGETDIRS=     $(DIRS)
  43 $(BUILD64)      TARGETDIRS += $(DIRS64)

  45 TARGETDIRS      += $(FILELINKS) $(DIRLINKS)

  47 i386_DIRS=                       \
  48         /boot/acpi               \
  49         /boot/acpi/tables        \
  50         /boot/grub               \
  51         /boot/grub/bin           \
  52         /platform/i86pc          \
  53         /usr/lib/brand/lx \
  54         /usr/lib/brand/lx/amd64 \
  55         /usr/lib/brand/lx/distros \
  56 #endif /* ! codereview */
  57         /usr/lib/xen             \
  58         /usr/lib/xen/bin

  60 sparc_DIRS=                           \
  61         /usr/lib/ldoms
```



```
  63 sparc_64ONLY= $(POUND_SIGN)
  64 64ONLY=  $($(MACH)_64ONLY)

  66 $(64ONLY) MACH32_DIRS=/usr/ucb/$(MACH32)

  68 DIRS= \
  69         /boot \
  70         /boot/solaris \
  71         /boot/solaris/bin \
  72         $($(MACH)_DIRS) \
  73         /dev \
  74         /dev/dsk \
  75         /dev/fd \
  76         /dev/ipnet \
  77         /dev/net \
  78         /dev/rdsk \
  79         /dev/rmt \
  80         /dev/pts \
  81         /dev/sad \
  82         /dev/swap \
  83         /dev/term \
  84         /dev/vt \
  85         /dev/zcons \
  86         /devices \
  87         /devices/pseudo \
  88         /etc \
  89         /etc/brand  \
  90         /etc/brand/solaris10  \
  91         /etc/cron.d \
  92         /etc/crypto \
  93         /etc/crypto/certs \
  94         /etc/crypto/crls \
  95         /etc/dbus-1 \
  96         /etc/dbus-1/system.d \
  97         /etc/default \
  98         /etc/devices  \
  99         /etc/dev  \
 100         /etc/dfs  \
 101         /etc/dladm \
 102         /etc/fs  \
 103         /etc/fs/nfs  \
 104         /etc/fs/zfs \
 105         /etc/ftpd  \
 106         /etc/hal \
 107         /etc/hal/fdi \
 108         /etc/hal/fdi/information \
 109         /etc/hal/fdi/information/10freedesktop \
 110         /etc/hal/fdi/information/20thirdparty \
 111         /etc/hal/fdi/information/30user \
 112         /etc/hal/fdi/policy \
 113         /etc/hal/fdi/policy/10osvendor \
 114         /etc/hal/fdi/policy/20thirdparty \
 115         /etc/hal/fdi/policy/30user \
 116         /etc/hal/fdi/preprobe \
 117         /etc/hal/fdi/preprobe/10osvendor \
 118         /etc/hal/fdi/preprobe/20thirdparty \
 119         /etc/hal/fdi/preprobe/30user \
 120         /etc/ipadm \
 121         /etc/iscsi \
 122         /etc/rpcsec     \
 123         /etc/security   \
 124         /etc/security/auth_attr.d \
 125         /etc/security/exec_attr.d \
 126         /etc/security/prof_attr.d \
 127         /etc/security/tsol      \
```

```
128         /etc/gss        \
129         /etc/init.d  \
130         /etc/dhcp       \
131         /etc/lib  \
132         /etc/mail  \
133         /etc/mail/cf  \
134         /etc/mail/cf/cf  \
135         /etc/mail/cf/domain  \
136         /etc/mail/cf/feature  \
137         /etc/mail/cf/m4  \
138         /etc/mail/cf/mailer  \
139         /etc/mail/cf/ostype  \
140         /etc/mail/cf/sh  \
141         /etc/net-snmp \
142         /etc/net-snmp/snmp \
143         /etc/opt  \
144         /etc/rc0.d  \
145         /etc/rc1.d  \
146         /etc/rc2.d  \
147         /etc/rc3.d  \
148         /etc/rcS.d  \
149         /etc/saf \
150         /etc/sasl       \
151         /etc/sfw \
152         /etc/skel       \
153         /etc/svc \
154         /etc/svc/profile \
155         /etc/svc/profile/site \
156         /etc/svc/volatile \
157         /etc/tm  \
158         /etc/usb   \
159         /etc/user_attr.d \
160         /etc/zfs  \
161         /etc/zones  \
162         /export  \
163         /home  \
164         /lib \
165         /lib/crypto \
166         /lib/inet \
167         /lib/fm \
168         /lib/secure \
169         /lib/svc \
170         /lib/svc/bin \
171         /lib/svc/capture \
172         /lib/svc/manifest \
173         /lib/svc/manifest/milestone \
174         /lib/svc/manifest/device \
175         /lib/svc/manifest/system \
176         /lib/svc/manifest/system/device \
177         /lib/svc/manifest/system/filesystem \
178         /lib/svc/manifest/system/security \
179         /lib/svc/manifest/system/svc \
180         /lib/svc/manifest/network \
181         /lib/svc/manifest/network/dns \
182         /lib/svc/manifest/network/ipsec \
183         /lib/svc/manifest/network/ldap \
184         /lib/svc/manifest/network/nfs \
185         /lib/svc/manifest/network/nis \
186         /lib/svc/manifest/network/rpc \
187         /lib/svc/manifest/network/security \
188         /lib/svc/manifest/network/shares \
189         /lib/svc/manifest/network/ssl \
190         /lib/svc/manifest/application \
191         /lib/svc/manifest/application/management \
192         /lib/svc/manifest/application/security \
193         /lib/svc/manifest/application/print \
```



```
194         /lib/svc/manifest/platform \
195         /lib/svc/manifest/platform/sun4u \
196         /lib/svc/manifest/platform/sun4v \
197         /lib/svc/manifest/site \
198         /lib/svc/method \
199         /lib/svc/monitor \
200         /lib/svc/seed \
201         /lib/svc/share \
202         /kernel  \
203         /mnt  \
204         /opt  \
205         /platform  \
206         /proc  \
207         /root  \
208         /sbin  \
209         /system \
210         /system/contract \
211         /system/object \
212         /tmp \
213         /usr  \
214         /usr/4lib \
215         /usr/ast \
216         /usr/ast/bin \
217         /usr/bin \
218         /usr/bin/$(MACH32) \
219         /usr/ccs \
220         /usr/ccs/bin \
221         /usr/ccs/lib \
222         /usr/demo \
223         /usr/demo/SOUND \
224         /usr/games \
225         /usr/has \
226         /usr/has/bin \
227         /usr/has/lib \
228         /usr/has/man \
229         /usr/has/man/man1has \
230         /usr/include \
231         /usr/include/ast \
232         /usr/include/fm \
233         /usr/include/gssapi \
234         /usr/include/hal \
235         /usr/include/kerberosv5 \
236         /usr/include/libmilter \
237         /usr/include/libpolkit \
238         /usr/include/sasl \
239         /usr/include/scsi \
240         /usr/include/security \
241         /usr/include/sys/crypto \
242         /usr/include/tsol \
243         /usr/kernel  \
244         /usr/kvm \
245         /usr/lib \
246         /usr/lib/abi \
247         /usr/lib/brand \
248         /usr/lib/brand/ipkg \
249         /usr/lib/brand/labeled \
250         /usr/lib/brand/shared \
251         /usr/lib/brand/sn1 \
252         /usr/lib/brand/solaris10 \
253         /usr/lib/class \
254         /usr/lib/class/FSS \
255         /usr/lib/class/FX \
256         /usr/lib/class/IA \
257         /usr/lib/class/RT \
258         /usr/lib/class/SDC \
259         /usr/lib/class/TS \
```

```
260         /usr/lib/crypto \
261         /usr/lib/drv \
262         /usr/lib/elfedit \
263         /usr/lib/fm \
264         /usr/lib/font \
265         /usr/lib/fs \
266         /usr/lib/fs/nfs \
267         /usr/lib/fs/proc \
268         /usr/lib/fs/smb \
269         /usr/lib/fs/zfs \
270         /usr/lib/gss \
271         /usr/lib/hal \
272         /usr/lib/inet \
273         /usr/lib/inet/dhcp \
274         /usr/lib/inet/dhcp/nsu \
275         /usr/lib/inet/dhcp/svc \
276         /usr/lib/inet/dhcp/svcadm \
277         /usr/lib/inet/ilb \
278         /usr/lib/inet/$(MACH32) \
279         /usr/lib/inet/wanboot \
280         /usr/lib/krb5 \
281         /usr/lib/link_audit \
282         /usr/lib/libp \
283         /usr/lib/lwp \
284         /usr/lib/mdb \
285         /usr/lib/mdb/kvm \
286         /usr/lib/mdb/proc \
287         /usr/lib/nfs \
288         /usr/net \
289         /usr/net/servers \
290         /usr/lib/pool \
291         /usr/lib/python2.6 \
292         /usr/lib/python2.6/vendor-packages \
293         /usr/lib/python2.6/vendor-packages/64 \
294         /usr/lib/python2.6/vendor-packages/solaris \
295         /usr/lib/python2.6/vendor-packages/zfs \
296         /usr/lib/python2.6/vendor-packages/beadm \
297         /usr/lib/rcap \
298         /usr/lib/rcap/$(MACH32) \
299         /usr/lib/sa  \
300         /usr/lib/saf \
301         /usr/lib/sasl \
302         /usr/lib/scsi \
303         /usr/lib/secure \
304         /usr/lib/security \
305         /usr/lib/smbsrv \
306         /usr/lib/vscan \
307         /usr/lib/zfs \
308         /usr/lib/zones \
309         /usr/old \
310         /usr/platform  \
311         /usr/proc \
312         /usr/proc/bin \
313         /usr/sadm \
314         /usr/sadm/install \
315         /usr/sadm/install/bin \
316         /usr/sadm/install/scripts \
317         /usr/sbin \
318         /usr/sbin/$(MACH32) \
319         /usr/share  \
320         /usr/share/applications \
321         /usr/share/audio \
322         /usr/share/audio/samples \
323         /usr/share/audio/samples/au \
324         /usr/share/gnome \
325         /usr/share/gnome/autostart \
```



```
326         /usr/share/hwdata \
327         /usr/share/lib \
328         /usr/share/lib/ccs \
329         /usr/share/lib/tmac  \
330         /usr/share/lib/ldif  \
331         /usr/share/lib/xml \
332         /usr/share/lib/xml/dtd \
333         /usr/share/man \
334         /usr/share/man/man1 \
335         /usr/share/man/man1b \
336         /usr/share/man/man1c \
337         /usr/share/man/man1m \
338         /usr/share/man/man2 \
339         /usr/share/man/man3 \
340         /usr/share/man/man3bsm \
341         /usr/share/man/man3c \
342         /usr/share/man/man3c_db \
343         /usr/share/man/man3cfgadm \
344         /usr/share/man/man3commputil \
345         /usr/share/man/man3contract \
346         /usr/share/man/man3cpc \
347         /usr/share/man/man3curses \
348         /usr/share/man/man3dat \
349         /usr/share/man/man3devid \
350         /usr/share/man/man3devinfo \
351         /usr/share/man/man3dlpi \
352         /usr/share/man/man3dns_sd \
353         /usr/share/man/man3elf \
354         /usr/share/man/man3exacct \
355         /usr/share/man/man3ext \
356         /usr/share/man/man3fcoe \
357         /usr/share/man/man3fstyp \
358         /usr/share/man/man3gen \
359         /usr/share/man/man3gss \
360         /usr/share/man/man3head \
361         /usr/share/man/man3iscsit \
362         /usr/share/man/man3kstat \
363         /usr/share/man/man3kvm \
364         /usr/share/man/man3ldap \
365         /usr/share/man/man3lgrp \
366         /usr/share/man/man3lib \
367         /usr/share/man/man3mail \
368         /usr/share/man/man3malloc \
369         /usr/share/man/man3mp \
370         /usr/share/man/man3mpapi \
371         /usr/share/man/man3nsl \
372         /usr/share/man/man3nvpair \
373         /usr/share/man/man3pam \
374         /usr/share/man/man3papi \
375         /usr/share/man/man3perl \
376         /usr/share/man/man3picl \
377         /usr/share/man/man3picltree \
378         /usr/share/man/man3pool \
379         /usr/share/man/man3proc \
380         /usr/share/man/man3project \
381         /usr/share/man/man3resolv \
382         /usr/share/man/man3rpc \
383         /usr/share/man/man3rsm \
384         /usr/share/man/man3sasl \
385         /usr/share/man/man3scf \
386         /usr/share/man/man3sec \
387         /usr/share/man/man3secdb \
388         /usr/share/man/man3sip \
389         /usr/share/man/man3slp \
390         /usr/share/man/man3socket \
391         /usr/share/man/man3stmf \
```

```
392         /usr/share/man/man3sysevent \
393         /usr/share/man/man3tecla \
394         /usr/share/man/man3tnf \
395         /usr/share/man/man3tsol \
396         /usr/share/man/man3uuid \
397         /usr/share/man/man3volmgt \
398         /usr/share/man/man3xcurses \
399         /usr/share/man/man3xnet \
400         /usr/share/man/man4 \
401         /usr/share/man/man5 \
402         /usr/share/man/man7 \
403         /usr/share/man/man7d \
404         /usr/share/man/man7fs \
405         /usr/share/man/man7i \
406         /usr/share/man/man7ipp \
407         /usr/share/man/man7m \
408         /usr/share/man/man7p \
409         /usr/share/man/man9 \
410         /usr/share/man/man9e \
411         /usr/share/man/man9f \
412         /usr/share/man/man9p \
413         /usr/share/man/man9s \
414         /usr/share/src \
415         /usr/snadm \
416         /usr/snadm/lib \
417         /usr/ucb \
418         $(MACH32_DIRS) \
419         /usr/ucblib \
420         /usr/xpg4 \
421         /usr/xpg4/bin \
422         /usr/xpg4/include \
423         /usr/xpg4/lib \
424         /usr/xpg6 \
425         /usr/xpg6/bin \
426         /var  \
427         /var/adm  \
428         /var/adm/exacct \
429         /var/adm/log \
430         /var/adm/pool \
431         /var/adm/sa \
432         /var/adm/sm.bin \
433         /var/adm/streams \
434         /var/cores \
435         /var/cron \
436         /var/db \
437         /var/db/ipf \
438         /var/games \
439         /var/idmap \
440         /var/krb5 \
441         /var/krb5/rcache  \
442         /var/krb5/rcache/root  \
443         /var/ld \
444         /var/log \
445         /var/log/pool \
446         /var/logadm \
447         /var/mail \
448         /var/news \
449         /var/opt \
450         /var/preserve \
451         /var/run \
452         /var/saf \
453         /var/sadm \
454         /var/sadm/install \
455         /var/sadm/install/admin \
456         /var/sadm/install/logs \
457         /var/sadm/pkg \
```



```
458         /var/sadm/security \
459         /var/smb \
460         /var/smb/cvol \
461         /var/smb/cvol/windows \
462         /var/smb/cvol/windows/system32 \
463         /var/smb/cvol/windows/system32/vss \
464         /var/spool \
465         /var/spool/cron \
466         /var/spool/cron/atjobs \
467         /var/spool/cron/crontabs \
468         /var/spool/lp \
469         /var/spool/pkg \
470         /var/spool/uucp \
471         /var/spool/uucppublic \
472         /var/svc \
473         /var/svc/log \
474         /var/svc/manifest \
475         /var/svc/manifest/milestone \
476         /var/svc/manifest/device \
477         /var/svc/manifest/system \
478         /var/svc/manifest/system/device \
479         /var/svc/manifest/system/filesystem \
480         /var/svc/manifest/system/security \
481         /var/svc/manifest/system/svc \
482         /var/svc/manifest/network \
483         /var/svc/manifest/network/dns \
484         /var/svc/manifest/network/ipsec \
485         /var/svc/manifest/network/ldap \
486         /var/svc/manifest/network/nfs \
487         /var/svc/manifest/network/nis \
488         /var/svc/manifest/network/rpc \
489         /var/svc/manifest/network/routing \
490         /var/svc/manifest/network/security \
491         /var/svc/manifest/network/shares \
492         /var/svc/manifest/network/ssl \
493         /var/svc/manifest/application \
494         /var/svc/manifest/application/management \
495         /var/svc/manifest/application/print \
496         /var/svc/manifest/application/security \
497         /var/svc/manifest/platform \
498         /var/svc/manifest/platform/sun4u \
499         /var/svc/manifest/platform/sun4v \
500         /var/svc/manifest/site \
501         /var/svc/profile \
502         /var/uucp \
503         /var/tmp \
504         /var/tsol \
505         /var/tsol/doors

507 sparcv9_DIRS64= \
508         /platform/sun4u \
509         /platform/sun4u/lib \
510         /platform/sun4u/lib/$(MACH64) \
511         /usr/platform/sun4u \
512         /usr/platform/sun4u/sbin \
513         /usr/platform/sun4u/lib \
514         /platform/sun4v/lib \
515         /platform/sun4v/lib/$(MACH64) \
516         /usr/platform/sun4v/sbin \
517         /usr/platform/sun4v/lib \
518         /usr/platform/sun4u-us3/lib \
519         /usr/platform/sun4u-opl/lib

521 amd64_DIRS64= \
522         /platform/i86pc/amd64
```

```
524 DIRS64= \
525         $($(MACH64)_DIRS64) \
526         /lib/$(MACH64) \
527         /lib/crypto/$(MACH64) \
528         /lib/fm/$(MACH64) \
529         /lib/secure/$(MACH64) \
530         /usr/bin/$(MACH64) \
531         /usr/ccs/bin/$(MACH64)  \
532         /usr/ccs/lib/$(MACH64) \
533         /usr/lib/$(MACH64) \
534         /usr/lib/$(MACH64)/gss \
535         /usr/lib/brand/sn1/$(MACH64) \
536         /usr/lib/brand/solaris10/$(MACH64) \
537         /usr/lib/elfedit/$(MACH64) \
538         /usr/lib/fm/$(MACH64) \
539         /usr/lib/fs/nfs/$(MACH64) \
540         /usr/lib/fs/smb/$(MACH64) \
541         /usr/lib/inet/$(MACH64) \
542         /usr/lib/krb5/$(MACH64) \
543         /usr/lib/libp/$(MACH64) \
544         /usr/lib/link_audit/$(MACH64) \
545         /usr/lib/lwp/$(MACH64) \
546         /usr/lib/mdb/kvm/$(MACH64) \
547         /usr/lib/mdb/proc/$(MACH64) \
548         /usr/lib/rcap/$(MACH64) \
549         /usr/lib/sasl/$(MACH64) \
550         /usr/lib/scsi/$(MACH64) \
551         /usr/lib/secure/$(MACH64) \
552         /usr/lib/security/$(MACH64) \
553         /usr/lib/smbsrv/$(MACH64) \
554         /usr/lib/abi/$(MACH64) \
555         /usr/sbin/$(MACH64) \
556         /usr/ucb/$(MACH64) \
557         /usr/ucblib/$(MACH64) \
558         /usr/xpg4/lib/$(MACH64) \
559         /var/ld/$(MACH64)

561 # /var/mail/:saved is built directly by the rootdirs target in
562 # /usr/src/Makefile because of the colon in its name.

564 # macros for symbolic links
565 SYM.DIRS= \
566         /bin \
567         /dev/stdin \
568         /dev/stdout \
569         /dev/stderr \
570         /etc/lib/ld.so.1 \
571         /etc/lib/libdl.so.1 \
572         /etc/lib/nss_files.so.1 \
573         /etc/log \
574         /lib/32 \
575         /lib/crypto/32 \
576         /lib/secure/32 \
577         /usr/adm \
578         /usr/spool \
579         /usr/lib/tmac \
580         /usr/ccs/lib/link_audit \
581         /usr/news \
582         /usr/preserve \
583         /usr/lib/32 \
584         /usr/lib/cron \
585         /usr/lib/elfedit/32 \
586         /usr/lib/libp/32 \
587         /usr/lib/lwp/32 \
588         /usr/lib/link_audit/32 \
589         /usr/lib/secure/32 \
```



```
590         /usr/mail \
591         /usr/man \
592         /usr/pub \
593         /usr/src \
594         /usr/tmp \
595         /usr/ucblib/32 \
596         /var/ld/32

598 i386_SYM.DIRS64= \
599         /usr/lib/brand/lx/64

601 #endif /* ! codereview */
602 sparc_SYM.DIRS64=

604 SYM.DIRS64= \
605         $($(MACH)_SYM.DIRS64) \
606         /lib/64 \
607         /lib/crypto/64 \
608         /lib/secure/64 \
609         /usr/lib/64 \
610         /usr/lib/brand/sn1/64 \
611         /usr/lib/brand/solaris10/64 \
612         /usr/lib/elfedit/64 \
613         /usr/lib/libp/64 \
614         /usr/lib/link_audit/64 \
615         /usr/lib/lwp/64 \
616         /usr/lib/secure/64 \
617         /usr/lib/security/64 \
618         /usr/xpg4/lib/64 \
619         /var/ld/64 \
620         /usr/ucblib/64

622 # prepend the ROOT prefix

624 ROOTDIRS=       $(TARGETDIRS:%=$(ROOT)%)

626 # conditional assignments
627 #
628 # Target directories with non-default values for owner and group must
629 # be referenced here, using their fully-prefixed names, and the non-
630 # default values assigned.  If a directory is mentioned above and not
631 # mentioned below, it has default values for attributes.
632 #
633 # The default value for DIRMODE is specified in usr/src/Makefile.master.
634 #

636 $(ROOT)/var/adm \
637 $(ROOT)/var/adm/sa :=           DIRMODE= 775

639 $(ROOT)/var/spool/lp:=          DIRMODE= 775

641 # file mode
642 #
643 $(ROOT)/tmp \
644 $(ROOT)/var/krb5/rcache \
645 $(ROOT)/var/preserve \
646 $(ROOT)/var/spool/pkg \
647 $(ROOT)/var/spool/uucppublic \
648 $(ROOT)/var/tmp:=       DIRMODE= 1777

650 $(ROOT)/root:=          DIRMODE= 700

652 $(ROOT)/var/krb5/rcache/root:=  DIRMODE= 700


655 #
```

```
# These permissions must match those set
# in the package manifests.
#
$(ROOT)/var/sadm/pkg \
$(ROOT)/var/sadm/security \
$(ROOT)/var/sadm/install/logs :=        DIRMODE= 555


#
# These permissions must match the ones set
# internally by fdfs and autofs.
#
$(ROOT)/dev/fd \
$(ROOT)/home:=          DIRMODE= 555

$(ROOT)/var/mail:=      DIRMODE=1777

$(ROOT)/proc:=          DIRMODE= 555

$(ROOT)/system/contract:=       DIRMODE= 555
$(ROOT)/system/object:=         DIRMODE= 555

# symlink assignments, LINKDEST is the value of the symlink
#
$(ROOT)/usr/lib/cron:=                  LINKDEST=../../etc/cron.d
$(ROOT)/bin:=                           LINKDEST=usr/bin
$(ROOT)/lib/32:=                        LINKDEST=.
$(ROOT)/lib/crypto/32:=                 LINKDEST=.
$(ROOT)/lib/secure/32:=                 LINKDEST=.
$(ROOT)/dev/stdin:=                     LINKDEST=fd/0
$(ROOT)/dev/stdout:=                    LINKDEST=fd/1
$(ROOT)/dev/stderr:=                    LINKDEST=fd/2
$(ROOT)/usr/pub:=                       LINKDEST=share/lib/pub
$(ROOT)/usr/man:=                       LINKDEST=share/man
$(ROOT)/usr/src:=                       LINKDEST=share/src
$(ROOT)/usr/adm:=                       LINKDEST=../var/adm
$(ROOT)/etc/lib/ld.so.1:=               LINKDEST=../../lib/ld.so.1
$(ROOT)/etc/lib/libdl.so.1:=            LINKDEST=../../lib/libdl.so.1
$(ROOT)/etc/lib/nss_files.so.1:=        LINKDEST=../../lib/nss_files.so.1
$(ROOT)/etc/log:=                       LINKDEST=../var/adm/log
$(ROOT)/usr/mail:=                      LINKDEST=../var/mail
$(ROOT)/usr/news:=                      LINKDEST=../var/news
$(ROOT)/usr/preserve:=                  LINKDEST=../var/preserve
$(ROOT)/usr/spool:=                     LINKDEST=../var/spool
$(ROOT)/usr/tmp:=                       LINKDEST=../var/tmp
$(ROOT)/usr/lib/tmac:=                  LINKDEST=../share/lib/tmac
$(ROOT)/usr/lib/32:=                    LINKDEST=.
$(ROOT)/usr/lib/elfedit/32:=            LINKDEST=.
$(ROOT)/usr/lib/libp/32:=               LINKDEST=.
$(ROOT)/usr/lib/lwp/32:=                LINKDEST=.
$(ROOT)/usr/lib/link_audit/32:=         LINKDEST=.
$(ROOT)/usr/lib/secure/32:=             LINKDEST=.
$(ROOT)/usr/ccs/lib/link_audit:=        LINKDEST=../../lib/link_audit
$(ROOT)/var/ld/32:=                     LINKDEST=.
$(ROOT)/usr/ucblib/32:=                 LINKDEST=.


$(BUILD64) $(ROOT)/lib/64:=             LINKDEST=$(MACH64)
$(BUILD64) $(ROOT)/lib/crypto/64:=      LINKDEST=$(MACH64)
$(BUILD64) $(ROOT)/lib/secure/64:=      LINKDEST=$(MACH64)
$(BUILD64) $(ROOT)/usr/lib/64:=         LINKDEST=$(MACH64)
$(BUILD64) $(ROOT)/usr/lib/elfedit/64:= LINKDEST=$(MACH64)
$(BUILD64) $(ROOT)/usr/lib/brand/lx/64:=        LINKDEST=$(MACH64)
#endif /* ! codereview */
$(BUILD64) $(ROOT)/usr/lib/brand/sn1/64:=       LINKDEST=$(MACH64)
$(BUILD64) $(ROOT)/usr/lib/brand/solaris10/64:= LINKDEST=$(MACH64)
```



```
$(BUILD64) $(ROOT)/usr/lib/libp/64:=    LINKDEST=$(MACH64)
$(BUILD64) $(ROOT)/usr/lib/lwp/64:=     LINKDEST=$(MACH64)
$(BUILD64) $(ROOT)/usr/lib/link_audit/64:=      LINKDEST=$(MACH64)
$(BUILD64) $(ROOT)/usr/lib/secure/64:=  LINKDEST=$(MACH64)
$(BUILD64) $(ROOT)/usr/lib/security/64:=        LINKDEST=$(MACH64)
$(BUILD64) $(ROOT)/usr/xpg4/lib/64:=    LINKDEST=$(MACH64)
$(BUILD64) $(ROOT)/var/ld/64:=          LINKDEST=$(MACH64)
$(BUILD64) $(ROOT)/usr/ucblib/64:=      LINKDEST=$(MACH64)

#
# Installing a directory symlink calls for overriding INS.dir to install
# a symlink.
#
$(DIRLINKS:%=$(ROOT)%):= \
    INS.dir= -$(RM) -r $@; $(SYMLINK) $(LINKDEST) $@

# Special symlinks to populate usr/ccs/lib, whose objects
# have actually been moved to usr/lib
# Rather than adding another set of rules, we add usr/lib/lwp files here
$(ROOT)/usr/ccs/lib/libcurses.so:=      REALPATH=../../../lib/libcurses.so.1
$(ROOT)/usr/ccs/lib/llib-lcurses:=      REALPATH=../../../lib/llib-lcurses
$(ROOT)/usr/ccs/lib/llib-lcurses.ln:=   REALPATH=../../../lib/llib-lcurses.ln
$(ROOT)/usr/ccs/lib/libform.so:=        REALPATH=../../lib/libform.so.1
$(ROOT)/usr/ccs/lib/llib-lform:=        REALPATH=../../lib/llib-lform
$(ROOT)/usr/ccs/lib/llib-lform.ln:=     REALPATH=../../lib/llib-lform.ln
$(ROOT)/usr/ccs/lib/libgen.so:=         REALPATH=../../../lib/libgen.so.1
$(ROOT)/usr/ccs/lib/llib-lgen:=         REALPATH=../../../lib/llib-lgen
$(ROOT)/usr/ccs/lib/llib-lgen.ln:=      REALPATH=../../../lib/llib-lgen.ln
$(ROOT)/usr/ccs/lib/libmalloc.so:=      REALPATH=../../lib/libmalloc.so.1
$(ROOT)/usr/ccs/lib/libmenu.so:=        REALPATH=../../lib/libmenu.so.1
$(ROOT)/usr/ccs/lib/llib-lmenu:=        REALPATH=../../lib/llib-lmenu
$(ROOT)/usr/ccs/lib/llib-lmenu.ln:=     REALPATH=../../lib/llib-lmenu.ln
$(ROOT)/usr/ccs/lib/libpanel.so:=       REALPATH=../../lib/libpanel.so.1
$(ROOT)/usr/ccs/lib/llib-lpanel:=       REALPATH=../../lib/llib-lpanel
$(ROOT)/usr/ccs/lib/llib-lpanel.ln:=    REALPATH=../../lib/llib-lpanel.ln
$(ROOT)/usr/ccs/lib/libtermlib.so:=     REALPATH=../../../lib/libcurses.so.1
$(ROOT)/usr/ccs/lib/llib-ltermlib:=     REALPATH=../../../lib/llib-lcurses
$(ROOT)/usr/ccs/lib/llib-ltermlib.ln:=  REALPATH=../../../lib/llib-lcurses.ln
$(ROOT)/usr/ccs/lib/libtermcap.so:=     REALPATH=../../../lib/libtermcap.so.1
$(ROOT)/usr/ccs/lib/llib-ltermcap:=     REALPATH=../../../lib/llib-ltermcap
$(ROOT)/usr/ccs/lib/llib-ltermcap.ln:=  REALPATH=../../../lib/llib-ltermcap.ln
$(ROOT)/usr/ccs/lib/values-Xa.o:=       REALPATH=../../lib/values-Xa.o
$(ROOT)/usr/ccs/lib/values-Xc.o:=       REALPATH=../../lib/values-Xc.o
$(ROOT)/usr/ccs/lib/values-Xs.o:=       REALPATH=../../lib/values-Xs.o
$(ROOT)/usr/ccs/lib/values-Xt.o:=       REALPATH=../../lib/values-Xt.o
$(ROOT)/usr/ccs/lib/values-xpg4.o:=     REALPATH=../../lib/values-xpg4.o
$(ROOT)/usr/ccs/lib/values-xpg6.o:=     REALPATH=../../lib/values-xpg6.o
$(ROOT)/usr/ccs/lib/libl.so:=           REALPATH=../../lib/libl.so.1
$(ROOT)/usr/ccs/lib/llib-ll.ln:=        REALPATH=../../lib/llib-ll.ln
$(ROOT)/usr/ccs/lib/liby.so:=           REALPATH=../../lib/liby.so.1
$(ROOT)/usr/ccs/lib/llib-ly.ln:=        REALPATH=../../lib/llib-ly.ln
$(ROOT)/usr/lib/libp/libc.so.1:=        REALPATH=../../../lib/libc.so.1
$(ROOT)/usr/lib/lwp/libthread.so.1:=    REALPATH=../libthread.so.1
$(ROOT)/usr/lib/lwp/libthread_db.so.1:= REALPATH=../libthread_db.so.1

# symlinks to populate usr/ccs/lib/$(MACH64)
$(ROOT)/usr/ccs/lib/$(MACH64)/libcurses.so:= \
        REALPATH=../../../../lib/$(MACH64)/libcurses.so.1
$(ROOT)/usr/ccs/lib/$(MACH64)/llib-lcurses.ln:= \
        REALPATH=../../../../lib/$(MACH64)/llib-lcurses.ln
$(ROOT)/usr/ccs/lib/$(MACH64)/libform.so:= \
        REALPATH=../../../lib/$(MACH64)/libform.so.1
$(ROOT)/usr/ccs/lib/$(MACH64)/llib-lform.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lform.ln
$(ROOT)/usr/ccs/lib/$(MACH64)/libgen.so:= \
        REALPATH=../../../../lib/$(MACH64)/libgen.so.1
```

```
$(ROOT)/usr/ccs/lib/$(MACH64)/llib-lgen.ln:= \
        REALPATH=../../../../lib/$(MACH64)/llib-lgen.ln
$(ROOT)/usr/ccs/lib/$(MACH64)/libmalloc.so:= \
        REALPATH=../../../lib/$(MACH64)/libmalloc.so.1
$(ROOT)/usr/ccs/lib/$(MACH64)/libmenu.so:= \
        REALPATH=../../../lib/$(MACH64)/libmenu.so.1
$(ROOT)/usr/ccs/lib/$(MACH64)/llib-lmenu.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lmenu.ln
$(ROOT)/usr/ccs/lib/$(MACH64)/libpanel.so:= \
        REALPATH=../../../lib/$(MACH64)/libpanel.so.1
$(ROOT)/usr/ccs/lib/$(MACH64)/llib-lpanel.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lpanel.ln
$(ROOT)/usr/ccs/lib/$(MACH64)/libtermlib.so:= \
        REALPATH=../../../../lib/$(MACH64)/libcurses.so.1
$(ROOT)/usr/ccs/lib/$(MACH64)/llib-ltermlib.ln:= \
        REALPATH=../../../../lib/$(MACH64)/llib-lcurses.ln
$(ROOT)/usr/ccs/lib/$(MACH64)/libtermcap.so:= \
        REALPATH=../../../../lib/$(MACH64)/libtermcap.so.1
$(ROOT)/usr/ccs/lib/$(MACH64)/llib-ltermcap.ln:= \
        REALPATH=../../../../lib/$(MACH64)/llib-ltermcap.ln
$(ROOT)/usr/ccs/lib/$(MACH64)/values-Xa.o:= \
        REALPATH=../../../lib/$(MACH64)/values-Xa.o
$(ROOT)/usr/ccs/lib/$(MACH64)/values-Xc.o:= \
        REALPATH=../../../lib/$(MACH64)/values-Xc.o
$(ROOT)/usr/ccs/lib/$(MACH64)/values-Xs.o:= \
        REALPATH=../../../lib/$(MACH64)/values-Xs.o
$(ROOT)/usr/ccs/lib/$(MACH64)/values-Xt.o:= \
        REALPATH=../../../lib/$(MACH64)/values-Xt.o
$(ROOT)/usr/ccs/lib/$(MACH64)/values-xpg4.o:= \
        REALPATH=../../../lib/$(MACH64)/values-xpg4.o
$(ROOT)/usr/ccs/lib/$(MACH64)/values-xpg6.o:= \
        REALPATH=../../../lib/$(MACH64)/values-xpg6.o
$(ROOT)/usr/ccs/lib/$(MACH64)/libl.so:= \
        REALPATH=../../../lib/$(MACH64)/libl.so.1
$(ROOT)/usr/ccs/lib/$(MACH64)/llib-ll.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-ll.ln
$(ROOT)/usr/ccs/lib/$(MACH64)/liby.so:= \
        REALPATH=../../../lib/$(MACH64)/liby.so.1
$(ROOT)/usr/ccs/lib/$(MACH64)/llib-ly.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-ly.ln
$(ROOT)/usr/lib/libp/$(MACH64)/libc.so.1:= \
        REALPATH=../../../../lib/$(MACH64)/libc.so.1
$(ROOT)/usr/lib/lwp/$(MACH64)/libthread.so.1:= \
        REALPATH=../../$(MACH64)/libthread.so.1
$(ROOT)/usr/lib/lwp/$(MACH64)/libthread_db.so.1:= \
        REALPATH=../../$(MACH64)/libthread_db.so.1

SYM.USRCCSLIB= \
        /usr/ccs/lib/libcurses.so \
        /usr/ccs/lib/llib-lcurses \
        /usr/ccs/lib/llib-lcurses.ln \
        /usr/ccs/lib/libform.so \
        /usr/ccs/lib/llib-lform \
        /usr/ccs/lib/llib-lform.ln \
        /usr/ccs/lib/libgen.so \
        /usr/ccs/lib/llib-lgen \
        /usr/ccs/lib/llib-lgen.ln \
        /usr/ccs/lib/libmalloc.so \
        /usr/ccs/lib/libmenu.so \
        /usr/ccs/lib/llib-lmenu \
        /usr/ccs/lib/llib-lmenu.ln \
        /usr/ccs/lib/libpanel.so \
        /usr/ccs/lib/llib-lpanel \
        /usr/ccs/lib/llib-lpanel.ln \
        /usr/ccs/lib/libtermlib.so \
        /usr/ccs/lib/llib-ltermlib \
```



```
        /usr/ccs/lib/llib-ltermlib.ln \
        /usr/ccs/lib/libtermcap.so \
        /usr/ccs/lib/llib-ltermcap \
        /usr/ccs/lib/llib-ltermcap.ln \
        /usr/ccs/lib/values-Xa.o \
        /usr/ccs/lib/values-Xc.o \
        /usr/ccs/lib/values-Xs.o \
        /usr/ccs/lib/values-Xt.o \
        /usr/ccs/lib/values-xpg4.o \
        /usr/ccs/lib/values-xpg6.o \
        /usr/ccs/lib/libl.so \
        /usr/ccs/lib/llib-ll.ln \
        /usr/ccs/lib/liby.so \
        /usr/ccs/lib/llib-ly.ln \
        /usr/lib/libp/libc.so.1 \
        /usr/lib/lwp/libthread.so.1 \
        /usr/lib/lwp/libthread_db.so.1

SYM.USRCCSLIB64= \
        /usr/ccs/lib/$(MACH64)/libcurses.so \
        /usr/ccs/lib/$(MACH64)/llib-lcurses.ln \
        /usr/ccs/lib/$(MACH64)/libform.so \
        /usr/ccs/lib/$(MACH64)/llib-lform.ln \
        /usr/ccs/lib/$(MACH64)/libgen.so \
        /usr/ccs/lib/$(MACH64)/llib-lgen.ln \
        /usr/ccs/lib/$(MACH64)/libmalloc.so \
        /usr/ccs/lib/$(MACH64)/libmenu.so \
        /usr/ccs/lib/$(MACH64)/llib-lmenu.ln \
        /usr/ccs/lib/$(MACH64)/libpanel.so \
        /usr/ccs/lib/$(MACH64)/llib-lpanel.ln \
        /usr/ccs/lib/$(MACH64)/libtermlib.so \
        /usr/ccs/lib/$(MACH64)/llib-ltermlib.ln \
        /usr/ccs/lib/$(MACH64)/libtermcap.so \
        /usr/ccs/lib/$(MACH64)/llib-ltermcap.ln \
        /usr/ccs/lib/$(MACH64)/values-Xa.o \
        /usr/ccs/lib/$(MACH64)/values-Xc.o \
        /usr/ccs/lib/$(MACH64)/values-Xs.o \
        /usr/ccs/lib/$(MACH64)/values-Xt.o \
        /usr/ccs/lib/$(MACH64)/values-xpg4.o \
        /usr/ccs/lib/$(MACH64)/values-xpg6.o \
        /usr/ccs/lib/$(MACH64)/libl.so \
        /usr/ccs/lib/$(MACH64)/llib-ll.ln \
        /usr/ccs/lib/$(MACH64)/liby.so \
        /usr/ccs/lib/$(MACH64)/llib-ly.ln \
        /usr/lib/libp/$(MACH64)/libc.so.1 \
        /usr/lib/lwp/$(MACH64)/libthread.so.1 \
        /usr/lib/lwp/$(MACH64)/libthread_db.so.1

# Special symlinks to direct libraries that have been moved
# from /usr/lib to /lib in order to live in the root filesystem.
$(ROOT)/lib/libposix4.so.1:=            REALPATH=librt.so.1
$(ROOT)/lib/libposix4.so:=              REALPATH=libposix4.so.1
$(ROOT)/lib/llib-lposix4:=              REALPATH=llib-lrt
$(ROOT)/lib/llib-lposix4.ln:=           REALPATH=llib-lrt.ln
$(ROOT)/lib/libthread_db.so.1:=         REALPATH=libc_db.so.1
$(ROOT)/lib/libthread_db.so:=           REALPATH=libc_db.so.1
$(ROOT)/usr/lib/ld.so.1:=               REALPATH=../../lib/ld.so.1
$(ROOT)/usr/lib/libadm.so.1:=           REALPATH=../../lib/libadm.so.1
$(ROOT)/usr/lib/libadm.so:=             REALPATH=../../lib/libadm.so.1
$(ROOT)/usr/lib/libaio.so.1:=           REALPATH=../../lib/libaio.so.1
$(ROOT)/usr/lib/libaio.so:=             REALPATH=../../lib/libaio.so.1
$(ROOT)/usr/lib/libavl.so.1:=           REALPATH=../../lib/libavl.so.1
$(ROOT)/usr/lib/libavl.so:=             REALPATH=../../lib/libavl.so.1
$(ROOT)/usr/lib/libbsm.so.1:=           REALPATH=../../lib/libbsm.so.1
$(ROOT)/usr/lib/libbsm.so:=             REALPATH=../../lib/libbsm.so.1
$(ROOT)/usr/lib/libc.so.1:=             REALPATH=../../lib/libc.so.1
```

```
$(ROOT)/usr/lib/libc.so:=                REALPATH=../../lib/libc.so.1
$(ROOT)/usr/lib/libc_db.so.1:=           REALPATH=../../lib/libc_db.so.1
$(ROOT)/usr/lib/libc_db.so:=             REALPATH=../../lib/libc_db.so.1
$(ROOT)/usr/lib/libcmdutils.so.1:=       REALPATH=../../lib/libcmdutils.so.1
$(ROOT)/usr/lib/libcmdutils.so:=         REALPATH=../../lib/libcmdutils.so.1
$(ROOT)/usr/lib/libcontract.so.1:=       REALPATH=../../lib/libcontract.so.1
$(ROOT)/usr/lib/libcontract.so:=         REALPATH=../../lib/libcontract.so.1
$(ROOT)/usr/lib/libcryptoutil.so.1:=     REALPATH=../../lib/libcryptoutil.so.1
$(ROOT)/usr/lib/libcryptoutil.so:=       REALPATH=../../lib/libcryptoutil.so.1
$(ROOT)/usr/lib/libctf.so.1:=            REALPATH=../../lib/libctf.so.1
$(ROOT)/usr/lib/libctf.so:=              REALPATH=../../lib/libctf.so.1
$(ROOT)/usr/lib/libcurses.so.1:=         REALPATH=../../lib/libcurses.so.1
$(ROOT)/usr/lib/libcurses.so:=           REALPATH=../../lib/libcurses.so.1
$(ROOT)/usr/lib/libdevice.so.1:=         REALPATH=../../lib/libdevice.so.1
$(ROOT)/usr/lib/libdevice.so:=           REALPATH=../../lib/libdevice.so.1
$(ROOT)/usr/lib/libdevid.so.1:=          REALPATH=../../lib/libdevid.so.1
$(ROOT)/usr/lib/libdevid.so:=            REALPATH=../../lib/libdevid.so.1
$(ROOT)/usr/lib/libdevinfo.so.1:=        REALPATH=../../lib/libdevinfo.so.1
$(ROOT)/usr/lib/libdevinfo.so:=          REALPATH=../../lib/libdevinfo.so.1
$(ROOT)/usr/lib/libdhcpagent.so.1:=      REALPATH=../../lib/libdhcpagent.so.1
$(ROOT)/usr/lib/libdhcpagent.so:=        REALPATH=../../lib/libdhcpagent.so.1
$(ROOT)/usr/lib/libdhcputil.so.1:=       REALPATH=../../lib/libdhcputil.so.1
$(ROOT)/usr/lib/libdhcputil.so:=         REALPATH=../../lib/libdhcputil.so.1
$(ROOT)/usr/lib/libdl.so.1:=             REALPATH=../../lib/libdl.so.1
$(ROOT)/usr/lib/libdl.so:=               REALPATH=../../lib/libdl.so.1
$(ROOT)/usr/lib/libdlpi.so.1:=           REALPATH=../../lib/libdlpi.so.1
$(ROOT)/usr/lib/libdlpi.so:=             REALPATH=../../lib/libdlpi.so.1
$(ROOT)/usr/lib/libdoor.so.1:=           REALPATH=../../lib/libdoor.so.1
$(ROOT)/usr/lib/libdoor.so:=             REALPATH=../../lib/libdoor.so.1
$(ROOT)/usr/lib/libefi.so.1:=            REALPATH=../../lib/libefi.so.1
$(ROOT)/usr/lib/libefi.so:=              REALPATH=../../lib/libefi.so.1
$(ROOT)/usr/lib/libelf.so.1:=            REALPATH=../../lib/libelf.so.1
$(ROOT)/usr/lib/libelf.so:=              REALPATH=../../lib/libelf.so.1
$(ROOT)/usr/lib/libfdisk.so.1:=          REALPATH=../../lib/libfdisk.so.1
$(ROOT)/usr/lib/libfdisk.so:=            REALPATH=../../lib/libfdisk.so.1
$(ROOT)/usr/lib/libgen.so.1:=            REALPATH=../../lib/libgen.so.1
$(ROOT)/usr/lib/libgen.so:=              REALPATH=../../lib/libgen.so.1
$(ROOT)/usr/lib/libinetutil.so.1:=       REALPATH=../../lib/libinetutil.so.1
$(ROOT)/usr/lib/libinetutil.so:=         REALPATH=../../lib/libinetutil.so.1
$(ROOT)/usr/lib/libintl.so.1:=           REALPATH=../../lib/libintl.so.1
$(ROOT)/usr/lib/libintl.so:=             REALPATH=../../lib/libintl.so.1
$(ROOT)/usr/lib/libkmf.so.1:=            REALPATH=../../lib/libkmf.so.1
$(ROOT)/usr/lib/libkmf.so:=              REALPATH=../../lib/libkmf.so.1
$(ROOT)/usr/lib/libkmfberder.so.1:=      REALPATH=../../lib/libkmfberder.so.1
$(ROOT)/usr/lib/libkmfberder.so:=        REALPATH=../../lib/libkmfberder.so.1
$(ROOT)/usr/lib/libkstat.so.1:=          REALPATH=../../lib/libkstat.so.1
$(ROOT)/usr/lib/libkstat.so:=            REALPATH=../../lib/libkstat.so.1
$(ROOT)/usr/lib/liblddbg.so.4:=          REALPATH=../../lib/liblddbg.so.4
$(ROOT)/usr/lib/libmd.so.1:=             REALPATH=../../lib/libmd.so.1
$(ROOT)/usr/lib/libmd.so:=               REALPATH=../../lib/libmd.so.1
$(ROOT)/usr/lib/libmd5.so.1:=            REALPATH=../../lib/libmd5.so.1
$(ROOT)/usr/lib/libmd5.so:=              REALPATH=../../lib/libmd5.so.1
$(ROOT)/usr/lib/libmeta.so.1:=           REALPATH=../../lib/libmeta.so.1
$(ROOT)/usr/lib/libmeta.so:=             REALPATH=../../lib/libmeta.so.1
$(ROOT)/usr/lib/libmp.so.1:=             REALPATH=../../lib/libmp.so.1
$(ROOT)/usr/lib/libmp.so.2:=             REALPATH=../../lib/libmp.so.2
$(ROOT)/usr/lib/libmp.so:=               REALPATH=../../lib/libmp.so.2
$(ROOT)/usr/lib/libnsl.so.1:=            REALPATH=../../lib/libnsl.so.1
$(ROOT)/usr/lib/libnsl.so:=              REALPATH=../../lib/libnsl.so.1
$(ROOT)/usr/lib/libnvpair.so.1:=         REALPATH=../../lib/libnvpair.so.1
$(ROOT)/usr/lib/libnvpair.so:=           REALPATH=../../lib/libnvpair.so.1
$(ROOT)/usr/lib/libpam.so.1:=            REALPATH=../../lib/libpam.so.1
$(ROOT)/usr/lib/libpam.so:=              REALPATH=../../lib/libpam.so.1
$(ROOT)/usr/lib/libposix4.so.1:=         REALPATH=../../lib/librt.so.1
$(ROOT)/usr/lib/libposix4.so:=           REALPATH=../../lib/librt.so.1
$(ROOT)/usr/lib/libproc.so.1:=           REALPATH=../../lib/libproc.so.1
```



```
$(ROOT)/usr/lib/libproc.so:=             REALPATH=../../lib/libproc.so.1
$(ROOT)/usr/lib/libpthread.so.1:=        REALPATH=../../lib/libpthread.so.1
$(ROOT)/usr/lib/libpthread.so:=          REALPATH=../../lib/libpthread.so.1
$(ROOT)/usr/lib/librcm.so.1:=            REALPATH=../../lib/librcm.so.1
$(ROOT)/usr/lib/librcm.so:=              REALPATH=../../lib/librcm.so.1
$(ROOT)/usr/lib/libresolv.so.1:=         REALPATH=../../lib/libresolv.so.1
$(ROOT)/usr/lib/libresolv.so.2:=         REALPATH=../../lib/libresolv.so.2
$(ROOT)/usr/lib/libresolv.so:=           REALPATH=../../lib/libresolv.so.2
$(ROOT)/usr/lib/librestart.so.1:=        REALPATH=../../lib/librestart.so.1
$(ROOT)/usr/lib/librestart.so:=          REALPATH=../../lib/librestart.so.1
$(ROOT)/usr/lib/librpcsvc.so.1:=         REALPATH=../../lib/librpcsvc.so.1
$(ROOT)/usr/lib/librpcsvc.so:=           REALPATH=../../lib/librpcsvc.so.1
$(ROOT)/usr/lib/librt.so.1:=             REALPATH=../../lib/librt.so.1
$(ROOT)/usr/lib/librt.so:=               REALPATH=../../lib/librt.so.1
$(ROOT)/usr/lib/librtld.so.1:=           REALPATH=../../lib/librtld.so.1
$(ROOT)/usr/lib/librtld_db.so.1:=        REALPATH=../../lib/librtld_db.so.1
$(ROOT)/usr/lib/librtld_db.so:=          REALPATH=../../lib/librtld_db.so.1
$(ROOT)/usr/lib/libscf.so.1:=            REALPATH=../../lib/libscf.so.1
$(ROOT)/usr/lib/libscf.so:=              REALPATH=../../lib/libscf.so.1
$(ROOT)/usr/lib/libsec.so.1:=            REALPATH=../../lib/libsec.so.1
$(ROOT)/usr/lib/libsec.so:=              REALPATH=../../lib/libsec.so.1
$(ROOT)/usr/lib/libsecdb.so.1:=          REALPATH=../../lib/libsecdb.so.1
$(ROOT)/usr/lib/libsecdb.so:=            REALPATH=../../lib/libsecdb.so.1
$(ROOT)/usr/lib/libsendfile.so.1:=       REALPATH=../../lib/libsendfile.so.1
$(ROOT)/usr/lib/libsendfile.so:=         REALPATH=../../lib/libsendfile.so.1
$(ROOT)/usr/lib/libsocket.so.1:=         REALPATH=../../lib/libsocket.so.1
$(ROOT)/usr/lib/libsocket.so:=           REALPATH=../../lib/libsocket.so.1
$(ROOT)/usr/lib/libsysevent.so.1:=       REALPATH=../../lib/libsysevent.so.1
$(ROOT)/usr/lib/libsysevent.so:=         REALPATH=../../lib/libsysevent.so.1
$(ROOT)/usr/lib/libtermcap.so.1:=        REALPATH=../../lib/libtermcap.so.1
$(ROOT)/usr/lib/libtermcap.so:=          REALPATH=../../lib/libtermcap.so.1
$(ROOT)/usr/lib/libtermlib.so.1:=        REALPATH=../../lib/libcurses.so.1
$(ROOT)/usr/lib/libtermlib.so:=          REALPATH=../../lib/libcurses.so.1
$(ROOT)/usr/lib/libthread.so.1:=         REALPATH=../../lib/libthread.so.1
$(ROOT)/usr/lib/libthread.so:=           REALPATH=../../lib/libthread.so.1
$(ROOT)/usr/lib/libthread_db.so.1:=      REALPATH=../../lib/libc_db.so.1
$(ROOT)/usr/lib/libthread_db.so:=        REALPATH=../../lib/libc_db.so.1
$(ROOT)/usr/lib/libtsnet.so.1:=          REALPATH=../../lib/libtsnet.so.1
$(ROOT)/usr/lib/libtsnet.so:=            REALPATH=../../lib/libtsnet.so.1
$(ROOT)/usr/lib/libtsol.so.2:=           REALPATH=../../lib/libtsol.so.2
$(ROOT)/usr/lib/libtsol.so:=             REALPATH=../../lib/libtsol.so.2
$(ROOT)/usr/lib/libumem.so.1:=           REALPATH=../../lib/libumem.so.1
$(ROOT)/usr/lib/libumem.so:=             REALPATH=../../lib/libumem.so.1
$(ROOT)/usr/lib/libuuid.so.1:=           REALPATH=../../lib/libuuid.so.1
$(ROOT)/usr/lib/libuuid.so:=             REALPATH=../../lib/libuuid.so.1
$(ROOT)/usr/lib/libuutil.so.1:=          REALPATH=../../lib/libuutil.so.1
$(ROOT)/usr/lib/libuutil.so:=            REALPATH=../../lib/libuutil.so.1
$(ROOT)/usr/lib/libw.so.1:=              REALPATH=../../lib/libw.so.1
$(ROOT)/usr/lib/libw.so:=                REALPATH=../../lib/libw.so.1
$(ROOT)/usr/lib/libxnet.so.1:=           REALPATH=../../lib/libxnet.so.1
$(ROOT)/usr/lib/libxnet.so:=             REALPATH=../../lib/libxnet.so.1
$(ROOT)/usr/lib/libzfs.so.1:=            REALPATH=../../lib/libzfs.so.1
$(ROOT)/usr/lib/libzfs.so:=              REALPATH=../../lib/libzfs.so.1
$(ROOT)/usr/lib/libzfs_core.so.1:=       REALPATH=../../lib/libzfs_core.so.1
$(ROOT)/usr/lib/libzfs_core.so:=         REALPATH=../../lib/libzfs_core.so.1
$(ROOT)/usr/lib/llib-ladm.ln:=           REALPATH=../../lib/llib-ladm.ln
$(ROOT)/usr/lib/llib-ladm:=              REALPATH=../../lib/llib-ladm
$(ROOT)/usr/lib/llib-laio.ln:=           REALPATH=../../lib/llib-laio.ln
$(ROOT)/usr/lib/llib-laio:=              REALPATH=../../lib/llib-laio
$(ROOT)/usr/lib/llib-lavl.ln:=           REALPATH=../../lib/llib-lavl.ln
$(ROOT)/usr/lib/llib-lavl:=              REALPATH=../../lib/llib-lavl
$(ROOT)/usr/lib/llib-lbsm.ln:=           REALPATH=../../lib/llib-lbsm.ln
$(ROOT)/usr/lib/llib-lbsm:=              REALPATH=../../lib/llib-lbsm
$(ROOT)/usr/lib/llib-lc.ln:=             REALPATH=../../lib/llib-lc.ln
$(ROOT)/usr/lib/llib-lc:=                REALPATH=../../lib/llib-lc
$(ROOT)/usr/lib/llib-lcmdutils.ln:=      REALPATH=../../lib/llib-lcmdutils.ln
```

```
$(ROOT)/usr/lib/llib-lcmdutils:=        REALPATH=../../lib/llib-lcmdutils
$(ROOT)/usr/lib/llib-lcontract.ln:=     REALPATH=../../lib/llib-lcontract.ln
$(ROOT)/usr/lib/llib-lcontract:=        REALPATH=../../lib/llib-lcontract
$(ROOT)/usr/lib/llib-lctf.ln:=          REALPATH=../../lib/llib-lctf.ln
$(ROOT)/usr/lib/llib-lctf:=             REALPATH=../../lib/llib-lctf
$(ROOT)/usr/lib/llib-lcurses.ln:=       REALPATH=../../lib/llib-lcurses.ln
$(ROOT)/usr/lib/llib-lcurses:=          REALPATH=../../lib/llib-lcurses
$(ROOT)/usr/lib/llib-ldevice.ln:=       REALPATH=../../lib/llib-ldevice.ln
$(ROOT)/usr/lib/llib-ldevice:=          REALPATH=../../lib/llib-ldevice
$(ROOT)/usr/lib/llib-ldevid.ln:=        REALPATH=../../lib/llib-ldevid.ln
$(ROOT)/usr/lib/llib-ldevid:=           REALPATH=../../lib/llib-ldevid
$(ROOT)/usr/lib/llib-ldevinfo.ln:=      REALPATH=../../lib/llib-ldevinfo.ln
$(ROOT)/usr/lib/llib-ldevinfo:=         REALPATH=../../lib/llib-ldevinfo
$(ROOT)/usr/lib/llib-ldhcpagent.ln:=    REALPATH=../../lib/llib-ldhcpagent.ln
$(ROOT)/usr/lib/llib-ldhcpagent:=       REALPATH=../../lib/llib-ldhcpagent
$(ROOT)/usr/lib/llib-ldhcputil.ln:=     REALPATH=../../lib/llib-ldhcputil.ln
$(ROOT)/usr/lib/llib-ldhcputil:=        REALPATH=../../lib/llib-ldhcputil
$(ROOT)/usr/lib/llib-ldl.ln:=           REALPATH=../../lib/llib-ldl.ln
$(ROOT)/usr/lib/llib-ldl:=              REALPATH=../../lib/llib-ldl
$(ROOT)/usr/lib/llib-ldoor.ln:=         REALPATH=../../lib/llib-ldoor.ln
$(ROOT)/usr/lib/llib-ldoor:=            REALPATH=../../lib/llib-ldoor
$(ROOT)/usr/lib/llib-lefi.ln:=          REALPATH=../../lib/llib-lefi.ln
$(ROOT)/usr/lib/llib-lefi:=             REALPATH=../../lib/llib-lefi
$(ROOT)/usr/lib/llib-lelf.ln:=          REALPATH=../../lib/llib-lelf.ln
$(ROOT)/usr/lib/llib-lelf:=             REALPATH=../../lib/llib-lelf
$(ROOT)/usr/lib/llib-lfdisk.ln:=        REALPATH=../../lib/llib-lfdisk.ln
$(ROOT)/usr/lib/llib-lfdisk:=           REALPATH=../../lib/llib-lfdisk
$(ROOT)/usr/lib/llib-lgen.ln:=          REALPATH=../../lib/llib-lgen.ln
$(ROOT)/usr/lib/llib-lgen:=             REALPATH=../../lib/llib-lgen
$(ROOT)/usr/lib/llib-linetutil.ln:=     REALPATH=../../lib/llib-linetutil.ln
$(ROOT)/usr/lib/llib-linetutil:=        REALPATH=../../lib/llib-linetutil
$(ROOT)/usr/lib/llib-lintl.ln:=         REALPATH=../../lib/llib-lintl.ln
$(ROOT)/usr/lib/llib-lintl:=            REALPATH=../../lib/llib-lintl
$(ROOT)/usr/lib/llib-lkstat.ln:=        REALPATH=../../lib/llib-lkstat.ln
$(ROOT)/usr/lib/llib-lkstat:=           REALPATH=../../lib/llib-lkstat
$(ROOT)/usr/lib/llib-lmd5.ln:=          REALPATH=../../lib/llib-lmd5.ln
$(ROOT)/usr/lib/llib-lmd5:=             REALPATH=../../lib/llib-lmd5
$(ROOT)/usr/lib/llib-lmeta.ln:=         REALPATH=../../lib/llib-lmeta.ln
$(ROOT)/usr/lib/llib-lmeta:=            REALPATH=../../lib/llib-lmeta
$(ROOT)/usr/lib/llib-lnsl.ln:=          REALPATH=../../lib/llib-lnsl.ln
$(ROOT)/usr/lib/llib-lnsl:=             REALPATH=../../lib/llib-lnsl
$(ROOT)/usr/lib/llib-lnvpair.ln:=       REALPATH=../../lib/llib-lnvpair.ln
$(ROOT)/usr/lib/llib-lnvpair:=          REALPATH=../../lib/llib-lnvpair
$(ROOT)/usr/lib/llib-lpam.ln:=          REALPATH=../../lib/llib-lpam.ln
$(ROOT)/usr/lib/llib-lpam:=             REALPATH=../../lib/llib-lpam
$(ROOT)/usr/lib/llib-lposix4.ln:=       REALPATH=../../lib/llib-lrt.ln
$(ROOT)/usr/lib/llib-lposix4:=          REALPATH=../../lib/llib-lrt
$(ROOT)/usr/lib/llib-lpthread.ln:=      REALPATH=../../lib/llib-lpthread.ln
$(ROOT)/usr/lib/llib-lpthread:=         REALPATH=../../lib/llib-lpthread
$(ROOT)/usr/lib/llib-lresolv.ln:=       REALPATH=../../lib/llib-lresolv.ln
$(ROOT)/usr/lib/llib-lresolv:=          REALPATH=../../lib/llib-lresolv
$(ROOT)/usr/lib/llib-lrpcsvc.ln:=       REALPATH=../../lib/llib-lrpcsvc.ln
$(ROOT)/usr/lib/llib-lrpcsvc:=          REALPATH=../../lib/llib-lrpcsvc
$(ROOT)/usr/lib/llib-lrt.ln:=           REALPATH=../../lib/llib-lrt.ln
$(ROOT)/usr/lib/llib-lrt:=              REALPATH=../../lib/llib-lrt
$(ROOT)/usr/lib/llib-lrtld_db.ln:=      REALPATH=../../lib/llib-lrtld_db.ln
$(ROOT)/usr/lib/llib-lrtld_db:=         REALPATH=../../lib/llib-lrtld_db
$(ROOT)/usr/lib/llib-lscf.ln:=          REALPATH=../../lib/llib-lscf.ln
$(ROOT)/usr/lib/llib-lscf:=             REALPATH=../../lib/llib-lscf
$(ROOT)/usr/lib/llib-lsec.ln:=          REALPATH=../../lib/llib-lsec.ln
$(ROOT)/usr/lib/llib-lsec:=             REALPATH=../../lib/llib-lsec
$(ROOT)/usr/lib/llib-lsecdb.ln:=        REALPATH=../../lib/llib-lsecdb.ln
$(ROOT)/usr/lib/llib-lsecdb:=           REALPATH=../../lib/llib-lsecdb
$(ROOT)/usr/lib/llib-lsendfile.ln:=     REALPATH=../../lib/llib-lsendfile.ln
$(ROOT)/usr/lib/llib-lsendfile:=        REALPATH=../../lib/llib-lsendfile
$(ROOT)/usr/lib/llib-lsocket.ln:=       REALPATH=../../lib/llib-lsocket.ln
```



```
$(ROOT)/usr/lib/llib-lsocket:=          REALPATH=../../lib/llib-lsocket
$(ROOT)/usr/lib/llib-lsysevent.ln:=     REALPATH=../../lib/llib-lsysevent.ln
$(ROOT)/usr/lib/llib-lsysevent:=        REALPATH=../../lib/llib-lsysevent
$(ROOT)/usr/lib/llib-ltermcap.ln:=      REALPATH=../../lib/llib-ltermcap.ln
$(ROOT)/usr/lib/llib-ltermcap:=         REALPATH=../../lib/llib-ltermcap
$(ROOT)/usr/lib/llib-ltermlib.ln:=      REALPATH=../../lib/llib-lcurses.ln
$(ROOT)/usr/lib/llib-ltermlib:=         REALPATH=../../lib/llib-lcurses
$(ROOT)/usr/lib/llib-lthread.ln:=       REALPATH=../../lib/llib-lthread.ln
$(ROOT)/usr/lib/llib-lthread:=          REALPATH=../../lib/llib-lthread
$(ROOT)/usr/lib/llib-lthread_db.ln:=    REALPATH=../../lib/llib-lc_db.ln
$(ROOT)/usr/lib/llib-lthread_db:=       REALPATH=../../lib/llib-lc_db
$(ROOT)/usr/lib/llib-ltsnet.ln:=        REALPATH=../../lib/llib-ltsnet.ln
$(ROOT)/usr/lib/llib-ltsnet:=           REALPATH=../../lib/llib-ltsnet
$(ROOT)/usr/lib/llib-ltsol.ln:=         REALPATH=../../lib/llib-ltsol.ln
$(ROOT)/usr/lib/llib-ltsol:=            REALPATH=../../lib/llib-ltsol
$(ROOT)/usr/lib/llib-lumem.ln:=         REALPATH=../../lib/llib-lumem.ln
$(ROOT)/usr/lib/llib-lumem:=            REALPATH=../../lib/llib-lumem
$(ROOT)/usr/lib/llib-luuid.ln:=         REALPATH=../../lib/llib-luuid.ln
$(ROOT)/usr/lib/llib-luuid:=            REALPATH=../../lib/llib-luuid
$(ROOT)/usr/lib/llib-lxnet.ln:=         REALPATH=../../lib/llib-lxnet.ln
$(ROOT)/usr/lib/llib-lxnet:=            REALPATH=../../lib/llib-lxnet
$(ROOT)/usr/lib/llib-lzfs.ln:=          REALPATH=../../lib/llib-lzfs.ln
$(ROOT)/usr/lib/llib-lzfs:=             REALPATH=../../lib/llib-lzfs
$(ROOT)/usr/lib/llib-lzfs_core.ln:=     REALPATH=../../lib/llib-lzfs_core.ln
$(ROOT)/usr/lib/llib-lzfs_core:=        REALPATH=../../lib/llib-lzfs_core
$(ROOT)/usr/lib/nss_compat.so.1:=       REALPATH=../../lib/nss_compat.so.1
$(ROOT)/usr/lib/nss_dns.so.1:=          REALPATH=../../lib/nss_dns.so.1
$(ROOT)/usr/lib/nss_files.so.1:=        REALPATH=../../lib/nss_files.so.1
$(ROOT)/usr/lib/nss_nis.so.1:=          REALPATH=../../lib/nss_nis.so.1
$(ROOT)/usr/lib/nss_user.so.1:=         REALPATH=../../lib/nss_user.so.1
$(ROOT)/usr/lib/fm/libfmevent.so.1:=    REALPATH=../../../lib/fm/libfmevent.so.1
$(ROOT)/usr/lib/fm/libfmevent.so:=      REALPATH=../../../lib/fm/libfmevent.so.1
$(ROOT)/usr/lib/fm/llib-lfmevent.ln:=   REALPATH=../../../lib/fm/llib-lfmevent.l
$(ROOT)/usr/lib/fm/llib-lfmevent:=      REALPATH=../../../lib/fm/llib-lfmevent

$(ROOT)/lib/$(MACH64)/libposix4.so.1:= \
        REALPATH=librt.so.1
$(ROOT)/lib/$(MACH64)/libposix4.so:= \
        REALPATH=libposix4.so.1
$(ROOT)/lib/$(MACH64)/llib-lposix4.ln:= \
        REALPATH=llib-lrt.ln
$(ROOT)/lib/$(MACH64)/libthread_db.so.1:= \
        REALPATH=libc_db.so.1
$(ROOT)/lib/$(MACH64)/libthread_db.so:= \
        REALPATH=libc_db.so.1
$(ROOT)/usr/lib/$(MACH64)/ld.so.1:= \
        REALPATH=../../../lib/$(MACH64)/ld.so.1
$(ROOT)/usr/lib/$(MACH64)/libadm.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libadm.so.1
$(ROOT)/usr/lib/$(MACH64)/libadm.so:= \
        REALPATH=../../../lib/$(MACH64)/libadm.so.1
$(ROOT)/usr/lib/$(MACH64)/libaio.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libaio.so.1
$(ROOT)/usr/lib/$(MACH64)/libaio.so:= \
        REALPATH=../../../lib/$(MACH64)/libaio.so.1
$(ROOT)/usr/lib/$(MACH64)/libavl.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libavl.so.1
$(ROOT)/usr/lib/$(MACH64)/libavl.so:= \
        REALPATH=../../../lib/$(MACH64)/libavl.so.1
$(ROOT)/usr/lib/$(MACH64)/libbsm.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libbsm.so.1
$(ROOT)/usr/lib/$(MACH64)/libbsm.so:= \
        REALPATH=../../../lib/$(MACH64)/libbsm.so.1
$(ROOT)/usr/lib/$(MACH64)/libc.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libc.so.1
$(ROOT)/usr/lib/$(MACH64)/libc.so:= \
```

```
1184         REALPATH=../../../lib/$(MACH64)/libc.so.1
1185 $(ROOT)/usr/lib/$(MACH64)/libc_db.so.1:= \
1186         REALPATH=../../../lib/$(MACH64)/libc_db.so.1
1187 $(ROOT)/usr/lib/$(MACH64)/libc_db.so:= \
1188         REALPATH=../../../lib/$(MACH64)/libc_db.so.1
1189 $(ROOT)/usr/lib/$(MACH64)/libcmdutils.so.1:= \
1190         REALPATH=../../../lib/$(MACH64)/libcmdutils.so.1
1191 $(ROOT)/usr/lib/$(MACH64)/libcmdutils.so:= \
1192         REALPATH=../../../lib/$(MACH64)/libcmdutils.so.1
1193 $(ROOT)/usr/lib/$(MACH64)/libcontract.so.1:= \
1194         REALPATH=../../../lib/$(MACH64)/libcontract.so.1
1195 $(ROOT)/usr/lib/$(MACH64)/libcontract.so:= \
1196         REALPATH=../../../lib/$(MACH64)/libcontract.so.1
1197 $(ROOT)/usr/lib/$(MACH64)/libctf.so.1:= \
1198         REALPATH=../../../lib/$(MACH64)/libctf.so.1
1199 $(ROOT)/usr/lib/$(MACH64)/libctf.so:= \
1200         REALPATH=../../../lib/$(MACH64)/libctf.so.1
1201 $(ROOT)/usr/lib/$(MACH64)/libcurses.so.1:= \
1202         REALPATH=../../../lib/$(MACH64)/libcurses.so.1
1203 $(ROOT)/usr/lib/$(MACH64)/libcurses.so:= \
1204         REALPATH=../../../lib/$(MACH64)/libcurses.so.1
1205 $(ROOT)/usr/lib/$(MACH64)/libdevice.so.1:= \
1206         REALPATH=../../../lib/$(MACH64)/libdevice.so.1
1207 $(ROOT)/usr/lib/$(MACH64)/libdevice.so:= \
1208         REALPATH=../../../lib/$(MACH64)/libdevice.so.1
1209 $(ROOT)/usr/lib/$(MACH64)/libdevid.so.1:= \
1210         REALPATH=../../../lib/$(MACH64)/libdevid.so.1
1211 $(ROOT)/usr/lib/$(MACH64)/libdevid.so:= \
1212         REALPATH=../../../lib/$(MACH64)/libdevid.so.1
1213 $(ROOT)/usr/lib/$(MACH64)/libdevinfo.so.1:= \
1214         REALPATH=../../../lib/$(MACH64)/libdevinfo.so.1
1215 $(ROOT)/usr/lib/$(MACH64)/libdevinfo.so:= \
1216         REALPATH=../../../lib/$(MACH64)/libdevinfo.so.1
1217 $(ROOT)/usr/lib/$(MACH64)/libdhcputil.so.1:= \
1218         REALPATH=../../../lib/$(MACH64)/libdhcputil.so.1
1219 $(ROOT)/usr/lib/$(MACH64)/libdhcputil.so:= \
1220         REALPATH=../../../lib/$(MACH64)/libdhcputil.so.1
1221 $(ROOT)/usr/lib/$(MACH64)/libdl.so.1:= \
1222         REALPATH=../../../lib/$(MACH64)/libdl.so.1
1223 $(ROOT)/usr/lib/$(MACH64)/libdl.so:= \
1224         REALPATH=../../../lib/$(MACH64)/libdl.so.1
1225 $(ROOT)/usr/lib/$(MACH64)/libdlpi.so.1:= \
1226         REALPATH=../../../lib/$(MACH64)/libdlpi.so.1
1227 $(ROOT)/usr/lib/$(MACH64)/libdlpi.so:= \
1228         REALPATH=../../../lib/$(MACH64)/libdlpi.so.1
1229 $(ROOT)/usr/lib/$(MACH64)/libdoor.so.1:= \
1230         REALPATH=../../../lib/$(MACH64)/libdoor.so.1
1231 $(ROOT)/usr/lib/$(MACH64)/libdoor.so:= \
1232         REALPATH=../../../lib/$(MACH64)/libdoor.so.1
1233 $(ROOT)/usr/lib/$(MACH64)/libefi.so.1:= \
1234         REALPATH=../../../lib/$(MACH64)/libefi.so.1
1235 $(ROOT)/usr/lib/$(MACH64)/libefi.so:= \
1236         REALPATH=../../../lib/$(MACH64)/libefi.so.1
1237 $(ROOT)/usr/lib/$(MACH64)/libelf.so.1:= \
1238         REALPATH=../../../lib/$(MACH64)/libelf.so.1
1239 $(ROOT)/usr/lib/$(MACH64)/libelf.so:= \
1240         REALPATH=../../../lib/$(MACH64)/libelf.so.1
1241 $(ROOT)/usr/lib/$(MACH64)/libgen.so.1:= \
1242         REALPATH=../../../lib/$(MACH64)/libgen.so.1
1243 $(ROOT)/usr/lib/$(MACH64)/libgen.so:= \
1244         REALPATH=../../../lib/$(MACH64)/libgen.so.1
1245 $(ROOT)/usr/lib/$(MACH64)/libinetutil.so.1:= \
1246         REALPATH=../../../lib/$(MACH64)/libinetutil.so.1
1247 $(ROOT)/usr/lib/$(MACH64)/libinetutil.so:= \
1248         REALPATH=../../../lib/$(MACH64)/libinetutil.so.1
1249 $(ROOT)/usr/lib/$(MACH64)/libintl.so.1:= \
```



```
1250         REALPATH=../../../lib/$(MACH64)/libintl.so.1
1251 $(ROOT)/usr/lib/$(MACH64)/libintl.so:= \
1252         REALPATH=../../../lib/$(MACH64)/libintl.so.1
1253 $(ROOT)/usr/lib/$(MACH64)/libkstat.so.1:= \
1254         REALPATH=../../../lib/$(MACH64)/libkstat.so.1
1255 $(ROOT)/usr/lib/$(MACH64)/libkstat.so:= \
1256         REALPATH=../../../lib/$(MACH64)/libkstat.so.1
1257 $(ROOT)/usr/lib/$(MACH64)/liblddbg.so.4:= \
1258         REALPATH=../../../lib/$(MACH64)/liblddbg.so.4
1259 $(ROOT)/usr/lib/$(MACH64)/libmd.so.1:= \
1260         REALPATH=../../../lib/$(MACH64)/libmd.so.1
1261 $(ROOT)/usr/lib/$(MACH64)/libmd.so:= \
1262         REALPATH=../../../lib/$(MACH64)/libmd.so.1
1263 $(ROOT)/usr/lib/$(MACH64)/libmd5.so.1:= \
1264         REALPATH=../../../lib/$(MACH64)/libmd5.so.1
1265 $(ROOT)/usr/lib/$(MACH64)/libmd5.so:= \
1266         REALPATH=../../../lib/$(MACH64)/libmd5.so.1
1267 $(ROOT)/usr/lib/$(MACH64)/libmp.so.2:= \
1268         REALPATH=../../../lib/$(MACH64)/libmp.so.2
1269 $(ROOT)/usr/lib/$(MACH64)/libmp.so:= \
1270         REALPATH=../../../lib/$(MACH64)/libmp.so.2
1271 $(ROOT)/usr/lib/$(MACH64)/libnsl.so.1:= \
1272         REALPATH=../../../lib/$(MACH64)/libnsl.so.1
1273 $(ROOT)/usr/lib/$(MACH64)/libnsl.so:= \
1274         REALPATH=../../../lib/$(MACH64)/libnsl.so.1
1275 $(ROOT)/usr/lib/$(MACH64)/libnvpair.so.1:= \
1276         REALPATH=../../../lib/$(MACH64)/libnvpair.so.1
1277 $(ROOT)/usr/lib/$(MACH64)/libnvpair.so:= \
1278         REALPATH=../../../lib/$(MACH64)/libnvpair.so.1
1279 $(ROOT)/usr/lib/$(MACH64)/libpam.so.1:= \
1280         REALPATH=../../../lib/$(MACH64)/libpam.so.1
1281 $(ROOT)/usr/lib/$(MACH64)/libpam.so:= \
1282         REALPATH=../../../lib/$(MACH64)/libpam.so.1
1283 $(ROOT)/usr/lib/$(MACH64)/libposix4.so.1:= \
1284         REALPATH=../../../lib/$(MACH64)/librt.so.1
1285 $(ROOT)/usr/lib/$(MACH64)/libposix4.so:= \
1286         REALPATH=../../../lib/$(MACH64)/librt.so.1
1287 $(ROOT)/usr/lib/$(MACH64)/libproc.so.1:= \
1288         REALPATH=../../../lib/$(MACH64)/libproc.so.1
1289 $(ROOT)/usr/lib/$(MACH64)/libproc.so:= \
1290         REALPATH=../../../lib/$(MACH64)/libproc.so.1
1291 $(ROOT)/usr/lib/$(MACH64)/libpthread.so.1:= \
1292         REALPATH=../../../lib/$(MACH64)/libpthread.so.1
1293 $(ROOT)/usr/lib/$(MACH64)/libpthread.so:= \
1294         REALPATH=../../../lib/$(MACH64)/libpthread.so.1
1295 $(ROOT)/usr/lib/$(MACH64)/librcm.so.1:= \
1296         REALPATH=../../../lib/$(MACH64)/librcm.so.1
1297 $(ROOT)/usr/lib/$(MACH64)/librcm.so:= \
1298         REALPATH=../../../lib/$(MACH64)/librcm.so.1
1299 $(ROOT)/usr/lib/$(MACH64)/libresolv.so.2:= \
1300         REALPATH=../../../lib/$(MACH64)/libresolv.so.2
1301 $(ROOT)/usr/lib/$(MACH64)/libresolv.so:= \
1302         REALPATH=../../../lib/$(MACH64)/libresolv.so.2
1303 $(ROOT)/usr/lib/$(MACH64)/librestart.so.1:= \
1304         REALPATH=../../../lib/$(MACH64)/librestart.so.1
1305 $(ROOT)/usr/lib/$(MACH64)/librestart.so:= \
1306         REALPATH=../../../lib/$(MACH64)/librestart.so.1
1307 $(ROOT)/usr/lib/$(MACH64)/librpcsvc.so.1:= \
1308         REALPATH=../../../lib/$(MACH64)/librpcsvc.so.1
1309 $(ROOT)/usr/lib/$(MACH64)/librpcsvc.so:= \
1310         REALPATH=../../../lib/$(MACH64)/librpcsvc.so.1
1311 $(ROOT)/usr/lib/$(MACH64)/librt.so.1:= \
1312         REALPATH=../../../lib/$(MACH64)/librt.so.1
1313 $(ROOT)/usr/lib/$(MACH64)/librt.so:= \
1314         REALPATH=../../../lib/$(MACH64)/librt.so.1
1315 $(ROOT)/usr/lib/$(MACH64)/librtld.so.1:= \
```

```
1316         REALPATH=../../../lib/$(MACH64)/librtld.so.1
1317 $(ROOT)/usr/lib/$(MACH64)/librtld_db.so.1:= \
1318         REALPATH=../../../lib/$(MACH64)/librtld_db.so.1
1319 $(ROOT)/usr/lib/$(MACH64)/librtld_db.so:= \
1320         REALPATH=../../../lib/$(MACH64)/librtld_db.so.1
1321 $(ROOT)/usr/lib/$(MACH64)/libscf.so.1:= \
1322         REALPATH=../../../lib/$(MACH64)/libscf.so.1
1323 $(ROOT)/usr/lib/$(MACH64)/libscf.so:= \
1324         REALPATH=../../../lib/$(MACH64)/libscf.so.1
1325 $(ROOT)/usr/lib/$(MACH64)/libsec.so.1:= \
1326         REALPATH=../../../lib/$(MACH64)/libsec.so.1
1327 $(ROOT)/usr/lib/$(MACH64)/libsec.so:= \
1328         REALPATH=../../../lib/$(MACH64)/libsec.so.1
1329 $(ROOT)/usr/lib/$(MACH64)/libsecdb.so.1:= \
1330         REALPATH=../../../lib/$(MACH64)/libsecdb.so.1
1331 $(ROOT)/usr/lib/$(MACH64)/libsecdb.so:= \
1332         REALPATH=../../../lib/$(MACH64)/libsecdb.so.1
1333 $(ROOT)/usr/lib/$(MACH64)/libsendfile.so.1:= \
1334         REALPATH=../../../lib/$(MACH64)/libsendfile.so.1
1335 $(ROOT)/usr/lib/$(MACH64)/libsendfile.so:= \
1336         REALPATH=../../../lib/$(MACH64)/libsendfile.so.1
1337 $(ROOT)/usr/lib/$(MACH64)/libsocket.so.1:= \
1338         REALPATH=../../../lib/$(MACH64)/libsocket.so.1
1339 $(ROOT)/usr/lib/$(MACH64)/libsocket.so:= \
1340         REALPATH=../../../lib/$(MACH64)/libsocket.so.1
1341 $(ROOT)/usr/lib/$(MACH64)/libsysevent.so.1:= \
1342         REALPATH=../../../lib/$(MACH64)/libsysevent.so.1
1343 $(ROOT)/usr/lib/$(MACH64)/libsysevent.so:= \
1344         REALPATH=../../../lib/$(MACH64)/libsysevent.so.1
1345 $(ROOT)/usr/lib/$(MACH64)/libtermcap.so.1:= \
1346         REALPATH=../../../lib/$(MACH64)/libtermcap.so.1
1347 $(ROOT)/usr/lib/$(MACH64)/libtermcap.so:= \
1348         REALPATH=../../../lib/$(MACH64)/libtermcap.so.1
1349 $(ROOT)/usr/lib/$(MACH64)/libtermlib.so.1:= \
1350         REALPATH=../../../lib/$(MACH64)/libcurses.so.1
1351 $(ROOT)/usr/lib/$(MACH64)/libtermlib.so:= \
1352         REALPATH=../../../lib/$(MACH64)/libcurses.so.1
1353 $(ROOT)/usr/lib/$(MACH64)/libthread.so.1:= \
1354         REALPATH=../../../lib/$(MACH64)/libthread.so.1
1355 $(ROOT)/usr/lib/$(MACH64)/libthread.so:= \
1356         REALPATH=../../../lib/$(MACH64)/libthread.so.1
1357 $(ROOT)/usr/lib/$(MACH64)/libthread_db.so.1:= \
1358         REALPATH=../../../lib/$(MACH64)/libc_db.so.1
1359 $(ROOT)/usr/lib/$(MACH64)/libthread_db.so:= \
1360         REALPATH=../../../lib/$(MACH64)/libc_db.so.1
1361 $(ROOT)/usr/lib/$(MACH64)/libtsnet.so.1:= \
1362         REALPATH=../../../lib/$(MACH64)/libtsnet.so.1
1363 $(ROOT)/usr/lib/$(MACH64)/libtsnet.so:= \
1364         REALPATH=../../../lib/$(MACH64)/libtsnet.so.1
1365 $(ROOT)/usr/lib/$(MACH64)/libtsol.so.2:= \
1366         REALPATH=../../../lib/$(MACH64)/libtsol.so.2
1367 $(ROOT)/usr/lib/$(MACH64)/libtsol.so:= \
1368         REALPATH=../../../lib/$(MACH64)/libtsol.so.2
1369 $(ROOT)/usr/lib/$(MACH64)/libumem.so.1:= \
1370         REALPATH=../../../lib/$(MACH64)/libumem.so.1
1371 $(ROOT)/usr/lib/$(MACH64)/libumem.so:= \
1372         REALPATH=../../../lib/$(MACH64)/libumem.so.1
1373 $(ROOT)/usr/lib/$(MACH64)/libuuid.so.1:= \
1374         REALPATH=../../../lib/$(MACH64)/libuuid.so.1
1375 $(ROOT)/usr/lib/$(MACH64)/libuuid.so:= \
1376         REALPATH=../../../lib/$(MACH64)/libuuid.so.1
1377 $(ROOT)/usr/lib/$(MACH64)/libuutil.so.1:= \
1378         REALPATH=../../../lib/$(MACH64)/libuutil.so.1
1379 $(ROOT)/usr/lib/$(MACH64)/libuutil.so:= \
1380         REALPATH=../../../lib/$(MACH64)/libuutil.so.1
1381 $(ROOT)/usr/lib/$(MACH64)/libw.so.1:= \
```



```
1382         REALPATH=../../../lib/$(MACH64)/libw.so.1
1383 $(ROOT)/usr/lib/$(MACH64)/libw.so:= \
1384         REALPATH=../../../lib/$(MACH64)/libw.so.1
1385 $(ROOT)/usr/lib/$(MACH64)/libxnet.so.1:= \
1386         REALPATH=../../../lib/$(MACH64)/libxnet.so.1
1387 $(ROOT)/usr/lib/$(MACH64)/libxnet.so:= \
1388         REALPATH=../../../lib/$(MACH64)/libxnet.so.1
1389 $(ROOT)/usr/lib/$(MACH64)/libzfs.so:= \
1390         REALPATH=../../../lib/$(MACH64)/libzfs.so.1
1391 $(ROOT)/usr/lib/$(MACH64)/libzfs.so.1:= \
1392         REALPATH=../../../lib/$(MACH64)/libzfs.so.1
1393 $(ROOT)/usr/lib/$(MACH64)/libzfs_core.so:= \
1394         REALPATH=../../../lib/$(MACH64)/libzfs_core.so.1
1395 $(ROOT)/usr/lib/$(MACH64)/libzfs_core.so.1:= \
1396         REALPATH=../../../lib/$(MACH64)/libzfs_core.so.1
1397 $(ROOT)/usr/lib/$(MACH64)/libfdisk.so.1:= \
1398         REALPATH=../../../lib/$(MACH64)/libfdisk.so.1
1399 $(ROOT)/usr/lib/$(MACH64)/libfdisk.so:= \
1400         REALPATH=../../../lib/$(MACH64)/libfdisk.so.1
1401 $(ROOT)/usr/lib/$(MACH64)/llib-ladm.ln:= \
1402         REALPATH=../../../lib/$(MACH64)/llib-ladm.ln
1403 $(ROOT)/usr/lib/$(MACH64)/llib-laio.ln:= \
1404         REALPATH=../../../lib/$(MACH64)/llib-laio.ln
1405 $(ROOT)/usr/lib/$(MACH64)/llib-lavl.ln:= \
1406         REALPATH=../../../lib/$(MACH64)/llib-lavl.ln
1407 $(ROOT)/usr/lib/$(MACH64)/llib-lbsm.ln:= \
1408         REALPATH=../../../lib/$(MACH64)/llib-lbsm.ln
1409 $(ROOT)/usr/lib/$(MACH64)/llib-lc.ln:= \
1410         REALPATH=../../../lib/$(MACH64)/llib-lc.ln
1411 $(ROOT)/usr/lib/$(MACH64)/llib-lcmdutils.ln:= \
1412         REALPATH=../../../lib/$(MACH64)/llib-lcmdutils.ln
1413 $(ROOT)/usr/lib/$(MACH64)/llib-lcontract.ln:= \
1414         REALPATH=../../../lib/$(MACH64)/llib-lcontract.ln
1415 $(ROOT)/usr/lib/$(MACH64)/llib-lctf.ln:= \
1416         REALPATH=../../../lib/$(MACH64)/llib-lctf.ln
1417 $(ROOT)/usr/lib/$(MACH64)/llib-lcurses.ln:= \
1418         REALPATH=../../../lib/$(MACH64)/llib-lcurses.ln
1419 $(ROOT)/usr/lib/$(MACH64)/llib-ldevice.ln:= \
1420         REALPATH=../../../lib/$(MACH64)/llib-ldevice.ln
1421 $(ROOT)/usr/lib/$(MACH64)/llib-ldevid.ln:= \
1422         REALPATH=../../../lib/$(MACH64)/llib-ldevid.ln
1423 $(ROOT)/usr/lib/$(MACH64)/llib-ldevinfo.ln:= \
1424         REALPATH=../../../lib/$(MACH64)/llib-ldevinfo.ln
1425 $(ROOT)/usr/lib/$(MACH64)/llib-ldhcputil.ln:= \
1426         REALPATH=../../../lib/$(MACH64)/llib-ldhcputil.ln
1427 $(ROOT)/usr/lib/$(MACH64)/llib-ldl.ln:= \
1428         REALPATH=../../../lib/$(MACH64)/llib-ldl.ln
1429 $(ROOT)/usr/lib/$(MACH64)/llib-ldoor.ln:= \
1430         REALPATH=../../../lib/$(MACH64)/llib-ldoor.ln
1431 $(ROOT)/usr/lib/$(MACH64)/llib-lefi.ln:= \
1432         REALPATH=../../../lib/$(MACH64)/llib-lefi.ln
1433 $(ROOT)/usr/lib/$(MACH64)/llib-lelf.ln:= \
1434         REALPATH=../../../lib/$(MACH64)/llib-lelf.ln
1435 $(ROOT)/usr/lib/$(MACH64)/llib-lgen.ln:= \
1436         REALPATH=../../../lib/$(MACH64)/llib-lgen.ln
1437 $(ROOT)/usr/lib/$(MACH64)/llib-linetutil.ln:= \
1438         REALPATH=../../../lib/$(MACH64)/llib-linetutil.ln
1439 $(ROOT)/usr/lib/$(MACH64)/llib-lintl.ln:= \
1440         REALPATH=../../../lib/$(MACH64)/llib-lintl.ln
1441 $(ROOT)/usr/lib/$(MACH64)/llib-lkstat.ln:= \
1442         REALPATH=../../../lib/$(MACH64)/llib-lkstat.ln
1443 $(ROOT)/usr/lib/$(MACH64)/llib-lmd5.ln:= \
1444         REALPATH=../../../lib/$(MACH64)/llib-lmd5.ln
1445 $(ROOT)/usr/lib/$(MACH64)/llib-lnsl.ln:= \
1446         REALPATH=../../../lib/$(MACH64)/llib-lnsl.ln
1447 $(ROOT)/usr/lib/$(MACH64)/llib-lnvpair.ln:= \
```

```
1448         REALPATH=../../../lib/$(MACH64)/llib-lnvpair.ln
1449 $(ROOT)/usr/lib/$(MACH64)/llib-lpam.ln:= \
1450         REALPATH=../../../lib/$(MACH64)/llib-lpam.ln
1451 $(ROOT)/usr/lib/$(MACH64)/llib-lposix4.ln:= \
1452         REALPATH=../../../lib/$(MACH64)/llib-lrt.ln
1453 $(ROOT)/usr/lib/$(MACH64)/llib-lpthread.ln:= \
1454         REALPATH=../../../lib/$(MACH64)/llib-lpthread.ln
1455 $(ROOT)/usr/lib/$(MACH64)/llib-lresolv.ln:= \
1456         REALPATH=../../../lib/$(MACH64)/llib-lresolv.ln
1457 $(ROOT)/usr/lib/$(MACH64)/llib-lrpcsvc.ln:= \
1458         REALPATH=../../../lib/$(MACH64)/llib-lrpcsvc.ln
1459 $(ROOT)/usr/lib/$(MACH64)/llib-lrt.ln:= \
1460         REALPATH=../../../lib/$(MACH64)/llib-lrt.ln
1461 $(ROOT)/usr/lib/$(MACH64)/llib-lrtld_db.ln:= \
1462         REALPATH=../../../lib/$(MACH64)/llib-lrtld_db.ln
1463 $(ROOT)/usr/lib/$(MACH64)/llib-lscf.ln:= \
1464         REALPATH=../../../lib/$(MACH64)/llib-lscf.ln
1465 $(ROOT)/usr/lib/$(MACH64)/llib-lsec.ln:= \
1466         REALPATH=../../../lib/$(MACH64)/llib-lsec.ln
1467 $(ROOT)/usr/lib/$(MACH64)/llib-lsecdb.ln:= \
1468         REALPATH=../../../lib/$(MACH64)/llib-lsecdb.ln
1469 $(ROOT)/usr/lib/$(MACH64)/llib-lsendfile.ln:= \
1470         REALPATH=../../../lib/$(MACH64)/llib-lsendfile.ln
1471 $(ROOT)/usr/lib/$(MACH64)/llib-lsocket.ln:= \
1472         REALPATH=../../../lib/$(MACH64)/llib-lsocket.ln
1473 $(ROOT)/usr/lib/$(MACH64)/llib-lsysevent.ln:= \
1474         REALPATH=../../../lib/$(MACH64)/llib-lsysevent.ln
1475 $(ROOT)/usr/lib/$(MACH64)/llib-ltermcap.ln:= \
1476         REALPATH=../../../lib/$(MACH64)/llib-ltermcap.ln
1477 $(ROOT)/usr/lib/$(MACH64)/llib-ltermlib.ln:= \
1478         REALPATH=../../../lib/$(MACH64)/llib-lcurses.ln
1479 $(ROOT)/usr/lib/$(MACH64)/llib-lthread.ln:= \
1480         REALPATH=../../../lib/$(MACH64)/llib-lthread.ln
1481 $(ROOT)/usr/lib/$(MACH64)/llib-lthread_db.ln:= \
1482         REALPATH=../../../lib/$(MACH64)/llib-lc_db.ln
1483 $(ROOT)/usr/lib/$(MACH64)/llib-ltsnet.ln:= \
1484         REALPATH=../../../lib/$(MACH64)/llib-ltsnet.ln
1485 $(ROOT)/usr/lib/$(MACH64)/llib-ltsol.ln:= \
1486         REALPATH=../../../lib/$(MACH64)/llib-ltsol.ln
1487 $(ROOT)/usr/lib/$(MACH64)/llib-lumem.ln:= \
1488         REALPATH=../../../lib/$(MACH64)/llib-lumem.ln
1489 $(ROOT)/usr/lib/$(MACH64)/llib-luuid.ln:= \
1490         REALPATH=../../../lib/$(MACH64)/llib-luuid.ln
1491 $(ROOT)/usr/lib/$(MACH64)/llib-lxnet.ln:= \
1492         REALPATH=../../../lib/$(MACH64)/llib-lxnet.ln
1493 $(ROOT)/usr/lib/$(MACH64)/llib-lzfs.ln:= \
1494         REALPATH=../../../lib/$(MACH64)/llib-lzfs.ln
1495 $(ROOT)/usr/lib/$(MACH64)/llib-lzfs_core.ln:= \
1496         REALPATH=../../../lib/$(MACH64)/llib-lzfs_core.ln
1497 $(ROOT)/usr/lib/$(MACH64)/llib-lfdisk.ln:= \
1498         REALPATH=../../../lib/$(MACH64)/llib-lfdisk.ln
1499 $(ROOT)/usr/lib/$(MACH64)/nss_compat.so.1:= \
1500         REALPATH=../../../lib/$(MACH64)/nss_compat.so.1
1501 $(ROOT)/usr/lib/$(MACH64)/nss_dns.so.1:= \
1502         REALPATH=../../../lib/$(MACH64)/nss_dns.so.1
1503 $(ROOT)/usr/lib/$(MACH64)/nss_files.so.1:= \
1504         REALPATH=../../../lib/$(MACH64)/nss_files.so.1
1505 $(ROOT)/usr/lib/$(MACH64)/nss_nis.so.1:= \
1506         REALPATH=../../../lib/$(MACH64)/nss_nis.so.1
1507 $(ROOT)/usr/lib/$(MACH64)/nss_user.so.1:= \
1508         REALPATH=../../../lib/$(MACH64)/nss_user.so.1
1509 $(ROOT)/usr/lib/fm/$(MACH64)/libfmevent.so.1:= \
1510         REALPATH=../../../../lib/fm/$(MACH64)/libfmevent.so.1
1511 $(ROOT)/usr/lib/fm/$(MACH64)/libfmevent.so:= \
1512         REALPATH=../../../../lib/fm/$(MACH64)/libfmevent.so.1
1513 $(ROOT)/usr/lib/fm/$(MACH64)/llib-lfmevent.ln:= \
```



```
1514         REALPATH=../../../../lib/fm/$(MACH64)/llib-lfmevent.ln

1516 i386_SYM.USRLIB= \
1517         /usr/lib/libfdisk.so \
1518         /usr/lib/libfdisk.so.1 \
1519         /usr/lib/llib-lfdisk \
1520         /usr/lib/llib-lfdisk.ln

1522 SYM.USRLIB= \
1523         $($(MACH)_SYM.USRLIB)   \
1524         /lib/libposix4.so \
1525         /lib/libposix4.so.1 \
1526         /lib/llib-lposix4 \
1527         /lib/llib-lposix4.ln \
1528         /lib/libthread_db.so \
1529         /lib/libthread_db.so.1 \
1530         /usr/lib/ld.so.1 \
1531         /usr/lib/libadm.so \
1532         /usr/lib/libadm.so.1 \
1533         /usr/lib/libaio.so \
1534         /usr/lib/libaio.so.1 \
1535         /usr/lib/libavl.so \
1536         /usr/lib/libavl.so.1 \
1537         /usr/lib/libbsm.so \
1538         /usr/lib/libbsm.so.1 \
1539         /usr/lib/libc.so \
1540         /usr/lib/libc.so.1 \
1541         /usr/lib/libc_db.so \
1542         /usr/lib/libc_db.so.1 \
1543         /usr/lib/libcmdutils.so \
1544         /usr/lib/libcmdutils.so.1 \
1545         /usr/lib/libcontract.so \
1546         /usr/lib/libcontract.so.1 \
1547         /usr/lib/libctf.so \
1548         /usr/lib/libctf.so.1 \
1549         /usr/lib/libcurses.so \
1550         /usr/lib/libcurses.so.1 \
1551         /usr/lib/libdevice.so \
1552         /usr/lib/libdevice.so.1 \
1553         /usr/lib/libdevid.so \
1554         /usr/lib/libdevid.so.1 \
1555         /usr/lib/libdevinfo.so \
1556         /usr/lib/libdevinfo.so.1 \
1557         /usr/lib/libdhcpagent.so \
1558         /usr/lib/libdhcpagent.so.1 \
1559         /usr/lib/libdhcputil.so \
1560         /usr/lib/libdhcputil.so.1 \
1561         /usr/lib/libdl.so \
1562         /usr/lib/libdl.so.1 \
1563         /usr/lib/libdlpi.so \
1564         /usr/lib/libdlpi.so.1 \
1565         /usr/lib/libdoor.so \
1566         /usr/lib/libdoor.so.1 \
1567         /usr/lib/libefi.so \
1568         /usr/lib/libefi.so.1 \
1569         /usr/lib/libelf.so \
1570         /usr/lib/libelf.so.1 \
1571         /usr/lib/libgen.so \
1572         /usr/lib/libgen.so.1 \
1573         /usr/lib/libinetutil.so \
1574         /usr/lib/libinetutil.so.1 \
1575         /usr/lib/libintl.so \
1576         /usr/lib/libintl.so.1 \
1577         /usr/lib/libkstat.so \
1578         /usr/lib/libkstat.so.1 \
1579         /usr/lib/liblddbg.so.4 \
```

```
1580         /usr/lib/libmd.so \
1581         /usr/lib/libmd.so.1 \
1582         /usr/lib/libmd5.so \
1583         /usr/lib/libmd5.so.1 \
1584         /usr/lib/libmeta.so \
1585         /usr/lib/libmeta.so.1 \
1586         /usr/lib/libmp.so \
1587         /usr/lib/libmp.so.1 \
1588         /usr/lib/libmp.so.2 \
1589         /usr/lib/libnsl.so \
1590         /usr/lib/libnsl.so.1 \
1591         /usr/lib/libnvpair.so \
1592         /usr/lib/libnvpair.so.1 \
1593         /usr/lib/libpam.so \
1594         /usr/lib/libpam.so.1 \
1595         /usr/lib/libposix4.so \
1596         /usr/lib/libposix4.so.1 \
1597         /usr/lib/libproc.so \
1598         /usr/lib/libproc.so.1 \
1599         /usr/lib/libpthread.so \
1600         /usr/lib/libpthread.so.1 \
1601         /usr/lib/librcm.so \
1602         /usr/lib/librcm.so.1 \
1603         /usr/lib/libresolv.so \
1604         /usr/lib/libresolv.so.1 \
1605         /usr/lib/libresolv.so.2 \
1606         /usr/lib/librestart.so \
1607         /usr/lib/librestart.so.1 \
1608         /usr/lib/librpcsvc.so \
1609         /usr/lib/librpcsvc.so.1 \
1610         /usr/lib/librt.so \
1611         /usr/lib/librt.so.1 \
1612         /usr/lib/librtld.so.1 \
1613         /usr/lib/librtld_db.so \
1614         /usr/lib/librtld_db.so.1 \
1615         /usr/lib/libscf.so \
1616         /usr/lib/libscf.so.1 \
1617         /usr/lib/libsec.so \
1618         /usr/lib/libsec.so.1 \
1619         /usr/lib/libsecdb.so \
1620         /usr/lib/libsecdb.so.1 \
1621         /usr/lib/libsendfile.so \
1622         /usr/lib/libsendfile.so.1 \
1623         /usr/lib/libsocket.so \
1624         /usr/lib/libsocket.so.1 \
1625         /usr/lib/libsysevent.so \
1626         /usr/lib/libsysevent.so.1 \
1627         /usr/lib/libtermcap.so \
1628         /usr/lib/libtermcap.so.1 \
1629         /usr/lib/libtermlib.so \
1630         /usr/lib/libtermlib.so.1 \
1631         /usr/lib/libthread.so \
1632         /usr/lib/libthread.so.1 \
1633         /usr/lib/libthread_db.so \
1634         /usr/lib/libthread_db.so.1 \
1635         /usr/lib/libtsnet.so \
1636         /usr/lib/libtsnet.so.1 \
1637         /usr/lib/libtsol.so \
1638         /usr/lib/libtsol.so.2 \
1639         /usr/lib/libumem.so \
1640         /usr/lib/libumem.so.1 \
1641         /usr/lib/libuuid.so \
1642         /usr/lib/libuuid.so.1 \
1643         /usr/lib/libuutil.so \
1644         /usr/lib/libuutil.so.1 \
1645         /usr/lib/libw.so \
```



```
1646         /usr/lib/libw.so.1 \
1647         /usr/lib/libxnet.so \
1648         /usr/lib/libxnet.so.1 \
1649         /usr/lib/libzfs.so \
1650         /usr/lib/libzfs.so.1 \
1651         /usr/lib/libzfs_core.so \
1652         /usr/lib/libzfs_core.so.1 \
1653         /usr/lib/llib-ladm \
1654         /usr/lib/llib-ladm.ln \
1655         /usr/lib/llib-laio \
1656         /usr/lib/llib-laio.ln \
1657         /usr/lib/llib-lavl \
1658         /usr/lib/llib-lavl.ln \
1659         /usr/lib/llib-lbsm \
1660         /usr/lib/llib-lbsm.ln \
1661         /usr/lib/llib-lc \
1662         /usr/lib/llib-lc.ln \
1663         /usr/lib/llib-lcmdutils \
1664         /usr/lib/llib-lcmdutils.ln \
1665         /usr/lib/llib-lcontract \
1666         /usr/lib/llib-lcontract.ln \
1667         /usr/lib/llib-lctf \
1668         /usr/lib/llib-lctf.ln \
1669         /usr/lib/llib-lcurses \
1670         /usr/lib/llib-lcurses.ln \
1671         /usr/lib/llib-ldevice \
1672         /usr/lib/llib-ldevice.ln \
1673         /usr/lib/llib-ldevid \
1674         /usr/lib/llib-ldevid.ln \
1675         /usr/lib/llib-ldevinfo \
1676         /usr/lib/llib-ldevinfo.ln \
1677         /usr/lib/llib-ldhcpagent \
1678         /usr/lib/llib-ldhcpagent.ln \
1679         /usr/lib/llib-ldhcputil \
1680         /usr/lib/llib-ldhcputil.ln \
1681         /usr/lib/llib-ldl \
1682         /usr/lib/llib-ldl.ln \
1683         /usr/lib/llib-ldoor \
1684         /usr/lib/llib-ldoor.ln \
1685         /usr/lib/llib-lefi \
1686         /usr/lib/llib-lefi.ln \
1687         /usr/lib/llib-lelf \
1688         /usr/lib/llib-lelf.ln \
1689         /usr/lib/llib-lgen \
1690         /usr/lib/llib-lgen.ln \
1691         /usr/lib/llib-linetutil \
1692         /usr/lib/llib-linetutil.ln \
1693         /usr/lib/llib-lintl \
1694         /usr/lib/llib-lintl.ln \
1695         /usr/lib/llib-lkstat \
1696         /usr/lib/llib-lkstat.ln \
1697         /usr/lib/llib-lmd5 \
1698         /usr/lib/llib-lmd5.ln \
1699         /usr/lib/llib-lmeta \
1700         /usr/lib/llib-lmeta.ln \
1701         /usr/lib/llib-lnsl \
1702         /usr/lib/llib-lnsl.ln \
1703         /usr/lib/llib-lnvpair \
1704         /usr/lib/llib-lnvpair.ln \
1705         /usr/lib/llib-lpam \
1706         /usr/lib/llib-lpam.ln \
1707         /usr/lib/llib-lposix4 \
1708         /usr/lib/llib-lposix4.ln \
1709         /usr/lib/llib-lpthread \
1710         /usr/lib/llib-lpthread.ln \
1711         /usr/lib/llib-lresolv \
```

```
1712         /usr/lib/llib-lresolv.ln \
1713         /usr/lib/llib-lrpcsvc \
1714         /usr/lib/llib-lrpcsvc.ln \
1715         /usr/lib/llib-lrt \
1716         /usr/lib/llib-lrt.ln \
1717         /usr/lib/llib-lrtld_db \
1718         /usr/lib/llib-lrtld_db.ln \
1719         /usr/lib/llib-lscf \
1720         /usr/lib/llib-lscf.ln \
1721         /usr/lib/llib-lsec \
1722         /usr/lib/llib-lsec.ln \
1723         /usr/lib/llib-lsecdb \
1724         /usr/lib/llib-lsecdb.ln \
1725         /usr/lib/llib-lsendfile \
1726         /usr/lib/llib-lsendfile.ln \
1727         /usr/lib/llib-lsocket \
1728         /usr/lib/llib-lsocket.ln \
1729         /usr/lib/llib-lsysevent \
1730         /usr/lib/llib-lsysevent.ln \
1731         /usr/lib/llib-ltermcap \
1732         /usr/lib/llib-ltermcap.ln \
1733         /usr/lib/llib-ltermlib \
1734         /usr/lib/llib-ltermlib.ln \
1735         /usr/lib/llib-lthread \
1736         /usr/lib/llib-lthread.ln \
1737         /usr/lib/llib-lthread_db \
1738         /usr/lib/llib-lthread_db.ln \
1739         /usr/lib/llib-ltsnet \
1740         /usr/lib/llib-ltsnet.ln \
1741         /usr/lib/llib-ltsol \
1742         /usr/lib/llib-ltsol.ln \
1743         /usr/lib/llib-lumem \
1744         /usr/lib/llib-lumem.ln \
1745         /usr/lib/llib-luuid \
1746         /usr/lib/llib-luuid.ln \
1747         /usr/lib/llib-lxnet \
1748         /usr/lib/llib-lxnet.ln \
1749         /usr/lib/llib-lzfs \
1750         /usr/lib/llib-lzfs.ln \
1751         /usr/lib/llib-lzfs_core \
1752         /usr/lib/llib-lzfs_core.ln \
1753         /usr/lib/nss_compat.so.1 \
1754         /usr/lib/nss_dns.so.1 \
1755         /usr/lib/nss_files.so.1 \
1756         /usr/lib/nss_nis.so.1 \
1757         /usr/lib/nss_user.so.1 \
1758         /usr/lib/fm/libfmevent.so \
1759         /usr/lib/fm/libfmevent.so.1 \
1760         /usr/lib/fm/llib-lfmevent \
1761         /usr/lib/fm/llib-lfmevent.ln

1763 sparcv9_SYM.USRLIB64=

1765 amd64_SYM.USRLIB64=     \
1766         /usr/lib/amd64/libfdisk.so      \
1767         /usr/lib/amd64/libfdisk.so.1    \
1768         /usr/lib/amd64/llib-lfdisk.ln


1771 SYM.USRLIB64= \
1772         $($(MACH64)_SYM.USRLIB64)               \
1773         /lib/$(MACH64)/libposix4.so \
1774         /lib/$(MACH64)/libposix4.so.1 \
1775         /lib/$(MACH64)/llib-lposix4.ln \
1776         /lib/$(MACH64)/libthread_db.so \
1777         /lib/$(MACH64)/libthread_db.so.1 \
```



```
1778         /usr/lib/$(MACH64)/ld.so.1 \
1779         /usr/lib/$(MACH64)/libadm.so \
1780         /usr/lib/$(MACH64)/libadm.so.1 \
1781         /usr/lib/$(MACH64)/libaio.so \
1782         /usr/lib/$(MACH64)/libaio.so.1 \
1783         /usr/lib/$(MACH64)/libavl.so \
1784         /usr/lib/$(MACH64)/libavl.so.1 \
1785         /usr/lib/$(MACH64)/libbsm.so \
1786         /usr/lib/$(MACH64)/libbsm.so.1 \
1787         /usr/lib/$(MACH64)/libc.so \
1788         /usr/lib/$(MACH64)/libc.so.1 \
1789         /usr/lib/$(MACH64)/libc_db.so \
1790         /usr/lib/$(MACH64)/libc_db.so.1 \
1791         /usr/lib/$(MACH64)/libcmdutils.so \
1792         /usr/lib/$(MACH64)/libcmdutils.so.1 \
1793         /usr/lib/$(MACH64)/libcontract.so \
1794         /usr/lib/$(MACH64)/libcontract.so.1 \
1795         /usr/lib/$(MACH64)/libctf.so \
1796         /usr/lib/$(MACH64)/libctf.so.1 \
1797         /usr/lib/$(MACH64)/libcurses.so \
1798         /usr/lib/$(MACH64)/libcurses.so.1 \
1799         /usr/lib/$(MACH64)/libdevice.so \
1800         /usr/lib/$(MACH64)/libdevice.so.1 \
1801         /usr/lib/$(MACH64)/libdevid.so \
1802         /usr/lib/$(MACH64)/libdevid.so.1 \
1803         /usr/lib/$(MACH64)/libdevinfo.so \
1804         /usr/lib/$(MACH64)/libdevinfo.so.1 \
1805         /usr/lib/$(MACH64)/libdhcputil.so \
1806         /usr/lib/$(MACH64)/libdhcputil.so.1 \
1807         /usr/lib/$(MACH64)/libdl.so \
1808         /usr/lib/$(MACH64)/libdl.so.1 \
1809         /usr/lib/$(MACH64)/libdlpi.so \
1810         /usr/lib/$(MACH64)/libdlpi.so.1 \
1811         /usr/lib/$(MACH64)/libdoor.so \
1812         /usr/lib/$(MACH64)/libdoor.so.1 \
1813         /usr/lib/$(MACH64)/libefi.so \
1814         /usr/lib/$(MACH64)/libefi.so.1 \
1815         /usr/lib/$(MACH64)/libelf.so \
1816         /usr/lib/$(MACH64)/libelf.so.1 \
1817         /usr/lib/$(MACH64)/libgen.so \
1818         /usr/lib/$(MACH64)/libgen.so.1 \
1819         /usr/lib/$(MACH64)/libinetutil.so \
1820         /usr/lib/$(MACH64)/libinetutil.so.1 \
1821         /usr/lib/$(MACH64)/libintl.so \
1822         /usr/lib/$(MACH64)/libintl.so.1 \
1823         /usr/lib/$(MACH64)/libkstat.so \
1824         /usr/lib/$(MACH64)/libkstat.so.1 \
1825         /usr/lib/$(MACH64)/liblddbg.so.4 \
1826         /usr/lib/$(MACH64)/libmd.so \
1827         /usr/lib/$(MACH64)/libmd.so.1 \
1828         /usr/lib/$(MACH64)/libmd5.so \
1829         /usr/lib/$(MACH64)/libmd5.so.1 \
1830         /usr/lib/$(MACH64)/libmp.so \
1831         /usr/lib/$(MACH64)/libmp.so.2 \
1832         /usr/lib/$(MACH64)/libnsl.so \
1833         /usr/lib/$(MACH64)/libnsl.so.1 \
1834         /usr/lib/$(MACH64)/libnvpair.so \
1835         /usr/lib/$(MACH64)/libnvpair.so.1 \
1836         /usr/lib/$(MACH64)/libpam.so \
1837         /usr/lib/$(MACH64)/libpam.so.1 \
1838         /usr/lib/$(MACH64)/libposix4.so \
1839         /usr/lib/$(MACH64)/libposix4.so.1 \
1840         /usr/lib/$(MACH64)/libproc.so \
1841         /usr/lib/$(MACH64)/libproc.so.1 \
1842         /usr/lib/$(MACH64)/libpthread.so \
1843         /usr/lib/$(MACH64)/libpthread.so.1 \
```

```
1844         /usr/lib/$(MACH64)/librcm.so \
1845         /usr/lib/$(MACH64)/librcm.so.1 \
1846         /usr/lib/$(MACH64)/libresolv.so \
1847         /usr/lib/$(MACH64)/libresolv.so.2 \
1848         /usr/lib/$(MACH64)/librestart.so \
1849         /usr/lib/$(MACH64)/librestart.so.1 \
1850         /usr/lib/$(MACH64)/librpcsvc.so \
1851         /usr/lib/$(MACH64)/librpcsvc.so.1 \
1852         /usr/lib/$(MACH64)/librt.so \
1853         /usr/lib/$(MACH64)/librt.so.1 \
1854         /usr/lib/$(MACH64)/librtld.so.1 \
1855         /usr/lib/$(MACH64)/librtld_db.so \
1856         /usr/lib/$(MACH64)/librtld_db.so.1 \
1857         /usr/lib/$(MACH64)/libscf.so \
1858         /usr/lib/$(MACH64)/libscf.so.1 \
1859         /usr/lib/$(MACH64)/libsec.so \
1860         /usr/lib/$(MACH64)/libsec.so.1 \
1861         /usr/lib/$(MACH64)/libsecdb.so \
1862         /usr/lib/$(MACH64)/libsecdb.so.1 \
1863         /usr/lib/$(MACH64)/libsendfile.so \
1864         /usr/lib/$(MACH64)/libsendfile.so.1 \
1865         /usr/lib/$(MACH64)/libsocket.so \
1866         /usr/lib/$(MACH64)/libsocket.so.1 \
1867         /usr/lib/$(MACH64)/libsysevent.so \
1868         /usr/lib/$(MACH64)/libsysevent.so.1 \
1869         /usr/lib/$(MACH64)/libtermcap.so \
1870         /usr/lib/$(MACH64)/libtermcap.so.1 \
1871         /usr/lib/$(MACH64)/libtermlib.so \
1872         /usr/lib/$(MACH64)/libtermlib.so.1 \
1873         /usr/lib/$(MACH64)/libthread.so \
1874         /usr/lib/$(MACH64)/libthread.so.1 \
1875         /usr/lib/$(MACH64)/libthread_db.so \
1876         /usr/lib/$(MACH64)/libthread_db.so.1 \
1877         /usr/lib/$(MACH64)/libtsnet.so \
1878         /usr/lib/$(MACH64)/libtsnet.so.1 \
1879         /usr/lib/$(MACH64)/libtsol.so \
1880         /usr/lib/$(MACH64)/libtsol.so.2 \
1881         /usr/lib/$(MACH64)/libumem.so \
1882         /usr/lib/$(MACH64)/libumem.so.1 \
1883         /usr/lib/$(MACH64)/libuuid.so \
1884         /usr/lib/$(MACH64)/libuuid.so.1 \
1885         /usr/lib/$(MACH64)/libuutil.so \
1886         /usr/lib/$(MACH64)/libuutil.so.1 \
1887         /usr/lib/$(MACH64)/libw.so \
1888         /usr/lib/$(MACH64)/libw.so.1 \
1889         /usr/lib/$(MACH64)/libxnet.so \
1890         /usr/lib/$(MACH64)/libxnet.so.1 \
1891         /usr/lib/$(MACH64)/libzfs.so \
1892         /usr/lib/$(MACH64)/libzfs.so.1 \
1893         /usr/lib/$(MACH64)/libzfs_core.so \
1894         /usr/lib/$(MACH64)/libzfs_core.so.1 \
1895         /usr/lib/$(MACH64)/llib-ladm.ln \
1896         /usr/lib/$(MACH64)/llib-laio.ln \
1897         /usr/lib/$(MACH64)/llib-lavl.ln \
1898         /usr/lib/$(MACH64)/llib-lbsm.ln \
1899         /usr/lib/$(MACH64)/llib-lc.ln \
1900         /usr/lib/$(MACH64)/llib-lcmdutils.ln \
1901         /usr/lib/$(MACH64)/llib-lcontract.ln \
1902         /usr/lib/$(MACH64)/llib-lctf.ln \
1903         /usr/lib/$(MACH64)/llib-lcurses.ln \
1904         /usr/lib/$(MACH64)/llib-ldevice.ln \
1905         /usr/lib/$(MACH64)/llib-ldevid.ln \
1906         /usr/lib/$(MACH64)/llib-ldevinfo.ln \
1907         /usr/lib/$(MACH64)/llib-ldhcputil.ln \
1908         /usr/lib/$(MACH64)/llib-ldl.ln \
1909         /usr/lib/$(MACH64)/llib-ldoor.ln \
```



```
1910         /usr/lib/$(MACH64)/llib-lefi.ln \
1911         /usr/lib/$(MACH64)/llib-lelf.ln \
1912         /usr/lib/$(MACH64)/llib-lgen.ln \
1913         /usr/lib/$(MACH64)/llib-linetutil.ln \
1914         /usr/lib/$(MACH64)/llib-lintl.ln \
1915         /usr/lib/$(MACH64)/llib-lkstat.ln \
1916         /usr/lib/$(MACH64)/llib-lmd5.ln \
1917         /usr/lib/$(MACH64)/llib-lnsl.ln \
1918         /usr/lib/$(MACH64)/llib-lnvpair.ln \
1919         /usr/lib/$(MACH64)/llib-lpam.ln \
1920         /usr/lib/$(MACH64)/llib-lposix4.ln \
1921         /usr/lib/$(MACH64)/llib-lpthread.ln \
1922         /usr/lib/$(MACH64)/llib-lresolv.ln \
1923         /usr/lib/$(MACH64)/llib-lrpcsvc.ln \
1924         /usr/lib/$(MACH64)/llib-lrt.ln \
1925         /usr/lib/$(MACH64)/llib-lrtld_db.ln \
1926         /usr/lib/$(MACH64)/llib-lscf.ln \
1927         /usr/lib/$(MACH64)/llib-lsec.ln \
1928         /usr/lib/$(MACH64)/llib-lsecdb.ln \
1929         /usr/lib/$(MACH64)/llib-lsendfile.ln \
1930         /usr/lib/$(MACH64)/llib-lsocket.ln \
1931         /usr/lib/$(MACH64)/llib-lsysevent.ln \
1932         /usr/lib/$(MACH64)/llib-ltermcap.ln \
1933         /usr/lib/$(MACH64)/llib-ltermlib.ln \
1934         /usr/lib/$(MACH64)/llib-lthread.ln \
1935         /usr/lib/$(MACH64)/llib-lthread_db.ln \
1936         /usr/lib/$(MACH64)/llib-ltsnet.ln \
1937         /usr/lib/$(MACH64)/llib-ltsol.ln \
1938         /usr/lib/$(MACH64)/llib-lumem.ln \
1939         /usr/lib/$(MACH64)/llib-luuid.ln \
1940         /usr/lib/$(MACH64)/llib-lxnet.ln \
1941         /usr/lib/$(MACH64)/llib-lzfs.ln \
1942         /usr/lib/$(MACH64)/llib-lzfs_core.ln \
1943         /usr/lib/$(MACH64)/nss_compat.so.1 \
1944         /usr/lib/$(MACH64)/nss_dns.so.1 \
1945         /usr/lib/$(MACH64)/nss_files.so.1 \
1946         /usr/lib/$(MACH64)/nss_nis.so.1 \
1947         /usr/lib/$(MACH64)/nss_user.so.1 \
1948         /usr/lib/fm/$(MACH64)/libfmevent.so \
1949         /usr/lib/fm/$(MACH64)/libfmevent.so.1 \
1950         /usr/lib/fm/$(MACH64)/llib-lfmevent.ln

1952 #
1953 # usr/src/Makefile uses INS.dir for any member of ROOTDIRS, the fact
1954 # these are symlinks to files has no bearing on this.
1955 #
1956 $(FILELINKS:%=$(ROOT)%):= \
1957     INS.dir= -$(RM) $@; $(SYMLINK) $(REALPATH) $@
```

```
**********************************************************
    1169 Tue Jan 14 16:16:54 2014
new/usr/src/cmd/devfsadm/i386/Makefile
Bring back LX zones.
**********************************************************
```

```
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #
  21 #
  22 # Copyright (c) 1998, 2010, Oracle and/or its affiliates. All rights reserved.
  23 #
  24 #ident  "%Z%%M% %I%     %E% SMI"
  25 #endif /* ! codereview */

  27 LINK_OBJS_i386 = \
  28         misc_link_i386.o \
  29         lx_link_i386.o \
  30 #endif /* ! codereview */
  31         xen_link.o

  33 lx_link_i386.o lx_link_i386.po lx_link_i386.ln := CPPFLAGS += -I$(UTSBASE)/commo
  34 #endif /* ! codereview */

  36 xen_link.o xen_link.ln xen_link.po := CPPFLAGS += -I$(UTSBASE)/i86xpv

  38 include ../Makefile.com
```

```
**********************************************************
    2555 Tue Jan 14 16:16:55 2014
new/usr/src/cmd/devfsadm/i386/lx_link_i386.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <devfsadm.h>
#include <strings.h>
#include <stdio.h>
#include <sys/lx_ptm.h>
#include <sys/lx_audio.h>

static int lx_ptm(di_minor_t minor, di_node_t node);
static int lx_audio(di_minor_t minor, di_node_t node);
static int lx_systrace(di_minor_t minor, di_node_t node);

static devfsadm_create_t lx_create_cbt[] = {
        { "pseudo", "ddi_pseudo", LX_PTM_DRV,
            TYPE_EXACT | DRV_EXACT, ILEVEL_0, lx_ptm },
        { "pseudo", "ddi_pseudo", LX_AUDIO_DRV,
            TYPE_EXACT | DRV_EXACT, ILEVEL_0, lx_audio },
        { "pseudo", "ddi_pseudo", "lx_systrace",
            TYPE_EXACT | DRV_EXACT, ILEVEL_0, lx_systrace },
};

DEVFSADM_CREATE_INIT_V0(lx_create_cbt);

static int
lx_ptm(di_minor_t minor, di_node_t node)
{
        char *mname = di_minor_name(minor);

        if (strcmp(LX_PTM_MINOR_NODE, mname) == 0)
                (void) devfsadm_mklink("brand/lx/ptmx", node, minor, 0);

        return (DEVFSADM_CONTINUE);
}

static int
```



```
lx_audio(di_minor_t minor, di_node_t node)
{
        char *mname = di_minor_name(minor);

        if (strcmp(LXA_MINORNAME_DEVCTL, mname) == 0)
                (void) devfsadm_mklink("brand/lx/audio_devctl", node, minor, 0);
        if (strcmp(LXA_MINORNAME_DSP, mname) == 0)
                (void) devfsadm_mklink("brand/lx/dsp", node, minor, 0);
        if (strcmp(LXA_MINORNAME_MIXER, mname) == 0)
                (void) devfsadm_mklink("brand/lx/mixer", node, minor, 0);

        return (DEVFSADM_CONTINUE);
}

static int
lx_systrace(di_minor_t minor, di_node_t node)
{
        char *mname = di_minor_name(minor);
        char path[MAXPATHLEN];

        (void) snprintf(path, sizeof (path), "dtrace/provider/%s", mname);
        (void) devfsadm_mklink(path, node, minor, 0);

        return (DEVFSADM_CONTINUE);
}
#endif /* ! codereview */
```

```
**********************************************************
   57877 Tue Jan 14 16:16:55 2014
new/usr/src/cmd/zlogin/zlogin.c
Bring back LX zones.
**********************************************************
______unchanged_portion_omitted_

1724 int
1725 main(int argc, char **argv)
1726 {
1727         int arg, console = 0;
1728         zoneid_t zoneid;
1729         zone_state_t st;
1730         char *login = "root";
1731         int lflag = 0;
1732         char *zonename = NULL;
1733         char **proc_args = NULL;
1734         char **new_args, **new_env;
1735         sigset_t block_cld;
1736         char devroot[MAXPATHLEN];
1737         char *slavename, slaveshortname[MAXPATHLEN];
1738         priv_set_t *privset;
1739         int tmpl_fd;
1740         char zonebrand[MAXNAMELEN];
1741         char default_brand[MAXNAMELEN];
1742         struct stat sb;
1743         char kernzone[ZONENAME_MAX];
1744         brand_handle_t bh;
1745         char user_cmd[MAXPATHLEN];
1746         char authname[MAXAUTHS];

1748         (void) setlocale(LC_ALL, "");
1749         (void) textdomain(TEXT_DOMAIN);

1751         (void) getpname(argv[0]);
1752         username = get_username();

1754         while ((arg = getopt(argc, argv, "ECR:Se:l:Q")) != EOF) {
1755                 switch (arg) {
1756                 case 'C':
1757                         console = 1;
1758                         break;
1759                 case 'E':
1760                         nocmdchar = 1;
1761                         break;
1762                 case 'R':       /* undocumented */
1763                         if (*optarg != '/') {
1764                                 zerror(gettext("root path must be absolute."));
1765                                 exit(2);
1766                         }
1767                         if (stat(optarg, &sb) == -1 || !S_ISDIR(sb.st_mode)) {
1768                                 zerror(
1769                                     gettext("root path must be a directory."));
1770                                 exit(2);
1771                         }
1772                         zonecfg_set_root(optarg);
1773                         break;
1774                 case 'Q':
1775                         quiet = 1;
1776                         break;
1777                 case 'S':
1778                         failsafe = 1;
1779                         break;
1780                 case 'e':
1781                         set_cmdchar(optarg);
1782                         break;
```



```
1783                 case 'l':
1784                         login = optarg;
1785                         lflag = 1;
1786                         break;
1787                 default:
1788                         usage();
1789                 }
1790         }

1792         if (console != 0 && lflag != 0) {
1793                 zerror(gettext("-l may not be specified for console login"));
1794                 usage();
1795         }

1797         if (console != 0 && failsafe != 0) {
1798                 zerror(gettext("-S may not be specified for console login"));
1799                 usage();
1800         }

1802         if (console != 0 && zonecfg_in_alt_root()) {
1803                 zerror(gettext("-R may not be specified for console login"));
1804                 exit(2);
1805         }

1807         if (failsafe != 0 && lflag != 0) {
1808                 zerror(gettext("-l may not be specified for failsafe login"));
1809                 usage();
1810         }

1812         if (optind == (argc - 1)) {
1813                 /*
1814                  * zone name, no process name; this should be an interactive
1815                  * as long as STDIN is really a tty.
1816                  */
1817                 if (isatty(STDIN_FILENO))
1818                         interactive = 1;
1819                 zonename = argv[optind];
1820         } else if (optind < (argc - 1)) {
1821                 if (console) {
1822                         zerror(gettext("Commands may not be specified for "
1823                             "console login."));
1824                         usage();
1825                 }
1826                 /* zone name and process name, and possibly some args */
1827                 zonename = argv[optind];
1828                 proc_args = &argv[optind + 1];
1829                 interactive = 0;
1830         } else {
1831                 usage();
1832         }

1834         if (getzoneid() != GLOBAL_ZONEID) {
1835                 zerror(gettext("'%s' may only be used from the global zone"),
1836                     pname);
1837                 return (1);
1838         }

1840         if (strcmp(zonename, GLOBAL_ZONENAME) == 0) {
1841                 zerror(gettext("'%s' not applicable to the global zone"),
1842                     pname);
1843                 return (1);
1844         }

1846         if (zone_get_state(zonename, &st) != Z_OK) {
1847                 zerror(gettext("zone '%s' unknown"), zonename);
1848                 return (1);
```

```
1849         }

1851         if (st < ZONE_STATE_INSTALLED) {
1852                 zerror(gettext("cannot login to a zone which is '%s'"),
1853                     zone_state_str(st));
1854                 return (1);
1855         }

1857         /*
1858          * In both console and non-console cases, we require all privs.
1859          * In the console case, because we may need to startup zoneadmd.
1860          * In the non-console case in order to do zone_enter(2), zonept()
1861          * and other tasks.
1862          */

1864         if ((privset = priv_allocset()) == NULL) {
1865                 zperror(gettext("priv_allocset failed"));
1866                 return (1);
1867         }

1869         if (getppriv(PRIV_EFFECTIVE, privset) != 0) {
1870                 zperror(gettext("getppriv failed"));
1871                 priv_freeset(privset);
1872                 return (1);
1873         }

1875         if (priv_isfullset(privset) == B_FALSE) {
1876                 zerror(gettext("You lack sufficient privilege to run "
1877                     "this command (all privs required)"));
1878                 priv_freeset(privset);
1879                 return (1);
1880         }
1881         priv_freeset(privset);

1883         /*
1884          * Check if user is authorized for requested usage of the zone
1885          */

1887         (void) snprintf(authname, MAXAUTHS, "%s%s%s",
1888             ZONE_MANAGE_AUTH, KV_OBJECT, zonename);
1889         if (chkauthattr(authname, username) == 0) {
1890                 if (console) {
1891                         zerror(gettext("%s is not authorized for console "
1892                             "access to  %s zone."),
1893                             username, zonename);
1894                         return (1);
1895                 } else {
1896                         (void) snprintf(authname, MAXAUTHS, "%s%s%s",
1897                             ZONE_LOGIN_AUTH, KV_OBJECT, zonename);
1898                         if (failsafe || !interactive) {
1899                                 zerror(gettext("%s is not authorized for  "
1900                                     "failsafe or non-interactive login "
1901                                     "to  %s zone."), username, zonename);
1902                                 return (1);
1903                         } else if (chkauthattr(authname, username) == 0) {
1904                                 zerror(gettext("%s is not authorized "
1905                                     " to login to %s zone."),
1906                                     username, zonename);
1907                                 return (1);
1908                         }
1909                 }
1910         } else {
1911                 forced_login = B_TRUE;
1912         }

1914         /*
```



```
1915          * The console is a separate case from the rest of the code; handle
1916          * it first.
1917          */
1918         if (console) {
1919                 /*
1920                  * Ensure that zoneadmd for this zone is running.
1921                  */
1922                 if (start_zoneadmd(zonename) == -1)
1923                         return (1);

1925                 /*
1926                  * Make contact with zoneadmd.
1927                  */
1928                 if (get_console_master(zonename) == -1)
1929                         return (1);

1931                 if (!quiet)
1932                         (void) printf(
1933                             gettext("[Connected to zone '%s' console]\n"),
1934                             zonename);

1936                 if (set_tty_rawmode(STDIN_FILENO) == -1) {
1937                         reset_tty();
1938                         zperror(gettext("failed to set stdin pty to raw mode"));
1939                         return (1);
1940                 }

1942                 (void) sigset(SIGWINCH, sigwinch);
1943                 (void) sigwinch(0);

1945                 /*
1946                  * Run the I/O loop until we get disconnected.
1947                  */
1948                 doio(masterfd, -1, masterfd, -1, -1, B_FALSE);
1949                 reset_tty();
1950                 if (!quiet)
1951                         (void) printf(
1952                             gettext("\n[Connection to zone '%s' console "
1953                             "closed]\n"), zonename);

1955                 return (0);
1956         }

1958         if (st != ZONE_STATE_RUNNING && st != ZONE_STATE_MOUNTED) {
1959                 zerror(gettext("login allowed only to running zones "
1960                     "(%s is '%s')."), zonename, zone_state_str(st));
1961                 return (1);
1962         }

1964         (void) strlcpy(kernzone, zonename, sizeof (kernzone));
1965         if (zonecfg_in_alt_root()) {
1966                 FILE *fp = zonecfg_open_scratch("", B_FALSE);

1968                 if (fp == NULL || zonecfg_find_scratch(fp, zonename,
1969                     zonecfg_get_root(), kernzone, sizeof (kernzone)) == -1) {
1970                         zerror(gettext("cannot find scratch zone %s"),
1971                             zonename);
1972                         if (fp != NULL)
1973                                 zonecfg_close_scratch(fp);
1974                         return (1);
1975                 }
1976                 zonecfg_close_scratch(fp);
1977         }

1979         if ((zoneid = getzoneidbyname(kernzone)) == -1) {
1980                 zerror(gettext("failed to get zoneid for zone '%s'"),
```

```
1981                    zonename);
1982                return (1);
1983        }

1985        /*
1986         * We need the zone root path only if we are setting up a pty.
1987         */
1988        if (zone_get_devroot(zonename, devroot, sizeof (devroot)) == -1) {
1989                zerror(gettext("could not get dev path for zone %s"),
1990                    zonename);
1991                return (1);
1992        }

1994        if (zone_get_brand(zonename, zonebrand, sizeof (zonebrand)) != Z_OK) {
1995                zerror(gettext("could not get brand for zone %s"), zonename);
1996                return (1);
1997        }
1998        /*
1999         * In the alternate root environment, the only supported
2000         * operations are mount and unmount.  In this case, just treat
2001         * the zone as native if it is cluster.  Cluster zones can be
2002         * native for the purpose of LU or upgrade, and the cluster
2003         * brand may not exist in the miniroot (such as in net install
2004         * upgrade).
2005         */
2006        if (zonecfg_default_brand(default_brand,
2007            sizeof (default_brand)) != Z_OK) {
2008                zerror(gettext("unable to determine default brand"));
2009                return (1);
2010        }
2011        if (zonecfg_in_alt_root() &&
2012            strcmp(zonebrand, CLUSTER_BRAND_NAME) == 0) {
2013                (void) strlcpy(zonebrand, default_brand, sizeof (zonebrand));
2014        }

2016        if ((bh = brand_open(zonebrand)) == NULL) {
2017                zerror(gettext("could not open brand for zone %s"), zonename);
2018                return (1);
2019        }

2021        if ((new_args = prep_args(bh, login, proc_args)) == NULL) {
2022                zperror(gettext("could not assemble new arguments"));
2023                brand_close(bh);
2024                return (1);
2025        }
2026        /*
2027         * Get the brand specific user_cmd.  This command is used to get
2028         * a passwd(4) entry for login.
2029         */
2030        if (!interactive && !failsafe) {
2031                if (zone_get_user_cmd(bh, login, user_cmd,
2032                    sizeof (user_cmd)) == NULL) {
2033                        zerror(gettext("could not get user_cmd for zone %s"),
2034                            zonename);
2035                        brand_close(bh);
2036                        return (1);
2037                }
2038        }
2039        brand_close(bh);

2041        if ((new_env = prep_env()) == NULL) {
2042                zperror(gettext("could not assemble new environment"));
2043                return (1);
2044        }

2046        if (!interactive)
```



```
2047                return (noninteractive_login(zonename, user_cmd, zoneid,
2048                    new_args, new_env));

2050        if (zonecfg_in_alt_root()) {
2051                zerror(gettext("cannot use interactive login with scratch "
2052                    "zone"));
2053                return (1);
2054        }

2056        /*
2057         * Things are more complex in interactive mode; we get the
2058         * master side of the pty, then place the user's terminal into
2059         * raw mode.
2060         */
2061        if (get_master_pty() == -1) {
2062                zerror(gettext("could not setup master pty device"));
2063                return (1);
2064        }

2066        /*
2067         * Compute the "short name" of the pts.  /dev/pts/2 --> pts/2
2068         */
2069        if ((slavename = ptsname(masterfd)) == NULL) {
2070                zperror(gettext("failed to get name for pseudo-tty"));
2071                return (1);
2072        }
2073        if (strncmp(slavename, "/dev/", strlen("/dev/")) == 0)
2074                (void) strlcpy(slaveshortname, slavename + strlen("/dev/"),
2075                    sizeof (slaveshortname));
2076        else
2077                (void) strlcpy(slaveshortname, slavename,
2078                    sizeof (slaveshortname));

2080        if (!quiet)
2081                (void) printf(gettext("[Connected to zone '%s' %s]\n"),
2082                    zonename, slaveshortname);

2084        if (set_tty_rawmode(STDIN_FILENO) == -1) {
2085                reset_tty();
2086                zperror(gettext("failed to set stdin pty to raw mode"));
2087                return (1);
2088        }

2090        if (prefork_dropprivs() != 0) {
2091                reset_tty();
2092                zperror(gettext("could not allocate privilege set"));
2093                return (1);
2094        }

2096        /*
2097         * We must mask SIGCLD until after we have coped with the fork
2098         * sufficiently to deal with it; otherwise we can race and receive the
2099         * signal before child_pid has been initialized (yes, this really
2100         * happens).
2101         */
2102        (void) sigset(SIGCLD, sigcld);
2103        (void) sigemptyset(&block_cld);
2104        (void) sigaddset(&block_cld, SIGCLD);
2105        (void) sigprocmask(SIG_BLOCK, &block_cld, NULL);

2107        /*
2108         * We activate the contract template at the last minute to
2109         * avoid intermediate functions that could be using fork(2)
2110         * internally.
2111         */
2112        if ((tmpl_fd = init_template()) == -1) {
```

```
2113                reset_tty();
2114                zperror(gettext("could not create contract"));
2115                return (1);
2116        }

2118        if ((child_pid = fork()) == -1) {
2119                (void) ct_tmpl_clear(tmpl_fd);
2120                reset_tty();
2121                zperror(gettext("could not fork"));
2122                return (1);
2123        } else if (child_pid == 0) { /* child process */
2124                int slavefd, newslave;

2126                (void) ct_tmpl_clear(tmpl_fd);
2127                (void) close(tmpl_fd);

2129                (void) sigprocmask(SIG_UNBLOCK, &block_cld, NULL);

2131                if ((slavefd = init_slave_pty(zoneid, devroot)) == -1)
2132                        return (1);

2134                /*
2135                 * Close all fds except for the slave pty.
2136                 */
2137                (void) fdwalk(close_func, &slavefd);

2139                /*
2140                 * Temporarily dup slavefd to stderr; that way if we have
2141                 * to print out that zone_enter failed, the output will
2142                 * have somewhere to go.
2143                 */
2144                if (slavefd != STDERR_FILENO)
2145                        (void) dup2(slavefd, STDERR_FILENO);

2147                if (zone_enter(zoneid) == -1) {
2148                        zerror(gettext("could not enter zone %s: %s"),
2149                            zonename, strerror(errno));
2150                        return (1);
2151                }

2153                if (slavefd != STDERR_FILENO)
2154                        (void) close(STDERR_FILENO);

2156                /*
2157                 * We take pains to get this process into a new process
2158                 * group, and subsequently a new session.  In this way,
2159                 * we'll have a session which doesn't yet have a controlling
2160                 * terminal.  When we open the slave, it will become the
2161                 * controlling terminal; no PIDs concerning pgrps or sids
2162                 * will leak inappropriately into the zone.
2163                 */
2164                (void) setpgrp();

2166                /*
2167                 * We need the slave pty to be referenced from the zone's
2168                 * /dev in order to ensure that the devt's, etc are all
2169                 * correct.  Otherwise we break ttyname and the like.
2170                 */
2171                if ((newslave = open(slavename, O_RDWR)) == -1) {
2172                        (void) close(slavefd);
2173                        return (1);
2174                }
2175                (void) close(slavefd);
2176                slavefd = newslave;

2178                /*
```



```
2179                 * dup the slave to the various FDs, so that when the
2180                 * spawned process does a write/read it maps to the slave
2181                 * pty.
2182                 */
2183                (void) dup2(slavefd, STDIN_FILENO);
2184                (void) dup2(slavefd, STDOUT_FILENO);
2185                (void) dup2(slavefd, STDERR_FILENO);
2186                if (slavefd != STDIN_FILENO && slavefd != STDOUT_FILENO &&
2187                    slavefd != STDERR_FILENO) {
2188                        (void) close(slavefd);
2189                }

2191                /*
2192                 * In failsafe mode, we don't use login(1), so don't try
2193                 * setting up a utmpx entry.
2194                 *
2195                 * A branded zone may have very different utmpx semantics.
2196                 * At the moment, we only have two brand types:
2197                 * Solaris-like (native, sn1) and Linux.  In the Solaris
2198                 * case, we know exactly how to do the necessary utmpx
2199                 * setup.  Fortunately for us, the Linux /bin/login is
2200                 * prepared to deal with a non-initialized utmpx entry, so
2201                 * we can simply skip it.  If future brands don't fall into
2202                 * either category, we'll have to add a per-brand utmpx
2203                 * setup hook.
2204 #endif /* ! codereview */
2205                 */
2206                if (!failsafe && (strcmp(zonebrand, "lx") != 0))
2194                if (!failsafe)
2207                        if (setup_utmpx(slaveshortname) == -1)
2208                                return (1);

2210                /*
2211                 * The child needs to run as root to
2212                 * execute the brand's login program.
2213                 */
2214                if (setuid(0) == -1) {
2215                        zperror(gettext("insufficient privilege"));
2216                        return (1);
2217                }

2219                (void) execve(new_args[0], new_args, new_env);
2220                zperror(gettext("exec failure"));
2221                return (1);
2222        }

2224        (void) ct_tmpl_clear(tmpl_fd);
2225        (void) close(tmpl_fd);

2227        /*
2228         * The rest is only for the parent process.
2229         */
2230        (void) sigset(SIGWINCH, sigwinch);

2232        postfork_dropprivs();

2234        (void) sigprocmask(SIG_UNBLOCK, &block_cld, NULL);
2235        doio(masterfd, -1, masterfd, -1, -1, B_FALSE);

2237        reset_tty();
2238        if (!quiet)
2239                (void) fprintf(stderr,
2240                    gettext("\n[Connection to zone '%s' %s closed]\n"),
2241                    zonename, slaveshortname);

2243        if (pollerr != 0) {
```

```
2244                 (void) fprintf(stderr, gettext("Error: connection closed due "
2245                     "to unexpected pollevents=0x%x.\n"), pollerr);
2246                 return (1);
2247         }

2249         return (0);
2250 }
```

_____***unchanged_portion_omitted_***

****
4592 Tue Jan 14 16:16:55 2014
new/usr/src/cmd/zoneadm/svc-zones
Bring back LX zones.
****

```
  1 #!/sbin/sh
  2 #
  3 # CDDL HEADER START
  4 #
  5 # The contents of this file are subject to the terms of the
  6 # Common Development and Distribution License (the "License").
  7 # You may not use this file except in compliance with the License.
  8 #
  9 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 10 # or http://www.opensolaris.org/os/licensing.
 11 # See the License for the specific language governing permissions
 12 # and limitations under the License.
 13 #
 14 # When distributing Covered Code, include this CDDL HEADER in each
 15 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 16 # If applicable, add the following below this CDDL HEADER, with the
 17 # fields enclosed by brackets "[]" replaced with your own identifying
 18 # information: Portions Copyright [yyyy] [name of copyright owner]
 19 #
 20 # CDDL HEADER END
 21 #
 22 #
 23 # Copyright (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.

 25 . /lib/svc/share/smf_include.sh

 27 #
 28 # Return a list of running, non-global zones for which a shutdown via
 29 # "/sbin/init 0" may work (typically only Solaris zones.)
 30 #
 31 shutdown_zones()
 32 {
 33         zoneadm list -p | nawk -F: '{
 34                 if (($5 != "lx") && ($2 != "global")) {
 34                 if ($2 != "global") {
 35                         print $2
 36                 }
 37         }'
 38 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    6232 Tue Jan 14 16:16:56 2014
new/usr/src/common/brand/lx/lx_signum.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/signal.h>
#include <lx_signum.h>

/*
 * Delivering signals to a Linux process is complicated by differences in
 * signal numbering, stack structure and contents, and the action taken when a
 * signal handler exits.  In addition, many signal-related structures, such as
 * sigset_ts, vary between Solaris and Linux.
 *
 * The simplest transformation that must be done when sending signals is to
 * translate between Linux and Solaris signal numbers.
 *
 * These are the major signal number differences between Linux and Solaris:
 *
 *      ===================================
 *      | Number |   Linux    |  Solaris   |
 *      | ====== | ========== | ========== |
 *      |    7   | SIGBUS     | SIGEMT     |
 *      |   10   | SIGUSR1    | SIGBUS     |
 *      |   12   | SIGUSR2    | SIGSYS     |
 *      |   16   | SIGSTKFLT  | SIGUSR1    |
 *      |   17   | SIGCHLD    | SIGUSR2    |
 *      |   18   | SIGCONT    | SIGCHLD    |
 *      |   19   | SIGSTOP    | SIGPWR     |
 *      |   20   | SIGTSTP    | SIGWINCH   |
 *      |   21   | SIGTTIN    | SIGURG     |
 *      |   22   | SIGTTOU    | SIGPOLL    |
 *      |   23   | SIGURG     | SIGSTOP    |
 *      |   24   | SIGXCPU    | SIGTSTP    |
 *      |   25   | SIGXFSZ    | SIGCONT    |
 *      |   26   | SIGVTALARM | SIGTTIN    |
 *      |   27   | SIGPROF    | SIGTTOU    |
 *      |   28   | SIGWINCH   | SIGVTALARM |
 *      |   29   | SIGPOLL    | SIGPROF    |
 *      |   30   | SIGPWR     | SIGXCPU    |
```



```
 *      |   31   | SIGSYS     | SIGXFSZ    |
 *      ===================================
 *
 * Not every Linux signal maps to a Solaris signal, nor does every Solaris
 * signal map to a Linux counterpart. However, when signals do map, the
 * mapping is unique.
 *
 * One mapping issue is that Linux supports 33 real time signals, with SIGRTMIN
 * typically starting at or near 32 (SIGRTMIN) and proceeding to 64 (SIGRTMAX)
 * (SIGRTMIN is "at or near" 32 because glibc usually "steals" one ore more of
 * these signals for its own internal use, adjusting SIGRTMIN and SIGRTMAX as
 * needed.)  Conversely, Solaris actively uses signals 32-40 for other purposes
 * and supports exactly 32 real time signals, in the range 41 (SIGRTMIN)
 * to 72 (SIGRTMAX).
 *
 * At present, attempting to translate a Linux signal equal to 63
 * will generate an error (we allow SIGRTMAX because a program
 * should be able to send SIGRTMAX without getting an EINVAL, though obviously
 * anything that loops through the signals from SIGRTMIN to SIGRTMAX will
 * fail.)
 *
 * Similarly, attempting to translate a native Solaris signal in the range
 * 32-40 will also generate an error as we don't want to support the receipt of
 * those signals from the Solaris global zone.
 */

/*
 * Linux to Solaris signal map
 *
 * Usage:  solaris_signal = ltos_signum[lx_signal];
 */
const int
ltos_signo[LX_NSIG] = {
        0,
        SIGHUP,
        SIGINT,
        SIGQUIT,
        SIGILL,
        SIGTRAP,
        SIGABRT,
        SIGBUS,
        SIGFPE,
        SIGKILL,
        SIGUSR1,
        SIGSEGV,
        SIGUSR2,
        SIGPIPE,
        SIGALRM,
        SIGTERM,
        SIGEMT,                 /* 16:  Linux SIGSTKFLT; use Solaris SIGEMT */
        SIGCHLD,
        SIGCONT,
        SIGSTOP,
        SIGTSTP,
        SIGTTIN,
        SIGTTOU,
        SIGURG,
        SIGXCPU,
        SIGXFSZ,
        SIGVTALRM,
        SIGPROF,
        SIGWINCH,
        SIGPOLL,
        SIGPWR,
        SIGSYS,
        _SIGRTMIN,              /* 32:  Linux SIGRTMIN */
```

```
        _SIGRTMIN + 1,
        _SIGRTMIN + 2,
        _SIGRTMIN + 3,
        _SIGRTMIN + 4,
        _SIGRTMIN + 5,
        _SIGRTMIN + 6,
        _SIGRTMIN + 7,
        _SIGRTMIN + 8,
        _SIGRTMIN + 9,
        _SIGRTMIN + 10,
        _SIGRTMIN + 11,
        _SIGRTMIN + 12,
        _SIGRTMIN + 13,
        _SIGRTMIN + 14,
        _SIGRTMIN + 15,
        _SIGRTMIN + 16,
        _SIGRTMIN + 17,
        _SIGRTMIN + 18,
        _SIGRTMIN + 19,
        _SIGRTMIN + 20,
        _SIGRTMIN + 21,
        _SIGRTMIN + 22,
        _SIGRTMIN + 23,
        _SIGRTMIN + 24,
        _SIGRTMIN + 25,
        _SIGRTMIN + 26,
        _SIGRTMIN + 27,
        _SIGRTMIN + 28,
        _SIGRTMIN + 29,
        _SIGRTMIN + 30,
        -1,                     /* 63:  Linux SIGRTMIN + 31, or SIGRTMAX - 1 */
        _SIGRTMAX,              /* 64:  Linux SIGRTMAX */
};

/*
 * Solaris to Linux signal map
 *
 * Usage:  lx_signal = stol_signo[solaris_signal];
 */
const int
stol_signo[NSIG] = {
        0,
        LX_SIGHUP,
        LX_SIGINT,
        LX_SIGQUIT,
        LX_SIGILL,
        LX_SIGTRAP,
        LX_SIGABRT,
        LX_SIGSTKFLT,           /* 7:  Solaris SIGEMT; use for LX_SIGSTKFLT */
        LX_SIGFPE,
        LX_SIGKILL,
        LX_SIGBUS,
        LX_SIGSEGV,
        LX_SIGSYS,
        LX_SIGPIPE,
        LX_SIGALRM,
        LX_SIGTERM,
        LX_SIGUSR1,
        LX_SIGUSR2,
        LX_SIGCHLD,
        LX_SIGPWR,
        LX_SIGWINCH,
        LX_SIGURG,
        LX_SIGPOLL,
        LX_SIGSTOP,
        LX_SIGTSTP,
```



```
        LX_SIGCONT,
        LX_SIGTTIN,
        LX_SIGTTOU,
        LX_SIGVTALRM,
        LX_SIGPROF,
        LX_SIGXCPU,
        LX_SIGXFSZ,
        -1,                     /* 32:  Solaris SIGWAITING */
        -1,                     /* 33:  Solaris SIGLWP */
        -1,                     /* 34:  Solaris SIGFREEZE */
        -1,                     /* 35:  Solaris SIGTHAW */
        -1,                     /* 36:  Solaris SIGCANCEL */
        -1,                     /* 37:  Solaris SIGLOST */
        -1,                     /* 38:  Solaris SIGXRES */
        -1,                     /* 39:  Solaris SIGJVM1 */
        -1,                     /* 40:  Solaris SIGJVM2 */
        LX_SIGRTMIN,            /* 41:  Solaris _SIGRTMIN */
        LX_SIGRTMIN + 1,
        LX_SIGRTMIN + 2,
        LX_SIGRTMIN + 3,
        LX_SIGRTMIN + 4,
        LX_SIGRTMIN + 5,
        LX_SIGRTMIN + 6,
        LX_SIGRTMIN + 7,
        LX_SIGRTMIN + 8,
        LX_SIGRTMIN + 9,
        LX_SIGRTMIN + 10,
        LX_SIGRTMIN + 11,
        LX_SIGRTMIN + 12,
        LX_SIGRTMIN + 13,
        LX_SIGRTMIN + 14,
        LX_SIGRTMIN + 15,
        LX_SIGRTMIN + 16,
        LX_SIGRTMIN + 17,
        LX_SIGRTMIN + 18,
        LX_SIGRTMIN + 19,
        LX_SIGRTMIN + 20,
        LX_SIGRTMIN + 21,
        LX_SIGRTMIN + 22,
        LX_SIGRTMIN + 23,
        LX_SIGRTMIN + 24,
        LX_SIGRTMIN + 25,
        LX_SIGRTMIN + 26,
        LX_SIGRTMIN + 27,
        LX_SIGRTMIN + 28,
        LX_SIGRTMIN + 29,
        LX_SIGRTMIN + 30,
        LX_SIGRTMAX,            /* 72: Solaris _SIGRTMAX */
};
#endif /* ! codereview */
```

```
**********************************************************
    2055 Tue Jan 14 16:16:56 2014
new/usr/src/common/brand/lx/lx_signum.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _LX_SIGNUM_H
#define _LX_SIGNUM_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

#ifdef  __cplusplus
extern "C" {
#endif

#define LX_SIGHUP       1
#define LX_SIGINT       2
#define LX_SIGQUIT      3
#define LX_SIGILL       4
#define LX_SIGTRAP      5
#define LX_SIGABRT      6
#define LX_SIGIOT       6
#define LX_SIGBUS       7
#define LX_SIGFPE       8
#define LX_SIGKILL      9
#define LX_SIGUSR1      10
#define LX_SIGSEGV      11
#define LX_SIGUSR2      12
#define LX_SIGPIPE      13
#define LX_SIGALRM      14
#define LX_SIGTERM      15
#define LX_SIGSTKFLT    16
#define LX_SIGCHLD      17
#define LX_SIGCONT      18
#define LX_SIGSTOP      19
#define LX_SIGTSTP      20
#define LX_SIGTTIN      21
#define LX_SIGTTOU      22
#define LX_SIGURG       23
#define LX_SIGXCPU      24
#define LX_SIGXFSZ      25
#define LX_SIGVTALRM    26
```



```
#define LX_SIGPROF      27
#define LX_SIGWINCH     28
#define LX_SIGIO        29
#define LX_SIGPOLL      LX_SIGIO
#define LX_SIGPWR       30
#define LX_SIGSYS       31
#define LX_SIGUNUSED    31

#define LX_NSIG_WORDS   2
#define LX_NBPW         32
#define LX_NSIG         ((LX_NBPW * LX_NSIG_WORDS) + 1)

#define LX_SIGRTMIN     32
#define LX_SIGRTMAX     LX_NSIG - 1

extern const int ltos_signo[];
extern const int stol_signo[];

#ifdef  __cplusplus
}
#endif

#endif  /* _LX_SIGNUM_H */
#endif /* ! codereview */
```

```
**********************************************************
   10059 Tue Jan 14 16:16:56 2014
new/usr/src/head/regexp.h
LX zone support should now build and packages of relevance produced.
**********************************************************
______unchanged_portion_omitted_
```

```
357 int
358 #ifdef  __STDC__
359 advance(const char *lp, const char *ep)
360 #else
361 advance(lp, ep)
362 register char *lp, *ep;
363 #endif
364 {
365 #ifdef  __STDC__
366         const char *curlp;
367 #else
368         register char *curlp;
369 #endif
370         int c;
371         char *bbeg;
372         register char neg;
373         size_t ct;

375         for (;;) {
376                 neg = 0;
377                 switch (*ep++) {

379                 case CCHR:
380                         if (*ep++ == *lp++)
381                                 continue;
382                         return (0);
383                         /*FALLTHRU*/

385                 case CDOT:
386                         if (*lp++)
387                                 continue;
388                         return (0);
389                         /*FALLTHRU*/

391                 case CDOL:
392                         if (*lp == 0)
393                                 continue;
394                         return (0);
395                         /*FALLTHRU*/

397                 case CCEOF:
398                         loc2 = (char *)lp;
399                         return (1);
400                         /*FALLTHRU*/

402                 case CXCL:
403                         c = (unsigned char)*lp++;
404                         if (ISTHERE(c)) {
405                                 ep += 32;
406                                 continue;
407                         }
408                         return (0);
409                         /*FALLTHRU*/

411                 case NCCL:
412                         neg = 1;
413                         /*FALLTHRU*/

415                 case CCL:
```



```
416                         c = *lp++;
417                         if (((c & 0200) == 0 && ISTHERE(c)) ^ neg) {
418                                 ep += 16;
419                                 continue;
420                         }
421                         return (0);
422                         /*FALLTHRU*/

424                 case CBRA:
425                         braslist[(int)*ep++] = (char *)lp;
425                         braslist[*ep++] = (char *)lp;
426                         continue;
427                         /*FALLTHRU*/

429                 case CKET:
430                         braelist[(int)*ep++] = (char *)lp;
430                         braelist[*ep++] = (char *)lp;
431                         continue;
432                         /*FALLTHRU*/

434                 case CCHR | RNGE:
435                         c = *ep++;
436                         getrnge(ep);
437                         while (low--)
438                                 if (*lp++ != c)
439                                         return (0);
440                         curlp = lp;
441                         while (size--)
442                                 if (*lp++ != c)
443                                         break;
444                         if (size < 0)
445                                 lp++;
446                         ep += 2;
447                         goto star;
448                         /*FALLTHRU*/

450                 case CDOT | RNGE:
451                         getrnge(ep);
452                         while (low--)
453                                 if (*lp++ == '\0')
454                                         return (0);
455                         curlp = lp;
456                         while (size--)
457                                 if (*lp++ == '\0')
458                                         break;
459                         if (size < 0)
460                                 lp++;
461                         ep += 2;
462                         goto star;
463                         /*FALLTHRU*/

465                 case CXCL | RNGE:
466                         getrnge(ep + 32);
467                         while (low--) {
468                                 c = (unsigned char)*lp++;
469                                 if (!ISTHERE(c))
470                                         return (0);
471                         }
472                         curlp = lp;
473                         while (size--) {
474                                 c = (unsigned char)*lp++;
475                                 if (!ISTHERE(c))
476                                         break;
477                         }
478                         if (size < 0)
479                                 lp++;
```

```
480                     ep += 34;               /* 32 + 2 */
481                     goto star;
482                     /*FALLTHRU*/

484             case NCCL | RNGE:
485                     neg = 1;
486                     /*FALLTHRU*/

488             case CCL | RNGE:
489                     getrnge(ep + 16);
490                     while (low--) {
491                             c = *lp++;
492                             if (((c & 0200) || !ISTHERE(c)) ^ neg)
493                                     return (0);
494                     }
495                     curlp = lp;
496                     while (size--) {
497                             c = *lp++;
498                             if (((c & 0200) || !ISTHERE(c)) ^ neg)
499                                     break;
500                     }
501                     if (size < 0)
502                             lp++;
503                     ep += 18;               /* 16 + 2 */
504                     goto star;
505                     /*FALLTHRU*/

507             case CBACK:
508                     bbeg = braslist[(int)*ep];
509                     ct = braelist[(int)*ep++] - bbeg;
508                     bbeg = braslist[*ep];
509                     ct = braelist[*ep++] - bbeg;

511                     if (ecmp(bbeg, lp, ct)) {
512                             lp += ct;
513                             continue;
514                     }
515                     return (0);
516                     /*FALLTHRU*/

518             case CBACK | STAR:
519                     bbeg = braslist[(int)*ep];
520                     ct = braelist[(int)*ep++] - bbeg;
519                     bbeg = braslist[*ep];
520                     ct = braelist[*ep++] - bbeg;
521                     curlp = lp;
522                     while (ecmp(bbeg, lp, ct))
523                             lp += ct;

525                     while (lp >= curlp) {
526                             if (advance(lp, ep))
527                                     return (1);
528                             lp -= ct;
529                     }
530                     return (0);
531                     /*FALLTHRU*/

533             case CDOT | STAR:
534                     curlp = lp;
535                     while (*lp++);
536                     goto star;
537                     /*FALLTHRU*/

539             case CCHR | STAR:
540                     curlp = lp;
541                     while (*lp++ == *ep);
```



```
542                     ep++;
543                     goto star;
544                     /*FALLTHRU*/

546             case CXCL | STAR:
547                     curlp = lp;
548                     do {
549                             c = (unsigned char)*lp++;
550                     } while (ISTHERE(c));
551                     ep += 32;
552                     goto star;
553                     /*FALLTHRU*/

555             case NCCL | STAR:
556                     neg = 1;
557                     /*FALLTHRU*/

559             case CCL | STAR:
560                     curlp = lp;
561                     do {
562                             c = *lp++;
563                     } while (((c & 0200) == 0 && ISTHERE(c)) ^ neg);
564                     ep += 16;
565                     goto star;
566                     /*FALLTHRU*/

568             star:
569                     do {
570                             if (--lp == locs)
571                                     break;
572                             if (advance(lp, ep))
573                                     return (1);
574                     } while (lp > curlp);
575                     return (0);

577             }
578     }
579     /*NOTREACHED*/
580 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    1471 Tue Jan 14 16:16:56 2014
new/usr/src/lib/brand/Makefile
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright (c) 2006, 2010, Oracle and/or its affiliates. All rights reserved.
#
# lib/brand/Makefile
#
# include global definitions
include ../../Makefile.master

#
# Build everything in parallel; use .WAIT for dependencies
.PARALLEL:

i386_SUBDIRS= lx
i386_MSGSUBDIRS= lx

#endif /* ! codereview */
SUBDIRS= shared .WAIT sn1 solaris10 ipkg labeled $($(MACH)_SUBDIRS)
MSGSUBDIRS= solaris10 shared $($(MACH)_MSGSUBDIRS)

all :=          TARGET= all
install :=      TARGET= install
clean :=        TARGET= clean
clobber :=      TARGET= clobber
lint :=         TARGET= lint
_msg :=         TARGET= _msg

.KEEP_STATE:

all install clean clobber lint: $(SUBDIRS)

_msg: $(MSGSUBDIRS)

$(SUBDIRS): FRC
        @cd $@; pwd; $(MAKE) $(TARGET)

FRC:
```

```
**********************************************************
    1443 Tue Jan 14 16:16:57 2014
new/usr/src/lib/brand/lx/Makefile
Bring back LX zones.
**********************************************************
```

```
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #

  22 #
  23 # Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  24 # Use is subject to license terms.
  25 #
  26 # ident "%Z%%M% %I%     %E% SMI"
  27 #

  29 default: all

  31 include Makefile.lx

  33 # Build everything in parallel; use .WAIT for dependencies
  34 .PARALLEL:

  36 SUBDIRS=        cmd librtld_db lx_support lx_brand lx_thunk netfiles zone \
  37                 .WAIT lx_nametoaddr
  38 MSGSUBDIRS=     lx_brand lx_support zone

  40 all :=          TARGET= all
  41 install :=      TARGET= install
  42 clean :=        TARGET= clean
  43 clobber :=      TARGET= clobber
  44 lint :=         TARGET= lint
  45 _msg :=         TARGET= _msg

  47 .KEEP_STATE:

  49 all install clean clobber lint: $(SUBDIRS)

  51 _msg: $(MSGSUBDIRS)

  53 $(SUBDIRS): FRC
  54         @cd $@; pwd; $(MAKE) $(TARGET)

  56 FRC:
  57 #endif /* ! codereview */
```

****
1042 Tue Jan 14 16:16:57 2014
new/usr/src/lib/brand/lx/Makefile.lx
Bring back LX zones.
****

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "%Z%%M% %I%     %E% SMI"
#
# lib/brand/lx/Makefile.lx
#
# include global definitions

BRAND=  lx

include $(SRC)/lib/brand/Makefile.brand

#endif /* ! codereview */
```

```
**********************************************************
    1212 Tue Jan 14 16:16:57 2014
new/usr/src/lib/brand/lx/cmd/Makefile
Bring back LX zones.
**********************************************************
```

```
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #

  22 #
  23 # Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  24 # Use is subject to license terms.
  25 #
  26 #ident  "%Z%%M% %I%     %E% SMI"

  28 PROGS =         lx_lockd lx_native lx_statd lx_thunk

  30 include ../Makefile.lx

  32 # override the install directory
  33 ROOTBIN =       $(ROOTBRANDDIR)
  34 CLOBBERFILES =  $(ROOTPROGS)

  36 .KEEP_STATE:

  38 lint:

  40 all:            $(PROGS)

  42 install:        all $(ROOTPROGS)

  44 clean:
  45         $(RM) $(PROGS)

  47 clobber: clean
  48         $(RM) $(ROOTPROGS)
  49 #endif /* ! codereview */
```

```
**********************************************************
    1172 Tue Jan 14 16:16:57 2014
new/usr/src/lib/brand/lx/cmd/lx_lockd.sh
Bring back LX zones.
**********************************************************
```

```
  1 #!/bin/sh
  2 #
  3 # CDDL HEADER START
  4 #
  5 # The contents of this file are subject to the terms of the
  6 # Common Development and Distribution License (the "License").
  7 # You may not use this file except in compliance with the License.
  8 #
  9 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 10 # or http://www.opensolaris.org/os/licensing.
 11 # See the License for the specific language governing permissions
 12 # and limitations under the License.
 13 #
 14 # When distributing Covered Code, include this CDDL HEADER in each
 15 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 16 # If applicable, add the following below this CDDL HEADER, with the
 17 # fields enclosed by brackets "[]" replaced with your own identifying
 18 # information: Portions Copyright [yyyy] [name of copyright owner]
 19 #
 20 # CDDL HEADER END
 21 #

 23 #
 24 # Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 25 # Use is subject to license terms.
 26 #
 27 # ident "%Z%%M% %I%     %E% SMI"
 28 #

 30 LD_LIBRARY_PATH=/usr/lib/brand/lx
 31 LD_PRELOAD=/native/usr/lib/brand/lx/lx_thunk.so.1
 32 LD_BIND_NOW=1
 33 export LD_LIBRARY_PATH LD_PRELOAD LD_BIND_NOW

 35 exec /native/usr/lib/brand/lx/lx_native \
 36         /native/usr/lib/nfs/lockd -P -U 29 -G 29
 37 #endif /* ! codereview */
```

****
949 Tue Jan 14 16:16:57 2014
new/usr/src/lib/brand/lx/cmd/lx_native.sh
Bring back LX zones.
****

```
#!/bin/sh
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "%Z%%M% %I%     %E% SMI"
#
exit 0
#endif /* ! codereview */
```

```
**********************************************************
    1172 Tue Jan 14 16:16:58 2014
new/usr/src/lib/brand/lx/cmd/lx_statd.sh
Bring back LX zones.
**********************************************************
```

```
#!/bin/sh
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "%Z%%M% %I%     %E% SMI"
#

LD_LIBRARY_PATH=/usr/lib/brand/lx
LD_PRELOAD=/native/usr/lib/brand/lx/lx_thunk.so.1
LD_BIND_NOW=1
export LD_LIBRARY_PATH LD_PRELOAD LD_BIND_NOW

exec /native/usr/lib/brand/lx/lx_native \
        /native/usr/lib/nfs/statd -P -U 29 -G 29
#endif /* ! codereview */
```

```
**********************************************************
     981 Tue Jan 14 16:16:58 2014
new/usr/src/lib/brand/lx/cmd/lx_thunk.sh
Bring back LX zones.
**********************************************************
```

```
   1 #!/bin/sh
   2 #
   3 # CDDL HEADER START
   4 #
   5 # The contents of this file are subject to the terms of the
   6 # Common Development and Distribution License (the "License").
   7 # You may not use this file except in compliance with the License.
   8 #
   9 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
  10 # or http://www.opensolaris.org/os/licensing.
  11 # See the License for the specific language governing permissions
  12 # and limitations under the License.
  13 #
  14 # When distributing Covered Code, include this CDDL HEADER in each
  15 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  16 # If applicable, add the following below this CDDL HEADER, with the
  17 # fields enclosed by brackets "[]" replaced with your own identifying
  18 # information: Portions Copyright [yyyy] [name of copyright owner]
  19 #
  20 # CDDL HEADER END
  21 #

  23 #
  24 # Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  25 # Use is subject to license terms.
  26 #
  27 # ident "%Z%%M% %I%     %E% SMI"
  28 #
  29 exec /native/usr/lib/brand/lx/lx_thunk
  30 #endif /* ! codereview */
```

```
**********************************************************
    1371 Tue Jan 14 16:16:58 2014
new/usr/src/lib/brand/lx/librtld_db/Makefile
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "%Z%%M% %I%     %E% SMI"
#

default: all

include $(SRC)/lib/Makefile.lib

SUBDIRS = $(MACH)
$(BUILD64)SUBDIRS += $(MACH64)

LINT_SUBDIRS=   $(MACH)
$(BUILD64)LINT_SUBDIRS += $(MACH64)

all :=          TARGET= all
clean :=        TARGET= clean
clobber :=      TARGET= clobber
install :=      TARGET= install
lint :=         TARGET= lint

.KEEP_STATE:

all install clean clobber: $(SUBDIRS)

lint: $(LINT_SUBDIRS)

$(SUBDIRS): FRC
        @cd $@; pwd; $(MAKE) $(TARGET)

FRC:
#endif /* ! codereview */
```

```
**********************************************************
    2453 Tue Jan 14 16:16:58 2014
new/usr/src/lib/brand/lx/librtld_db/Makefile.com
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "%Z%%M% %I%     %E% SMI"
#

LIBRARY =       lx_librtld_db.a
VERS    =       .1
COBJS   =       lx_librtld_db.o
OBJECTS =       $(COBJS) $(COBJS64)

include $(SRC)/lib/Makefile.lib
include ../../Makefile.lx

CSRCS =       $(COBJS:%o=../common/%c)
SRCS  =       $(CSRCS)

SRCDIR =        ../common
UTSBASE =       $(SRC)/uts

#
# ATTENTION:
#       Librtl_db brand plugin libraries should NOT directly invoke any
#       libproc.so interfaces or be linked against libproc.  If a librtl_db
#       brand plugin library uses libproc.so interfaces then it may break
#       any other librtld_db consumers (like mdb) that tries to attach
#       to a branded process.  The only safe interfaces that the a librtld_db
#       brand plugin library can use to access a target process are the
#       proc_service(3PROC) apis.
#
DYNFLAGS +=     $(VERSREF) -M../common/mapfile-vers
LIBS =          $(DYNLIB)
LDLIBS +=       -lc -lrtld_db
CFLAGS +=       $(CCVERBOSE)
CPPFLAGS +=     -D_REENTRANT -I../ -I$(UTSBASE)/common/brand/lx \
                        -I$(SRC)/cmd/sgs/librtld_db/common \
                        -I$(SRC)/cmd/sgs/include \
                        -I$(SRC)/cmd/sgs/include/$(MACH)
```



```
ROOTLIBDIR =    $(ROOT)/usr/lib/brand/lx
ROOTLIBDIR64 =  $(ROOT)/usr/lib/brand/lx/$(MACH64)

#
# The top level Makefiles define define TEXT_DOMAIN.  But librtld_db.so.1
# isn't internationalized and this library won't be either.  The only
# messages that this library can generate are messages used for debugging
# the operation of the library itself.
#
DTEXTDOM =

.KEEP_STATE:

all: $(LIBS)

lint: lintcheck

pics/%64.o:     ../common/%.c
                $(COMPILE.c) -D_ELF64 $(PICFLAGS) -o $@ $<
                $(POST_PROCESS_O)

include $(SRC)/lib/Makefile.targ
#endif /* ! codereview */
```

```
**********************************************************
    1119 Tue Jan 14 16:16:58 2014
new/usr/src/lib/brand/lx/librtld_db/amd64/Makefile
Bring back LX zones.
**********************************************************
```

```
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #

  22 #
  23 # Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
  24 # Use is subject to license terms.
  25 #
  26 # ident "%Z%%M% %I%     %E% SMI"
  27 #

  29 COBJS64 =       lx_librtld_db64.o

  31 include ../Makefile.com
  32 include $(SRC)/lib/Makefile.lib.64

  34 DYNFLAGS +=     -Mmapfile-vers

  36 CLOBBERFILES = $(ROOTLIBDIR64)/$(DYNLIB)

  38 install: all $(ROOTLIBS64)
  39 #endif /* ! codereview */
```

```
**********************************************************
    1304 Tue Jan 14 16:16:59 2014
new/usr/src/lib/brand/lx/librtld_db/amd64/mapfile-vers
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

#
# MAPFILE HEADER START
#
# WARNING:  STOP NOW.  DO NOT MODIFY THIS FILE.
# Object versioning must comply with the rules detailed in
#
#       usr/src/lib/README.mapfiles
#
# You should not be making modifications here until you've read the most current
# copy of that file. If you need help, contact a gatekeeper for guidance.
#
# MAPFILE HEADER END
#

SUNWprivate_1.1 {
        global:
                rtld_db_brand_ops64;
};
#endif /* ! codereview */
```

```
**********************************************************
   17226 Tue Jan 14 16:16:59 2014
new/usr/src/lib/brand/lx/librtld_db/common/lx_librtld_db.c
LX zone support should now build and packages of relevance produced.
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/link.h>
#include <libproc.h>
#include <proc_service.h>
#include <rtld_db.h>
#include <synch.h>

#include <sys/lx_brand.h>

/*
 * ATTENTION:
 *      Librtl_db brand plugin libraries should NOT directly invoke any
 *      libproc.so interfaces or be linked against libproc.  If a librtl_db
 *      brand plugin library uses libproc.so interfaces then it may break
 *      any other librtld_db consumers (like mdb) that tries to attach
 *      to a branded process.  The only safe interfaces that the a librtld_db
 *      brand plugin library can use to access a target process are the
 *      proc_service(3PROC) apis.
 */

/*
 * M_DATA comes from some streams header file but is also redifined in
 * _rtld_db.h, so nuke the old streams definition here.
 */
#ifdef M_DATA
#undef M_DATA
#endif /* M_DATA */

/*
```



```
 * For 32-bit versions of this library, this file get's compiled once.
 * For 64-bit versions of this library, this file get's compiled twice,
 * once with _ELF64 defined and once without.  The expectation is that
 * the 64-bit version of the library can properly deal with both 32-bit
 * and 64-bit elf files, hence in the 64-bit library there are two copies
 * of all the interfaces in this file, one set named *32 and one named *64.
 *
 * This also means that we need to be careful when declaring local pointers
 * that point to objects in another processes address space, since these
 * pointers may not match the current processes pointer width.  Basically,
 * we should avoid using data types that change size between 32 and 64 bit
 * modes like: long, void *, uintprt_t, caddr_t, psaddr_t, size_t, etc.
 * Instead we should declare all pointers as uint32_t.  Then when we
 * are compiled to deal with 64-bit targets we'll re-define uint32_t
 * to be a uint64_t.
 *
 * Finally, one last importante note.  All the 64-bit elf file code
 * is never used and can't be tested.  This is because we don't actually
 * support 64-bit Linux processes yet.  The reason that we have it here
 * is because we want to support debugging 32-bit elf targets with the
 * 64-bit version of this library, so we need to have a 64-bit version
 * of this library.  But a 64-bit version of this library is expected
 * to provide debugging interfaces for both 32 and 64-bit elf targets.
 * So we provide the 64-bit elf target interfaces, but they will never
 * be invoked and are untested.  If we ever add support for 64-bit elf
 * Linux processes, we'll need to verify that this code works correctly
 * for those targets.
 */
#ifdef _LP64
#ifdef _ELF64
#define lx_ldb_get_dyns32               lx_ldb_get_dyns64
#define lx_ldb_init32                   lx_ldb_init64
#define lx_ldb_fini32                   lx_ldb_fini64
#define lx_ldb_loadobj_iter32           lx_ldb_loadobj_iter64
#define lx_ldb_getauxval32              lx_ldb_getauxval64
#define lx_elf_props32                  lx_elf_props64
#define _rd_get_dyns32                  _rd_get_dyns64
#define _rd_get_ehdr32                  _rd_get_ehdr64
#define uint32_t                        uint64_t
#define Elf32_Dyn                       Elf64_Dyn
#define Elf32_Ehdr                      Elf64_Ehdr
#define Elf32_Phdr                      Elf64_Phdr
#endif /* _ELF64 */
#endif /* _LP64 */

/* Included from usr/src/cmd/sgs/librtld_db/common */
#include <_rtld_db.h>

typedef struct lx_rd {
        rd_agent_t              *lr_rap;
        struct ps_prochandle    *lr_php;        /* proc handle pointer */
        uint32_t                lr_rdebug;      /* address of lx r_debug */
        uint32_t                lr_exec;        /* base address of executable */
} lx_rd_t;

typedef struct lx_link_map {
        uint32_t lxm_addr;      /* Base address shared object is loaded at.  */
        uint32_t lxm_name;      /* Absolute file name object was found in.  */
        uint32_t lxm_ld;        /* Dynamic section of the shared object.  */
        uint32_t lxm_next;      /* Chain of loaded objects.  */
} lx_link_map_t;

typedef struct lx_r_debug {
        int r_version;          /* Version number for this protocol.  */
        uint32_t        r_map;  /* Head of the chain of loaded objects. */
```

```
127         /*
128          * This is the address of a function internal to the run-time linker,
129          * that will always be called when the linker begins to map in a
130          * library or unmap it, and again when the mapping change is complete.
131          * The debugger can set a breakpoint at this address if it wants to
132          * notice shared object mapping changes.
133          */
134         uint32_t        r_brk;
135         r_state_e       r_state; /* defined the same way between lx/solaris */
136         uint32_t        r_ldbase; /* Base address the linker is loaded at. */
137 } lx_r_debug_t;

139 static uint32_t
140 lx_ldb_getauxval32(struct ps_prochandle *php, int type)
141 {
142         const auxv_t            *auxvp = NULL;

144         if (ps_pauxv(php, &auxvp) != PS_OK)
145                 return ((uint32_t)-1);

147         while (auxvp->a_type != AT_NULL) {
148                 if (auxvp->a_type == type)
149                         return ((uint32_t)(uintptr_t)auxvp->a_un.a_ptr);
150                 auxvp++;
151         }
152         return ((uint32_t)-1);
153 }

155 /*
156  * A key difference between the linux linker and ours' is that the linux
157  * linker adds the base address of segments to certain values in the
158  * segments' ELF header. As an example, look at the address of the
159  * DT_HASH hash table in a Solaris section - it is a relative address
160  * which locates the start of the hash table, relative to the beginning
161  * of the ELF file. However, when the linux linker loads a section, it
162  * modifies the in-memory ELF image by changing address of the hash
163  * table to be an absolute address. This is only done for libraries - not for
164  * executables.
165  *
166  * Solaris tools expect the relative address to remain relative, so
167  * here we will modify the in-memory ELF image so that it once again
168  * contains relative addresses.
169  *
170  * To accomplish this, we walk through all sections in the target.
171  * Linux sections are identified by pointing to the linux linker or libc in the
172  * DT_NEEDED section. For all matching sections, we subtract the segment
173  * base address to get back to relative addresses.
174  */
175 static rd_err_e
176 lx_ldb_get_dyns32(rd_helper_data_t rhd,
177     psaddr_t addr, void **dynpp, size_t *dynpp_sz)
178 {
179         lx_rd_t                 *lx_rd = (lx_rd_t *)rhd;
180         rd_agent_t              *rap = lx_rd->lr_rap;
181         Elf32_Ehdr              ehdr;
182         Elf32_Dyn               *dynp = NULL;
183         size_t                  dynp_sz;
184         uint_t                  ndyns;
185         int                     i;

187         ps_plog("lx_ldb_get_dyns: invoked for object at 0x%p", addr);

189         /* Read in a copy of the ehdr */
190         if (_rd_get_ehdr32(rap, addr, &ehdr, NULL) != RD_OK) {
191                 ps_plog("lx_ldb_get_dyns: _rd_get_ehdr() failed");
192                 return (RD_ERR);
```



```
193         }

195         /* read out the PT_DYNAMIC elements for this object */
196         if (_rd_get_dyns32(rap, addr, &dynp, &dynp_sz) != RD_OK) {
197                 ps_plog("lx_ldb_get_dyns: _rd_get_dyns() failed");
198                 return (RD_ERR);
199         }

201         /*
202          * From here on out if we encounter an error we'll just return
203          * success and pass back the unmolested dynamic elements that
204          * we've already obtained.
205          */
206         *dynpp = dynp;
207         *dynpp_sz = dynp_sz;
208         ndyns = dynp_sz / sizeof (Elf32_Dyn);

210         /* If this isn't a dynamic object, there's nothing left todo */
211         if (ehdr.e_type != ET_DYN) {
212                 ps_plog("lx_ldb_get_dyns: done: not a shared object");
213                 return (RD_OK);
214         }

216         /*
217          * Before we blindly start changing dynamic section addresses
218          * we need to figure out if the current object that we're looking
219          * at is a linux object or a solaris object.  To do this first
220          * we need to find the string tab dynamic section element.
221          */
222         for (i = 0; i < ndyns; i++) {
223                 if (dynp[i].d_tag == DT_STRTAB)
224                         break;
225         }
226         if (i == ndyns) {
227                 ps_plog("lx_ldb_get_dyns: "
228                     "failed to find string tab in the dynamic section");
229                 return (RD_OK);
230         }

232         /*
233          * Check if the strtab value looks like an offset or an address.
234          * It's an offset if the value is less then the base address that
235          * the object is loaded at, or if the value is less than the offset
236          * of the section headers in the same elf object.  This check isn't
237          * perfect, but in practice it's good enough.
238          */
239         if ((dynp[i].d_un.d_ptr < addr) ||
240             (dynp[i].d_un.d_ptr < ehdr.e_shoff)) {
241                 ps_plog("lx_ldb_get_dyns: "
242                     "doesn't appear to be an lx object");
243                 return (RD_OK);
244         }

246         /*
247          * This seems to be a a linux object, so we'll patch up the dynamic
248          * section addresses
249          */
250         ps_plog("lx_ldb_get_dyns: "
251             "patching up lx object dynamic section addresses");
252         for (i = 0; i < ndyns; i++) {
253                 switch (dynp[i].d_tag) {
254                 case DT_PLTGOT:
255                 case DT_HASH:
256                 case DT_STRTAB:
257                 case DT_SYMTAB:
258                 case DT_RELA:
```

```
259                 case DT_REL:
260                 case DT_DEBUG:
261                 case DT_JMPREL:
262                 case DT_VERSYM:
263                         if (dynp[i].d_un.d_val > addr) {
264                                 dynp[i].d_un.d_ptr -= addr;
265                         }
266                         break;
267                 default:
268                         break;
269                 }
270         }
271         return (RD_OK);
272 }

274 static void
275 lx_ldb_fini32(rd_helper_data_t rhd)
276 {
277         lx_rd_t *lx_rd = (lx_rd_t *)rhd;
278         ps_plog("lx_ldb_fini: cleaning up lx helper");
279         free(lx_rd);
280 }

282 /*
283  * The linux linker has an r_debug structure somewhere in its data section that
284  * contains the address of the head of the link map list. To find this, we will
285  * use the DT_DEBUG token in the executable's dynamic section. The linux linker
286  * wrote the address of its r_debug structure to the DT_DEBUG dynamic entry. We
287  * get the address of the executable's program headers from the
288  * AT_SUN_BRAND_LX_PHDR aux vector entry. From there, we calculate the
289  * address of the Elf header, and from there we can easily get to the DT_DEBUG
290  * entry.
291  */
292 static rd_helper_data_t
293 lx_ldb_init32(rd_agent_t *rap, struct ps_prochandle *php)
294 {
295         lx_rd_t         *lx_rd;
296         uint32_t        addr, phdr_addr, dyn_addr;
297         Elf32_Dyn       *dyn;
298         Elf32_Phdr      phdr, *ph, *phdrs;
299         Elf32_Ehdr      ehdr;
300         int             i, dyn_count;

302         lx_rd = calloc(sizeof (lx_rd_t), 1);
303         if (lx_rd == NULL) {
304                 ps_plog("lx_ldb_init: cannot allocate memory");
305                 return (NULL);
306         }
307         lx_rd->lr_rap = rap;
308         lx_rd->lr_php = php;

310         phdr_addr = lx_ldb_getauxval32(php, AT_SUN_BRAND_LX_PHDR);
311         if (phdr_addr == (uint32_t)-1) {
312                 ps_plog("lx_ldb_init: no LX_PHDR found in aux vector");
313                 return (NULL);
314         }
315         ps_plog("lx_ldb_init: found LX_PHDR auxv phdr at: 0x%p",
316             phdr_addr);

318         if (ps_pread(php, phdr_addr, &phdr, sizeof (phdr)) != PS_OK) {
319                 ps_plog("lx_ldb_init: couldn't read phdr at 0x%p",
320                     phdr_addr);
321                 free(lx_rd);
322                 return (NULL);
323         }
```



```
325         /* The ELF headher should be before the program header in memory */
326         lx_rd->lr_exec = addr = phdr_addr - phdr.p_offset;
327         if (ps_pread(php, addr, &ehdr, sizeof (ehdr)) != PS_OK) {
328                 ps_plog("lx_ldb_init: couldn't read ehdr at 0x%p",
329                     lx_rd->lr_exec);
330                 free(lx_rd);
331                 return (NULL);
332         }
333         ps_plog("lx_ldb_init: read ehdr at: 0x%p", addr);

335         if ((phdrs = malloc(ehdr.e_phnum * ehdr.e_phentsize)) == NULL) {
336                 ps_plog("lx_ldb_init: couldn't alloc phdrs memory");
337                 free(lx_rd);
338                 return (NULL);
339         }

341         if (ps_pread(php, phdr_addr, phdrs, ehdr.e_phnum * ehdr.e_phentsize) !=
342             PS_OK) {
343                 ps_plog("lx_ldb_init: couldn't read phdrs at 0x%p",
344                     phdr_addr);
345                 free(lx_rd);
346                 free(phdrs);
347                 return (NULL);
348         }
349         ps_plog("lx_ldb_init: read %d phdrs at: 0x%p",
350             ehdr.e_phnum, phdr_addr);

352         for (i = 0, ph = phdrs; i < ehdr.e_phnum; i++,
353             /*LINTED */
354             ph = (Elf32_Phdr *)((char *)ph + ehdr.e_phentsize)) {
355                 if (ph->p_type == PT_DYNAMIC)
356                         break;
357         }
358         if (i == ehdr.e_phnum) {
359                 ps_plog("lx_ldb_init: no PT_DYNAMIC in executable");
360                 free(lx_rd);
361                 free(phdrs);
362                 return (NULL);
363         }
364         ps_plog("lx_ldb_init: found PT_DYNAMIC phdr[%d] at: 0x%p",
365             i, (phdr_addr + ((char *)ph - (char *)phdrs)));

367         if ((dyn = malloc(ph->p_filesz)) == NULL) {
368                 ps_plog("lx_ldb_init: couldn't alloc for PT_DYNAMIC");
369                 free(lx_rd);
370                 free(phdrs);
371                 return (NULL);
372         }

374         dyn_addr = addr + ph->p_offset;
375         dyn_count = ph->p_filesz / sizeof (Elf32_Dyn);
376         if (ps_pread(php, dyn_addr, dyn, ph->p_filesz) != PS_OK) {
377                 ps_plog("lx_ldb_init: couldn't read dynamic at 0x%p",
378                     dyn_addr);
379                 free(lx_rd);
380                 free(phdrs);
381                 free(dyn);
382                 return (NULL);
383         }
384         ps_plog("lx_ldb_init: read %d dynamic headers at: 0x%p",
385             dyn_count, dyn_addr);

387         for (i = 0; i < dyn_count; i++) {
388                 if (dyn[i].d_tag == DT_DEBUG) {
389                         lx_rd->lr_rdebug = dyn[i].d_un.d_ptr;
390                         break;
```

```
391                 }
392         }
393         free(phdrs);
394         free(dyn);
395
396         if (lx_rd->lr_rdebug == 0) {
397                 ps_plog("lx_ldb_init: no DT_DEBUG found in exe");
398                 free(lx_rd);
399                 return (NULL);
400         }
401         ps_plog("lx_ldb_init: found DT_DEBUG: 0x%p", lx_rd->lr_rdebug);
402
403         return ((rd_helper_data_t)lx_rd);
404 }
405
406 /*
407  * Given the address of an ELF object in the target, return its size and
408  * the proper link map ID.
409  */
410 static size_t
411 lx_elf_props32(struct ps_prochandle *php, uint32_t addr, psaddr_t *data_addr)
412 {
413         Elf32_Ehdr      ehdr;
414         Elf32_Phdr      *phdrs, *ph;
415         int             i;
416         uint32_t        min = (uint32_t)-1;
417         uint32_t        max = 0;
418         size_t          sz = NULL;
419
420         if (ps_pread(php, addr, &ehdr, sizeof (ehdr)) != PS_OK) {
421                 ps_plog("lx_elf_props: Couldn't read ELF header at 0x%p",
422                     addr);
423                 return (0);
424         }
425
426         if ((phdrs = malloc(ehdr.e_phnum * ehdr.e_phentsize)) == NULL)
427                 return (0);
428
429         if (ps_pread(php, addr + ehdr.e_phoff, phdrs, ehdr.e_phnum *
430             ehdr.e_phentsize) != PS_OK) {
431                 ps_plog("lx_elf_props: Couldn't read program headers at 0x%p",
432                     addr + ehdr.e_phoff);
433                 return (0);
434         }
435
436         for (i = 0, ph = phdrs; i < ehdr.e_phnum; i++,
437             /*LINTED */
438             ph = (Elf32_Phdr *)((char *)ph + ehdr.e_phentsize)) {
439
440                 if (ph->p_type != PT_LOAD)
441                         continue;
442
443                 if ((ph->p_flags & (PF_W | PF_R)) == (PF_W | PF_R)) {
444                         *data_addr = ph->p_vaddr;
445                         if (ehdr.e_type == ET_DYN)
446                                 *data_addr += addr;
447                         if (*data_addr & (ph->p_align - 1))
448                                 *data_addr = *data_addr & (~(ph->p_align -1));
449                 }
450
451                 if (ph->p_vaddr < min)
452                         min = ph->p_vaddr;
453
454                 if (ph->p_vaddr > max) {
455                         max = ph->p_vaddr;
456                         sz = ph->p_memsz + max - min;
```



```
457                         if (sz & (ph->p_align - 1))
458                                 sz = (sz & (~(ph->p_align - 1))) + ph->p_align;
459                 }
460         }
461
462         free(phdrs);
463         return (sz);
464 }
465
466 static int
467 lx_ldb_loadobj_iter32(rd_helper_data_t rhd, rl_iter_f *cb, void *client_data)
468 {
469         lx_rd_t                 *lx_rd = (lx_rd_t *)rhd;
470         struct ps_prochandle    *php = lx_rd->lr_php;
471         lx_r_debug_t            r_debug;
472         lx_link_map_t           map;
473         uint32_t                p = NULL;
474         int                     rc;
475         rd_loadobj_t            exec;
476
477         if ((rc = ps_pread(php, (psaddr_t)lx_rd->lr_rdebug, &r_debug,
478             sizeof (r_debug))) != PS_OK) {
479                 ps_plog("lx_ldb_loadobj_iter: "
480                     "Couldn't read linux r_debug at 0x%p", lx_rd->lr_rdebug);
481                 return (rc);
482         }
483
484         p = r_debug.r_map;
485
486         /*
487          * The first item on the link map list is for the executable, but it
488          * doesn't give us any useful information about it. We need to
489          * synthesize a rd_loadobj_t for the client.
490          *
491          * Linux doesn't give us the executable name, so we'll get it from
492          * the AT_EXECNAME entry instead.
493          */
494         if ((rc = ps_pread(php, (psaddr_t)p, &map, sizeof (map))) != PS_OK) {
495                 ps_plog("lx_ldb_loadobj_iter: "
496                     "Couldn't read linux link map at 0x%p", p);
497                 return (rc);
498         }
499
500         bzero(&exec, sizeof (exec));
501         exec.rl_base = lx_rd->lr_exec;
502         exec.rl_dynamic = map.lxm_ld;
503         exec.rl_nameaddr = lx_ldb_getauxval32(php, AT_SUN_EXECNAME);
504         exec.rl_lmident = LM_ID_BASE;
505
506         exec.rl_bend = exec.rl_base +
507             lx_elf_props32(php, lx_rd->lr_exec, &exec.rl_data_base);
508
509         if ((*cb)(&exec, client_data) == 0) {
510                 ps_plog("lx_ldb_loadobj_iter: "
511                     "client callb failed for executable");
512                 return (PS_ERR);
513         }
514
515         for (p = map.lxm_next; p != NULL; p = map.lxm_next) {
516                 rd_loadobj_t    obj;
517
518                 if ((rc = ps_pread(php, (psaddr_t)p, &map, sizeof (map))) !=
519                     PS_OK) {
520                         ps_plog("lx_ldb_loadobj_iter: "
521                             "Couldn't read lk map at %p", p);
522                         return (rc);
```

```
523                 }
525                 /*
526                  * The linux link map has less information than the Solaris one.
527                  * We need to go fetch the missing information from the ELF
528                  * headers.
529                  */
531                 obj.rl_nameaddr = (psaddr_t)map.lxm_name;
532                 obj.rl_base = map.lxm_addr;
533                 obj.rl_refnameaddr = (psaddr_t)map.lxm_name;
534                 obj.rl_plt_base = NULL;
535                 obj.rl_plt_size = 0;
536                 obj.rl_lmident = LM_ID_BASE;
538                 /*
539                  * Ugh - we have to walk the ELF stuff, find the PT_LOAD
540                  * sections, and calculate the end of the file's mappings
541                  * ourselves.
542                  */
544                 obj.rl_bend = map.lxm_addr +
545                     lx_elf_props32(php, map.lxm_addr, &obj.rl_data_base);
546                 obj.rl_padstart = obj.rl_base;
547                 obj.rl_padend = obj.rl_bend;
548                 obj.rl_dynamic = map.lxm_ld;
549                 obj.rl_tlsmodid = 0;
551                 ps_plog("lx_ldb_loadobj_iter: 0x%p to 0x%p",
552                     obj.rl_base, obj.rl_bend);
554                 if ((*cb)(&obj, client_data) == 0) {
555                         ps_plog("lx_ldb_loadobj_iter: "
556                             "client callback failed on %s", map.lxm_name);
557                         return (rc);
558                 }
559         }
560         return (RD_OK);
561 }
563 /*
564  * Librtld_db plugin linkage struct.
565  *
566  * When we get loaded by librtld_db, it will look for the symbol below
567  * to find our plugin entry points.
568  */
569 rd_helper_ops_t RTLD_DB_BRAND_OPS = {
570         LM_ID_BRAND,
571         lx_ldb_init32,
572         lx_ldb_fini32,
573         lx_ldb_loadobj_iter32,
574         lx_ldb_get_dyns32
575 };
576 #endif /* ! codereview */
```

****
1572 Tue Jan 14 16:16:59 2014
new/usr/src/lib/brand/lx/librtld_db/common/mapfile-vers
Bring back LX zones.
****

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

#
# MAPFILE HEADER START
#
# WARNING:  STOP NOW.  DO NOT MODIFY THIS FILE.
# Object versioning must comply with the rules detailed in
#
#       usr/src/lib/README.mapfiles
#
# You should not be making modifications here until you've read the most current
# copy of that file. If you need help, contact a gatekeeper for guidance.
#
# MAPFILE HEADER END
#

{
        global:
                rtld_db_brand_ops32;
        local:
                *;
};

#Externally defined symbols
{
    global:
        ps_pauxv =              NODIRECT PARENT;
        ps_pdmodel =            NODIRECT PARENT;
        ps_pglobal_lookup =     NODIRECT PARENT;
        ps_pglobal_sym =        NODIRECT PARENT;
        ps_plog =               NODIRECT PARENT;
        ps_pread =              NODIRECT PARENT;
        ps_pwrite =             NODIRECT PARENT;
};
#endif /* ! codereview */
```

```
**********************************************************
    1022 Tue Jan 14 16:16:59 2014
new/usr/src/lib/brand/lx/librtld_db/i386/Makefile
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
#ident  "%Z%%M% %I%     %E% SMI"
#

include ../Makefile.com

CLOBBERFILES =  $(ROOTLIBDIR)/$(DYNLIB)

install: all $(ROOTLIBS)
#endif /* ! codereview */
```

****
1328 Tue Jan 14 16:16:59 2014
new/usr/src/lib/brand/lx/lx_brand/Makefile
Bring back LX zones.
****

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "%Z%%M% %I%     %E% SMI"
#

include ../../../Makefile.lib

default:        all

SUBDIRS=        $(MACH)

LINT_SUBDIRS=   $(MACH)

all :=          TARGET= all
clean :=        TARGET= clean
clobber :=      TARGET= clobber
install :=      TARGET= install
lint :=         TARGET= lint
_msg :=         TARGET= _msg

.KEEP_STATE:

all install clean clobber _msg: $(SUBDIRS)

lint: $(LINT_SUBDIRS)

$(SUBDIRS): FRC
        @cd $@; pwd; $(MAKE) $(TARGET)

FRC:
#endif /* ! codereview */
```

```
**********************************************************
    2351 Tue Jan 14 16:16:59 2014
new/usr/src/lib/brand/lx/lx_brand/Makefile.com
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "%Z%%M% %I%     %E% SMI"
#

LX_CMN  =       $(SRC)/common/brand/lx

LIBRARY =       lx_brand.a
VERS    =       .1
COBJS   =       clock.o                 \
                clone.o                 \
                debug.o                 \
                dir.o                   \
                file.o                  \
                fcntl.o                 \
                fork.o                  \
                id.o                    \
                ioctl.o                 \
                iovec.o                 \
                lx_brand.o              \
                lx_thunk_server.o       \
                mem.o                   \
                misc.o                  \
                module.o                \
                mount.o                 \
                open.o                  \
                pgrp.o                  \
                poll_select.o           \
                priority.o              \
                ptrace.o                \
                rlimit.o                \
                sched.o                 \
                sendfile.o              \
                signal.o                \
                socket.o                \
                stat.o                  \
                statfs.o                \
                sysctl.o                \
                sysv_ipc.o              \
```



```
                time.o                  \
                truncate.o              \
                wait.o

CMNOBJS =       lx_signum.o
ASOBJS  =       lx_handler.o lx_runexe.o lx_crt.o
OBJECTS =       $(CMNOBJS) $(COBJS) $(ASOBJS)

include ../../Makefile.lx
include ../../../../Makefile.lib

CSRCS   =       $(COBJS:%o=../common/%c) $(CMNOBJS:%o=$(LX_CMN)/%c)
ASSRCS  =       $(ASOBJS:%o=$(ISASRCDIR)/%s)
SRCS    =       $(CSRCS) $(ASSRCS)

SRCDIR =        ../common
UTSBASE =       ../../../../../uts

LIBS =          $(DYNLIB)
LDLIBS +=       -lc -lsocket -lmapmalloc -lproc -lrtld_db
DYNFLAGS +=     -Wl,-e_start -Wl,-I/native/lib/ld.so.1 -M../common/mapfile
CFLAGS +=       $(CCVERBOSE)
CPPFLAGS +=     -D_REENTRANT -I../ -I$(UTSBASE)/common/brand/lx -I$(LX_CMN)
ASFLAGS =       -P $(ASFLAGS_$(CURTYPE)) -D_ASM -I../   \
                        -I$(UTSBASE)/common/brand/lx

.KEEP_STATE:

all: $(LIBS)

lint: lintcheck

include ../../../../Makefile.targ

pics/%.o: $(ISASRCDIR)/%.s
        $(COMPILE.s) -o $@ $<
        $(POST_PROCESS_O)

pics/%.o: $(LX_CMN)/%.c
        $(COMPILE.c) -o $@ $<
        $(POST_PROCESS_O)
#endif /* ! codereview */
```

```
**********************************************************
    2994 Tue Jan 14 16:17:00 2014
new/usr/src/lib/brand/lx/lx_brand/common/clock.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <errno.h>
#include <string.h>
#include <time.h>
#include <sys/lx_misc.h>

/*
 * Linux uses different values for it clock identifiers, so we have to do basic
 * translations between the two.  Thankfully, both Linux and Solaris implement
 * the same POSIX SUSv3 clock types, so the semantics should be identical.
 */

static int ltos_clock[] = {
        CLOCK_REALTIME,
        CLOCK_MONOTONIC,
        CLOCK_PROCESS_CPUTIME_ID,
        CLOCK_THREAD_CPUTIME_ID
};

#define LX_CLOCK_MAX    (sizeof (ltos_clock) / sizeof (ltos_clock[0]))

int
lx_clock_gettime(int clock, struct timespec *tp)
{
        struct timespec ts;

        if (clock < 0 || clock > LX_CLOCK_MAX)
                return (-EINVAL);

        if (clock_gettime(ltos_clock[clock], &ts) < 0)
                return (-errno);

        return ((uucopy(&ts, tp, sizeof (struct timespec)) < 0) ? -EFAULT : 0);
}
```



```
int
lx_clock_settime(int clock, struct timespec *tp)
{
        struct timespec ts;

        if (clock < 0 || clock > LX_CLOCK_MAX)
                return (-EINVAL);

        if (uucopy(tp, &ts, sizeof (struct timespec)) < 0)
                return (-EFAULT);

        return ((clock_settime(ltos_clock[clock], &ts) < 0) ? -errno : 0);
}

int
lx_clock_getres(int clock, struct timespec *tp)
{
        struct timespec ts;

        if (clock < 0 || clock > LX_CLOCK_MAX)
                return (-EINVAL);

        if (clock_getres(ltos_clock[clock], &ts) < 0)
                return (-errno);

        return ((uucopy(&ts, tp, sizeof (struct timespec)) < 0) ? -EFAULT : 0);
}

int
lx_clock_nanosleep(int clock, int flags, struct timespec *rqtp,
    struct timespec *rmtp)
{
        struct timespec rqt, rmt;

        if (clock < 0 || clock > LX_CLOCK_MAX)
                return (-EINVAL);

        if (uucopy(rqtp, &rqt, sizeof (struct timespec)) < 0)
                return (-EFAULT);

        /* the TIMER_RELTIME and TIMER_ABSTIME flags are the same on Linux */
        if (clock_nanosleep(ltos_clock[clock], flags, &rqt, &rmt) < 0)
                return (-errno);

        /*
         * Only copy values to rmtp if the timer is TIMER_RELTIME and rmtp is
         * non-NULL.
         */
        if (((flags & TIMER_RELTIME) == TIMER_RELTIME) && (rmtp != NULL) &&
            (uucopy(&rmt, rmtp, sizeof (struct timespec)) < 0))
                return (-EFAULT);

        return (0);
}
#endif /* ! codereview */
```

```
**********************************************************
   14435 Tue Jan 14 16:17:00 2014
new/usr/src/lib/brand/lx/lx_brand/common/clone.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <assert.h>
#include <errno.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <ucontext.h>
#include <thread.h>
#include <strings.h>
#include <libintl.h>
#include <sys/regset.h>
#include <sys/syscall.h>
#include <sys/inttypes.h>
#include <sys/param.h>
#include <sys/types.h>
#include <sys/segments.h>
#include <signal.h>
#include <sys/lx_misc.h>
#include <sys/lx_types.h>
#include <sys/lx_signal.h>
#include <sys/lx_syscall.h>
#include <sys/lx_brand.h>
#include <sys/lx_debug.h>
#include <sys/lx_thread.h>

#define LX_CSIGNAL              0x000000ff
#define LX_CLONE_VM             0x00000100
#define LX_CLONE_FS             0x00000200
#define LX_CLONE_FILES          0x00000400
#define LX_CLONE_SIGHAND        0x00000800
#define LX_CLONE_PID            0x00001000
#define LX_CLONE_PTRACE         0x00002000
#define LX_CLONE_VFORK          0x00004000
#define LX_CLONE_PARENT         0x00008000
```



```
#define LX_CLONE_THREAD         0x00010000
#define LX_CLONE_SYSVSEM        0x00040000
#define LX_CLONE_SETTLS         0x00080000
#define LX_CLONE_PARENT_SETTID  0x00100000
#define LX_CLONE_CHILD_CLEARTID 0x00200000
#define LX_CLONE_DETACH         0x00400000
#define LX_CLONE_CHILD_SETTID   0x01000000

#define SHARED_AS       \
        (LX_CLONE_VM | LX_CLONE_FS | LX_CLONE_FILES | LX_CLONE_SIGHAND)
#define CLONE_VFORK (LX_CLONE_VM | LX_CLONE_VFORK)
#define CLONE_TD (LX_CLONE_THREAD|LX_CLONE_DETACH)

#define IS_FORK(f)      (((f) & SHARED_AS) == 0)
#define IS_VFORK(f)     (((f) & CLONE_VFORK) == CLONE_VFORK)

#define LX_EXIT         1
#define LX_EXIT_GROUP   2

/*
 * This is dicey.  This seems to be an internal glibc structure, and not
 * part of any external interface.  Thus, it is subject to change without
 * notice.  FWIW, clone(2) itself seems to be an internal (or at least
 * unstable) interface, since strace(1) shows it differently than the man
 * page.
 */
struct lx_desc
{
        uint32_t entry_number;
        uint32_t base_addr;
        uint32_t limit;
        uint32_t seg_32bit:1;
        uint32_t contents:2;
        uint32_t read_exec_only:1;
        uint32_t limit_in_pages:1;
        uint32_t seg_not_present:1;
        uint32_t useable:1;
        uint32_t empty:25;
};

struct clone_state {
        void            *c_retaddr;     /* instr after clone()'s int80 */
        int             c_flags;        /* flags to clone(2) */
        int             c_sig;          /* signal to send on thread exit */
        void            *c_stk;         /* %esp of new thread */
        void            *c_ptidp;
        struct lx_desc  *c_ldtinfo;     /* thread-specific segment */
        void            *c_ctidp;
        uintptr_t       c_gs;           /* Linux's %gs */
        sigset_t        c_sigmask;      /* signal mask */
        lx_affmask_t    c_affmask;      /* CPU affinity mask */
        volatile int    *c_clone_res;   /* pid/error returned to cloner */
};

extern void lx_setup_clone(uintptr_t, void *, void *);

/*
 * Counter incremented when we vfork(2) ourselves, and decremented when the
 * vfork(2)ed child exit(2)s or exec(2)s.
 */
static int is_vforked = 0;

int
lx_exit(uintptr_t p1)
{
        int             ret, status = (int)p1;
```

```
128         lx_tsd_t        *lx_tsd;
129
130         /*
131          * If we are a vfork(2)ed child, we need to exit as quickly and
132          * cleanly as possible to avoid corrupting our parent.
133          */
134         if (is_vforked != 0) {
135                 is_vforked--;
136                 _exit(status);
137         }
138
139         if ((ret = thr_getspecific(lx_tsd_key, (void **)&lx_tsd)) != 0)
140                 lx_err_fatal(gettext(
141                     "%s: unable to read thread-specific data: %s"),
142                     "exit", strerror(ret));
143
144         assert(lx_tsd != 0);
145
146         lx_tsd->lxtsd_exit = LX_EXIT;
147         lx_tsd->lxtsd_exit_status = status;
148
149         /*
150          * Block all signals in the exit context to avoid taking any signals
151          * (to the degree possible) while exiting.
152          */
153         (void) sigfillset(&lx_tsd->lxtsd_exit_context.uc_sigmask);
154
155         /*
156          * This thread is exiting.  Restore the state of the thread to
157          * what it was before we started running linux code.
158          */
159         (void) setcontext(&lx_tsd->lxtsd_exit_context);
160
161         /*
162          * If we returned from the setcontext(2), something is very wrong.
163          */
164         lx_err_fatal(gettext("%s: unable to set exit context: %s"),
165             "exit", strerror(errno));
166
167         /*NOTREACHED*/
168         return (0);
169 }
170
171 int
172 lx_group_exit(uintptr_t p1)
173 {
174         int             ret, status = (int)p1;
175         lx_tsd_t        *lx_tsd;
176
177         /*
178          * If we are a vfork(2)ed child, we need to exit as quickly and
179          * cleanly as possible to avoid corrupting our parent.
180          */
181         if (is_vforked != 0) {
182                 is_vforked--;
183                 _exit(status);
184         }
185
186         if ((ret = thr_getspecific(lx_tsd_key, (void **)&lx_tsd)) != 0)
187                 lx_err_fatal(gettext(
188                     "%s: unable to read thread-specific data: %s"),
189                     "group_exit", strerror(ret));
190
191         assert(lx_tsd != 0);
192
193         lx_tsd->lxtsd_exit = LX_EXIT_GROUP;
```



```
194         lx_tsd->lxtsd_exit_status = status;
195
196         /*
197          * Block all signals in the exit context to avoid taking any signals
198          * (to the degree possible) while exiting.
199          */
200         (void) sigfillset(&lx_tsd->lxtsd_exit_context.uc_sigmask);
201
202         /*
203          * This thread is exiting.  Restore the state of the thread to
204          * what it was before we started running linux code.
205          */
206         (void) setcontext(&lx_tsd->lxtsd_exit_context);
207
208         /*
209          * If we returned from the setcontext(2), something is very wrong.
210          */
211         lx_err_fatal(gettext("%s: unable to set exit context: %s"),
212             "group_exit", strerror(errno));
213
214         /*NOTREACHED*/
215         return (0);
216 }
217
218 static void *
219 clone_start(void *arg)
220 {
221         int rval;
222         struct clone_state *cs = (struct clone_state *)arg;
223         lx_tsd_t lx_tsd;
224
225         /*
226          * Let the kernel finish setting up all the needed state for this
227          * new thread.
228          *
229          * We already created the thread using the thr_create(3C) library
230          * call, so most of the work required to emulate lx_clone(2) has
231          * been done by the time we get to this point.  Instead of creating
232          * a new brandsys(2) subcommand to perform the last few bits of
233          * bookkeeping, we just use the lx_clone() slot in the syscall
234          * table.
235          */
236         lx_debug("\tre-vectoring to lx kernel module to complete lx_clone()");
237         lx_debug("\tLX_SYS_clone(0x%x, 0x%p, 0x%p, 0x%p, 0x%p)",
238             cs->c_flags, cs->c_stk, cs->c_ptidp, cs->c_ldtinfo, cs->c_ctidp);
239
240         rval = syscall(SYS_brand, B_EMULATE_SYSCALL + LX_SYS_clone,
241             cs->c_flags, cs->c_stk, cs->c_ptidp, cs->c_ldtinfo, cs->c_ctidp,
242             NULL);
243
244         /*
245          * At this point the parent is waiting for cs->c_clone_res to go
246          * non-zero to indicate the thread has been cloned.  The value set
247          * in cs->c_clone_res will be used for the return value from
248          * clone().
249          */
250         if (rval < 0) {
251                 *(cs->c_clone_res) = -errno;
252                 lx_debug("\tkernel clone failed, errno %d\n", errno);
253                 return (NULL);
254         }
255
256         if (lx_sched_setaffinity(0, sizeof (cs->c_affmask),
257             (uintptr_t)&cs->c_affmask) != 0) {
258                 *(cs->c_clone_res) = -errno;
```

```
260                 lx_err_fatal(gettext(
261                     "Unable to set affinity mask in child thread: %s"),
262                     strerror(errno));
263         }

265         /* Initialize the thread specific data for this thread. */
266         bzero(&lx_tsd, sizeof (lx_tsd));
267         lx_tsd.lxtsd_gs = cs->c_gs;

269         /*
270          * Use the address of the stack-allocated lx_tsd as the
271          * per-thread storage area to cache various values for later
272          * use.
273          *
274          * This address is only used by this thread, so there is no
275          * danger of other threads using this storage area, nor of it
276          * being accessed once this stack frame has been freed.
277          */
278         if (thr_setspecific(lx_tsd_key, &lx_tsd) != 0) {
279                 *(cs->c_clone_res) = -errno;
280                 lx_err_fatal(
281                     gettext("Unable to set thread-specific ptr for clone: %s"),
282                     strerror(rval));
283         }

285         /*
286          * Save the current context of this thread.
287          *
288          * We'll restore this context when this thread attempts to exit.
289          */
290         if (getcontext(&lx_tsd.lxtsd_exit_context) != 0) {
291                 *(cs->c_clone_res) = -errno;

293                 lx_err_fatal(gettext(
294                     "Unable to initialize thread-specific exit context: %s"),
295                     strerror(errno));
296         }

298         /*
299          * Do the final stack twiddling, reset %gs, and return to the
300          * clone(2) path.
301          */
302         if (lx_tsd.lxtsd_exit == 0) {
303                 if (sigprocmask(SIG_SETMASK, &cs->c_sigmask, NULL) < 0) {
304                         *(cs->c_clone_res) = -errno;

306                         lx_err_fatal(gettext(
307                             "Unable to release held signals for child "
308                             "thread: %s"), strerror(errno));
309                 }

311                 /*
312                  * Let the parent know that the clone has (effectively) been
313                  * completed.
314                  */
315                 *(cs->c_clone_res) = rval;

317                 lx_setup_clone(cs->c_gs, cs->c_retaddr, cs->c_stk);

319                 /* lx_setup_clone() should never return. */
320                 assert(0);
321         }

323         /*
324          * We are here because the Linux application called the exit() or
325          * exit_group() system call.  In turn the brand library did a
```



```
326          * setcontext() to jump to the thread context state saved in
327          * getcontext(), above.
328          */
329         if (lx_tsd.lxtsd_exit == LX_EXIT)
330                 thr_exit((void *)lx_tsd.lxtsd_exit_status);
331         else
332                 exit(lx_tsd.lxtsd_exit_status);

334         assert(0);
335         /*NOTREACHED*/
336 }

338 int
339 lx_clone(uintptr_t p1, uintptr_t p2, uintptr_t p3, uintptr_t p4,
340         uintptr_t p5)
341 {
342         struct clone_state *cs;
343         int flags = (int)p1;
344         void *cldstk = (void *)p2;
345         void *ptidp = (void *)p3;
346         struct lx_desc *ldtinfo = (void *)p4;
347         void *ctidp = (void *)p5;
348         thread_t tid;
349         volatile int clone_res;
350         int sig;
351         int rval;
352         int pid;
353         lx_regs_t *rp;
354         sigset_t sigmask;

356         if (flags & LX_CLONE_SETTLS) {
357                 lx_debug("lx_clone(flags=0x%x stk=0x%p ptidp=0x%p ldt=0x%p "
358                     "ctidp=0x%p", flags, cldstk, ptidp, ldtinfo, ctidp);
359         } else {
360                 lx_debug("lx_clone(flags=0x%x stk=0x%p ptidp=0x%p)",
361                     flags, cldstk, ptidp);
362         }

364         /*
365          * Only supported for pid 0 on Linux
366          */
367         if (flags & LX_CLONE_PID)
368                 return (-EINVAL);

370         /*
371          * CLONE_THREAD requires CLONE_SIGHAND.
372          *
373          * CLONE_THREAD and CLONE_DETACHED must both be either set or cleared
374          * in kernel 2.4 and prior.
375          * In kernel 2.6 CLONE_DETACHED was dropped completely, so we no
376          * longer have this requirement.
377          */

379         if (flags & CLONE_TD) {
380                 if (!(flags & LX_CLONE_SIGHAND))
381                         return (-EINVAL);
382                 if ((lx_get_kern_version() <= LX_KERN_2_4) &&
383                     (flags & CLONE_TD) != CLONE_TD)
384                         return (-EINVAL);
385         }

387         rp = lx_syscall_regs();

389         /* test if pointer passed by user are writable */
390         if (flags & LX_CLONE_PARENT_SETTID) {
391                 if (uucopy(ptidp, &pid, sizeof (int)) != 0)
```

```
392                     return (-EFAULT);
393             if (uucopy(&pid, ptidp, sizeof (int)) != 0)
394                     return (-EFAULT);
395     }
396     if (flags & LX_CLONE_CHILD_SETTID) {
397             if (uucopy(ctidp, &pid, sizeof (int)) != 0)
398                     return (-EFAULT);
399             if (uucopy(&pid, ctidp, sizeof (int)) != 0)
400                     return (-EFAULT);
401     }

403     /* See if this is a fork() operation or a thr_create().  */
404     if (IS_FORK(flags) || IS_VFORK(flags)) {
405             if (flags & LX_CLONE_PARENT) {
406                     lx_unsupported(gettext(
407                         "clone(2) only supports CLONE_PARENT "
408                         "for threads.\n"));
409                     return (-ENOTSUP);
410             }

412             if (flags & LX_CLONE_PTRACE)
413                     lx_ptrace_fork();

415             if (flags & LX_CLONE_VFORK) {
416                     is_vforked++;
417                     rval = vfork();
418                     if (rval != 0)
419                             is_vforked--;
420             } else {
421                     rval = fork1();
422                     if (rval == 0 && lx_is_rpm)
423                             (void) sleep(lx_rpm_delay);
424             }

426             /*
427              * Since we've already forked, we can't do much if uucopy fails,
428              * so we just ignore failure. Failure is unlikely since we've
429              * tested the memory before we did the fork.
430              */
431             if (rval > 0 && (flags & LX_CLONE_PARENT_SETTID)) {
432                     (void) uucopy(&rval, ptidp, sizeof (int));
433             }

435             if (rval == 0 && (flags & LX_CLONE_CHILD_SETTID)) {
436                     /*
437                      * lx_getpid should not fail, and if it does, there's
438                      * not much we can do about it since we've already
439                      * forked, so on failure, we just don't copy the
440                      * memory.
441                      */
442                     pid = lx_getpid();
443                     if (pid >= 0)
444                             (void) uucopy(&pid, ctidp, sizeof (int));
445             }

447             /* Parent just returns */
448             if (rval != 0)
449                     return ((rval < 0) ? -errno : rval);

451             /*
452              * If provided, the child needs its new stack set up.
453              */
454             if (cldstk)
455                     lx_setup_clone(rp->lxr_gs, (void *)rp->lxr_eip, cldstk);

457             return (0);
```



```
458     }

460     /*
461      * We have very restricted support.... only exactly these flags are
462      * supported
463      */
464     if (((flags & SHARED_AS) != SHARED_AS)) {
465             lx_unsupported(gettext(
466                 "clone(2) requires that all or none of CLONE_VM "
467                 "CLONE_FS, CLONE_FILES, and CLONE_SIGHAND be set.\n"));
468             return (-ENOTSUP);
469     }

471     if (cldstk == NULL) {
472             lx_unsupported(gettext(
473                 "clone(2) requires the caller to allocate the "
474                 "child's stack.\n"));
475             return (-ENOTSUP);
476     }

478     /*
479      * If we want a signal-on-exit, ensure that the signal is valid.
480      */
481     if ((sig = ltos_signo[flags & LX_CSIGNAL]) == -1) {
482             lx_unsupported(gettext(
483                 "clone(2) passed unsupported signal: %d"), sig);
484             return (-ENOTSUP);
485     }

487     /*
488      * To avoid malloc() here, we steal a part of the new thread's
489      * stack to store all the info that thread might need for
490      * initialization.  We also make it 64-bit aligned for good
491      * measure.
492      */
493     cs = (struct clone_state *)
494         ((p2 - sizeof (struct clone_state)) & -((uintptr_t)8));
495     cs->c_flags = flags;
496     cs->c_sig = sig;
497     cs->c_stk = cldstk;
498     cs->c_ptidp = ptidp;
499     cs->c_ldtinfo = ldtinfo;
500     cs->c_ctidp = ctidp;
501     cs->c_clone_res = &clone_res;
502     cs->c_gs = rp->lxr_gs;

504     if (lx_sched_getaffinity(0, sizeof (cs->c_affmask),
505         (uintptr_t)&cs->c_affmask) == -1)
506             lx_err_fatal(gettext(
507                 "Unable to get affinity mask for parent thread: %s"),
508                 strerror(errno));

510     /*
511      * We want the new thread to return directly to the return site for
512      * the system call.
513      */
514     cs->c_retaddr = (void *)rp->lxr_eip;
515     clone_res = 0;

517     (void) sigfillset(&sigmask);

519     /*
520      * Block all signals because the thread we create won't be able to
521      * properly handle them until it's fully set up.
522      */
523     if (sigprocmask(SIG_BLOCK, &sigmask, &cs->c_sigmask) < 0) {
```

```
524                 lx_debug("lx_clone sigprocmask() failed: %s", strerror(errno));
525                 return (-errno);
526         }

528         rval = thr_create(NULL, NULL, clone_start, cs, THR_DETACHED, &tid);

530         /*
531          * Release any pending signals
532          */
533         (void) sigprocmask(SIG_SETMASK, &cs->c_sigmask, NULL);

535         /*
536          * Wait for the child to be created and have its tid assigned.
537          */
538         if (rval == 0) {
539                 while (clone_res == 0)
540                         ;

542                 rval = clone_res;
543         }

545         return (rval);
546 }
547 #endif /* ! codereview */
```

```
**********************************************************
    3775 Tue Jan 14 16:17:00 2014
new/usr/src/lib/brand/lx/lx_brand/common/debug.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <assert.h>
#include <errno.h>
#include <fcntl.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <thread.h>
#include <unistd.h>

#include <sys/modctl.h>
#include <sys/stat.h>
#include <sys/types.h>

#include <sys/lx_brand.h>
#include <sys/lx_debug.h>
#include <sys/lx_misc.h>

/* internal debugging state */
static char     *lx_debug_path = NULL;          /* debug output file path */
static char     lx_debug_path_buf[MAXPATHLEN];

void
lx_debug_enable(void)
{
        /* send all debugging output to /dev/tty */
        lx_debug_path = "/dev/tty";
        lx_debug("lx_debug: debugging output enabled: %s", lx_debug_path);
}

void
lx_debug_init(void)
{
        if (getenv("LX_DEBUG") == NULL)
```



```
                return;

        /*
         * It's OK to use this value without any locking, as all callers can
         * use the return value to decide whether extra work should be done
         * before calling lx_debug().
         *
         * If debugging is disabled after a routine calls this function it
         * doesn't really matter as lx_debug() will see debugging is disabled
         * and will not output anything.
         */
        lx_debug_enabled = 1;

        /* check if there's a debug log file specified */
        lx_debug_path = getenv("LX_DEBUG_FILE");
        if (lx_debug_path == NULL) {
                /* send all debugging output to /dev/tty */
                lx_debug_path = "/dev/tty";
        }

        (void) strlcpy(lx_debug_path_buf, lx_debug_path,
            sizeof (lx_debug_path_buf));
        lx_debug_path = lx_debug_path_buf;

        lx_debug("lx_debug: debugging output ENABLED to path: \"%s\"",
            lx_debug_path);
}

void
lx_debug(const char *msg, ...)
{
        va_list         ap;
        char            buf[LX_MSG_MAXLEN + 1];
        int             rv, fd, n;
        int             errno_backup;

        if (lx_debug_enabled == 0)
                return;

        errno_backup = errno;

        /* prefix the message with pid/tid */
        if ((n = snprintf(buf, sizeof (buf), "%u/%u: ",
            getpid(), thr_self())) == -1) {
                errno = errno_backup;
                return;
        }

        /* format the message */
        va_start(ap, msg);
        rv = vsnprintf(&buf[n], sizeof (buf) - n, msg, ap);
        va_end(ap);
        if (rv == -1) {
                errno = errno_backup;
                return;
        }

        /* add a carrige return if there isn't one already */
        if ((buf[strlen(buf) - 1] != '\n') &&
            (strlcat(buf, "\n", sizeof (buf)) >= sizeof (buf))) {
                errno = errno_backup;
                return;
        }

        /*
         * Open the debugging output file.  note that we don't protect
```

```
128          * ourselves against exec or fork1 here.  if an mt process were
129          * to exec/fork1 while we're doing this they'd end up with an
130          * extra open desciptor in their fd space.  a'well.  shouldn't
131          * really matter.
132          */
133         if ((fd = open(lx_debug_path,
134             O_WRONLY|O_APPEND|O_CREAT|O_NDELAY|O_NOCTTY, 0666)) == -1) {
135                 return;
136         }
137         (void) fchmod(fd, 0666);

139         /* we retry in case of EINTR */
140         do {
141                 rv = write(fd, buf, strlen(buf));
142         } while ((rv == -1) && (errno == EINTR));
143         (void) fsync(fd);

145         (void) close(fd);
146         errno = errno_backup;
147 }
148 #endif /* ! codereview */
```

```
**********************************************************
    3976 Tue Jan 14 16:17:00 2014
new/usr/src/lib/brand/lx/lx_brand/common/dir.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <string.h>
#include <stddef.h>
#include <errno.h>
#include <unistd.h>
#include <assert.h>
#include <sys/types.h>
#include <sys/systm.h>
#include <sys/dirent.h>
#include <sys/lx_misc.h>
#include <sys/lx_debug.h>

#define LX_NAMEMAX      256

struct lx_dirent {
        long            d_ino;  /* not l_ino_t */
        long            d_off;
        ushort_t        d_reclen;
        char            d_name[LX_NAMEMAX];
};

struct lx_dirent64 {
        uint64_t        d_ino;
        int64_t         d_off;
        ushort_t        d_reclen;
        uchar_t         d_type;
        char            d_name[LX_NAMEMAX];
};

#define LX_RECLEN(namelen)      \
        ((offsetof(struct lx_dirent64, d_name) + 1 + (namelen) + 7) & ~7)

/*
 * Read in one dirent structure from fd into dirp.
 * p3 (count) is ignored.
```



```
 */
/*ARGSUSED*/
int
lx_readdir(uintptr_t p1, uintptr_t p2, uintptr_t p3)
{
        int fd = (int)p1;
        struct lx_dirent *dirp = (struct lx_dirent *)p2;
        uint_t count = sizeof (struct lx_dirent);
        int rc = 0;
        struct lx_dirent _ld;
        struct dirent *sd = (struct dirent *)&_ld;

        /*
         * The return value from getdents is not applicable, as
         * it might have squeezed more than one dirent in the buffer
         * we provided.
         *
         * getdents() will deal with the case of dirp == NULL
         */
        if ((rc = getdents(fd, sd, count)) < 0)
                return (-errno);

        /*
         * Set rc 1 (pass), or 0 (end of directory).
         */
        rc = (sd->d_reclen == 0) ? 0 : 1;

        if (uucopy(sd, dirp, count) != 0)
                return (-errno);

        return (rc);
}

/*
 * Read in dirent64 structures from p1 (fd) into p2 (buffer).
 * p3 (count) is the size of the memory area.
 */
int
lx_getdents64(uintptr_t p1, uintptr_t p2, uintptr_t p3)
{
        int fd = (uint_t)p1;
        void *buf = (void *)p2;
        void *sbuf, *lbuf;
        int lbufsz = (uint_t)p3;
        int sbufsz;
        int namelen;
        struct dirent *sd;
        struct lx_dirent64 *ld;
        int bytes, rc;

        if (lbufsz < sizeof (struct lx_dirent64))
                return (-EINVAL);

        /*
         * The Linux dirent64 is bigger than the Solaris dirent64.  To
         * avoid inadvertently consuming more of the directory than we can
         * pass back to the Linux app, we hand the kernel a smaller buffer
         * than the app handed us.
         */
        sbufsz = (lbufsz / 32) * 24;

        sbuf = SAFE_ALLOCA(sbufsz);
        lbuf = SAFE_ALLOCA(lbufsz);
        if (sbuf == NULL || lbuf == NULL)
                return (-ENOMEM);
```

```
128         if ((bytes = getdents(fd, sbuf, sbufsz)) < 0)
129                 return (-errno);

131         /* munge the Solaris buffer to a linux buffer. */
132         sd = (struct dirent *)sbuf;
133         ld = (struct lx_dirent64 *)lbuf;
134         rc = 0;
135         while (bytes > 0) {
136                 namelen = strlen(sd->d_name);
137                 if (namelen >= LX_NAMEMAX)
138                         namelen = LX_NAMEMAX - 1;
139                 ld->d_ino = (uint64_t)sd->d_ino;
140                 ld->d_off = (int64_t)sd->d_off;
141                 ld->d_type = 0;

143                 (void) strncpy(ld->d_name, sd->d_name, namelen);
144                 ld->d_name[namelen] = 0;
145                 ld->d_reclen = (ushort_t)LX_RECLEN(namelen);

147                 bytes -= (int)sd->d_reclen;
148                 rc += (int)ld->d_reclen;

150                 sd = (struct dirent *)(void *)((caddr_t)sd + sd->d_reclen);
151                 ld = (struct lx_dirent64 *)(void *)((caddr_t)ld + ld->d_reclen);
152         }

154         /* now copy the lbuf to the userland buffer */
155         assert(rc <= lbufsz);
156         if (uucopy(lbuf, buf, rc) != 0)
157                 return (-EFAULT);

159         return (rc);
160 }
161 #endif /* ! codereview */
```

```
**********************************************************
    8042 Tue Jan 14 16:17:00 2014
new/usr/src/lib/brand/lx/lx_brand/common/fcntl.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */


#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/types.h>
#include <sys/filio.h>
#include <unistd.h>
#include <fcntl.h>
#include <stropts.h>
#include <libintl.h>
#include <errno.h>
#include <string.h>

#include <sys/lx_fcntl.h>
#include <sys/lx_debug.h>
#include <sys/lx_misc.h>

static int lx_fcntl_com(int fd, int cmd, ulong_t arg);
static void ltos_flock(struct lx_flock *l, struct flock *s);
static void stol_flock(struct flock *s, struct lx_flock *l);
static void ltos_flock64(struct lx_flock64 *l, struct flock64 *s);
static void stol_flock64(struct flock64 *s, struct lx_flock64 *l);
static short ltos_type(short l_type);
static short stol_type(short l_type);
static int lx_fcntl_getfl(int fd);
static int lx_fcntl_setfl(int fd, ulong_t arg);

int
lx_dup2(uintptr_t p1, uintptr_t p2)
{
        int oldfd = (int)p1;
        int newfd = (int)p2;
        int rc;

        rc = fcntl(oldfd, F_DUP2FD, newfd);
        return ((rc == -1) ? -errno : rc);
```



```
}

int
lx_fcntl(uintptr_t p1, uintptr_t p2, uintptr_t p3)
{
        int             fd = (int)p1;
        int             cmd = (int)p2;
        ulong_t         arg = (ulong_t)p3;
        struct lx_flock lxflk;
        struct flock    fl;
        int             lk = 0;
        int             rc;

        /*
         * The 64-bit fcntl commands must go through fcntl64().
         */
        if (cmd == LX_F_GETLK64 || cmd == LX_F_SETLK64 ||
            cmd == LX_F_SETLKW64)
                return (-EINVAL);

        if (cmd == LX_F_SETSIG || cmd == LX_F_GETSIG || cmd == LX_F_SETLEASE ||
            cmd == LX_F_GETLEASE) {
                lx_unsupported(gettext("%s(): unsupported command: %d"),
                    "fcntl", cmd);
                return (-ENOTSUP);
        }

        if (cmd == LX_F_GETLK || cmd == LX_F_SETLK ||
            cmd == LX_F_SETLKW) {
                if (uucopy((void *)p3, (void *)&lxflk,
                    sizeof (struct lx_flock)) != 0)
                        return (-errno);
                lk = 1;
                ltos_flock(&lxflk, &fl);
                arg = (ulong_t)&fl;
        }

        rc = lx_fcntl_com(fd, cmd, arg);

        if (lk)
                stol_flock(&fl, (struct lx_flock *)p3);

        return (rc);
}

int
lx_fcntl64(uintptr_t p1, uintptr_t p2, uintptr_t p3)
{
        int             fd = (int)p1;
        int             cmd = (int)p2;
        struct lx_flock lxflk;
        struct lx_flock64 lxflk64;
        struct flock    fl;
        struct flock64  fl64;
        int             rc;

        if (cmd == LX_F_SETSIG || cmd == LX_F_GETSIG || cmd == LX_F_SETLEASE ||
            cmd == LX_F_GETLEASE) {
                lx_unsupported(gettext("%s(): unsupported command: %d"),
                    "fcntl64", cmd);
                return (-ENOTSUP);
        }

        if (cmd == LX_F_GETLK || cmd == LX_F_SETLK || cmd == LX_F_SETLKW) {
                if (uucopy((void *)p3, (void *)&lxflk,
                    sizeof (struct lx_flock)) != 0)
```

```
128                         return (-errno);
129                 ltos_flock(&lxflk, &fl);
130                 rc = lx_fcntl_com(fd, cmd, (ulong_t)&fl);
131                 stol_flock(&fl, (struct lx_flock *)p3);
132         } else if (cmd == LX_F_GETLK64 || cmd == LX_F_SETLKW64 || \
133             cmd == LX_F_SETLK64) {
134                 if (uucopy((void *)p3, (void *)&lxflk64,
135                     sizeof (struct lx_flock64)) != 0)
136                         return (-errno);
137                 ltos_flock64(&lxflk64, &fl64);
138                 rc = lx_fcntl_com(fd, cmd, (ulong_t)&fl64);
139                 stol_flock64(&fl64, (struct lx_flock64 *)p3);
140         } else {
141                 rc = lx_fcntl_com(fd, cmd, (ulong_t)p3);
142         }

144         return (rc);
145 }

147 static int
148 lx_fcntl_com(int fd, int cmd, ulong_t arg)
149 {
150         int             rc = 0;

152         switch (cmd) {
153         case LX_F_DUPFD:
154                 rc = fcntl(fd, F_DUPFD, arg);
155                 break;

157         case LX_F_GETFD:
158                 rc = fcntl(fd, F_GETFD, 0);
159                 break;

161         case LX_F_SETFD:
162                 rc = fcntl(fd, F_SETFD, arg);
163                 break;

165         case LX_F_GETFL:
166                 rc = lx_fcntl_getfl(fd);
167                 break;

169         case LX_F_SETFL:
170                 rc = lx_fcntl_setfl(fd, arg);
171                 break;

173         case LX_F_GETLK:
174                 rc = fcntl(fd, F_GETLK, arg);
175                 break;

177         case LX_F_SETLK:
178                 rc = fcntl(fd, F_SETLK, arg);
179                 break;

181         case LX_F_SETLKW:
182                 rc = fcntl(fd, F_SETLKW, arg);
183                 break;

185         case LX_F_GETLK64:
186                 rc = fcntl(fd, F_GETLK64, arg);
187                 break;

189         case LX_F_SETLK64:
190                 rc = fcntl(fd, F_SETLK64, arg);
191                 break;

193         case LX_F_SETLKW64:
```



```
194                 rc = fcntl(fd, F_SETLKW64, arg);
195                 break;

197         case LX_F_SETOWN:
198                 rc = fcntl(fd, F_SETOWN, arg);
199                 break;

201         case LX_F_GETOWN:
202                 rc = fcntl(fd, F_GETOWN, arg);
203                 break;

205         default:
206                 return (-EINVAL);
207         }

209         return ((rc == -1) ? -errno : rc);
210 }


213 #define LTOS_FLOCK(l, s)                                                \
214 {                                                                       \
215         s->l_type = ltos_type(l->l_type);                             \
216         s->l_whence = l->l_whence;                                    \
217         s->l_start = l->l_start;                                      \
218         s->l_len = l->l_len;                                          \
219         s->l_sysid = 0;                 /* not defined in linux */    \
220         s->l_pid = (pid_t)l->l_pid;                                   \
221 }

223 #define STOL_FLOCK(s, l)                                                \
224 {                                                                       \
225         l->l_type = stol_type(s->l_type);                             \
226         l->l_whence = s->l_whence;                                    \
227         l->l_start = s->l_start;                                      \
228         l->l_len = s->l_len;                                          \
229         l->l_pid = (int)s->l_pid;                                     \
230 }

232 static void
233 ltos_flock(struct lx_flock *l, struct flock *s)
234 {
235         LTOS_FLOCK(l, s)
236 }

238 static void
239 stol_flock(struct flock *s, struct lx_flock *l)
240 {
241         STOL_FLOCK(s, l)
242 }

244 static void
245 ltos_flock64(struct lx_flock64 *l, struct flock64 *s)
246 {
247         LTOS_FLOCK(l, s)
248 }

250 static void
251 stol_flock64(struct flock64 *s, struct lx_flock64 *l)
252 {
253         STOL_FLOCK(s, l)
254 }

256 static short
257 ltos_type(short l_type)
258 {
259         switch (l_type) {
```

```
260         case LX_F_RDLCK:
261                 return (F_RDLCK);
262         case LX_F_WRLCK:
263                 return (F_WRLCK);
264         case LX_F_UNLCK:
265                 return (F_UNLCK);
266         default:
267                 return (-1);
268         }
269 }

271 static short
272 stol_type(short l_type)
273 {
274         switch (l_type) {
275         case F_RDLCK:
276                 return (LX_F_RDLCK);
277         case F_WRLCK:
278                 return (LX_F_WRLCK);
279         case F_UNLCK:
280                 return (LX_F_UNLCK);
281         default:
282                 /* can't ever happen */
283                 return (0);
284         }
285 }

287 int
288 lx_fcntl_getfl(int fd)
289 {
290         int retval;
291         int rc;

293         retval = fcntl(fd, F_GETFL, 0);

295         if ((retval & O_ACCMODE) == O_RDONLY)
296                 rc = LX_O_RDONLY;
297         else if ((retval & O_ACCMODE) == O_WRONLY)
298                 rc = LX_O_WRONLY;
299         else
300                 rc = LX_O_RDWR;
301         /* O_NDELAY != O_NONBLOCK, so we need to check for both */
302         if (retval & O_NDELAY)
303                 rc |= LX_O_NDELAY;
304         if (retval & O_NONBLOCK)
305                 rc |= LX_O_NONBLOCK;
306         if (retval & O_APPEND)
307                 rc |= LX_O_APPEND;
308         if (retval & O_SYNC)
309                 rc |= LX_O_SYNC;
310         if (retval & O_LARGEFILE)
311                 rc |= LX_O_LARGEFILE;
312         if (retval & FASYNC)
313                 rc |= LX_O_ASYNC;

315         return (rc);
316 }

318 int
319 lx_fcntl_setfl(int fd, ulong_t arg)
320 {
321         int new_arg;

323         new_arg = 0;
324         /* LX_O_NDELAY == LX_O_NONBLOCK, so we only check for one */
325         if (arg & LX_O_NDELAY)
```



```
326                 new_arg |= O_NONBLOCK;
327         if (arg & LX_O_APPEND)
328                 new_arg |= O_APPEND;
329         if (arg & LX_O_SYNC)
330                 new_arg |= O_SYNC;
331         if (arg & LX_O_LARGEFILE)
332                 new_arg |= O_LARGEFILE;
333         if (arg & LX_O_ASYNC)
334                 new_arg |= FASYNC;

336         return ((fcntl(fd, F_SETFL, new_arg) == 0) ? 0 : -errno);
337 }

339 /*
340  * flock() applies or removes an advisory lock on the file
341  * associated with the file descriptor fd.
342  *
343  * Stolen verbatim from usr/src/ucblib/libucb/port/sys/flock.c
344  *
345  * operation is: LX_LOCK_SH, LX_LOCK_EX, LX_LOCK_UN, LX_LOCK_NB
346  */
347 int
348 lx_flock(uintptr_t p1, uintptr_t p2)
349 {
350         int                     fd = (int)p1;
351         int                     operation = (int)p2;
352         struct flock            fl;
353         int                     cmd;
354         int                     ret;

356         /* In non-blocking lock, use F_SETLK for cmd, F_SETLKW otherwise */
357         if (operation & LX_LOCK_NB) {
358                 cmd = F_SETLK;
359                 operation &= ~LX_LOCK_NB; /* turn off this bit */
360         } else
361                 cmd = F_SETLKW;

363         switch (operation) {
364                 case LX_LOCK_UN:
365                         fl.l_type = F_UNLCK;
366                         break;
367                 case LX_LOCK_SH:
368                         fl.l_type = F_RDLCK;
369                         break;
370                 case LX_LOCK_EX:
371                         fl.l_type = F_WRLCK;
372                         break;
373                 default:
374                         return (-EINVAL);
375         }

377         fl.l_whence = 0;
378         fl.l_start = 0;
379         fl.l_len = 0;

381         ret = fcntl(fd, cmd, &fl);

383         if (ret == -1 && errno == EACCES)
384                 return (-EWOULDBLOCK);

386         return ((ret == -1) ? -errno : ret);
387 }
388 #endif /* ! codereview */
```

```
**********************************************************
   16742 Tue Jan 14 16:17:00 2014
new/usr/src/lib/brand/lx/lx_brand/common/file.c
LX zone support should now build and packages of relevance produced.
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/fstyp.h>
#include <sys/fsid.h>

#include <errno.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/vnode.h>
#include <fcntl.h>
#include <string.h>
#include <utime.h>
#include <atomic.h>

#include <sys/lx_syscall.h>
#include <sys/lx_types.h>
#include <sys/lx_debug.h>
#include <sys/lx_misc.h>
#include <sys/lx_fcntl.h>

static int
install_checkpath(uintptr_t p1)
{
        int saved_errno = errno;
        char path[MAXPATHLEN];

        /*
         * The "dev" RPM package wants to modify /dev/pts, but /dev/pts is a
         * lofs mounted copy of /native/dev/pts, so that won't work.
         *
         * Instead, if we're trying to modify /dev/pts from install mode, just
         * act as if it succeded.
         */
        if (uucopystr((void *)p1, path, MAXPATHLEN) == -1)
```



```
                return (-errno);

        if (strcmp(path, "/dev/pts") == 0)
                return (0);

        errno = saved_errno;
        return (-errno);
}

/*
 * Convert linux LX_AT_* flags to solaris AT_* flags, while verifying allowed
 * flags have been passed. This also allows EACCESS/REMOVEDIR to be translated
 * correctly since on linux they have the same value.
 */
int
ltos_at_flag(int lflag, int allow)
{
        int sflag = 0;

        if ((lflag & LX_AT_EACCESS) && (allow & AT_EACCESS)) {
                lflag &= ~LX_AT_EACCESS;
                sflag |= AT_EACCESS;
        }

        if ((lflag & LX_AT_REMOVEDIR) && (allow & AT_REMOVEDIR)) {
                lflag &= ~LX_AT_REMOVEDIR;
                sflag |= AT_REMOVEDIR;
        }

        if ((lflag & LX_AT_SYMLINK_NOFOLLOW) && (allow & AT_SYMLINK_NOFOLLOW)) {
                lflag &= ~LX_AT_SYMLINK_NOFOLLOW;
                sflag |= AT_SYMLINK_NOFOLLOW;
        }

        /* right now solaris doesn't have a _FOLLOW flag, so use a fake one */
        if ((lflag & LX_AT_SYMLINK_FOLLOW) && (allow & LX_AT_SYMLINK_FOLLOW)) {
                lflag &= ~LX_AT_SYMLINK_FOLLOW;
                sflag |= LX_AT_SYMLINK_FOLLOW;
        }

        /* if flag is not zero than some flags did not hit the above code */
        if (lflag)
                return (-EINVAL);

        return (sflag);
}


/*
 * Miscellaneous file-related system calls.
 */

/*
 * Linux creates half-duplex unnamed pipes and Solaris creates full-duplex
 * pipes.  Thus, to get the correct semantics, our simple pipe() system
 * call actually needs to create a named pipe, do three opens, a close, and
 * an unlink.  This is woefully expensive.  If performance becomes a real
 * issue, we can implement a half-duplex pipe() in the brand module.
 */
#define PIPENAMESZ      32 /* enough room for /tmp/.pipe.<pid>.<num> */

int
lx_pipe(uintptr_t p1)
{
        static uint32_t pipecnt = 0;
        int cnt;
```

```
127         char pipename[PIPENAMESZ];
128         int fds[3];
129         int r = 0;

131         fds[0] = -1;
132         fds[1] = -1;
133         fds[2] = -1;

135         /*
136          * Construct a name for the named pipe: /tmp/.pipe.<pid>.<++cnt>
137          */
138         cnt = atomic_inc_32_nv(&pipecnt);

140         (void) snprintf(pipename, PIPENAMESZ, "/tmp/.pipe.%d.%d",
141             getpid(), cnt);

143         if (mkfifo(pipename, 0600))
144                 return (-errno);

146         /*
147          * To prevent either the read-only or write-only open from
148          * blocking, we first need to open the pipe for both reading and
149          * writing.
150          */
151         if (((fds[2] = open(pipename, O_RDWR)) < 0) ||
152             ((fds[0] = open(pipename, O_RDONLY)) < 0) ||
153             ((fds[1] = open(pipename, O_WRONLY)) < 0)) {
154                 r = errno;
155         } else {
156                 /*
157                  * Copy the two one-way fds back to the app's address
158                  * space.
159                  */
160                 if (uucopy(fds, (void *)p1, 2 * sizeof (int)))
161                         r = errno;
162         }

164         if (fds[2] >= 0)
165                 (void) close(fds[2]);
166         (void) unlink(pipename);

168         if (r != 0) {
169                 if (fds[0] >= 0)
170                         (void) close(fds[0]);
171                 if (fds[1] >= 0)
172                         (void) close(fds[1]);
173         }

175         return (-r);
176 }

178 /*
179  * On Linux, even root cannot create a link to a directory, so we have to
180  * add an explicit check.
181  */
182 int
183 lx_link(uintptr_t p1, uintptr_t p2)
184 {
185         char *from = (char *)p1;
186         char *to = (char *)p2;
187         struct stat64 statbuf;

189         if ((stat64(from, &statbuf) == 0) && S_ISDIR(statbuf.st_mode))
190                 return (-EPERM);

192         return (link(from, to) ? -errno : 0);
```



```
193 }

195 /*
196  * On Linux, an unlink of a directory returns EISDIR, not EPERM.
197  */
198 int
199 lx_unlink(uintptr_t p)
200 {
201         char *pathname = (char *)p;
202         struct stat64 statbuf;

204         if ((lstat64(pathname, &statbuf) == 0) && S_ISDIR(statbuf.st_mode))
205                 return (-EISDIR);

207         return (unlink(pathname) ? -errno : 0);
208 }

210 int
211 lx_unlinkat(uintptr_t ext1, uintptr_t p1, uintptr_t p2)
212 {
213         int atfd = (int)ext1;
214         char *pathname = (char *)p1;
215         int flag = (int)p2;
216         struct stat64 statbuf;

218         if (atfd == LX_AT_FDCWD)
219                 atfd = AT_FDCWD;

221         flag = ltos_at_flag(flag, AT_REMOVEDIR);
222         if (flag < 0)
223                 return (-EINVAL);

225         if (!(flag & AT_REMOVEDIR)) {
226                 /* Behave like unlink() */
227                 if ((fstatat64(atfd, pathname, &statbuf, AT_SYMLINK_NOFOLLOW) ==
228                     0) && S_ISDIR(statbuf.st_mode))
229                         return (-EISDIR);
230         }

232         return (unlinkat(atfd, pathname, flag) ? -errno : 0);
233 }

235 /*
236  * fsync() and fdatasync() - On Solaris, these calls translate into a common
237  * fsync() syscall with a different parameter, so we layer on top of the librt
238  * functions instead.
239  */
240 int
241 lx_fsync(uintptr_t fd)
242 {
243         int fildes = (int)fd;
244         struct stat64 statbuf;

246         if ((fstat64(fildes, &statbuf) == 0) &&
247             (S_ISCHR(statbuf.st_mode) || S_ISFIFO(statbuf.st_mode)))
248                 return (-EINVAL);

250         return (fsync((int)fd) ? -errno : 0);
251 }

253 int
254 lx_fdatasync(uintptr_t fd)
255 {
256         int fildes = (int)fd;
257         struct stat64 statbuf;
```

```
259         if ((fstat64(fildes, &statbuf) == 0) &&
260             (S_ISCHR(statbuf.st_mode) || S_ISFIFO(statbuf.st_mode)))
261                 return (-EINVAL);

263         return (fdatasync((int)fd) ? -errno : 0);
264 }

266 /*
267  * Linux, unlike Solaris, ALWAYS resets the setuid and setgid bits on a
268  * chown/fchown  regardless of whether it was done by root or not.  Therefore,
269  * we must do extra work after each chown/fchown call to emulate this behavior.
270  */
271 #define SETUGID (S_ISUID | S_ISGID)

273 /*
274  * [lf]chown16() - Translate the uid/gid and pass onto the real functions.
275  */
276 int
277 lx_chown16(uintptr_t p1, uintptr_t p2, uintptr_t p3)
278 {
279         char *filename = (char *)p1;
280         struct stat64 statbuf;

282         if (chown(filename, LX_UID16_TO_UID32((lx_gid16_t)p2),
283             LX_GID16_TO_GID32((lx_gid16_t)p3)))
284                 return (-errno);

286         if (stat64(filename, &statbuf) == 0) {
287                 statbuf.st_mode &= ~S_ISUID;
288                 if (statbuf.st_mode & S_IXGRP)
289                         statbuf.st_mode &= ~S_ISGID;
290                 (void) chmod(filename, (statbuf.st_mode & MODEMASK));
291         }

293         return (0);
294 }

296 int
297 lx_fchown16(uintptr_t p1, uintptr_t p2, uintptr_t p3)
298 {
299         int fd = (int)p1;
300         struct stat64 statbuf;

302         if (fchown(fd, LX_UID16_TO_UID32((lx_gid16_t)p2),
303             LX_GID16_TO_GID32((lx_gid16_t)p3)))
304                 return (-errno);

306         if (fstat64(fd, &statbuf) == 0) {
307                 statbuf.st_mode &= ~S_ISUID;
308                 if (statbuf.st_mode & S_IXGRP)
309                         statbuf.st_mode &= ~S_ISGID;
310                 (void) fchmod(fd, (statbuf.st_mode & MODEMASK));
311         }

313         return (0);
314 }

316 int
317 lx_lchown16(uintptr_t p1, uintptr_t p2, uintptr_t p3)
318 {
319         return (lchown((char *)p1, LX_UID16_TO_UID32((lx_gid16_t)p2),
320             LX_GID16_TO_GID32((lx_gid16_t)p3)) ? -errno : 0);
321 }

323 int
324 lx_chown(uintptr_t p1, uintptr_t p2, uintptr_t p3)
```



```
325 {
326         char *filename = (char *)p1;
327         struct stat64 statbuf;
328         int ret;

330         ret = chown(filename, (uid_t)p2, (gid_t)p3);

332         if (ret < 0) {
333                 /*
334                  * If chown() failed and we're in install mode, return success
335                  * if the the reason we failed was because the source file
336                  * didn't actually exist or if we're trying to modify /dev/pts.
337                  */
338                 if ((lx_install != 0) &&
339                     ((errno == ENOENT) || (install_checkpath(p1) == 0)))
340                         return (0);

342                 return (-errno);
343         }

345         if (stat64(filename, &statbuf) == 0) {
346                 statbuf.st_mode &= ~S_ISUID;
347                 if (statbuf.st_mode & S_IXGRP)
348                         statbuf.st_mode &= ~S_ISGID;
349                 (void) chmod(filename, (statbuf.st_mode & MODEMASK));
350         }

352         return (0);
353 }

355 int
356 lx_fchown(uintptr_t p1, uintptr_t p2, uintptr_t p3)
357 {
358         int fd = (int)p1;
359         struct stat64 statbuf;

361         if (fchown(fd, (uid_t)p2, (gid_t)p3))
362                 return (-errno);

364         if (fstat64(fd, &statbuf) == 0) {
365                 statbuf.st_mode &= ~S_ISUID;
366                 if (statbuf.st_mode & S_IXGRP)
367                         statbuf.st_mode &= ~S_ISGID;
368                 (void) fchmod(fd, (statbuf.st_mode & MODEMASK));
369         }

371         return (0);
372 }

374 int
375 lx_chmod(uintptr_t p1, uintptr_t p2)
376 {
377         int ret;

379         ret = chmod((const char *)p1, (mode_t)p2);

381         if (ret < 0) {
382                 /*
383                  * If chown() failed and we're in install mode, return success
384                  * if the the reason we failed was because the source file
385                  * didn't actually exist or if we're trying to modify /dev/pts.
386                  */
387                 if ((lx_install != 0) &&
388                     ((errno == ENOENT) || (install_checkpath(p1) == 0)))
389                         return (0);
```

```
391                 return (-errno);
392         }
393
394         return (0);
395 }
396
397 int
398 lx_utime(uintptr_t p1, uintptr_t p2)
399 {
400         int ret;
401
402         ret = utime((const char *)p1, (const struct utimbuf *)p2);
403
404         if (ret < 0) {
405                 /*
406                  * If chown() failed and we're in install mode, return success
407                  * if the the reason we failed was because the source file
408                  * didn't actually exist or if we're trying to modify /dev/pts.
409                  */
410                 if ((lx_install != 0) &&
411                     ((errno == ENOENT) || (install_checkpath(p1) == 0)))
412                         return (0);
413
414                 return (-errno);
415         }
416
417         return (0);
418 }
419
420 /*
421  * llseek() - The Linux implementation takes an additional parameter, which is
422  * the resulting position in the file.
423  */
424 int
425 lx_llseek(uintptr_t p1, uintptr_t p2, uintptr_t p3, uintptr_t p4,
426     uintptr_t p5)
427 {
428         offset_t ret;
429         offset_t *res = (offset_t *)p4;
430
431         /* SEEK_DATA and SEEK_HOLE are only valid in Solaris */
432         if ((int)p5 > SEEK_END)
433                 return (-EINVAL);
434
435         if ((ret = llseek((int)p1, LX_32TO64(p3, p2), p5)) < 0)
436                 return (-errno);
437
438         *res = ret;
439         return (0);
440 }
441
442 /*
443  * seek() - When the resultant file offset cannot be represented in 32 bits,
444  * Linux performs the seek but Solaris doesn't, though both set EOVERFLOW.  We
445  * call llseek() and then check to see if we need to return EOVERFLOW.
446  */
447 int
448 lx_lseek(uintptr_t p1, uintptr_t p2, uintptr_t p3)
449 {
450         offset_t offset = (offset_t)(off_t)(p2);        /* sign extend */
451         offset_t ret;
452         off_t ret32;
453
454         /* SEEK_DATA and SEEK_HOLE are only valid in Solaris */
455         if ((int)p3 > SEEK_END)
456                 return (-EINVAL);
```



```
457
458         if ((ret = llseek((int)p1, offset, p3)) < 0)
459                 return (-errno);
460
461         ret32 = (off_t)ret;
462         if ((offset_t)ret32 == ret)
463                 return (ret32);
464         else
465                 return (-EOVERFLOW);
466 }
467
468 /*
469  * Neither Solaris nor Linux actually returns anything to the caller, but glibc
470  * expects to see SOME value returned, so placate it and return 0.
471  */
472 int
473 lx_sync(void)
474 {
475         sync();
476         return (0);
477 }
478
479 int
480 lx_rmdir(uintptr_t p1)
481 {
482         int r;
483
484         r = rmdir((char *)p1);
485         if (r < 0)
486                 return ((errno == EEXIST) ? -ENOTEMPTY : -errno);
487         return (0);
488 }
489
490 /*
491  * Exactly the same as Solaris' sysfs(2), except Linux numbers their fs indices
492  * starting at 0, and Solaris starts at 1.
493  */
494 int
495 lx_sysfs(uintptr_t p1, uintptr_t p2, uintptr_t p3)
496 {
497         int option = (int)p1;
498         int res;
499
500         /*
501          * Linux actually doesn't have #defines for these; their sysfs(2)
502          * man page literally defines the "option" field as being 1, 2 or 3,
503          * corresponding to Solaris' GETFSIND, GETFSTYP and GETNFSTYP,
504          * respectively.
505          */
506         switch (option) {
507                 case 1:
508                         if ((res = sysfs(GETFSIND, (const char *)p2)) < 0)
509                                 return (-errno);
510
511                         return (res - 1);
512
513                 case 2:
514                         if ((res = sysfs(GETFSTYP, (int)p2 + 1,
515                             (char *)p3)) < 0)
516                                 return (-errno);
517
518                         return (0);
519
520                 case 3:
521                         if ((res = sysfs(GETNFSTYP)) < 0)
522                                 return (-errno);
```

```
524                         return (res);

526                 default:
527                         break;
528         }

530         return (-EINVAL);
531 }

533 int
534 lx_faccessat(uintptr_t p1, uintptr_t p2, uintptr_t p3, uintptr_t p4)
535 {
536         int atfd = (int)p1;
537         char *path = (char *)p2;
538         int mode = (mode_t)p3;
539         int flag = (int)p4;

541         if (atfd == LX_AT_FDCWD)
542                 atfd = AT_FDCWD;

544         flag = ltos_at_flag(flag, AT_EACCESS);
545         if (flag < 0)
546                 return (-EINVAL);

548         return (faccessat(atfd, path, mode, flag) ? -errno : 0);
549 }

551 int
552 lx_futimesat(uintptr_t p1, uintptr_t p2, uintptr_t p3)
553 {
554         int atfd = (int)p1;
555         char *path = (char *)p2;
556         struct timeval *times = (struct timeval *)p3;

558         if (atfd == LX_AT_FDCWD)
559                 atfd = AT_FDCWD;

561         return (futimesat(atfd, path, times) ? -errno : 0);
562 }


565 /*
566  * Constructs an absolute path string in buf from the path of fd and the
567  * relative path string pointed to by "p1". This is required for emulating
568  * *at() system calls.
569  * Example:
570  *    If the path of fd is "/foo/bar" and path is "etc" the string returned is
571  *    "/foo/bar/etc", if the fd is a file fd then it fails with ENOTDIR.
572  *    If path is absolute then no modifcations are made to it when copied.
573  */
574 static int
575 getpathat(int fd, uintptr_t p1, char *outbuf, size_t outbuf_size)
576 {
577         char pathbuf[MAXPATHLEN];
578         char fdpathbuf[MAXPATHLEN];
579         char *fdpath;
580         struct stat64 statbuf;

582         if (uucopystr((void *)p1, pathbuf, MAXPATHLEN) == -1)
583                 return (-errno);

585         /* If the path is absolute then we can early out */
586         if ((pathbuf[0] == '/') || (fd == LX_AT_FDCWD)) {
587                 (void) strlcpy(outbuf, pathbuf, outbuf_size);
588                 return (0);
```



```
589         }

591         fdpath = lx_fd_to_path(fd, fdpathbuf, sizeof (fdpathbuf));
592         if (fdpath == NULL)
593                 return (-EBADF);

595         if ((fstat64(fd, &statbuf) < 0))
596                 return (-EBADF);

598         if (!S_ISDIR(statbuf.st_mode))
599                 return (-ENOTDIR);

601         if (snprintf(outbuf, outbuf_size, "%s/%s", fdpath, pathbuf) >
602             (outbuf_size-1))
603                 return (-ENAMETOOLONG);

605         return (0);
606 }

608 int
609 lx_mkdirat(uintptr_t p1, uintptr_t p2, uintptr_t p3)
610 {
611         int atfd = (int)p1;
612         mode_t mode = (mode_t)p3;
613         char pathbuf[MAXPATHLEN];
614         int ret;

616         ret = getpathat(atfd, p2, pathbuf, sizeof (pathbuf));
617         if (ret < 0)
618                 return (ret);

620         return (mkdir(pathbuf, mode) ? -errno : 0);
621 }

623 int
624 lx_mknodat(uintptr_t ext1, uintptr_t p1, uintptr_t p2, uintptr_t p3)
625 {
626         int atfd = (int)ext1;
627         char pathbuf[MAXPATHLEN];
628         int ret;

630         ret = getpathat(atfd, p1, pathbuf, sizeof (pathbuf));
631         if (ret < 0)
632                 return (ret);

634         return (lx_mknod((uintptr_t)pathbuf, p2, p3));
635 }

637 int
638 lx_symlinkat(uintptr_t p1, uintptr_t ext1, uintptr_t p2)
639 {
640         int atfd = (int)ext1;
641         char pathbuf[MAXPATHLEN];
642         int ret;

644         ret = getpathat(atfd, p2, pathbuf, sizeof (pathbuf));
645         if (ret < 0)
646                 return (ret);

648         return (symlink((char *)p1, pathbuf) ? -errno : 0);
649 }

651 int
652 lx_linkat(uintptr_t ext1, uintptr_t p1, uintptr_t ext2, uintptr_t p2,
653     uintptr_t p3)
654 {
```

```
655         int atfd1 = (int)ext1;
656         int atfd2 = (int)ext2;
657         char pathbuf1[MAXPATHLEN];
658         char pathbuf2[MAXPATHLEN];
659         int ret;

661         /*
662          * The flag specifies whether the hardlink will point to a symlink or
663          * not, on solaris the default behaviour of link() is to dereference a
664          * symlink and there is no obvious way to trigger the other behaviour.
665          * So for now we just ignore this flag and act like link().
666          */
667         /* LINTED [set but not used in function] */
668         int flag = p3;

670         if (flag != p3)
671                 return (flag); // workaround.

673         ret = getpathat(atfd1, p1, pathbuf1, sizeof (pathbuf1));
674         if (ret < 0)
675                 return (ret);

677         ret = getpathat(atfd2, p2, pathbuf2, sizeof (pathbuf2));
678         if (ret < 0)
679                 return (ret);

681         return (lx_link((uintptr_t)pathbuf1, (uintptr_t)pathbuf2));
682 }

684 int
685 lx_readlinkat(uintptr_t ext1, uintptr_t p1, uintptr_t p2, uintptr_t p3)
686 {
687         int atfd = (int)ext1;
688         char pathbuf[MAXPATHLEN];
689         int ret;

691         ret = getpathat(atfd, p1, pathbuf, sizeof (pathbuf));
692         if (ret < 0)
693                 return (ret);

695         ret = readlink(pathbuf, (char *)p2, (size_t)p3);
696         if (ret < 0)
697                 return (-errno);

699         return (ret);
700 }

702 int
703 lx_fchownat(uintptr_t ext1, uintptr_t p1, uintptr_t p2, uintptr_t p3,
704     uintptr_t p4)
705 {
706         int flag;
707         int atfd = (int)ext1;
708         char pathbuf[MAXPATHLEN];
709         int ret;

711         flag = ltos_at_flag(p4, AT_SYMLINK_NOFOLLOW);
712         if (flag < 0)
713                 return (-EINVAL);

715         ret = getpathat(atfd, p1, pathbuf, sizeof (pathbuf));
716         if (ret < 0)
717                 return (ret);

719         if (flag & AT_SYMLINK_NOFOLLOW)
720                 return (lchown(pathbuf, (uid_t)p2, (gid_t)p3) ? -errno : 0);
```



```
721         else
722                 return (lx_chown((uintptr_t)pathbuf, p2, p3));
723 }

725 int
726 lx_fchmodat(uintptr_t ext1, uintptr_t p1, uintptr_t p2, uintptr_t p3)
727 {
728         int atfd = (int)ext1;
729         char pathbuf[MAXPATHLEN];
730         int ret;

732         /*
733          * It seems that at least some versions of glibc do not set or clear
734          * the flags arg, so checking them will result in random behaviour.
735          */
736         /* LINTED [set but not used in function] */
737         int flag = p3;

739         if (flag != p3)
740                 return (flag); // workaround.

742         ret = getpathat(atfd, p1, pathbuf, sizeof (pathbuf));
743         if (ret < 0)
744                 return (ret);

746         return (lx_chmod((uintptr_t)pathbuf, p2));
747 }
748 #endif /* ! codereview */
```

```
**********************************************************
    1850 Tue Jan 14 16:17:01 2014
new/usr/src/lib/brand/lx/lx_brand/common/fork.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <errno.h>
#include <unistd.h>
#include <sys/lx_misc.h>

/*
 * fork() and vfork()
 *
 * These cannot be pass thru system calls because we need libc to do its own
 * initialization or else bad things will happen (i.e. ending up with a bad
 * schedctl page).  On Linux, there is no such thing as forkall(), so we use
 * fork1() here.
 */
int
lx_fork(void)
{
        int ret = fork1();

        if (ret == 0 && lx_is_rpm)
                (void) sleep(lx_rpm_delay);

        return (ret == -1 ? -errno : ret);
}

/*
 * For vfork(), we have a serious problem because the child is not allowed to
 * return from the current frame because it will corrupt the parent's stack.
 * Since the semantics of vfork() are rather ill-defined (other than "it's
 * faster than fork"), we should theoretically be safe by falling back to
 * fork1().
 */
int
lx_vfork(void)
{
```



```
        int ret = fork1();

        return (ret == -1 ? -errno : ret);
}
#endif /* ! codereview */
```

```
**********************************************************
    6526 Tue Jan 14 16:17:01 2014
new/usr/src/lib/brand/lx/lx_brand/common/id.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/types.h>
#include <sys/systm.h>
#include <sys/errno.h>
#include <sys/zone.h>
#include <sys/lx_types.h>
#include <sys/lx_syscall.h>
#include <sys/cred_impl.h>
#include <sys/policy.h>
#include <sys/ucred.h>
#include <sys/syscall.h>
#include <alloca.h>
#include <errno.h>
#include <ucred.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/lx_misc.h>

int
lx_setuid16(uintptr_t uid)
{
        return ((setuid(LX_UID16_TO_UID32((lx_uid16_t)uid))) ? -errno : 0);
}

int
lx_getuid16(void)
{
        return ((int)LX_UID32_TO_UID16(getuid()));
}

int
lx_setgid16(uintptr_t gid)
{
```



```
        return ((setgid(LX_GID16_TO_GID32((lx_gid16_t)gid))) ? -errno : 0);
}

int
lx_getgid16(void)
{
        return ((int)LX_GID32_TO_GID16(getgid()));
}

int
lx_geteuid16(void)
{
        return ((int)LX_UID32_TO_UID16(geteuid()));
}

int
lx_getegid16(void)
{
        return ((int)LX_GID32_TO_GID16(getegid()));
}

int
lx_geteuid(void)
{
        return ((int)geteuid());
}

int
lx_getegid(void)
{
        return ((int)getegid());
}

int
lx_getresuid(uintptr_t ruid, uintptr_t euid, uintptr_t suid)
{
        lx_uid_t lx_ruid, lx_euid, lx_suid;
        ucred_t *cr;
        size_t sz;

        /*
         * We allocate a ucred_t ourselves rather than call ucred_get(3C)
         * because ucred_get() calls malloc(3C), which the brand library cannot
         * use.  Because we allocate the space with SAFE_ALLOCA(), there's
         * no need to free it when we're done.
         */
        sz = ucred_size();
        cr = (ucred_t *)SAFE_ALLOCA(sz);
        if (cr == NULL)
                return (-ENOMEM);

        if (syscall(SYS_ucredsys, UCREDSYS_UCREDGET, P_MYID, cr) != 0)
                return (-errno);

        if (((lx_ruid = (lx_uid_t)ucred_getruid(cr)) == (lx_uid_t)-1) ||
            ((lx_euid = (lx_uid_t)ucred_geteuid(cr)) == (lx_uid_t)-1) ||
            ((lx_suid = (lx_uid_t)ucred_getsuid(cr)) == (lx_uid_t)-1)) {
                return (-errno);
        }

        if (uucopy(&lx_ruid, (void *)ruid, sizeof (lx_uid_t)) != 0)
                return (-errno);

        if (uucopy(&lx_euid, (void *)euid, sizeof (lx_uid_t)) != 0)
                return (-errno);
```

```
128         return ((uucopy(&lx_suid, (void *)suid, sizeof (lx_uid_t)) != 0)
129             ? -errno : 0);
130 }

132 int
133 lx_getresuid16(uintptr_t ruid16, uintptr_t euid16, uintptr_t suid16)
134 {
135         lx_uid_t lx_ruid, lx_euid, lx_suid;
136         lx_uid16_t lx_ruid16, lx_euid16, lx_suid16;
137         int rv;

139         if ((rv = lx_getresuid((uintptr_t)&lx_ruid, (uintptr_t)&lx_euid,
140             (uintptr_t)&lx_suid)) != 0)
141                 return (rv);

143         lx_ruid16 = LX_UID32_TO_UID16(lx_ruid);
144         lx_euid16 = LX_UID32_TO_UID16(lx_euid);
145         lx_suid16 = LX_UID32_TO_UID16(lx_suid);

147         if (uucopy(&lx_ruid16, (void *)ruid16, sizeof (lx_uid16_t)) != 0)
148                 return (-errno);

150         if (uucopy(&lx_euid16, (void *)euid16, sizeof (lx_uid16_t)) != 0)
151                 return (-errno);

153         return ((uucopy(&lx_suid16, (void *)suid16, sizeof (lx_uid16_t)) != 0)
154             ? -errno : 0);
155 }

157 int
158 lx_getresgid(uintptr_t rgid, uintptr_t egid, uintptr_t sgid)
159 {
160         ucred_t *cr;
161         lx_gid_t lx_rgid, lx_egid, lx_sgid;
162         size_t sz;

164         /*
165          * We allocate a ucred_t ourselves rather than call ucred_get(3C)
166          * because ucred_get() calls malloc(3C), which the brand library cannot
167          * use.  Because we allocate the space with SAFE_ALLOCA(), there's
168          * no need to free it when we're done.
169          */
170         sz = ucred_size();
171         cr = (ucred_t *)SAFE_ALLOCA(sz);
172         if (cr == NULL)
173                 return (-ENOMEM);

175         if (syscall(SYS_ucredsys, UCREDSYS_UCREDGET, P_MYID, cr) != 0)
176                 return (-errno);

178         if (((lx_rgid = (lx_gid_t)ucred_getrgid(cr)) == (lx_gid_t)-1) ||
179             ((lx_egid = (lx_gid_t)ucred_getegid(cr)) == (lx_gid_t)-1) ||
180             ((lx_sgid = (lx_gid_t)ucred_getsgid(cr)) == (lx_gid_t)-1)) {
181                 return (-errno);
182         }

184         if (uucopy(&lx_rgid, (void *)rgid, sizeof (lx_gid_t)) != 0)
185                 return (-errno);

187         if (uucopy(&lx_egid, (void *)egid, sizeof (lx_gid_t)) != 0)
188                 return (-errno);

190         return ((uucopy(&lx_sgid, (void *)sgid, sizeof (lx_gid_t)) != 0)
191             ? -errno : 0);
192 }
```



```
194 int
195 lx_getresgid16(uintptr_t rgid16, uintptr_t egid16, uintptr_t sgid16)
196 {
197         lx_gid_t lx_rgid, lx_egid, lx_sgid;
198         lx_gid16_t lx_rgid16, lx_egid16, lx_sgid16;
199         int rv;

201         if ((rv = lx_getresgid((uintptr_t)&lx_rgid, (uintptr_t)&lx_egid,
202             (uintptr_t)&lx_sgid)) != 0)
203                 return (rv);

205         lx_rgid16 = LX_UID32_TO_UID16(lx_rgid);
206         lx_egid16 = LX_UID32_TO_UID16(lx_egid);
207         lx_sgid16 = LX_UID32_TO_UID16(lx_sgid);

209         if (uucopy(&lx_rgid16, (void *)rgid16, sizeof (lx_gid16_t)) != 0)
210                 return (-errno);

212         if (uucopy(&lx_egid16, (void *)egid16, sizeof (lx_gid16_t)) != 0)
213                 return (-errno);

215         return ((uucopy(&lx_sgid16, (void *)sgid16, sizeof (lx_gid16_t)) != 0)
216             ? -errno : 0);
217 }

219 int
220 lx_setreuid16(uintptr_t ruid, uintptr_t euid)
221 {
222         return ((setreuid(LX_UID16_TO_UID32((lx_uid16_t)ruid),
223             LX_UID16_TO_UID32((lx_uid16_t)euid))) ? -errno : 0);
224 }

226 int
227 lx_setregid16(uintptr_t rgid, uintptr_t egid)
228 {
229         return ((setregid(LX_UID16_TO_UID32((lx_gid16_t)rgid),
230             LX_UID16_TO_UID32((lx_gid16_t)egid))) ? -errno : 0);
231 }

233 /*
234  * The lx brand cannot support the setfs[ug]id16/setfs[ug]id calls as that
235  * would require significant rework of Solaris' privilege mechanisms, so
236  * instead return the current effective [ug]id.
237  *
238  * In Linux, fsids track effective IDs, so returning the effective IDs works
239  * as a substitute; returning the current value also denotes failure of the
240  * call if the caller had specified something different.  We don't need to
241  * worry about setting error codes because the Linux calls don't set any.
242  */
243 /*ARGSUSED*/
244 int
245 lx_setfsuid16(uintptr_t fsuid16)
246 {
247         return (lx_geteuid16());
248 }

250 /*ARGSUSED*/
251 int
252 lx_setfsgid16(uintptr_t fsgid16)
253 {
254         return (lx_getegid16());
255 }

257 /*ARGSUSED*/
258 int
259 lx_setfsuid(uintptr_t fsuid)
```

```
260 {
261         return (geteuid());
262 }

264 /*ARGSUSED*/
265 int
266 lx_setfsgid(uintptr_t fsgid)
267 {
268         return (getegid());
269 }
270 #endif /* ! codereview */
```

```
**********************************************************
   73335 Tue Jan 14 16:17:01 2014
new/usr/src/lib/brand/lx/lx_brand/common/ioctl.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <assert.h>
#include <fcntl.h>
#include <sys/types.h>
#include <signal.h>
#include <sys/stat.h>
#include <unistd.h>
#include <limits.h>
#include <stdio.h>
#include <stdarg.h>
#include <stdlib.h>
#include <stropts.h>
#include <strings.h>
#include <thread.h>
#include <errno.h>
#include <libintl.h>
#include <sys/bitmap.h>
#include <sys/lx_autofs.h>
#include <sys/modctl.h>
#include <sys/filio.h>
#include <sys/termios.h>
#include <sys/termio.h>
#include <sys/sockio.h>
#include <net/if.h>
#include <net/if_arp.h>
#include <sys/ptms.h>
#include <sys/ldlinux.h>
#include <sys/lx_ptm.h>
#include <sys/lx_socket.h>
#include <sys/syscall.h>
#include <sys/brand.h>
#include <sys/lx_audio.h>
#include <sys/lx_ioctl.h>
#include <sys/lx_misc.h>
```



```
#include <sys/lx_debug.h>
#include <sys/ptyvar.h>
#include <sys/audio.h>
#include <sys/mixer.h>

/* Define _KERNEL to get the devt manipulation macros. */
#define _KERNEL
#include <sys/sysmacros.h>
#undef  _KERNEL

/* Maximum number of modules on a stream that we can handle. */
#define MAX_STRMODS     10

/* Maximum buffer size for debugging messages. */
#define MSGBUF          1024

/* Structure used to define an ioctl translator. */
typedef struct ioc_cmd_translator {
        int     ict_lx_cmd;
        char    *ict_lx_cmd_str;
        int     ict_cmd;
        char    *ict_cmd_str;
        int     (*ict_func)(int fd, struct stat *stat,
            int cmd, char *cmd_str, intptr_t arg);
} ioc_cmd_translator_t;

/*
 * Structures used to associate a group of ioctl translators with
 * a specific device.
 */
typedef struct ioc_dev_translator {
        char                    *idt_driver;
        major_t                 idt_major;

        /* Array of command translators. */
        ioc_cmd_translator_t    *idt_cmds;
} ioc_dev_translator_t;

/*
 * Structures used to associate a group of ioctl translators with
 * a specific filesystem.
 */
typedef struct ioc_fs_translator {
        char                    *ift_filesystem;

        /* Array of command translators. */
        ioc_cmd_translator_t    *ift_cmds;
} ioc_fs_translator_t;

/* Structure used to define a unsupported ioctl error codes. */
typedef struct ioc_errno_translator {
        int     iet_lx_cmd;
        char    *iet_lx_cmd_str;
        int     iet_errno;
} ioc_errno_translator_t;

/* Structure used to convert oss format flags into Solaris options. */
typedef struct oss_fmt_translator {
        int     oft_oss_fmt;
        int     oft_encoding;
        int     oft_precision;
} oss_fmt_translator_t;

/* Translator forward declerations. */
static oss_fmt_translator_t oft_table[];
static ioc_cmd_translator_t ioc_translators_file[];
```

```
128 static ioc_cmd_translator_t ioc_translators_fifo[];
129 static ioc_cmd_translator_t ioc_translators_sock[];
130 static ioc_dev_translator_t ioc_translator_ptm;
131 static ioc_dev_translator_t *ioc_translators_dev[];
132 static ioc_fs_translator_t *ioc_translators_fs[];
133 static ioc_errno_translator_t ioc_translators_errno[];

135 /*
136  * Interface name table.
137  */
138 typedef struct ifname_map {
139         char    im_linux[IFNAMSIZ];
140         char    im_solaris[IFNAMSIZ];
141         struct ifname_map *im_next;
142 } ifname_map_t;

144 static ifname_map_t *ifname_map;
145 static mutex_t ifname_mtx;

147 /*
148  * Macros and structures to help convert integers to string
149  * values that they represent (for displaying in debug output).
150  */
151 #define I2S_ENTRY(x)    { x, #x },
152 #define I2S_END         { 0, NULL }

154 typedef struct int2str {
155         int     i2s_int;
156         char    *i2s_str;
157 } int2str_t;

159 static int2str_t st_mode_strings[] = {
160         I2S_ENTRY(S_IFIFO)
161         I2S_ENTRY(S_IFCHR)
162         I2S_ENTRY(S_IFDIR)
163         I2S_ENTRY(S_IFBLK)
164         I2S_ENTRY(S_IFREG)
165         I2S_ENTRY(S_IFLNK)
166         I2S_ENTRY(S_IFSOCK)
167         I2S_ENTRY(S_IFDOOR)
168         I2S_ENTRY(S_IFPORT)
169         I2S_END
170 };

172 static int2str_t oss_fmt_str[] = {
173         I2S_ENTRY(LX_OSS_AFMT_QUERY)
174         I2S_ENTRY(LX_OSS_AFMT_MU_LAW)
175         I2S_ENTRY(LX_OSS_AFMT_A_LAW)
176         I2S_ENTRY(LX_OSS_AFMT_IMA_ADPCM)
177         I2S_ENTRY(LX_OSS_AFMT_U8)
178         I2S_ENTRY(LX_OSS_AFMT_S16_LE)
179         I2S_ENTRY(LX_OSS_AFMT_S16_BE)
180         I2S_ENTRY(LX_OSS_AFMT_S8)
181         I2S_ENTRY(LX_OSS_AFMT_U16_LE)
182         I2S_ENTRY(LX_OSS_AFMT_U16_BE)
183         I2S_ENTRY(LX_OSS_AFMT_MPEG)
184         I2S_END
185 };

187 static void
188 lx_ioctl_msg(int fd, int cmd, char *lx_cmd_str, struct stat *stat, char *msg)
189 {
190         int     errno_backup = errno;
191         char    *path, path_buf[MAXPATHLEN];

193         assert(msg != NULL);
```



```
195         if (lx_debug_enabled == 0)
196                 return;

198         path = lx_fd_to_path(fd, path_buf, sizeof (path_buf));
199         if (path == NULL)
200                 path = "?";

202         if (lx_cmd_str == NULL)
203                 lx_cmd_str = "?";

205         /* Display the initial error message and extended ioctl information. */
206         lx_debug("\t%s", msg);
207         lx_debug("\tlx_ioctl(): cmd = 0x%x - %s, fd = %d - %s",
208             cmd, lx_cmd_str, fd, path);

210         /* Display information about the target file, if it's available. */
211         if (stat != NULL) {
212                 major_t fd_major = getmajor(stat->st_rdev);
213                 minor_t fd_minor = getminor(stat->st_rdev);
214                 int     fd_mode = stat->st_mode & S_IFMT;
215                 char    *fd_mode_str = "unknown";
216                 char    buf[LX_MSG_MAXLEN];
217                 int     i;

219                 /* Translate the file type bits into a string. */
220                 for (i = 0; st_mode_strings[i].i2s_str != NULL; i++) {
221                         if (fd_mode != st_mode_strings[i].i2s_int)
222                                 continue;
223                         fd_mode_str = st_mode_strings[i].i2s_str;
224                         break;
225                 }

227                 (void) snprintf(buf, sizeof (buf),
228                     "\tlx_ioctl(): mode = %s", fd_mode_str);

230                 if ((fd_mode == S_IFCHR) || (fd_mode == S_IFBLK)) {
231                         char    *fd_driver[MODMAXNAMELEN + 1];
232                         int     i;

234                         /* This is a device so display the devt. */
235                         i = strlen(buf);
236                         (void) snprintf(buf + i, sizeof (buf) - i,
237                             "; rdev = [%d, %d]", fd_major, fd_minor);

239                         /* Try to display the drivers name. */
240                         if (modctl(MODGETNAME,
241                             fd_driver, sizeof (fd_driver), &fd_major) == 0)
242                                 i = strlen(buf);
243                                 (void) snprintf(buf + i, sizeof (buf) - i,
244                                     "; driver = %s", fd_driver);
245                 }
246                 lx_debug(buf);
247         }

249         /* Restore errno. */
250         errno = errno_backup;
251 }

253 static int
254 ldlinux_check(int fd)
255 {
256         struct str_mlist        mlist[MAX_STRMODS];
257         struct str_list         strlist;
258         int                     i;
```

```
260         /* Get the number of modules on the stream. */
261         lx_debug("\tioctl(%d, 0x%x - %s, ...)",
262             fd, I_LIST, "I_LIST");
263         if ((i = ioctl(fd, I_LIST, (struct str_list *)NULL)) < 0) {
264                 lx_debug("\tldlinux_check(): unable to count stream modules");
265                 return (-errno);
266         }

268         /* Sanity check the number of modules on the stream. */
269         assert(i <= MAX_STRMODS);

271         /* Get the list of modules on the stream. */
272         strlist.sl_nmods = i;
273         strlist.sl_modlist = mlist;
274         lx_debug("\tioctl(%d, 0x%x - %s, ...)",
275             fd, I_LIST, "I_LIST");
276         if (ioctl(fd, I_LIST, &strlist) < 0) {
277                 lx_debug("\tldlinux_check(): unable to list stream modules");
278                 return (-errno);
279         }

281         for (i = 0; i < strlist.sl_nmods; i++)
282                 if (strcmp(strlist.sl_modlist[i].l_name, LDLINUX_MOD) == 0)
283                         return (1);

285         return (0);
286 }

288 static int
289 ioctl_istr(int fd, int cmd, char *cmd_str, void *arg, int arg_len)
290 {
291         struct strioctl istr;

293         istr.ic_cmd = cmd;
294         istr.ic_len = arg_len;
295         istr.ic_timout = 0;
296         istr.ic_dp = arg;

298         lx_debug("\tioctl_istr(%d, 0x%x - %s, ...)", fd, cmd, cmd_str);
299         if (ioctl(fd, I_STR, &istr) < 0)
300                 return (-1);
301         return (0);
302 }

304 /*
305  * Add an interface name mapping if it doesn't already exist.
306  *
307  * Interfaces with IFF_LOOPBACK flag get renamed to loXXX.
308  * Interfaces with IFF_BROADCAST flag get renamed to ethXXX.
309  *
310  * Caller locks the name table.
311  */
312 static int
313 ifname_add(char *if_name, int if_flags)
314 {
315         static int eth_index = 0;
316         static int lo_index = 0;
317         ifname_map_t **im_pp;

319         for (im_pp = &ifname_map; *im_pp; im_pp = &(*im_pp)->im_next)
320                 if (strncmp((*im_pp)->im_solaris, if_name, IFNAMSIZ) == 0)
321                         return (0);

323         *im_pp = calloc(1, sizeof (ifname_map_t));
324         if (*im_pp == NULL)
325                 return (-1);
```



```
327         (void) strlcpy((*im_pp)->im_solaris, if_name, IFNAMSIZ);
328         if (if_flags & IFF_LOOPBACK) {
329                 /* Loopback */
330                 if (lo_index == 0)
331                         (void) strlcpy((*im_pp)->im_linux, "lo", IFNAMSIZ);
332                 else
333                         (void) snprintf((*im_pp)->im_linux, IFNAMSIZ,
334                             "lo:%d", lo_index);
335                 lo_index++;
336         } else if (if_flags & IFF_BROADCAST) {
337                 /* Assume ether if it has a broadcast address */
338                 (void) snprintf((*im_pp)->im_linux, IFNAMSIZ,
339                     "eth%d", eth_index);
340                 eth_index++;
341         } else {
342                 /* Do not translate unknown interfaces */
343                 (void) strlcpy((*im_pp)->im_linux, if_name, IFNAMSIZ);
344         }

346         lx_debug("map interface %s -> %s", if_name, (*im_pp)->im_linux);

348         return (0);
349 }

351 static int
352 ifname_cmp(const void *p1, const void *p2)
353 {
354         struct ifreq *rp1 = (struct ifreq *)p1;
355         struct ifreq *rp2 = (struct ifreq *)p2;

357         return (strncmp(rp1->ifr_name, rp2->ifr_name, IFNAMSIZ));
358 }

360 /*
361  * (Re-)scan all interfaces and add them to the name table.
362  * Caller locks the name table.
363  */
364 static int
365 ifname_scan(void)
366 {
367         struct ifconf conf;
368         int i, fd, ifcount;

370         conf.ifc_buf = NULL;

372         if ((fd = socket(PF_INET, SOCK_DGRAM, 0)) < 0)
373                 goto fail;
374         lx_debug("\tioctl(%d, 0x%x - %s, ...)", fd, SIOCGIFNUM, "SIOCGIFNUM");
375         if (ioctl(fd, SIOCGIFNUM, &ifcount) < 0) {
376                 lx_debug("\tifname_scan(): unable to get number of interfaces");
377                 goto fail;
378         }

380         conf.ifc_len = ifcount * sizeof (struct ifreq);
381         if ((conf.ifc_buf = calloc(ifcount, sizeof (struct ifreq))) == NULL)
382                 goto fail;
383         lx_debug("\tioctl(%d, 0x%x - %s, ...)", fd, SIOCGIFCONF, "SIOCGIFCONF");
384         if (ioctl(fd, SIOCGIFCONF, &conf) < 0) {
385                 lx_debug("\tifname_scan(): unable to get interfaces");
386                 goto fail;
387         }

389         /* Get the interface flags */
390         for (i = 0; i < ifcount; i++) {
391                 lx_debug("\tioctl(%d, 0x%x - %s, ...)",
```

```
392                     fd, SIOCGIFFLAGS, "SIOCGIFFLAGS");
393                 if (ioctl(fd, SIOCGIFFLAGS, &conf.ifc_req[i]) < 0) {
394                         conf.ifc_req[i].ifr_flags = 0;
395                         lx_debug("\tifname_scan(): unable to get flags for %s",
396                             conf.ifc_req[i].ifr_name);
397                 }
398         }

400         /*
401          * Sort the interfaces by name to preserve the order
402          * across reboots of this zone.  Note that the order of
403          * interface names won't be consistent across network
404          * configuration changes.  ie.  If network interfaces
405          * are added or removed from a zone (either dynamically
406          * or statically) the network interfaces names to physical
407          * network interface mappings that linux apps see may
408          * change.
409          */
410         qsort(conf.ifc_req, ifcount, sizeof (struct ifreq), ifname_cmp);

412         /* Add to the name table */
413         for (i = 0; i < ifcount; i++)
414                 if (ifname_add(conf.ifc_req[i].ifr_name,
415                     conf.ifc_req[i].ifr_flags) != 0)
416                         goto fail;

418         (void) close(fd);
419         free(conf.ifc_buf);

421         return (0);

423 fail:
424         if (fd >= 0)
425                 (void) close(fd);
426         if (conf.ifc_buf != NULL)
427                 free(conf.ifc_buf);

429         return (-1);
430 }

432 static int
433 ifname_from_linux(char *name)
434 {
435         int pass;
436         ifname_map_t *im_p;

438         (void) mutex_lock(&ifname_mtx);

440         for (pass = 0; pass < 2; pass++) {
441                 for (im_p = ifname_map; im_p; im_p = im_p->im_next)
442                         if (strncmp(im_p->im_linux, name, IFNAMSIZ) == 0)
443                                 break;
444                 if (im_p != NULL || (pass == 0 && ifname_scan() != 0))
445                         break;
446         }

448         (void) mutex_unlock(&ifname_mtx);

450         if (im_p) {
451                 (void) strlcpy(name, im_p->im_solaris, IFNAMSIZ);
452                 return (0);
453         }

455         return (-1);
456 }
```



```
458 static int
459 ifname_from_solaris(char *name)
460 {
461         int pass;
462         ifname_map_t *im_p;

464         (void) mutex_lock(&ifname_mtx);

466         for (pass = 0; pass < 2; pass++) {
467                 for (im_p = ifname_map; im_p; im_p = im_p->im_next)
468                         if (strncmp(im_p->im_solaris, name, IFNAMSIZ) == 0)
469                                 break;
470                 if (im_p != NULL || (pass == 0 && ifname_scan() != 0))
471                         break;
472         }

474         (void) mutex_unlock(&ifname_mtx);

476         if (im_p) {
477                 (void) strlcpy(name, im_p->im_linux, IFNAMSIZ);
478                 return (0);
479         }

481         return (-1);
482 }

484 /*
485  * Called to initialize the ioctl translation subsystem.
486  */
487 int
488 lx_ioctl_init()
489 {
490         int i, ret;

492         /* Figure out the major numbers for our devices translators. */
493         for (i = 0; ioc_translators_dev[i] != NULL; i++) {
494                 ioc_dev_translator_t *idt = ioc_translators_dev[i];

496                 ret = modctl(MODGETMAJBIND,
497                     idt->idt_driver, strlen(idt->idt_driver) + 1,
498                     &idt->idt_major);

500                 if (ret != 0) {
501                         lx_err(gettext("%s%s) failed: %s\n"),
502                             "lx_ioctl_init(): modctl(MODGETMAJBIND, ",
503                             idt->idt_driver, strerror(errno));
504                         lx_err(gettext("%s: %s translator disabled for: %s\n"),
505                             "lx_ioctl_init()", "ioctl", idt->idt_driver);
506                         idt->idt_major = (major_t)-1;
507                 }
508         }

510         /* Create the interface name table */
511         if (ifname_scan() != 0)
512                 lx_err("lx_ioctl_init(): ifname_scan() failed\n");

514         return (0);
515 }

517 static ioc_cmd_translator_t *
518 lx_ioctl_find_ict_cmd(ioc_cmd_translator_t *ict, int cmd)
519 {
520         assert(ict != NULL);
521         while ((ict != NULL) && (ict->ict_func != NULL)) {
522                 if (cmd == ict->ict_lx_cmd)
523                         return (ict);
```

```
524                 ict++;
525         }
526         return (NULL);
527 }
528
529 /*
530  * Main entry point for the ioctl translater.
531  */
532 int
533 lx_ioctl(uintptr_t p1, uintptr_t p2, uintptr_t p3)
534 {
535         int                     fd = (int)p1;
536         int                     cmd = (int)p2;
537         intptr_t                arg = (uintptr_t)p3;
538         struct stat             stat;
539         ioc_cmd_translator_t    *ict = NULL;
540         ioc_errno_translator_t  *iet = NULL;
541         major_t                 fd_major;
542         int                     i, ret;
543
544         if (fstat(fd, &stat) != 0) {
545                 lx_ioctl_msg(fd, cmd, NULL, NULL,
546                     "lx_ioctl(): fstat() failed");
547
548                 /*
549                  * Linux ioctl(2) is only documented to return EBADF, EFAULT,
550                  * EINVAL or ENOTTY.
551                  *
552                  * EINVAL is documented to be "Request or argp is not valid",
553                  * so it's reasonable to force any errno that's not EBADF,
554                  * EFAULT or ENOTTY to be EINVAL.
555                  */
556                 if ((errno != EBADF) && (errno != EFAULT) && (errno != ENOTTY))
557                         errno = EINVAL;
558
559                 return (-errno);        /* errno already set. */
560         }
561
562         switch (stat.st_mode & S_IFMT) {
563         default:
564                 break;
565         case S_IFREG:
566                 /* Use file translators. */
567                 ict = ioc_translators_file;
568                 break;
569
570         case S_IFSOCK:
571                 /* Use socket translators. */
572                 ict = ioc_translators_sock;
573                 break;
574
575         case S_IFIFO:
576                 /* Use fifo translators. */
577                 ict = ioc_translators_fifo;
578                 break;
579
580         case S_IFCHR:
581                 fd_major = getmajor(stat.st_rdev);
582
583                 /*
584                  * Look through all the device translators to see if there
585                  * is one for this device.
586                  */
587                 for (i = 0; ioc_translators_dev[i] != NULL; i++) {
588                         if (fd_major != ioc_translators_dev[i]->idt_major)
589                                 continue;
```



```
590
591                         /* We found a translator for this device. */
592                         ict = ioc_translators_dev[i]->idt_cmds;
593                         break;
594                 }
595                 break;
596         }
597
598         /*
599          * Search the selected translator group to see if we have a
600          * translator for this specific command.
601          */
602         if ((ict != NULL) &&
603             ((ict = lx_ioctl_find_ict_cmd(ict, cmd)) != NULL)) {
604                 /* We found a translator for this command, invoke it. */
605                 lx_ioctl_msg(fd, cmd, ict->ict_lx_cmd_str, &stat,
606                     "lx_ioctl(): emulating ioctl");
607
608                 ret = ict->ict_func(fd, &stat, ict->ict_cmd, ict->ict_cmd_str,
609                     arg);
610
611                 if ((ret < 0) && (ret != -EBADF) && (ret != -EFAULT) &&
612                     (ret != -ENOTTY))
613                         ret = -EINVAL;
614
615                 return (ret);
616         }
617
618         /*
619          * If we didn't find a file or device translator for this
620          * command then try to find a filesystem translator for
621          * this command.
622          */
623         for (i = 0; ioc_translators_fs[i] != NULL; i++) {
624                 if (strcmp(stat.st_fstype,
625                     ioc_translators_fs[i]->ift_filesystem) != 0)
626                         continue;
627
628                 /* We found a translator for this filesystem. */
629                 ict = ioc_translators_fs[i]->ift_cmds;
630                 break;
631         }
632
633         /*
634          * Search the selected translator group to see if we have a
635          * translator for this specific command.
636          */
637         if ((ict != NULL) &&
638             ((ict = lx_ioctl_find_ict_cmd(ict, cmd)) != NULL)) {
639                 /* We found a translator for this command, invoke it. */
640                 lx_ioctl_msg(fd, cmd, ict->ict_lx_cmd_str, &stat,
641                     "lx_ioctl(): emulating ioctl");
642                 ret = ict->ict_func(fd, &stat, ict->ict_cmd, ict->ict_cmd_str,
643                     arg);
644
645                 if ((ret < 0) && (ret != -EBADF) && (ret != -EFAULT) &&
646                     (ret != -ENOTTY))
647                         ret = -EINVAL;
648
649                 return (ret);
650         }
651
652         /*
653          * No translator for this ioctl was found.
654          * Check if there is an errno translator.
655          */
```

```
656         for (iet = ioc_translators_errno; iet->iet_lx_cmd_str != NULL; iet++) {
657                 if (cmd != iet->iet_lx_cmd)
658                         continue;

660                 /* We found a an errno translator for this ioctl. */
661                 lx_ioctl_msg(fd, cmd, iet->iet_lx_cmd_str, &stat,
662                     "lx_ioctl(): emulating errno");

664                 ret = -iet->iet_errno;

666                 if ((ret < 0) && (ret != -EBADF) && (ret != -EFAULT) &&
667                     (ret != -ENOTTY))
668                         ret = -EINVAL;

670                 return (ret);
671         }

673         lx_ioctl_msg(fd, cmd, NULL, &stat,
674             "lx_ioctl(): unsupported linux ioctl");
675         lx_unsupported(gettext("lx_ioctl(): unsupported linux ioctl (%d)"),
676             cmd);
677         return (-EINVAL);
678 }


681 /*
682  * Ioctl translator functions.
683  */
684 /*
685  * Used by translators that want to explicitly return EINVAL for an
686  * ioctl(2) instead of having the translation framework do it implicitly.
687  * This allows us to indicate which unsupported ioctl(2)s should not
688  * trigger a SIGSYS when running in LX_STRICT mode.
689  */
690 /* ARGSUSED */
691 static int
692 ict_einval(int fd, struct stat *stat, int cmd, char *cmd_str, intptr_t arg)
693 {
694         return (-EINVAL);
695 }

697 static int
698 /*ARGSUSED*/
699 ict_pass(int fd, struct stat *stat, int cmd, char *cmd_str, intptr_t arg)
700 {
701         int ret;

703         lx_debug("\tioctl(%d, 0x%x - %s, ...)",
704             fd, cmd, cmd_str);
705         ret = ioctl(fd, cmd, arg);
706         return (ret < 0 ? -errno : ret);
707 }

709 static int
710 /*ARGSUSED*/
711 ict_tcsbrkp(int fd, struct stat *stat, int cmd, char *cmd_str, intptr_t arg)
712 {
713         int ret, dur = 0;

715         assert(cmd == LX_TCSBRKP);
716         lx_debug("\tioctl(%d, 0x%x - %s, ...)",
717             fd, TCSBRK, "TCSBRK");
718         ret = ioctl(fd, TCSBRK, (intptr_t)&dur);
719         return (ret < 0 ? -errno : ret);
720 }
```



```
722 static int
723 /*ARGSUSED*/
724 ict_sioifoob(int fd, struct stat *stat, int cmd, char *cmd_str, intptr_t arg)
725 {
726         int req, *reqp = (int *)arg;
727         int len, val;

729         assert(cmd == SIOCATMARK);

731         if (uucopy(reqp, &req, sizeof (req)) != 0)
732                 return (-errno);

734         len = sizeof (val);

736         /*
737          * Linux expects a SIOCATMARK of a UDP socket to return EINVAL, while
738          * Solaris allows it.
739          */
740         if (getsockopt(fd, SOL_SOCKET, SO_TYPE, &val, &len) < 0) {
741                 lx_debug("ict_siofmark: getsockopt failed, errno %d", errno);
742                 return (-EINVAL);
743         }

745         if ((len != sizeof (val)) || (val != SOCK_STREAM))
746                 return (-EINVAL);

748         if (ioctl(fd, cmd, &req) < 0)
749                 return (-errno);

751         if (uucopy(&req, reqp, sizeof (req)) != 0)
752                 return (-errno);

754         return (0);
755 }

757 static int
758 /*ARGSUSED*/
759 ict_sioifreq(int fd, struct stat *stat, int cmd, char *cmd_str, intptr_t arg)
760 {
761         struct ifreq req, *reqp = (struct ifreq *)arg;

763         assert(cmd == SIOCGIFFLAGS || cmd == SIOCSIFFLAGS ||
764             cmd == SIOCGIFADDR || cmd == SIOCSIFADDR ||
765             cmd == SIOCGIFDSTADDR || cmd == SIOCSIFDSTADDR ||
766             cmd == SIOCGIFBRDADDR || cmd == SIOCSIFBRDADDR ||
767             cmd == SIOCGIFNETMASK || cmd == SIOCSIFNETMASK ||
768             cmd == SIOCGIFMETRIC || cmd == SIOCSIFMETRIC ||
769             cmd == SIOCGIFMTU || cmd == SIOCSIFMTU);

771         /* Copy in the data */
772         if (uucopy(reqp, &req, sizeof (struct ifreq)) != 0)
773                 return (-errno);

775         if (ifname_from_linux(req.ifr_name) < 0)
776                 return (-EINVAL);

778         lx_debug("\tioctl(%d, 0x%x - %s, %.14s",
779             fd, cmd, cmd_str, req.ifr_name);

781         if (ioctl(fd, cmd, &req) < 0)
782                 return (-errno);

784         if (ifname_from_solaris(req.ifr_name) < 0)
785                 return (-EINVAL);

787         /* Copy out the data */
```

```
788         if (uucopy(&req, reqp, sizeof (struct ifreq)) != 0)
789                 return (-errno);
790
791         return (0);
792 }
793
794 static int
795 /*ARGSUSED*/
796 ict_siocgifconf(int fd, struct stat *stat, int cmd, char *cmd_str, intptr_t arg)
797 {
798         struct ifconf   conf, *confp = (struct ifconf *)arg;
799         int             i, ifcount, ret;
800
801         assert(cmd == LX_SIOCGIFCONF);
802
803         /* Copy in the data. */
804         if (uucopy(confp, &conf, sizeof (conf)) != 0)
805                 return (-errno);
806
807         if (conf.ifc_len == 0) {
808                 /* They want to know how many interfaces there are. */
809                 lx_debug("\tioctl(%d, 0x%x - %s, ...)",
810                     fd, SIOCGIFNUM, "SIOCGIFNUM");
811                 if (ioctl(fd, SIOCGIFNUM, (intptr_t)&ifcount) < 0)
812                         return (-errno);
813                 conf.ifc_len = ifcount * sizeof (struct ifreq);
814
815                 /* Check if we're done. */
816                 if (conf.ifc_buf == NULL) {
817                         /* Copy out the data. */
818                         if (uucopy(&conf, confp, sizeof (conf)) != 0)
819                                 return (-errno);
820                         return (0);
821                 }
822         }
823
824         /* Get interface configuration list. */
825         lx_debug("\tioctl(%d, 0x%x - %s, ...)", fd, SIOCGIFCONF, "SIOCGIFCONF");
826         ret = ioctl(fd, SIOCGIFCONF, &conf);
827         if (ret < 0)
828                 return (-errno);
829
830         /* Rename interfaces to linux */
831         for (i = 0; i < conf.ifc_len / sizeof (struct ifreq); i++)
832                 if (ifname_from_solaris(conf.ifc_req[i].ifr_name) < 0)
833                         return (-EINVAL);
834
835         /* Copy out the data */
836         if (uucopy(&conf, confp, sizeof (conf)) != 0)
837                 return (-errno);
838
839         return (0);
840 }
841
842 static int
843 /*ARGSUSED*/
844 ict_siocifhwaddr(int fd, struct stat *stat, int cmd, char *cmd_str,
845     intptr_t arg)
846 {
847         struct ifreq req, *reqp = (struct ifreq *)arg;
848         struct arpreq arpreq;
849
850         assert(cmd == LX_SIOCGIFHWADDR || cmd == LX_SIOCSIFHWADDR);
851
852         /* Copy in the data */
853         if (uucopy(reqp, &req, sizeof (struct ifreq)) != 0)
```



```
854                 return (-errno);
855
856         lx_debug("\tioctl(%d, 0x%x - %s, lx %.14s)",
857             fd, cmd,
858             (cmd == LX_SIOCGIFHWADDR) ? "SIOCGIFHWADDR" : "SIOCSIFHWADDR",
859             req.ifr_name);
860
861         /*
862          * We're not going to support SIOCSIFHWADDR, but we need to be
863          * able to check the result of the uucopy first to see if the command
864          * should have returned EFAULT.
865          */
866         if (cmd == LX_SIOCSIFHWADDR) {
867                 lx_unsupported(gettext(
868                     "lx_ioctl(): unsupported linux ioctl: %s"),
869                     "SIOCSIFHWADDR");
870                 return (-EINVAL);
871         }
872
873         if (strcmp(req.ifr_name, "lo") == 0 ||
874             strncmp(req.ifr_name, "lo:", 3) == 0) {
875                 /* Abuse ifr_addr for linux ifr_hwaddr */
876                 bzero(&req.ifr_addr, sizeof (struct sockaddr));
877                 req.ifr_addr.sa_family = LX_ARPHRD_LOOPBACK;
878
879                 /* Copy out the data */
880                 if (uucopy(&req, reqp, sizeof (struct ifreq)) != 0)
881                         return (-errno);
882
883                 return (0);
884         }
885
886         if (ifname_from_linux(req.ifr_name) < 0)
887                 return (-EINVAL);
888
889         lx_debug("\tioctl(%d, 0x%x - %s, %.14s)",
890             fd, SIOCGIFADDR, "SIOCGIFADDR", req.ifr_name);
891
892         if (ioctl(fd, SIOCGIFADDR, &req) < 0)
893                 return (-errno);
894
895         bcopy(&req.ifr_addr, &arpreq.arp_pa, sizeof (struct sockaddr));
896
897         lx_debug("\tioctl(%d, 0x%x - %s, ...)", fd, SIOCGARP, "SIOCGARP");
898
899         if (ioctl(fd, SIOCGARP, &arpreq) < 0)
900                 return (-errno);
901
902         if (ifname_from_solaris(req.ifr_name) < 0)
903                 return (-EINVAL);
904
905         /* Abuse ifr_addr for linux ifr_hwaddr */
906         bcopy(&arpreq.arp_ha, &req.ifr_addr, sizeof (struct sockaddr));
907         if (strncmp(req.ifr_name, "eth", 3) == 0)
908                 req.ifr_addr.sa_family = LX_ARPHRD_ETHER;
909         else
910                 req.ifr_addr.sa_family = LX_ARPHRD_VOID;
911
912         /* Copy out the data */
913         if (uucopy(&req, reqp, sizeof (struct ifreq)) != 0)
914                 return (-errno);
915
916         return (0);
917 }
918
919 static void
```

```
920 l2s_termios(struct lx_termios *l_tios, struct termios *s_tios)
921 {
922         assert((l_tios != NULL) && (s_tios != NULL));

924         bzero(s_tios, sizeof (*s_tios));

926         s_tios->c_iflag = l_tios->c_iflag;
927         s_tios->c_oflag = l_tios->c_oflag;
928         s_tios->c_cflag = l_tios->c_cflag;

930         s_tios->c_lflag = l_tios->c_lflag;
931         if (s_tios->c_lflag & ICANON) {
932                 s_tios->c_cc[VEOF] = l_tios->c_cc[LX_VEOF];
933                 s_tios->c_cc[VEOL] = l_tios->c_cc[LX_VEOL];
934         } else {
935                 s_tios->c_cc[VMIN] = l_tios->c_cc[LX_VMIN];
936                 s_tios->c_cc[VTIME] = l_tios->c_cc[LX_VTIME];
937         }

939         s_tios->c_cc[VEOL2] = l_tios->c_cc[LX_VEOL2];
940         s_tios->c_cc[VERASE] = l_tios->c_cc[LX_VERASE];
941         s_tios->c_cc[VKILL] = l_tios->c_cc[LX_VKILL];
942         s_tios->c_cc[VREPRINT] = l_tios->c_cc[LX_VREPRINT];
943         s_tios->c_cc[VLNEXT] = l_tios->c_cc[LX_VLNEXT];
944         s_tios->c_cc[VWERASE] = l_tios->c_cc[LX_VWERASE];
945         s_tios->c_cc[VINTR] = l_tios->c_cc[LX_VINTR];
946         s_tios->c_cc[VQUIT] = l_tios->c_cc[LX_VQUIT];
947         s_tios->c_cc[VSWTCH] = l_tios->c_cc[LX_VSWTC];
948         s_tios->c_cc[VSTART] = l_tios->c_cc[LX_VSTART];
949         s_tios->c_cc[VSTOP] = l_tios->c_cc[LX_VSTOP];
950         s_tios->c_cc[VSUSP] = l_tios->c_cc[LX_VSUSP];
951         s_tios->c_cc[VDISCARD] = l_tios->c_cc[LX_VDISCARD];
952 }

954 static void
955 l2s_termio(struct lx_termio *l_tio, struct termio *s_tio)
956 {
957         assert((l_tio != NULL) && (s_tio != NULL));

959         bzero(s_tio, sizeof (*s_tio));

961         s_tio->c_iflag = l_tio->c_iflag;
962         s_tio->c_oflag = l_tio->c_oflag;
963         s_tio->c_cflag = l_tio->c_cflag;

965         s_tio->c_lflag = l_tio->c_lflag;
966         if (s_tio->c_lflag & ICANON) {
967                 s_tio->c_cc[VEOF] = l_tio->c_cc[LX_VEOF];
968         } else {
969                 s_tio->c_cc[VMIN] = l_tio->c_cc[LX_VMIN];
970                 s_tio->c_cc[VTIME] = l_tio->c_cc[LX_VTIME];
971         }

973         s_tio->c_cc[VINTR] = l_tio->c_cc[LX_VINTR];
974         s_tio->c_cc[VQUIT] = l_tio->c_cc[LX_VQUIT];
975         s_tio->c_cc[VERASE] = l_tio->c_cc[LX_VERASE];
976         s_tio->c_cc[VKILL] = l_tio->c_cc[LX_VKILL];
977         s_tio->c_cc[VSWTCH] = l_tio->c_cc[LX_VSWTC];
978 }

980 static void
981 termios2lx_cc(struct lx_termios *l_tios, struct lx_cc *lio)
982 {
983         assert((l_tios != NULL) && (lio != NULL));

985         bzero(lio, sizeof (*lio));
```



```
987         lio->veof = l_tios->c_cc[LX_VEOF];
988         lio->veol = l_tios->c_cc[LX_VEOL];
989         lio->vmin = l_tios->c_cc[LX_VMIN];
990         lio->vtime = l_tios->c_cc[LX_VTIME];
991 }

993 static void
994 termio2lx_cc(struct lx_termio *l_tio, struct lx_cc *lio)
995 {
996         assert((l_tio != NULL) && (lio != NULL));

998         bzero(lio, sizeof (*lio));

1000        lio->veof = l_tio->c_cc[LX_VEOF];
1001        lio->veol = 0;
1002        lio->vmin = l_tio->c_cc[LX_VMIN];
1003        lio->vtime = l_tio->c_cc[LX_VTIME];
1004 }

1006 static void
1007 s2l_termios(struct termios *s_tios, struct lx_termios *l_tios)
1008 {
1009        assert((s_tios != NULL) && (l_tios != NULL));

1011        bzero(l_tios, sizeof (*l_tios));

1013        l_tios->c_iflag = s_tios->c_iflag;
1014        l_tios->c_oflag = s_tios->c_oflag;
1015        l_tios->c_cflag = s_tios->c_cflag;
1016        l_tios->c_lflag = s_tios->c_lflag;

1018        if (s_tios->c_lflag & ICANON) {
1019                l_tios->c_cc[LX_VEOF] = s_tios->c_cc[VEOF];
1020                l_tios->c_cc[LX_VEOL] = s_tios->c_cc[VEOL];
1021        } else {
1022                l_tios->c_cc[LX_VMIN] = s_tios->c_cc[VMIN];
1023                l_tios->c_cc[LX_VTIME] = s_tios->c_cc[VTIME];
1024        }

1026        l_tios->c_cc[LX_VEOL2] = s_tios->c_cc[VEOL2];
1027        l_tios->c_cc[LX_VERASE] = s_tios->c_cc[VERASE];
1028        l_tios->c_cc[LX_VKILL] = s_tios->c_cc[VKILL];
1029        l_tios->c_cc[LX_VREPRINT] = s_tios->c_cc[VREPRINT];
1030        l_tios->c_cc[LX_VLNEXT] = s_tios->c_cc[VLNEXT];
1031        l_tios->c_cc[LX_VWERASE] = s_tios->c_cc[VWERASE];
1032        l_tios->c_cc[LX_VINTR] = s_tios->c_cc[VINTR];
1033        l_tios->c_cc[LX_VQUIT] = s_tios->c_cc[VQUIT];
1034        l_tios->c_cc[LX_VSWTC] = s_tios->c_cc[VSWTCH];
1035        l_tios->c_cc[LX_VSTART] = s_tios->c_cc[VSTART];
1036        l_tios->c_cc[LX_VSTOP] = s_tios->c_cc[VSTOP];
1037        l_tios->c_cc[LX_VSUSP] = s_tios->c_cc[VSUSP];
1038        l_tios->c_cc[LX_VDISCARD] = s_tios->c_cc[VDISCARD];
1039 }

1041 static void
1042 s2l_termio(struct termio *s_tio, struct lx_termio *l_tio)
1043 {
1044        assert((s_tio != NULL) && (l_tio != NULL));

1046        bzero(l_tio, sizeof (*l_tio));

1048        l_tio->c_iflag = s_tio->c_iflag;
1049        l_tio->c_oflag = s_tio->c_oflag;
1050        l_tio->c_cflag = s_tio->c_cflag;
1051        l_tio->c_lflag = s_tio->c_lflag;
```

```
1053         if (s_tio->c_lflag & ICANON) {
1054                 l_tio->c_cc[LX_VEOF] = s_tio->c_cc[VEOF];
1055         } else {
1056                 l_tio->c_cc[LX_VMIN] = s_tio->c_cc[VMIN];
1057                 l_tio->c_cc[LX_VTIME] = s_tio->c_cc[VTIME];
1058         }

1060         l_tio->c_cc[LX_VINTR] = s_tio->c_cc[VINTR];
1061         l_tio->c_cc[LX_VQUIT] = s_tio->c_cc[VQUIT];
1062         l_tio->c_cc[LX_VERASE] = s_tio->c_cc[VERASE];
1063         l_tio->c_cc[LX_VKILL] = s_tio->c_cc[VKILL];
1064         l_tio->c_cc[LX_VSWTC] = s_tio->c_cc[VSWTCH];
1065 }

1067 static int
1068 /*ARGSUSED*/
1069 ict_tcsets(int fd, struct stat *stat, int cmd, char *cmd_str, intptr_t arg)
1070 {
1071         struct lx_termios       l_tios, *l_tiosp = (struct lx_termios *)arg;
1072         struct termios          s_tios;
1073         struct lx_cc            lio;
1074         int                     ldlinux, ret;

1076         assert(cmd == TCSETS || cmd == TCSETSW || cmd == TCSETSF);

1078         /* Copy in the data. */
1079         if (uucopy(l_tiosp, &l_tios, sizeof (l_tios)) != 0)
1080                 return (-errno);

1082         /*
1083          * The TIOCSETLD/TIOCGETLD ioctls are only supported by the
1084          * ldlinux strmod.  So make sure the module exists on the
1085          * target stream before we invoke the ioctl.
1086          */
1087         if ((ldlinux = ldlinux_check(fd)) < 0)
1088                 return (ldlinux);

1090         if (ldlinux == 1) {
1091                 termios2lx_cc(&l_tios, &lio);
1092                 if (ioctl_istr(fd, TIOCSETLD, "TIOCSETLD",
1093                     &lio, sizeof (lio)) < 0)
1094                         return (-errno);
1095         }

1097         l2s_termios(&l_tios, &s_tios);
1098         lx_debug("\tioctl(%d, 0x%x - %s, ...)",
1099             fd, cmd, cmd_str);
1100         ret = ioctl(fd, cmd, (intptr_t)&s_tios);
1101         return ((ret < 0) ? -errno : ret);
1102 }

1104 static int
1105 /*ARGSUSED*/
1106 ict_tcseta(int fd, struct stat *stat, int cmd, char *cmd_str, intptr_t arg)
1107 {
1108         struct lx_termio        l_tio, *l_tiop = (struct lx_termio *)arg;
1109         struct termio           s_tio;
1110         struct lx_cc            lio;
1111         int                     ldlinux, ret;

1113         assert(cmd == TCSETA || cmd == TCSETAW || cmd == TCSETAF);

1115         /* Copy in the data. */
1116         if (uucopy(l_tiop, &l_tio, sizeof (l_tio)) != 0)
1117                 return (-errno);
```



```
1119         /*
1120          * The TIOCSETLD/TIOCGETLD ioctls are only supported by the
1121          * ldlinux strmod.  So make sure the module exists on the
1122          * target stream before we invoke the ioctl.
1123          */
1124         if ((ldlinux = ldlinux_check(fd)) < 0)
1125                 return (ldlinux);

1127         if (ldlinux == 1) {
1128                 termio2lx_cc(&l_tio, &lio);
1129                 if (ioctl_istr(fd, TIOCSETLD, "TIOCSETLD",
1130                     &lio, sizeof (lio)) < 0)
1131                         return (-errno);
1132         }

1134         l2s_termio(&l_tio, &s_tio);
1135         lx_debug("\tioctl(%d, 0x%x - %s, ...)",
1136             fd, cmd, cmd_str);
1137         ret = ioctl(fd, cmd, (intptr_t)&s_tio);
1138         return ((ret < 0) ? -errno : ret);
1139 }

1141 /*
1142  * The Solaris TIOCGPGRP ioctl does not have exactly the same semantics as
1143  * the Linux one. To mimic Linux semantics we have to do some extra work
1144  * normally done by the Solaris version of tcgetpgrp().
1145  */
1146 static int
1147 /*ARGSUSED*/
1148 ict_tiocgpgrp(int fd, struct stat *stat, int cmd, char *cmd_str, intptr_t arg)
1149 {
1150         pid_t   ttysid, mysid;
1151         int     ret;

1153         assert(cmd == LX_TIOCGPGRP);

1155         lx_debug("\tioctl(%d, 0x%x - %s, ...)",
1156             fd, TIOCGSID, "TIOCGSID");
1157         if (ioctl(fd, TIOCGSID, (intptr_t)&ttysid) < 0)
1158                 return (-errno);
1159         if ((mysid = getsid(0)) < 0)
1160                 return (-errno);
1161         if (mysid != ttysid)
1162                 return (-ENOTTY);

1164         lx_debug("\tioctl(%d, 0x%x - %s, ...)",
1165             fd, TIOCGPGRP, "TIOCGPGRP");
1166         ret = ioctl(fd, TIOCGPGRP, arg);
1167         return ((ret < 0) ? -errno : ret);
1168 }

1170 static int
1171 /*ARGSUSED*/
1172 ict_sptlock(int fd, struct stat *stat, int cmd, char *cmd_str, intptr_t arg)
1173 {
1174         assert(cmd == LX_TIOCSPTLCK);

1176         /*
1177          * The success/fail return values are different between Linux
1178          * and Solaris.   Linux expects 0 or -1.  Solaris can return
1179          * positive number on success.
1180          */
1181         if (ioctl_istr(fd, UNLKPT, "UNLKPT", NULL, 0) < 0)
1182                 return (-errno);
1183         return (0);
```

```
1184 }

1186 static int
1187 /*ARGSUSED*/
1188 ict_gptn(int fd, struct stat *stat, int cmd, char *cmd_str, intptr_t arg)
1189 {
1190         int             ptyno, *ptynop = (int *)arg;
1191         pt_own_t        pto;

1193         assert(cmd == LX_TIOCGPTN);
1194         assert(getmajor(stat->st_rdev) == ioc_translator_ptm.idt_major);

1196         /* This operation is only valid for the lx_ptm device. */
1197         ptyno = LX_PTM_DEV_TO_PTS(stat->st_rdev);

1199         /*
1200          * We'd like to just use grantpt() directly, but we can't since
1201          * it assumes the fd node that's passed to it is a ptm node,
1202          * and in our case it's an lx_ptm node.  It also relies on
1203          * naming services to get the current process group name.
1204          * Hence we have to invoke the OWNERPT ioctl directly here.
1205          */
1206         pto.pto_ruid = getuid();
1207         pto.pto_rgid = getgid();
1208         if (ioctl_istr(fd, OWNERPT, "OWNERPT", &pto, sizeof (pto)) != 0)
1209                 return (-EACCES);

1211         /* Copy out the data. */
1212         if (uucopy(&ptyno, ptynop, sizeof (ptyno)) != 0)
1213                 return (-errno);

1215         return (0);
1216 }

1218 static int
1219 /*ARGSUSED*/
1220 ict_tiocgwinsz(int fd, struct stat *stat, int cmd, char *cmd_str, intptr_t arg)
1221 {
1222         struct winsize  winsize, *winsizep = (struct winsize *)arg;

1224         assert(cmd == LX_TIOCGWINSZ);

1226         lx_debug("\tioctl(%d, 0x%x - %s, ...)", fd, TIOCGWINSZ, "TIOCGWINSZ");
1227         if (ioctl(fd, TIOCGWINSZ, arg) >= 0)
1228                 return (0);
1229         if (errno != EINVAL)
1230                 return (-errno);

1232         bzero(&winsize, sizeof (winsize));
1233         if (uucopy(&winsize, winsizep, sizeof (winsize)) != 0)
1234                 return (-errno);

1236         return (0);
1237 }

1239 static int
1240 /*ARGSUSED*/
1241 ict_tcgets_emulate(int fd, struct stat *stat,
1242     int cmd, char *cmd_str, intptr_t arg)
1243 {
1244         struct lx_termios       l_tios, *l_tiosp = (struct lx_termios *)arg;
1245         struct termios          s_tios;

1247         assert(cmd == LX_TCGETS);

1249         if (syscall(SYS_brand, B_TTYMODES, &s_tios) < 0)
```



```
1250                 return (-errno);

1252         /* Now munge the data to how Linux wants it. */
1253         s2l_termios(&s_tios, &l_tios);
1254         if (uucopy(&l_tios, l_tiosp, sizeof (l_tios)) != 0)
1255                 return (-errno);

1257         return (0);
1258 }

1260 static int
1261 /*ARGSUSED*/
1262 ict_tcgets_native(int fd, struct stat *stat,
1263     int cmd, char *cmd_str, intptr_t arg)
1264 {
1265         struct lx_termios       l_tios, *l_tiosp = (struct lx_termios *)arg;
1266         struct termios          s_tios;
1267         struct lx_cc            lio;
1268         int                     ldlinux;

1270         assert(cmd == LX_TCGETS);

1272         if ((ldlinux = ldlinux_check(fd)) < 0)
1273                 return (ldlinux);

1275         lx_debug("\tioctl(%d, 0x%x - %s, ...)",
1276             fd, TCGETS, "TCGETS");
1277         if (ioctl(fd, TCGETS, (intptr_t)&s_tios) < 0)
1278                 return (-errno);

1280         /* Now munge the data to how Linux wants it. */
1281         s2l_termios(&s_tios, &l_tios);

1283         /*
1284          * The TIOCSETLD/TIOCGETLD ioctls are only supported by the
1285          * ldlinux strmod.  So make sure the module exists on the
1286          * target stream before we invoke the ioctl.
1287          */
1288         if (ldlinux != 0) {
1289                 if (ioctl_istr(fd, TIOCGETLD, "TIOCGETLD",
1290                     &lio, sizeof (lio)) < 0)
1291                         return (-errno);

1293                 l_tios.c_cc[LX_VEOF] = lio.veof;
1294                 l_tios.c_cc[LX_VEOL] = lio.veol;
1295                 l_tios.c_cc[LX_VMIN] = lio.vmin;
1296                 l_tios.c_cc[LX_VTIME] = lio.vtime;
1297         }

1299         /* Copy out the data. */
1300         if (uucopy(&l_tios, l_tiosp, sizeof (l_tios)) != 0)
1301                 return (-errno);

1303         return (0);
1304 }

1306 static int
1307 /*ARGSUSED*/
1308 ict_tcgeta(int fd, struct stat *stat, int cmd, char *cmd_str, intptr_t arg)
1309 {
1310         struct lx_termio        l_tio, *l_tiop = (struct lx_termio *)arg;
1311         struct termio           s_tio;
1312         struct lx_cc            lio;
1313         int                     ldlinux;

1315         assert(cmd == LX_TCGETA);
```

```
1317         if ((ldlinux = ldlinux_check(fd)) < 0)
1318                 return (ldlinux);

1320         lx_debug("\tioctl(%d, 0x%x - %s, ...)",
1321             fd, TCGETA, "TCGETA");
1322         if (ioctl(fd, TCGETA, (intptr_t)&s_tio) < 0)
1323                 return (-errno);

1325         /* Now munge the data to how Linux wants it. */
1326         s2l_termio(&s_tio, &l_tio);

1328         /*
1329          * The TIOCSETLD/TIOCGETLD ioctls are only supported by the
1330          * ldlinux strmod.  So make sure the module exists on the
1331          * target stream before we invoke the ioctl.
1332          */
1333         if (ldlinux != 0) {
1334                 if (ioctl_istr(fd, TIOCGETLD, "TIOCGETLD",
1335                     &lio, sizeof (lio)) < 0)
1336                         return (-errno);

1338                 l_tio.c_cc[LX_VEOF] = lio.veof;
1339                 l_tio.c_cc[LX_VMIN] = lio.vmin;
1340                 l_tio.c_cc[LX_VTIME] = lio.vtime;
1341         }

1343         /* Copy out the data. */
1344         if (uucopy(&l_tio, l_tiop, sizeof (l_tio)) != 0)
1345                 return (-errno);

1347         return (0);
1348 }

1350 static int
1351 /*ARGSUSED*/
1352 ict_tiocsctty(int fd, struct stat *stat, int cmd, char *cmd_str, intptr_t arg)
1353 {
1354         pid_t   mysid, ttysid;

1356         if ((mysid = getsid(0)) < 0)
1357                 return (-errno);

1359         /* Check if this fd is already our ctty. */
1360         lx_debug("\tioctl(%d, 0x%x - %s, ...)",
1361             fd, TIOCGSID, "TIOCGSID");
1362         if (ioctl(fd, TIOCGSID, (intptr_t)&ttysid) >= 0)
1363                 if (mysid == ttysid)
1364                         return (0);

1366         /*
1367          * Need to make sure we're a session leader, otherwise the
1368          * TIOCSCTTY ioctl will fail.
1369          */
1370         if (mysid != getpid())
1371                 (void) setpgrp();

1373         lx_debug("\tioctl(%d, 0x%x - %s, ...)",
1374             fd, TIOCSCTTY, "TIOCSCTTY");
1375         if (ioctl(fd, TIOCSCTTY, 0) < 0)
1376                 return (-errno);
1377         return (0);
1378 }

1380 /*
1381  * /dev/dsp ioctl translators and support
```



```
1382  */
1383 static int
1384 i_is_dsp_dev(int fd)
1385 {
1386         int minor;

1388         /*
1389          * This is a cloning device so we have to ask the driver
1390          * what kind of minor node this is.
1391          */
1392         lx_debug("\tioctl(%d, 0x%x - %s, ...)",
1393             fd, LXA_IOC_GETMINORNUM, "LXA_IOC_GETMINORNUM");
1394         if (ioctl(fd, LXA_IOC_GETMINORNUM, &minor) < 0)
1395                 return (-EINVAL);
1396         if (minor != LXA_MINORNUM_DSP)
1397                 return (-EINVAL);
1398         return (0);
1399 }

1401 static int
1402 /*ARGSUSED*/
1403 ict_oss_sndctl_dsp_reset(int fd, struct stat *stat,
1404     int cmd, char *cmd_str, intptr_t arg)
1405 {
1406         int err;

1408         /* Ioctl is only supported on dsp audio devices. */
1409         if ((err = i_is_dsp_dev(fd)) != 0)
1410                 return (err);

1412         /* Nothing to really do on Solaris. */
1413         return (0);
1414 }

1416 static void
1417 i_oss_fmt_str(char *buf, int buf_size, uint_t mask)
1418 {
1419         int i, first = 1;

1421         assert(buf != NULL);

1423         buf[0] = '\0';
1424         for (i = 0; oss_fmt_str[i].i2s_str != NULL; i++) {
1425                 if ((oss_fmt_str[i].i2s_int != mask) &&
1426                     ((oss_fmt_str[i].i2s_int & mask) == 0))
1427                         continue;
1428                 if (first)
1429                         first = 0;
1430                 else
1431                         (void) strlcat(buf, " | ", buf_size);
1432                 (void) strlcat(buf, oss_fmt_str[i].i2s_str, buf_size);
1433         }
1434 }

1436 static int
1437 /*ARGSUSED*/
1438 ict_oss_sndctl_dsp_getfmts(int fd, struct stat *stat,
1439     int cmd, char *cmd_str, intptr_t arg)
1440 {
1441         audio_info_t    sa_info;
1442         char            buf[MSGBUF];
1443         uint_t          *maskp = (uint_t *)arg;
1444         uint_t          mask = 0;
1445         int             i, amode, err;

1447         assert(cmd == LX_OSS_SNDCTL_DSP_GETFMTS);
```

```
1449         /* Ioctl is only supported on dsp audio devices. */
1450         if ((err = i_is_dsp_dev(fd)) != 0)
1451                 return (err);
1452
1453         /* We need to know the access mode for the file. */
1454         if ((amode = fcntl(fd, F_GETFL)) < 0)
1455                 return (-EINVAL);
1456         amode &= O_ACCMODE;
1457         assert((amode == O_RDONLY) || (amode == O_WRONLY) || (amode == O_RDWR));
1458
1459         /* Test to see what Linux oss formats the target device supports. */
1460         for (i = 0; oft_table[i].oft_oss_fmt != 0; i++) {
1461
1462                 /* Initialize the mode request. */
1463                 AUDIO_INITINFO(&sa_info);
1464
1465                 /* Translate a Linux oss format into Solaris settings. */
1466                 if ((amode == O_RDONLY) || (amode == O_RDWR)) {
1467                         sa_info.record.encoding = oft_table[i].oft_encoding;
1468                         sa_info.record.precision = oft_table[i].oft_precision;
1469                 }
1470                 if ((amode == O_WRONLY) || (amode == O_RDWR)) {
1471                         sa_info.play.encoding = oft_table[i].oft_encoding;
1472                         sa_info.play.precision = oft_table[i].oft_precision;
1473                 }
1474
1475                 /* Send the request. */
1476                 lx_debug("\tioctl(%d, 0x%x - %s, ...)",
1477                     fd, AUDIO_SETINFO, "AUDIO_SETINFO");
1478                 if (ioctl(fd, AUDIO_SETINFO, &sa_info) < 0)
1479                         continue;
1480
1481                 /* This Linux oss format is supported. */
1482                 mask |= oft_table[i].oft_oss_fmt;
1483         }
1484
1485         if (lx_debug_enabled != 0) {
1486                 i_oss_fmt_str(buf, sizeof (buf), mask);
1487                 lx_debug("\toss formats supported = 0x%x (%s)", mask, buf);
1488         }
1489         if (uucopy(&mask, maskp, sizeof (mask)) != 0)
1490                 return (-errno);
1491         return (0);
1492 }
1493
1494 static int
1495 /*ARGSUSED*/
1496 ict_oss_sndctl_dsp_setfmts(int fd, struct stat *stat,
1497     int cmd, char *cmd_str, intptr_t arg)
1498 {
1499         audio_info_t    sa_info;
1500         char            buf[MSGBUF];
1501         uint_t          *maskp = (uint_t *)arg;
1502         uint_t          mask;
1503         int             i, amode, err;
1504
1505         assert(cmd == LX_OSS_SNDCTL_DSP_SETFMTS);
1506
1507         /* Ioctl is only supported on dsp audio devices. */
1508         if ((err = i_is_dsp_dev(fd)) != 0)
1509                 return (err);
1510
1511         if (uucopy(maskp, &mask, sizeof (mask)) != 0)
1512                 return (-errno);
```



```
1513
1514         if (lx_debug_enabled != 0) {
1515                 i_oss_fmt_str(buf, sizeof (buf), mask);
1516                 lx_debug("\toss formats request = 0x%x (%s)", mask, buf);
1517         }
1518
1519         if ((mask == (uint_t)-1) || (mask == 0)) {
1520                 lx_debug("\tXXX: possible oss formats query?");
1521                 return (-EINVAL);
1522         }
1523
1524         /* Check if multiple format bits were specified. */
1525         if (!BIT_ONLYONESET(mask))
1526                 return (-EINVAL);
1527
1528         /* Decode the oss format request into a native format. */
1529         for (i = 0; oft_table[i].oft_oss_fmt != 0; i++) {
1530                 if (oft_table[i].oft_oss_fmt == mask)
1531                         break;
1532         }
1533         if (oft_table[i].oft_oss_fmt == 0)
1534                 return (-EINVAL);
1535
1536         /* We need to know the access mode for the file. */
1537         if ((amode = fcntl(fd, F_GETFL)) < 0)
1538                 return (-EINVAL);
1539         amode &= O_ACCMODE;
1540         assert((amode == O_RDONLY) || (amode == O_WRONLY) || (amode == O_RDWR));
1541
1542         /* Initialize the mode request. */
1543         AUDIO_INITINFO(&sa_info);
1544
1545         /* Translate the Linux oss request into a Solaris request. */
1546         if ((amode == O_RDONLY) || (amode == O_RDWR)) {
1547                 sa_info.record.encoding = oft_table[i].oft_encoding;
1548                 sa_info.record.precision = oft_table[i].oft_precision;
1549         }
1550         if ((amode == O_WRONLY) || (amode == O_RDWR)) {
1551                 sa_info.play.encoding = oft_table[i].oft_encoding;
1552                 sa_info.play.precision = oft_table[i].oft_precision;
1553         }
1554
1555         /* Send the request. */
1556         lx_debug("\tioctl(%d, 0x%x - %s, ...)",
1557             fd, AUDIO_SETINFO, "AUDIO_SETINFO");
1558         return ((ioctl(fd, AUDIO_SETINFO, &sa_info) < 0) ? -errno : 0);
1559 }
1560
1561 static int
1562 /*ARGSUSED*/
1563 ict_oss_sndctl_dsp_channels(int fd, struct stat *stat,
1564     int cmd, char *cmd_str, intptr_t arg)
1565 {
1566         audio_info_t    sa_info;
1567         uint_t          *channelsp = (uint_t *)arg;
1568         uint_t          channels;
1569         int             amode, err;
1570
1571         assert((cmd == LX_OSS_SNDCTL_DSP_CHANNELS) ||
1572             (cmd == LX_OSS_SNDCTL_DSP_STEREO));
1573
1574         /* Ioctl is only supported on dsp audio devices. */
1575         if ((err = i_is_dsp_dev(fd)) != 0)
1576                 return (err);
1577
1578         if (uucopy(channelsp, &channels, sizeof (channels)) != 0)
1579                 return (-errno);
```

```
1581         lx_debug("\toss %s request = 0x%x (%u)",
1582             (cmd == LX_OSS_SNDCTL_DSP_CHANNELS) ? "channel" : "stereo",
1583             channels, channels);

1585         if (channels == (uint_t)-1) {
1586                 lx_debug("\tXXX: possible channel/stereo query?");
1587                 return (-EINVAL);
1588         }

1590         if (cmd == LX_OSS_SNDCTL_DSP_STEREO) {
1591                 /*
1592                  * There doesn't seem to be any documentation for
1593                  * SNDCTL_DSP_STEREO.  Looking at source that uses or
1594                  * used this ioctl seems to indicate that the
1595                  * functionality provided by this ioctl has been
1596                  * subsumed by the SNDCTL_DSP_CHANNELS ioctl.  It
1597                  * seems that the only arguments ever passed to
1598                  * the SNDCTL_DSP_STEREO.  Ioctl are boolean values
1599                  * of '0' or '1'.  Hence we'll start out strict and
1600                  * only support those values.
1601                  *
1602                  * Some online forum discussions about this ioctl
1603                  * seemed to indicate that in case of success it
1604                  * returns the "stereo" setting (ie, either
1605                  * '0' for mono or '1' for stereo).
1606                  */
1607                 if ((channels != 0) && (channels != 1)) {
1608                         lx_debug("\tinvalid stereo request");
1609                         return (-EINVAL);
1610                 }
1611                 channels += 1;
1612         } else {
1613                 /* Limit the system to one or two channels. */
1614                 if ((channels != 1) && (channels != 2)) {
1615                         lx_debug("\tinvalid channel request");
1616                         return (-EINVAL);
1617                 }
1618         }

1620         /* We need to know the access mode for the file. */
1621         if ((amode = fcntl(fd, F_GETFL)) < 0)
1622                 return (-EINVAL);
1623         amode &= O_ACCMODE;
1624         assert((amode == O_RDONLY) || (amode == O_WRONLY) || (amode == O_RDWR));

1626         /* Initialize the channel request. */
1627         AUDIO_INITINFO(&sa_info);

1629         /* Translate the Linux oss request into a Solaris request. */
1630         if ((amode == O_RDONLY) || (amode == O_RDWR))
1631                 sa_info.record.channels = channels;
1632         if ((amode == O_WRONLY) || (amode == O_RDWR))
1633                 sa_info.play.channels = channels;

1635         /* Send the request. */
1636         lx_debug("\tioctl(%d, 0x%x - %s, ...)",
1637             fd, AUDIO_SETINFO, "AUDIO_SETINFO");
1638         if (ioctl(fd, AUDIO_SETINFO, &sa_info) < 0)
1639                 return (-errno);

1641         if (cmd == LX_OSS_SNDCTL_DSP_STEREO)
1642                 return (channels - 1);
1643         return (0);
1644 }
```



```
1646 static int
1647 /*ARGSUSED*/
1648 ict_oss_sndctl_dsp_speed(int fd, struct stat *stat,
1649     int cmd, char *cmd_str, intptr_t arg)
1650 {
1651         audio_info_t    sa_info;
1652         uint_t          *speedp = (uint_t *)arg;
1653         uint_t          speed;
1654         int             amode, err;

1656         assert(cmd == LX_OSS_SNDCTL_DSP_SPEED);

1658         /* Ioctl is only supported on dsp audio devices. */
1659         if ((err = i_is_dsp_dev(fd)) != 0)
1660                 return (err);

1662         if (uucopy(speedp, &speed, sizeof (speed)) != 0)
1663                 return (-errno);

1665         lx_debug("\toss speed request = 0x%x (%u)", speed, speed);

1667         if (speed == (uint_t)-1) {
1668                 lx_debug("\tXXX: possible oss speed query?");
1669                 return (-EINVAL);
1670         }

1672         /* We need to know the access mode for the file. */
1673         if ((amode = fcntl(fd, F_GETFL)) < 0)
1674                 return (-EINVAL);
1675         amode &= O_ACCMODE;
1676         assert((amode == O_RDONLY) || (amode == O_WRONLY) || (amode == O_RDWR));

1678         /* Initialize the speed request. */
1679         AUDIO_INITINFO(&sa_info);

1681         /* Translate the Linux oss request into a Solaris request. */
1682         if ((amode == O_RDONLY) || (amode == O_RDWR))
1683                 sa_info.record.sample_rate = speed;
1684         if ((amode == O_WRONLY) || (amode == O_RDWR))
1685                 sa_info.play.sample_rate = speed;

1687         /* Send the request. */
1688         lx_debug("\tioctl(%d, 0x%x - %s, ...)",
1689             fd, AUDIO_SETINFO, "AUDIO_SETINFO");
1690         return ((ioctl(fd, AUDIO_SETINFO, &sa_info) < 0) ? -errno : 0);
1691 }

1693 static int
1694 /*ARGSUSED*/
1695 ict_oss_sndctl_dsp_getblksize(int fd, struct stat *stat,
1696     int cmd, char *cmd_str, intptr_t arg)
1697 {
1698         lxa_frag_info_t fi;
1699         uint_t          *blksizep = (uint_t *)arg;
1700         uint_t          blksize;
1701         int             err;

1703         assert(cmd == LX_OSS_SNDCTL_DSP_GETBLKSIZE);

1705         /* Ioctl is only supported on dsp audio devices. */
1706         if ((err = i_is_dsp_dev(fd)) != 0)
1707                 return (err);

1709         /* Query the current fragment count and size. */
1710         lx_debug("\tioctl(%d, 0x%x - %s, ...)",
1711             fd, LXA_IOC_GET_FRAG_INFO, "LXA_IOC_GET_FRAG_INFO");
```

```
1712         if (ioctl(fd, LXA_IOC_GET_FRAG_INFO, &fi) < 0)
1713                 return (-errno);

1715         blksize = fi.lxa_fi_size;

1717         if (uucopy(&blksize, blksizep, sizeof (blksize)) != 0)
1718                 return (-errno);
1719         return (0);
1720 }

1722 static int
1723 /*ARGSUSED*/
1724 ict_oss_sndctl_dsp_getspace(int fd, struct stat *stat,
1725     int cmd, char *cmd_str, intptr_t arg)
1726 {
1727         lx_oss_audio_buf_info_t *spacep = (lx_oss_audio_buf_info_t *)arg;
1728         lx_oss_audio_buf_info_t space;
1729         lxa_frag_info_t         fi;
1730         int                     err;

1732         assert((cmd == LX_OSS_SNDCTL_DSP_GETOSPACE) ||
1733             (cmd == LX_OSS_SNDCTL_DSP_GETISPACE));

1735         /* Ioctl is only supported on dsp audio devices. */
1736         if ((err = i_is_dsp_dev(fd)) != 0)
1737                 return (err);

1739         /* Query the current fragment count and size. */
1740         lx_debug("\tioctl(%d, 0x%x - %s, ...)",
1741             fd, LXA_IOC_GET_FRAG_INFO, "LXA_IOC_GET_FRAG_INFO");
1742         if (ioctl(fd, LXA_IOC_GET_FRAG_INFO, &fi) < 0)
1743                 return (-errno);

1745         /* Return the current fragment count and size. */
1746         space.fragstotal = fi.lxa_fi_cnt;
1747         space.fragsize = fi.lxa_fi_size;

1749         /*
1750          * We'll lie and tell applications that they can always write
1751          * out at least one fragment without blocking.
1752          */
1753         space.fragments = 1;
1754         space.bytes = space.fragsize;

1756         if (cmd == LX_OSS_SNDCTL_DSP_GETOSPACE)
1757                 lx_debug("\toss get output space result = ");
1758         if (cmd == LX_OSS_SNDCTL_DSP_GETISPACE)
1759                 lx_debug("\toss get input space result = ");

1761         lx_debug("\t\tbytes = 0x%x (%u), fragments = 0x%x (%u)",
1762             space.bytes, space.bytes, space.fragments, space.fragments);
1763         lx_debug("\t\tfragtotal = 0x%x (%u), fragsize = 0x%x (%u)",
1764             space.fragstotal, space.fragstotal,
1765             space.fragsize, space.fragsize);

1767         if (uucopy(&space, spacep, sizeof (space)) != 0)
1768                 return (-errno);
1769         return (0);
1770 }

1772 static int
1773 /*ARGSUSED*/
1774 ict_oss_sndctl_dsp_setfragment(int fd, struct stat *stat,
1775     int cmd, char *cmd_str, intptr_t arg)
1776 {
1777         lxa_frag_info_t fi;
```



```
1778         uint_t          *fraginfop = (uint_t *)arg;
1779         uint_t          fraginfo, frag_size, frag_cnt;
1780         int             err;

1782         assert(cmd == LX_OSS_SNDCTL_DSP_SETFRAGMENT);

1784         /* Ioctl is only supported on dsp audio devices. */
1785         if ((err = i_is_dsp_dev(fd)) != 0)
1786                 return (err);

1788         if (uucopy(fraginfop, &fraginfo, sizeof (fraginfo)) != 0)
1789                 return (-errno);

1791         /*
1792          * The argument to this ioctl is a 32-bit integer of the
1793          * format 0x MMMM SSSS where:
1794          *      SSSS - requests a fragment size of 2^SSSS
1795          *      MMMM - requests a maximum fragment count of 2^MMMM
1796          * if MMMM is 0x7fff then the application is requesting
1797          * no limits on the number of fragments.
1798          */

1800         frag_size = fraginfo & 0xffff;
1801         frag_cnt = fraginfo >> 16;

1803         lx_debug("\toss fragment request: "
1804             "power size = 0x%x (%u), power cnt = 0x%x (%u)",
1805             frag_size, frag_size, frag_cnt, frag_cnt);

1807         /* Limit the supported fragment size from 2^4 to 2^31. */
1808         if ((frag_size < 4) || (frag_size > 31))
1809                 return (-EINVAL);

1811         /* Limit the number of fragments from 2^1 to 2^32. */
1812         if (((frag_cnt < 1) || (frag_cnt > 32)) && (frag_cnt != 0x7fff))
1813                 return (-EINVAL);

1815         /* Expand the fragment values. */
1816         frag_size = 1 << frag_size;
1817         if ((frag_cnt == 32) || (frag_cnt == 0x7fff)) {
1818                 frag_cnt = UINT_MAX;
1819         } else {
1820                 frag_cnt = 1 << frag_cnt;
1821         }

1823         lx_debug("\toss fragment request: "
1824             "translated size = 0x%x (%u), translated cnt = 0x%x (%u)",
1825             frag_size, frag_size, frag_cnt, frag_cnt);

1827         fi.lxa_fi_size = frag_size;
1828         fi.lxa_fi_cnt = frag_cnt;

1830         /* Set the current fragment count and size. */
1831         lx_debug("\tioctl(%d, 0x%x - %s, ...)",
1832             fd, LXA_IOC_SET_FRAG_INFO, "LXA_IOC_SET_FRAG_INFO");
1833         return ((ioctl(fd, LXA_IOC_SET_FRAG_INFO, &fi) < 0) ? -errno : 0);
1834 }

1836 static int
1837 /*ARGSUSED*/
1838 ict_oss_sndctl_dsp_getcaps(int fd, struct stat *stat,
1839     int cmd, char *cmd_str, intptr_t arg)
1840 {
1841         uint_t  *capsp = (uint_t *)arg;
1842         uint_t  caps;
1843         int     err;
```

```
1845         assert(cmd == LX_OSS_SNDCTL_DSP_GETCAPS);

1847         /* Ioctl is only supported on dsp audio devices. */
1848         if ((err = i_is_dsp_dev(fd)) != 0)
1849                 return (err);

1851         /*
1852          * Report that we support mmap access
1853          * this is where things start to get fun.
1854          */
1855         caps = LX_OSS_DSP_CAP_MMAP | LX_OSS_DSP_CAP_TRIGGER;

1857         if (uucopy(&caps, capsp, sizeof (caps)) != 0)
1858                 return (-errno);
1859         return (0);
1860 }

1862 static int
1863 /*ARGSUSED*/
1864 ict_oss_sndctl_dsp_settrigger(int fd, struct stat *stat,
1865     int cmd, char *cmd_str, intptr_t arg)
1866 {
1867         uint_t          *triggerp = (uint_t *)arg;
1868         uint_t          trigger;
1869         int             err;

1871         assert(cmd == LX_OSS_SNDCTL_DSP_SETTRIGGER);

1873         /* Ioctl is only supported on dsp audio devices. */
1874         if ((err = i_is_dsp_dev(fd)) != 0)
1875                 return (err);

1877         if (uucopy(triggerp, &trigger, sizeof (trigger)) != 0)
1878                 return (-errno);

1880         lx_debug("\toss set trigger request = 0x%x (%u)",
1881             trigger, trigger);

1883         /* We only support two types of trigger requests. */
1884         if ((trigger != LX_OSS_PCM_DISABLE_OUTPUT) &&
1885             (trigger != LX_OSS_PCM_ENABLE_OUTPUT))
1886                 return (-EINVAL);

1888         /*
1889          * We only support triggers on devices open for write access,
1890          * but we don't need to check for that here since the driver will
1891          * verify this for us.
1892          */

1894         /* Send the trigger command to the audio device. */
1895         lx_debug("\tioctl(%d, 0x%x - %s, ...)",
1896             fd, LXA_IOC_MMAP_OUTPUT, "LXA_IOC_MMAP_OUTPUT");
1897         return ((ioctl(fd, LXA_IOC_MMAP_OUTPUT, &trigger) < 0) ? -errno : 0);
1898 }

1900 static int
1901 /*ARGSUSED*/
1902 ict_oss_sndctl_dsp_getoptr(int fd, struct stat *stat,
1903     int cmd, char *cmd_str, intptr_t arg)
1904 {
1905         static uint_t           bytes = 0;
1906         lx_oss_count_info_t     ci;
1907         lxa_frag_info_t         fi;
1908         audio_info_t            ai;
1909         int                     ptr, err;
```



```
1911         assert(cmd == LX_OSS_SNDCTL_DSP_GETOPTR);

1913         /* Ioctl is only supported on dsp audio devices. */
1914         if ((err = i_is_dsp_dev(fd)) != 0)
1915                 return (err);

1917         /* Query the current fragment size. */
1918         lx_debug("\tioctl(%d, 0x%x - %s, ...)",
1919             fd, LXA_IOC_GET_FRAG_INFO, "LXA_IOC_GET_FRAG_INFO");
1920         if (ioctl(fd, LXA_IOC_GET_FRAG_INFO, &fi) < 0)
1921                 return (-errno);

1923         /* Figure out how many samples have been played. */
1924         lx_debug("\tioctl(%d, 0x%x - %s, ...)",
1925             fd, AUDIO_GETINFO, "AUDIO_GETINFO");
1926         if (ioctl(fd, AUDIO_GETINFO, &ai) < 0)
1927                 return (-errno);
1928         ci.bytes = ai.play.samples + ai.record.samples;

1930         /*
1931          * Figure out how many fragments of audio have gone out since
1932          * the last call to this ioctl.
1933          */
1934         ci.blocks = (ci.bytes - bytes) / fi.lxa_fi_size;
1935         bytes = ci.bytes;

1937         /* Figure out the current fragment offset for mmap audio output. */
1938         lx_debug("\tioctl(%d, 0x%x - %s, ...)",
1939             fd, LXA_IOC_MMAP_PTR, "LXA_IOC_MMAP_PTR");
1940         if (ioctl(fd, LXA_IOC_MMAP_PTR, &ptr) < 0) {
1941                 /*
1942                  * We really should return an error here, but some
1943                  * application (*cough* *cough* flash) expect this
1944                  * ioctl to work even if they haven't mmaped the
1945                  * device.
1946                  */
1947                 ci.ptr = 0;
1948         } else {
1949                 ci.ptr = ptr;
1950         }

1952         lx_debug("\toss get output ptr result = ");
1953         lx_debug("\t\t"
1954             "bytes = 0x%x (%u), blocks = 0x%x (%u), ptr = 0x%x (%u)",
1955             ci.bytes, ci.bytes, ci.blocks, ci.blocks, ci.ptr, ci.ptr);

1957         if (uucopy(&ci, (void *)arg, sizeof (ci)) != 0)
1958                 return (-errno);
1959         return (0);
1960 }

1962 static int
1963 /*ARGSUSED*/
1964 ict_oss_sndctl_dsp_sync(int fd, struct stat *stat,
1965     int cmd, char *cmd_str, intptr_t arg)
1966 {
1967         int             amode, err;

1969         assert(cmd == LX_OSS_SNDCTL_DSP_SYNC);

1971         /* Ioctl is only supported on dsp audio devices. */
1972         if ((err = i_is_dsp_dev(fd)) != 0)
1973                 return (err);

1975         /* We need to know the access mode for the file. */
```

```
1976         if ((amode = fcntl(fd, F_GETFL)) < 0)
1977                 return (-EINVAL);
1978         amode &= O_ACCMODE;
1979         assert((amode == O_RDONLY) || (amode == O_WRONLY) || (amode == O_RDWR));

1981         /*
1982          * A sync is basically a noop for record only device.
1983          * We check for this here because on Linux a sync on a record
1984          * only device returns success immediately.  But the Solaris
1985          * equivalent to a drain operation is a AUDIO_DRAIN, and if
1986          * it's issued to a record only device it will fail and return
1987          * EINVAL.
1988          */
1989         if (amode == O_RDONLY)
1990                 return (0);

1992         /* Drain any pending output. */
1993         lx_debug("\tioctl(%d, 0x%x - %s, ...)",
1994             fd, AUDIO_DRAIN, "AUDIO_DRAIN");
1995         return ((ioctl(fd, AUDIO_DRAIN, NULL) < 0) ? -errno : 0);
1996 }

1998 /*
1999  * /dev/mixer ioctl translators and support
2000  *
2001  * There are some interesting things to take note of for supporting
2002  * /dev/mixer ioctls.
2003  *
2004  * 1) We report support for the following mixer resources:
2005  *      VOLUME, PCM, MIC
2006  *
2007  * 2) We assume the following number of channels for each mixer resource:
2008  *      VOLUME: 2 channels
2009  *      PCM:    2 channels
2010  *      MIC:    1 channel
2011  *
2012  * 3) OSS sets the gain on each channel independently but on Solaris
2013  *    there is only one gain value and a balance value.  So we need
2014  *    to do some translation back and forth.
2015  *
2016  * 4) OSS assumes direct access to hardware but Solaris provides
2017  *    virtualized audio device access (where everyone who opens /dev/audio
2018  *    get a virtualized audio channel stream, all of which are merged
2019  *    together by a software mixer before reaching the hardware).  Hence
2020  *    mapping OSS mixer resources to Solaris mixer resources takes some
2021  *    work.  VOLUME and Mic resources are mapped to the actual underlying
2022  *    audio hardware resources.  PCM resource are mapped to the virtual
2023  *    audio channel output level.  This mapping becomes more complicated
2024  *    if there are no open audio output channels.  In this case the
2025  *    lx_audio device caches the PCM channels setting for us and applies
2026  *    them to any new audio output channels that get opened.  (This
2027  *    is the reason that we don't use AUDIO_SETINFO ioctls directly
2028  *    but instead the lx_audio driver custom LXA_IOC_MIXER_SET_*
2029  *    and LXA_IOC_MIXER_GET_* ioctls.)  For more information see
2030  *    the comments in lx_audio.c.
2031  */
2032 static int
2033 i_is_mixer_dev(int fd)
2034 {
2035         int minor;

2037         /*
2038          * This is a cloning device so we have to ask the driver
2039          * what kind of minor node this is.
2040          */
2041         lx_debug("\tioctl(%d, 0x%x - %s, ...)",
```



```
2042             fd, LXA_IOC_GETMINORNUM, "LXA_IOC_GETMINORNUM");
2043         if (ioctl(fd, LXA_IOC_GETMINORNUM, &minor) < 0)
2044                 return (-EINVAL);
2045         if (minor != LXA_MINORNUM_MIXER)
2046                 return (-EINVAL);
2047         return (0);
2048 }

2050 static int
2051 i_oss_mixer_ml_to_val(lxa_mixer_levels_t *ml, uint_t *val)
2052 {
2053         int range, val1, val2;

2055         /* Deal with the other easy case, both channels have the same level. */
2056         if (ml->lxa_ml_balance == AUDIO_MID_BALANCE) {
2057                 *val = LX_OSS_MIXER_ENC2(
2058                     LX_OSS_S2L_GAIN(ml->lxa_ml_gain),
2059                     LX_OSS_S2L_GAIN(ml->lxa_ml_gain));
2060                 assert(LX_OSS_MIXER_2CH_OK(*val));
2061                 return (0);
2062         }

2064         /* Decode the balance/gain into two separate levels. */
2065         if (ml->lxa_ml_balance > AUDIO_MID_BALANCE) {
2066                 val2 = ml->lxa_ml_gain;

2068                 range = AUDIO_RIGHT_BALANCE - AUDIO_MID_BALANCE;
2069                 val1 = AUDIO_RIGHT_BALANCE - ml->lxa_ml_balance;
2070                 val1 = (val2 * val1) / range;
2071         } else {
2072                 assert(ml->lxa_ml_balance < AUDIO_MID_BALANCE);
2073                 val1 = ml->lxa_ml_gain;

2075                 range = AUDIO_MID_BALANCE - AUDIO_LEFT_BALANCE;
2076                 val2 = ml->lxa_ml_balance;
2077                 val2 = (val1 * val2) / range;
2078         }

2080         *val = LX_OSS_MIXER_ENC2(LX_OSS_S2L_GAIN(val1),
2081             LX_OSS_S2L_GAIN(val2));
2082         return (0);
2083 }

2085 static int
2086 i_oss_mixer_val_to_ml(uint_t val, lxa_mixer_levels_t *ml_old,
2087     lxa_mixer_levels_t *ml)
2088 {
2089         int range, val1, val2;

2091         if (!LX_OSS_MIXER_2CH_OK(val))
2092                 return (-EINVAL);

2094         val1 = LX_OSS_MIXER_DEC1(val);
2095         val2 = LX_OSS_MIXER_DEC2(val);

2097         /*
2098          * Deal with the easy case.
2099          * Both channels have the same non-zero level.
2100          */
2101         if ((val1 != 0) && (val1 == val2)) {
2102                 ml->lxa_ml_gain = LX_OSS_L2S_GAIN(val1);
2103                 ml->lxa_ml_balance = AUDIO_MID_BALANCE;
2104                 return (0);
2105         }

2107         /* If both levels are zero, preserve the current balance setting. */
```

```
2108         if ((val1 == 0) && (val2 == 0)) {
2109                 ml->lxa_ml_gain = 0;
2110                 ml->lxa_ml_balance = ml_old->lxa_ml_balance;
2111                 return (0);
2112         }

2114         /*
2115          * First set the gain to match the highest channel value volume.
2116          * Then use the balance to simulate lower volume on the second
2117          * channel.
2118          */
2119         if (val1 > val2) {
2120                 ml->lxa_ml_gain = LX_OSS_L2S_GAIN(val1);

2122                 range = AUDIO_MID_BALANCE - AUDIO_LEFT_BALANCE;
2123                 ml->lxa_ml_balance = 0;
2124                 ml->lxa_ml_balance += ((val2 * range) / val1);
2125         } else {
2126                 assert(val1 < val2);

2128                 ml->lxa_ml_gain = LX_OSS_L2S_GAIN(val2);

2130                 range = AUDIO_RIGHT_BALANCE - AUDIO_MID_BALANCE;
2131                 ml->lxa_ml_balance = AUDIO_RIGHT_BALANCE;
2132                 ml->lxa_ml_balance -= ((val1 * range) / val2);
2133         }

2135         return (0);
2136 }

2138 static int
2139 /*ARGSUSED*/
2140 ict_oss_mixer_read_volume(int fd, struct stat *stat,
2141     int cmd, char *cmd_str, intptr_t arg)
2142 {
2143         lxa_mixer_levels_t      ml;
2144         uint_t                  *valp = (uint_t *)arg;
2145         uint_t                  val;
2146         char                    *cmd_txt;
2147         int                     err, cmd_new;

2149         assert((cmd == LX_OSS_SOUND_MIXER_READ_VOLUME) ||
2150             (cmd == LX_OSS_SOUND_MIXER_READ_PCM));

2152         /* Ioctl is only supported on mixer audio devices. */
2153         if ((err = i_is_mixer_dev(fd)) != 0)
2154                 return (err);

2156         if (cmd == LX_OSS_SOUND_MIXER_READ_VOLUME) {
2157                 cmd_new = LXA_IOC_MIXER_GET_VOL;
2158                 cmd_txt = "LXA_IOC_MIXER_GET_VOL";
2159         }
2160         if (cmd == LX_OSS_SOUND_MIXER_READ_PCM) {
2161                 cmd_new = LXA_IOC_MIXER_GET_PCM;
2162                 cmd_txt = "LXA_IOC_MIXER_GET_PCM";
2163         }

2165         /* Attempt to set the device output gain. */
2166         lx_debug("\tioctl(%d, 0x%x - %s, ...)", fd, cmd_new, cmd_txt);
2167         if (ioctl(fd, cmd_new, &ml) < 0)
2168                 return (-errno);

2170         lx_debug("\tlx_audio mixer results, "
2171             "gain = 0x%x (%u), balance = 0x%x (%u)",
2172             ml.lxa_ml_gain, ml.lxa_ml_gain,
2173             ml.lxa_ml_balance, ml.lxa_ml_balance);
```



```
2175         assert(LXA_MIXER_LEVELS_OK(&ml));

2177         /* Translate the mixer levels struct to an OSS mixer value. */
2178         if ((err = i_oss_mixer_ml_to_val(&ml, &val)) != 0)
2179                 return (err);
2180         assert(LX_OSS_MIXER_2CH_OK(val));

2182         lx_debug("\toss get mixer %s result = 0x%x (%u)",
2183             (cmd == LX_OSS_SOUND_MIXER_READ_VOLUME) ? "volume" : "pcm",
2184             val, val);

2186         if (uucopy(&val, valp, sizeof (val)) != 0)
2187                 return (-errno);
2188         return (0);
2189 }

2191 static int
2192 /*ARGSUSED*/
2193 ict_oss_mixer_write_volume(int fd, struct stat *stat,
2194     int cmd, char *cmd_str, intptr_t arg)
2195 {
2196         lxa_mixer_levels_t      ml, ml_old;
2197         uint_t                  *valp = (uint_t *)arg;
2198         uint_t                  val;
2199         char                    *cmd_txt;
2200         int                     err, cmd_new;

2202         assert((cmd == LX_OSS_SOUND_MIXER_WRITE_VOLUME) ||
2203             (cmd == LX_OSS_SOUND_MIXER_WRITE_PCM));

2205         /* Ioctl is only supported on mixer audio devices. */
2206         if ((err = i_is_mixer_dev(fd)) != 0)
2207                 return (err);

2209         if (uucopy(valp, &val, sizeof (val)) != 0)
2210                 return (-errno);

2212         if (cmd == LX_OSS_SOUND_MIXER_WRITE_VOLUME) {
2213                 cmd_new = LXA_IOC_MIXER_SET_VOL;
2214                 cmd_txt = "LXA_IOC_MIXER_SET_VOL";

2216                 /* Attempt to get the device output gain. */
2217                 lx_debug("\tioctl(%d, 0x%x - %s, ...)", fd,
2218                     LXA_IOC_MIXER_GET_VOL, "LXA_IOC_MIXER_GET_VOL");
2219                 if (ioctl(fd, LXA_IOC_MIXER_GET_VOL, &ml_old) < 0)
2220                         return (-errno);
2221         }

2223         if (cmd == LX_OSS_SOUND_MIXER_WRITE_PCM) {
2224                 cmd_new = LXA_IOC_MIXER_SET_PCM;
2225                 cmd_txt = "LXA_IOC_MIXER_SET_PCM";

2227                 /* Attempt to get the device output gain. */
2228                 lx_debug("\tioctl(%d, 0x%x - %s, ...)", fd,
2229                     LXA_IOC_MIXER_GET_PCM, "LXA_IOC_MIXER_GET_PCM");
2230                 if (ioctl(fd, LXA_IOC_MIXER_GET_PCM, &ml_old) < 0)
2231                         return (-errno);
2232         }

2234         lx_debug("\toss set mixer %s request = 0x%x (%u)",
2235             (cmd == LX_OSS_SOUND_MIXER_WRITE_VOLUME) ? "volume" : "pcm",
2236             val, val);

2238         /* Translate an OSS mixer value to mixer levels. */
2239         if ((err = i_oss_mixer_val_to_ml(val, &ml_old, &ml)) != 0)
```

```
2240                return (err);
2241        assert(LXA_MIXER_LEVELS_OK(&ml));

2243        lx_debug("\tlx_audio mixer request, "
2244            "gain = 0x%x (%u), balance = 0x%x (%u)",
2245            ml.lxa_ml_gain, ml.lxa_ml_gain,
2246            ml.lxa_ml_balance, ml.lxa_ml_balance);

2248        /* Attempt to set the device output gain. */
2249        lx_debug("\tioctl(%d, 0x%x - %s, ...)", fd, cmd_new, cmd_txt);
2250        if (ioctl(fd, cmd_new, &ml) < 0)
2251                return (-errno);

2253        return (0);
2254 }

2256 static int
2257 /*ARGSUSED*/
2258 ict_oss_mixer_read_mic(int fd, struct stat *stat,
2259     int cmd, char *cmd_str, intptr_t arg)
2260 {
2261        lxa_mixer_levels_t      ml;
2262        uint_t                  *valp = (uint_t *)arg;
2263        uint_t                  val;
2264        int                     err;

2266        assert((cmd == LX_OSS_SOUND_MIXER_READ_MIC) ||
2267            (cmd == LX_OSS_SOUND_MIXER_READ_IGAIN));

2269        /* Ioctl is only supported on mixer audio devices. */
2270        if ((err = i_is_mixer_dev(fd)) != 0)
2271                return (err);

2273        /* Attempt to get the device input gain. */
2274        lx_debug("\tioctl(%d, 0x%x - %s, ...)",
2275            fd, LXA_IOC_MIXER_GET_MIC, "LXA_IOC_MIXER_GET_MIC");
2276        if (ioctl(fd, LXA_IOC_MIXER_GET_MIC, &ml) < 0)
2277                return (-errno);

2279        /* Report the mixer as having two channels. */
2280        val = LX_OSS_MIXER_ENC2(
2281            LX_OSS_S2L_GAIN(ml.lxa_ml_gain),
2282            LX_OSS_S2L_GAIN(ml.lxa_ml_gain));

2284        if (cmd == LX_OSS_SOUND_MIXER_READ_MIC)
2285                lx_debug("\toss get mixer mic result = 0x%x (%u)", val, val);
2286        if (cmd == LX_OSS_SOUND_MIXER_READ_IGAIN)
2287                lx_debug("\toss get mixer igain result = 0x%x (%u)", val, val);

2289        if (uucopy(&val, valp, sizeof (val)) != 0)
2290                return (-errno);
2291        return (0);
2292 }

2294 static int
2295 /*ARGSUSED*/
2296 ict_oss_mixer_write_mic(int fd, struct stat *stat,
2297     int cmd, char *cmd_str, intptr_t arg)
2298 {
2299        lxa_mixer_levels_t      ml;
2300        uint_t                  *valp = (uint_t *)arg;
2301        uint_t                  val;
2302        int                     err;

2304        assert((cmd == LX_OSS_SOUND_MIXER_WRITE_MIC) ||
2305            (cmd == LX_OSS_SOUND_MIXER_WRITE_IGAIN));
```



```
2307        /* Ioctl is only supported on mixer audio devices. */
2308        if ((err = i_is_mixer_dev(fd)) != 0)
2309                return (err);

2311        if (uucopy(valp, &val, sizeof (val)) != 0)
2312                return (-errno);

2314        if (cmd == LX_OSS_SOUND_MIXER_WRITE_MIC)
2315                lx_debug("\toss set mixer mic request = 0x%x (%u)", val, val);
2316        if (cmd == LX_OSS_SOUND_MIXER_WRITE_IGAIN)
2317                lx_debug("\toss set mixer igain request = 0x%x (%u)", val, val);

2319        /* The mic only supports one channel. */
2320        val = LX_OSS_MIXER_DEC1(val);

2322        ml.lxa_ml_balance = AUDIO_MID_BALANCE;
2323        ml.lxa_ml_gain = LX_OSS_L2S_GAIN(val);

2325        /* Attempt to set the device input gain. */
2326        lx_debug("\tioctl(%d, 0x%x - %s, ...)",
2327            fd, LXA_IOC_MIXER_SET_MIC, "LXA_IOC_MIXER_SET_MIC");
2328        if (ioctl(fd, LXA_IOC_MIXER_SET_MIC, &ml) < 0)
2329                return (-errno);

2331        return (0);
2332 }

2334 static int
2335 /*ARGSUSED*/
2336 ict_oss_mixer_read_devs(int fd, struct stat *stat,
2337     int cmd, char *cmd_str, intptr_t arg)
2338 {
2339        uint_t          *resultp = (uint_t *)arg;
2340        uint_t          result = 0;
2341        int             err;

2343        if (cmd == LX_OSS_SOUND_MIXER_READ_DEVMASK) {
2344                /* Bitmap of all the mixer channels we supposedly support. */
2345                result = ((1 << LX_OSS_SM_PCM) |
2346                    (1 << LX_OSS_SM_MIC) |
2347                    (1 << LX_OSS_SM_VOLUME));
2348        }
2349        if (cmd == LX_OSS_SOUND_MIXER_READ_STEREODEVS) {
2350                /* Bitmap of the stereo mixer channels we supposedly support. */
2351                result = ((1 << LX_OSS_SM_PCM) |
2352                    (1 << LX_OSS_SM_VOLUME));
2353        }
2354        if ((cmd == LX_OSS_SOUND_MIXER_READ_RECMASK) ||
2355            (cmd == LX_OSS_SOUND_MIXER_READ_RECSRC)) {
2356                /* Bitmap of the mixer input channels we supposedly support. */
2357                result = (1 << LX_OSS_SM_MIC);
2358        }
2359        assert(result != 0);

2361        /* Ioctl is only supported on mixer audio devices. */
2362        if ((err = i_is_mixer_dev(fd)) != 0)
2363                return (err);

2365        if (uucopy(&result, resultp, sizeof (result)) != 0)
2366                return (-errno);

2368        return (0);
2369 }

2371 /*
```

```
 * Audio ioctl conversion support structures.
 */
static oss_fmt_translator_t oft_table[] = {
        { LX_OSS_AFMT_MU_LAW,           AUDIO_ENCODING_ULAW,    8 },
        { LX_OSS_AFMT_A_LAW,            AUDIO_ENCODING_ALAW,    8 },
        { LX_OSS_AFMT_S8,               AUDIO_ENCODING_LINEAR,  8 },
        { LX_OSS_AFMT_U8,               AUDIO_ENCODING_LINEAR8, 8 },
        { LX_OSS_AFMT_S16_NE,           AUDIO_ENCODING_LINEAR,  16 },
        { 0,                            0,                      0 }
};

/*
 * Ioctl translator definitions.
 */

/*
 * Defines to help with creating ioctl translators.
 *
 * IOC_CMD_TRANSLATOR_NONE - Ioctl has the same semantics and argument
 * values on Solaris and Linux but may have different command values.
 * (Macro assumes the symbolic Linux name assigned to the ioctl command
 * value is the same as the Solaris symbol but pre-pended with an "LX_")
 *
 * IOC_CMD_TRANSLATOR_PASS - Ioctl is a Linux specific ioctl and should
 * be passed through unmodified.
 *
 * IOC_CMD_TRANSLATOR_FILTER - Ioctl has the same command name on
 * Solaris and Linux and needs a translation function that is common to
 * more than one ioctl. (Macro assumes the symbolic Linux name assigned
 * to the ioctl command value is the same as the Solaris symbol but
 * pre-pended with an "LX_")
 *
 * IOC_CMD_TRANSLATOR_CUSTOM - Ioctl needs special handling via a
 * translation function.
 */
#define IOC_CMD_TRANSLATOR_NONE(ioc_cmd_sym)                            \
        { (int)LX_##ioc_cmd_sym, "LX_" #ioc_cmd_sym,                    \
                ioc_cmd_sym, #ioc_cmd_sym, ict_pass },

#define IOC_CMD_TRANSLATOR_PASS(ioc_cmd_sym)                            \
        { (int)ioc_cmd_sym, #ioc_cmd_sym,                               \
                ioc_cmd_sym, #ioc_cmd_sym, ict_pass },

#define IOC_CMD_TRANSLATOR_FILTER(ioc_cmd_sym, ioct_handler)            \
        { (int)LX_##ioc_cmd_sym, "LX_" #ioc_cmd_sym,                    \
                ioc_cmd_sym, #ioc_cmd_sym, ioct_handler },

#define IOC_CMD_TRANSLATOR_CUSTOM(ioc_cmd_sym, ioct_handler)            \
        { (int)ioc_cmd_sym, #ioc_cmd_sym,                               \
                (int)ioc_cmd_sym, #ioc_cmd_sym, ioct_handler },

#define IOC_CMD_TRANSLATOR_END                                          \
        { 0, NULL, 0, NULL, NULL }

/* All files will need to support these ioctls. */
#define IOC_CMD_TRANSLATORS_ALL                                         \
        IOC_CMD_TRANSLATOR_NONE(FIONREAD)                               \
        IOC_CMD_TRANSLATOR_NONE(FIONBIO)

/* Any files supporting streams semantics will need these ioctls. */
#define IOC_CMD_TRANSLATORS_STREAMS                                     \
        IOC_CMD_TRANSLATOR_NONE(TCXONC)                                 \
        IOC_CMD_TRANSLATOR_NONE(TCFLSH)                                 \
        IOC_CMD_TRANSLATOR_NONE(TIOCEXCL)                               \
        IOC_CMD_TRANSLATOR_NONE(TIOCNXCL)                               \
        IOC_CMD_TRANSLATOR_NONE(TIOCSPGRP)                              \
```



```
        IOC_CMD_TRANSLATOR_NONE(TIOCSTI)                                \
        IOC_CMD_TRANSLATOR_NONE(TIOCSWINSZ)                             \
        IOC_CMD_TRANSLATOR_NONE(TIOCMBIS)                               \
        IOC_CMD_TRANSLATOR_NONE(TIOCMBIC)                               \
        IOC_CMD_TRANSLATOR_NONE(TIOCMSET)                               \
        IOC_CMD_TRANSLATOR_NONE(TIOCSETD)                               \
        IOC_CMD_TRANSLATOR_NONE(FIOASYNC)                               \
        IOC_CMD_TRANSLATOR_NONE(FIOSETOWN)                              \
        IOC_CMD_TRANSLATOR_NONE(TCSBRK)                                 \
                                                                        \
        IOC_CMD_TRANSLATOR_FILTER(TCSETS,               ict_tcsets)     \
        IOC_CMD_TRANSLATOR_FILTER(TCSETSW,              ict_tcsets)     \
        IOC_CMD_TRANSLATOR_FILTER(TCSETSF,              ict_tcsets)     \
        IOC_CMD_TRANSLATOR_FILTER(TCSETA,               ict_tcseta)     \
        IOC_CMD_TRANSLATOR_FILTER(TCSETAW,              ict_tcseta)     \
        IOC_CMD_TRANSLATOR_FILTER(TCSETAF,              ict_tcseta)     \
                                                                        \
        IOC_CMD_TRANSLATOR_CUSTOM(LX_TCSBRKP,           ict_tcsbrkp)


/*
 * Translators for non-device files.
 */
static ioc_cmd_translator_t ioc_translators_file[] = {
        IOC_CMD_TRANSLATORS_ALL
        IOC_CMD_TRANSLATOR_END
};

static ioc_cmd_translator_t ioc_translators_fifo[] = {
        IOC_CMD_TRANSLATORS_ALL
        IOC_CMD_TRANSLATORS_STREAMS
        IOC_CMD_TRANSLATOR_END
};

static ioc_cmd_translator_t ioc_translators_sock[] = {
        IOC_CMD_TRANSLATORS_ALL

        IOC_CMD_TRANSLATOR_NONE(FIOASYNC)
        IOC_CMD_TRANSLATOR_NONE(FIOGETOWN)
        IOC_CMD_TRANSLATOR_NONE(FIOSETOWN)
        IOC_CMD_TRANSLATOR_NONE(SIOCSPGRP)
        IOC_CMD_TRANSLATOR_NONE(SIOCGPGRP)

        IOC_CMD_TRANSLATOR_FILTER(SIOCATMARK,           ict_sioifoob)

        IOC_CMD_TRANSLATOR_FILTER(SIOCGIFFLAGS,         ict_sioifreq)
        IOC_CMD_TRANSLATOR_FILTER(SIOCSIFFLAGS,         ict_sioifreq)
        IOC_CMD_TRANSLATOR_FILTER(SIOCGIFADDR,          ict_sioifreq)
        IOC_CMD_TRANSLATOR_FILTER(SIOCSIFADDR,          ict_sioifreq)
        IOC_CMD_TRANSLATOR_FILTER(SIOCGIFDSTADDR,       ict_sioifreq)
        IOC_CMD_TRANSLATOR_FILTER(SIOCSIFDSTADDR,       ict_sioifreq)
        IOC_CMD_TRANSLATOR_FILTER(SIOCGIFBRDADDR,       ict_sioifreq)
        IOC_CMD_TRANSLATOR_FILTER(SIOCSIFBRDADDR,       ict_sioifreq)
        IOC_CMD_TRANSLATOR_FILTER(SIOCGIFNETMASK,       ict_sioifreq)
        IOC_CMD_TRANSLATOR_FILTER(SIOCSIFNETMASK,       ict_sioifreq)
        IOC_CMD_TRANSLATOR_FILTER(SIOCGIFMETRIC,        ict_sioifreq)
        IOC_CMD_TRANSLATOR_FILTER(SIOCSIFMETRIC,        ict_sioifreq)
        IOC_CMD_TRANSLATOR_FILTER(SIOCGIFMTU,           ict_sioifreq)
        IOC_CMD_TRANSLATOR_FILTER(SIOCSIFMTU,           ict_sioifreq)

        IOC_CMD_TRANSLATOR_CUSTOM(LX_SIOCGIFCONF,       ict_siocgifconf)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_SIOCGIFHWADDR,     ict_siocifhwaddr)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_SIOCSIFHWADDR,     ict_siocifhwaddr)

        IOC_CMD_TRANSLATOR_END
};
```

```
/*
 * Translators for devices.
 */
static ioc_cmd_translator_t ioc_cmd_translators_ptm[] = {
        IOC_CMD_TRANSLATORS_ALL
        IOC_CMD_TRANSLATORS_STREAMS

        IOC_CMD_TRANSLATOR_NONE(TIOCPKT)

        IOC_CMD_TRANSLATOR_CUSTOM(LX_TIOCGPGRP,         ict_tiocgpgrp)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_TIOCSPTLCK,        ict_sptlock)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_TIOCGPTN,          ict_gptn)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_TIOCGWINSZ,        ict_tiocgwinsz)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_TCGETS,            ict_tcgets_emulate)

        IOC_CMD_TRANSLATOR_END
};
static ioc_dev_translator_t ioc_translator_ptm = {
        LX_PTM_DRV,     /* idt_driver */
        0,              /* idt_major */
        ioc_cmd_translators_ptm
};

static ioc_cmd_translator_t ioc_cmd_translators_pts[] = {
        IOC_CMD_TRANSLATORS_ALL
        IOC_CMD_TRANSLATORS_STREAMS

        IOC_CMD_TRANSLATOR_NONE(TIOCGETD)
        IOC_CMD_TRANSLATOR_NONE(TIOCGSID)
        IOC_CMD_TRANSLATOR_NONE(TIOCNOTTY)

        IOC_CMD_TRANSLATOR_CUSTOM(LX_TIOCGPGRP,         ict_tiocgpgrp)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_TCGETS,            ict_tcgets_native)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_TCGETA,            ict_tcgeta)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_TIOCGWINSZ,        ict_tiocgwinsz)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_TIOCSCTTY,         ict_tiocsctty)

        IOC_CMD_TRANSLATOR_END
};
static ioc_dev_translator_t ioc_translator_pts = {
        "pts",          /* idt_driver */
        0,              /* idt_major */
        ioc_cmd_translators_pts
};

static ioc_dev_translator_t ioc_translator_sy = {
        "sy",           /* idt_driver */
        0,              /* idt_major */

        /*
         * /dev/tty (which is implemented via the "sy" driver) is basically
         * a layered driver that passes on requests to the ctty for the
         * current process.  Since ctty's are currently always implemented
         * via the pts driver, we should make sure to support all the
         * same ioctls on the sy driver as we do on the pts driver.
         */
        ioc_cmd_translators_pts
};

static ioc_cmd_translator_t ioc_cmd_translators_zcons[] = {
        IOC_CMD_TRANSLATORS_ALL
        IOC_CMD_TRANSLATORS_STREAMS

        IOC_CMD_TRANSLATOR_NONE(TIOCNOTTY)
```



```
        IOC_CMD_TRANSLATOR_CUSTOM(LX_TCGETS,            ict_tcgets_native)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_TCGETA,            ict_tcgeta)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_TIOCGWINSZ,        ict_tiocgwinsz)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_TIOCSCTTY,         ict_tiocsctty)

        IOC_CMD_TRANSLATOR_CUSTOM(LX_TIOCLINUX,         ict_einval)

        IOC_CMD_TRANSLATOR_END
};
static ioc_dev_translator_t ioc_translator_zcons = {
        "zcons",        /* idt_driver */
        0,              /* idt_major */
        ioc_cmd_translators_zcons
};

static ioc_cmd_translator_t ioc_cmd_translators_lx_audio[] = {
        IOC_CMD_TRANSLATORS_ALL

        /* /dev/dsp ioctls */
        IOC_CMD_TRANSLATOR_CUSTOM(LX_OSS_SNDCTL_DSP_RESET,
            ict_oss_sndctl_dsp_reset)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_OSS_SNDCTL_DSP_GETFMTS,
            ict_oss_sndctl_dsp_getfmts)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_OSS_SNDCTL_DSP_SETFMTS,
            ict_oss_sndctl_dsp_setfmts)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_OSS_SNDCTL_DSP_CHANNELS,
            ict_oss_sndctl_dsp_channels)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_OSS_SNDCTL_DSP_STEREO,
            ict_oss_sndctl_dsp_channels)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_OSS_SNDCTL_DSP_SPEED,
            ict_oss_sndctl_dsp_speed)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_OSS_SNDCTL_DSP_GETBLKSIZE,
            ict_oss_sndctl_dsp_getblksize)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_OSS_SNDCTL_DSP_SYNC,
            ict_oss_sndctl_dsp_sync)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_OSS_SNDCTL_DSP_SETFRAGMENT,
            ict_oss_sndctl_dsp_setfragment)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_OSS_SNDCTL_DSP_GETOSPACE,
            ict_oss_sndctl_dsp_getspace)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_OSS_SNDCTL_DSP_GETCAPS,
            ict_oss_sndctl_dsp_getcaps)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_OSS_SNDCTL_DSP_SETTRIGGER,
            ict_oss_sndctl_dsp_settrigger)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_OSS_SNDCTL_DSP_GETOPTR,
            ict_oss_sndctl_dsp_getoptr)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_OSS_SNDCTL_DSP_GETISPACE,
            ict_oss_sndctl_dsp_getspace)

        /* /dev/mixer level ioctls */
        IOC_CMD_TRANSLATOR_CUSTOM(LX_OSS_SOUND_MIXER_READ_VOLUME,
            ict_oss_mixer_read_volume)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_OSS_SOUND_MIXER_READ_PCM,
            ict_oss_mixer_read_volume)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_OSS_SOUND_MIXER_READ_MIC,
            ict_oss_mixer_read_mic)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_OSS_SOUND_MIXER_READ_IGAIN,
            ict_oss_mixer_read_mic)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_OSS_SOUND_MIXER_WRITE_VOLUME,
            ict_oss_mixer_write_volume)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_OSS_SOUND_MIXER_WRITE_PCM,
            ict_oss_mixer_write_volume)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_OSS_SOUND_MIXER_WRITE_MIC,
            ict_oss_mixer_write_mic)
        IOC_CMD_TRANSLATOR_CUSTOM(LX_OSS_SOUND_MIXER_WRITE_IGAIN,
            ict_oss_mixer_write_mic)
```

```
2636         /* /dev/mixer capability ioctls */
2637         IOC_CMD_TRANSLATOR_CUSTOM(LX_OSS_SOUND_MIXER_READ_STEREODEVS,
2638             ict_oss_mixer_read_devs)
2639         IOC_CMD_TRANSLATOR_CUSTOM(LX_OSS_SOUND_MIXER_READ_DEVMASK,
2640             ict_oss_mixer_read_devs)
2641         IOC_CMD_TRANSLATOR_CUSTOM(LX_OSS_SOUND_MIXER_READ_RECMASK,
2642             ict_oss_mixer_read_devs)
2643         IOC_CMD_TRANSLATOR_CUSTOM(LX_OSS_SOUND_MIXER_READ_RECSRC,
2644             ict_oss_mixer_read_devs)

2646         IOC_CMD_TRANSLATOR_END
2647 };
2648 static ioc_dev_translator_t ioc_translator_lx_audio = {
2649         "lx_audio",     /* idt_driver */
2650         0,              /* idt_major */
2651         ioc_cmd_translators_lx_audio
2652 };

2654 /*
2655  * An array of all the device translators.
2656  */
2657 static ioc_dev_translator_t *ioc_translators_dev[] = {
2658         &ioc_translator_lx_audio,
2659         &ioc_translator_ptm,
2660         &ioc_translator_pts,
2661         &ioc_translator_sy,
2662         &ioc_translator_zcons,
2663         NULL
2664 };

2666 /*
2667  * Translators for filesystems.
2668  */
2669 static ioc_cmd_translator_t ioc_cmd_translators_autofs[] = {
2670         IOC_CMD_TRANSLATOR_PASS(LX_AUTOFS_IOC_READY)
2671         IOC_CMD_TRANSLATOR_PASS(LX_AUTOFS_IOC_FAIL)
2672         IOC_CMD_TRANSLATOR_PASS(LX_AUTOFS_IOC_CATATONIC)
2673         IOC_CMD_TRANSLATOR_END
2674 };

2676 static ioc_fs_translator_t ioc_translator_autofs = {
2677         LX_AUTOFS_NAME, /* ift_filesystem */
2678         ioc_cmd_translators_autofs
2679 };

2681 /*
2682  * An array of all the filesystem translators.
2683  */
2684 static ioc_fs_translator_t *ioc_translators_fs[] = {
2685         &ioc_translator_autofs,
2686         NULL
2687 };

2689 /*
2690  * Ioctl error translator definitions.
2691  */
2692 #define IOC_ERRNO_TRANSLATOR(iet_cmd_sym, iet_errno)                    \
2693         { (int)LX_##iet_cmd_sym, "LX_" #iet_cmd_sym, iet_errno },

2695 #define IOC_ERRNO_TRANSLATOR_END                                        \
2696         { 0, NULL, 0 }

2698 static ioc_errno_translator_t ioc_translators_errno[] = {
2699         IOC_ERRNO_TRANSLATOR(TCGETS, ENOTTY)
2700         IOC_ERRNO_TRANSLATOR(TCSETS, ENOTTY)
2701         IOC_ERRNO_TRANSLATOR(TCSBRK, ENOTTY)
```



```
2702         IOC_ERRNO_TRANSLATOR(TCXONC, ENOTTY)
2703         IOC_ERRNO_TRANSLATOR(TCFLSH, ENOTTY)
2704         IOC_ERRNO_TRANSLATOR(TIOCGPGRP, ENOTTY)
2705         IOC_ERRNO_TRANSLATOR(TIOCSPGRP, ENOTTY)
2706         IOC_ERRNO_TRANSLATOR(TIOCGWINSZ, ENOTTY)
2707         IOC_ERRNO_TRANSLATOR_END
2708 };

2710 int
2711 lx_vhangup(void)
2712 {
2713         if (geteuid() != 0)
2714                 return (-EPERM);

2716         vhangup();

2718         return (0);
2719 }
2720 #endif /* ! codereview */
```

```
**********************************************************
    5577 Tue Jan 14 16:17:01 2014
new/usr/src/lib/brand/lx/lx_brand/common/iovec.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <errno.h>
#include <unistd.h>
#include <sys/uio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <alloca.h>
#include <string.h>
#include <sys/lx_syscall.h>
#include <sys/lx_misc.h>
#include <sys/lx_types.h>

static int
lx_is_directory(int fd)
{
        struct stat64 sbuf;

        if (fstat64(fd, &sbuf) < 0)
                sbuf.st_mode = 0;

        return ((sbuf.st_mode & S_IFMT) == S_IFDIR);
}

int
lx_read(uintptr_t p1, uintptr_t p2, uintptr_t p3)
{
        int             fd = (int)p1;
        void            *buf = (void *)p2;
        size_t          nbyte = (size_t)p3;
        ssize_t         ret;

        if (lx_is_directory(fd))
                return (-EISDIR);
```



```
        if ((ret = read(fd, buf, nbyte)) < 0)
                return (-errno);

        return (ret);
}

int
lx_pread64(uintptr_t p1, uintptr_t p2, uintptr_t p3, uintptr_t p4, uintptr_t p5)
{
        int             fd = (int)p1;
        void            *buf = (void *)p2;
        size_t          nbyte = (size_t)p3;
        uintptr_t       off_lo = p4;
        uintptr_t       off_hi = p5;
        ssize_t         ret;

        if (lx_is_directory(fd))
                return (-EISDIR);

        ret = pread64(fd, buf, nbyte, (off64_t)LX_32TO64(off_lo, off_hi));

        if (ret < 0)
                return (-errno);

        return (ret);
}

/*
 * On Linux, the pwrite(2) system call behaves identically to Solaris except
 * in the case of the file being opened with O_APPEND. In that case Linux's
 * pwrite(2) ignores the offset parameter and instead appends the data to the
 * file without modifying the current seek pointer.
 */
int
lx_pwrite64(uintptr_t p1, uintptr_t p2, uintptr_t p3, uintptr_t p4,
    uintptr_t p5)
{
        int fd = (int)p1;
        void *buf = (void *)p2;
        size_t nbyte = (size_t)p3;
        uintptr_t off_lo = p4;
        uintptr_t off_hi = p5;
        ssize_t ret;
        int rval;
        struct stat64 statbuf;

        if ((rval = fcntl(fd, F_GETFL, 0)) < 0)
                return (-errno);

        if (!(rval & O_APPEND)) {
                ret = pwrite64(fd, buf, nbyte,
                    (off64_t)LX_32TO64(off_lo, off_hi));
        } else if ((ret = fstat64(fd, &statbuf)) == 0) {
                ret = pwrite64(fd, buf, nbyte, statbuf.st_size);
        }

        if (ret < 0)
                return (-errno);

        return (ret);
}

/*
 * Implementation of Linux readv() and writev() system calls.
 *
```

```
 * The Linux system calls differ from the Solaris system calls in a few key
 * areas:
 *
 * - On Solaris, the maximum number of I/O vectors that can be passed to readv()
 *   or writev() is IOV_MAX (16).  Linux has a much larger restriction (1024).
 *
 * - Passing 0 as a vector count is an error on Solaris, but on Linux results
 *   in a return value of 0. Even though the man page says the opposite.
 *
 * - If the Nth vector results in an error, Solaris will return an error code
 *   for the entire operation.  Linux only returns an error if there has been
 *   no data transferred yet.  Otherwise, it returns the number of bytes
 *   transferred up until that point.
 *
 * In order to accomodate these differences, we implement these functions as a
 * series of ordinary read() or write() calls.
 */

#define LX_IOV_MAX 1024         /* Also called MAX_IOVEC */

static int
lx_iovec_copy_and_check(const struct iovec *iovp, struct iovec *iov, int count)
{
        int     i;
        ssize_t cnt = 0;

        if (uucopy(iovp, (void *)iov, count * sizeof (struct iovec)) != 0)
                return (-errno);

        for (i = 0; i < count; i++) {
                cnt += iov[i].iov_len;
                if (iov[i].iov_len < 0 || cnt < 0)
                        return (-EINVAL);
        }

        return (0);
}

int
lx_readv(uintptr_t p1, uintptr_t p2, uintptr_t p3)
{
        int                     fd = (int)p1;
        const struct iovec      *iovp = (const struct iovec *)p2;
        int                     count = (int)p3;
        struct iovec            *iov;
        ssize_t                 total = 0, ret;
        int                     i;

        if (count == 0)
                return (0);

        if (count < 0 || count > LX_IOV_MAX)
                return (-EINVAL);

        if (lx_is_directory(fd))
                return (-EISDIR);

        iov = SAFE_ALLOCA(count * sizeof (struct iovec));
        if (iov == NULL)
                return (-ENOMEM);
        if ((ret = lx_iovec_copy_and_check(iovp, iov, count)) != 0)
                return (ret);

        for (i = 0; i < count; i++) {
                ret = read(fd, iov[i].iov_base, iov[i].iov_len);
```



```
                if (ret < 0) {
                        if (total > 0)
                                return (total);
                        return (-errno);
                }

                total += ret;
        }

        return (total);
}

int
lx_writev(uintptr_t p1, uintptr_t p2, uintptr_t p3)
{
        int                     fd = (int)p1;
        const struct iovec      *iovp = (const struct iovec *)p2;
        int                     count = (int)p3;
        struct iovec            *iov;
        ssize_t                 total = 0, ret;
        int                     i;

        if (count == 0)
                return (0);

        if (count < 0 || count > LX_IOV_MAX)
                return (-EINVAL);

        iov = SAFE_ALLOCA(count * sizeof (struct iovec));
        if (iov == NULL)
                return (-ENOMEM);
        if ((ret = lx_iovec_copy_and_check(iovp, iov, count)) != 0)
                return (ret);

        for (i = 0; i < count; i++) {
                ret = write(fd, iov[i].iov_base, iov[i].iov_len);

                if (ret < 0) {
                        if (total > 0)
                                return (total);
                        return (-errno);
                }

                total += ret;
        }

        return (total);
}
#endif /* ! codereview */
```

```
**********************************************************
   39942 Tue Jan 14 16:17:01 2014
new/usr/src/lib/brand/lx/lx_brand/common/lx_brand.c
LX zone support should now build and packages of relevance produced.
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/types.h>
#include <sys/syscall.h>
#include <sys/utsname.h>
#include <sys/inttypes.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <sys/fstyp.h>
#include <sys/fsid.h>
#include <sys/systm.h>
#include <sys/auxv.h>
#include <sys/frame.h>
#include <sys/brand.h>

#include <assert.h>
#include <stdio.h>
#include <stdarg.h>
#include <stdlib.h>
#include <strings.h>
#include <unistd.h>
#include <errno.h>
#include <syslog.h>
#include <signal.h>
#include <fcntl.h>
#include <synch.h>
#include <libelf.h>
#include <libgen.h>
#include <pthread.h>
#include <utime.h>
#include <dirent.h>
#include <ucontext.h>
#include <libintl.h>
#include <locale.h>

#include <sys/lx_misc.h>
```



```
#include <sys/lx_debug.h>
#include <sys/lx_brand.h>
#include <sys/lx_types.h>
#include <sys/lx_stat.h>
#include <sys/lx_statfs.h>
#include <sys/lx_ioctl.h>
#include <sys/lx_signal.h>
#include <sys/lx_syscall.h>
#include <sys/lx_thread.h>
#include <sys/lx_thunk_server.h>

/*
 * Map solaris errno to the linux equivalent.
 */
static int stol_errno[] = {
        0,   1,   2,   3,   4,   5,   6,   7,   8,   9,
        10,  11,  12,  13,  14,  15,  16,  17,  18,  19,
        20,  21,  22,  23,  24,  25,  26,  27,  28,  29,
        30,  31,  32,  33,  34,  42,  43,  44,  45,  46,
        47,  48,  49,  50,  51,  35,  47,  22,  38,  22, /* 49 */
        52,  53,  54,  55,  56,  57,  58,  59,  22,  22,
        61,  61,  62,  63,  64,  65,  66,  67,  68,  69,
        70,  71,  22,  22,  72,  22,  22,  74,  36,  75,
        76,  77,  78,  79,  80,  81,  82,  83,  84,  38,
        40,  85,  86,  39,  87,  88,  89,  90,  91,  92, /* 99 */
        22,  22,  22,  22,  22,  22,  22,  22,  22,  22,
        22,  22,  22,  22,  22,  22,  22,  22,  22,  22,
        93,  94,  95,  96,  97,  98,  99, 100, 101, 102,
        103, 104, 105, 106, 107,  22,  22,  22,  22,  22,
        22,  22,  22, 108, 109, 110, 111, 112, 113, 114, /* 149 */
        115, 116
};

char lx_release[128];

/*
 * Map a linux locale ending string to the solaris equivalent.
 */
struct lx_locale_ending {
        const char      *linux_end;     /* linux ending string */
        const char      *solaris_end;   /* to transform with this string */
        int             le_size;        /* linux ending string length */
        int             se_size;        /* solaris ending string length */
};

#define l2s_locale(lname, sname) \
        {(lname), (sname), sizeof ((lname)) - 1, sizeof ((sname)) - 1}

/*static struct lx_locale_ending lx_locales[] = {
        l2s_locale(".utf8",      ".UTF-8"),
        l2s_locale(".utf8@euro", ".UTF-8"),
        l2s_locale("@euro",      ".ISO8859-15"),
        l2s_locale(".iso885915", ".ISO8859-15"),
        l2s_locale(".euckr",     ".EUC"),
        l2s_locale(".euctw",     ".EUC"),
        l2s_locale(".koi8r",     ".KOI8-R"),
        l2s_locale(".gb18030",   ".GB18030"),
        l2s_locale(".gbk",       ".GBK"),
        l2s_locale("@cyrillic",  ".ISO8859-5")
};*/

#define MAXLOCALENAMELEN        30
#if !defined(TEXT_DOMAIN)               /* should be defined by cc -D */
#define TEXT_DOMAIN     "SYS_TEST"      /* Use this only if it wasn't */
#endif
```

```
127 /*
128  * This flag is part of the registration with the in-kernel brand module. It's
129  * used in lx_handler() to determine if we should go back into the kernel after
130  * a system call in case the kernel needs to perform some post-syscall work
131  * like tracing for example.
132  */
133 int lx_traceflag;

135 #define NOSYS_NULL              1
136 #define NOSYS_NO_EQUIV          2
137 #define NOSYS_KERNEL            3
138 #define NOSYS_UNDOC             4
139 #define NOSYS_OBSOLETE          5

141 /*
142  * SYS_PASSTHRU denotes a system call we can just call on behalf of the
143  * branded process without having to translate the arguments.
144  *
145  * The restriction on this is that the call in question MUST return -1 to
146  * denote an error.
147  */
148 #define SYS_PASSTHRU            5

150 static char *nosys_msgs[] = {
151         "Either not yet done, or we haven't come up with an excuse",
152         "No such Linux system call",
153         "No equivalent Solaris functionality",
154         "Reads/modifies Linux kernel state",
155         "Undocumented and/or rarely used system call",
156         "Unsupported, obsolete system call"
157 };

159 struct lx_sysent {
160         char    *sy_name;
161         int     (*sy_callc)();
162         char    sy_flags;
163         char    sy_narg;
164 };

166 static struct lx_sysent sysents[LX_NSYSCALLS + 1];

168 static uintptr_t stack_bottom;

170 int lx_install = 0;             /* install mode enabled if non-zero */
171 boolean_t lx_is_rpm = B_FALSE;
172 int lx_rpm_delay = 1;
173 int lx_strict = 0;              /* "strict" mode enabled if non-zero */
174 int lx_verbose = 0;             /* verbose mode enabled if non-zero */
175 int lx_debug_enabled = 0;       /* debugging output enabled if non-zero */

177 pid_t zoneinit_pid;             /* zone init PID */

179 thread_key_t lx_tsd_key;

181 int
182 uucopy_unsafe(const void *src, void *dst, size_t n)
183 {
184         bcopy(src, dst, n);
185         return (0);
186 }

188 int
189 uucopystr_unsafe(const void *src, void *dst, size_t n)
190 {
191         (void) strncpy((char *)src, dst, n);
192         return (0);
```



```
193 }

195 static void
196 i_lx_msg(int fd, char *msg, va_list ap)
197 {
198         int     i;
199         char    buf[LX_MSG_MAXLEN];

201         /* LINTED [possible expansion issues] */
202         i = vsnprintf(buf, sizeof (buf), msg, ap);
203         buf[LX_MSG_MAXLEN - 1] = '\0';
204         if (i == -1)
205                 return;

207         /* if debugging is enabled, send this message to debug output */
208         if (lx_debug_enabled != 0)
209                 lx_debug(buf);

211         /*
212          * If we are trying to print to stderr, we also want to send the
213          * message to syslog.
214          */
215         if (fd == 2) {
216                 syslog(LOG_ERR, "%s", buf);

218                 /*
219                  * We let the user choose whether or not to see these
220                  * messages on the console.
221                  */
222                 if (lx_verbose == 0)
223                         return;
224         }

226         /* we retry in case of EINTR */
227         do {
228                 i = write(fd, buf, strlen(buf));
229         } while ((i == -1) && (errno == EINTR));
230 }

232 /*PRINTFLIKE1*/
233 void
234 lx_err(char *msg, ...)
235 {
236         va_list ap;

238         assert(msg != NULL);

240         va_start(ap, msg);
241         i_lx_msg(STDERR_FILENO, msg, ap);
242         va_end(ap);
243 }

245 /*
246  * This is just a non-zero exit value which also isn't one that would allow
247  * us to easily detect if a branded process exited because of a recursive
248  * fatal error.
249  */
250 #define LX_ERR_FATAL    42

252 /*
253  * Our own custom version of abort(), this routine will be used in place
254  * of the one located in libc.  The primary difference is that this version
255  * will first reset the signal handler for SIGABRT to SIG_DFL, ensuring the
256  * SIGABRT sent causes us to dump core and is not caught by a user program.
257  */
258 void
```

```
abort(void)
{
        static int aborting = 0;

        struct sigaction sa;
        sigset_t sigmask;

        /* watch out for recursive calls to this function */
        if (aborting != 0)
                exit(LX_ERR_FATAL);

        aborting = 1;

        /*
         * Block all signals here to avoid taking any signals while exiting
         * in an effort to avoid any strange user interaction with our death.
         */
        (void) sigfillset(&sigmask);
        (void) sigprocmask(SIG_BLOCK, &sigmask, NULL);

        /*
         * Our own version of abort(3C) that we know will never call
         * a user-installed SIGABRT handler first.  We WANT to die.
         *
         * Do this by resetting the handler to SIG_DFL, and releasing any
         * held SIGABRTs.
         *
         * If no SIGABRTs are pending, send ourselves one.
         *
         * The while loop is a bit of overkill, but abort(3C) does it to
         * assure it never returns so we will as well.
         */
        (void) sigemptyset(&sa.sa_mask);
        sa.sa_sigaction = SIG_DFL;
        sa.sa_flags = 0;

        for (;;) {
                (void) sigaction(SIGABRT, &sa, NULL);
                (void) sigrelse(SIGABRT);
                (void) thr_kill(thr_self(), SIGABRT);
        }

        /*NOTREACHED*/
}

/*PRINTFLIKE1*/
void
lx_msg(char *msg, ...)
{
        va_list ap;

        assert(msg != NULL);
        va_start(ap, msg);
        i_lx_msg(STDOUT_FILENO, msg, ap);
        va_end(ap);
}

/*PRINTFLIKE1*/
void
lx_err_fatal(char *msg, ...)
{
        va_list ap;

        assert(msg != NULL);

        va_start(ap, msg);
```



```
        i_lx_msg(STDERR_FILENO, msg, ap);
        va_end(ap);
        abort();
}

/*
 * See if it is safe to alloca() sz bytes.  Return 1 for yes, 0 for no.
 */
int
lx_check_alloca(size_t sz)
{
        uintptr_t sp = (uintptr_t)&sz;
        uintptr_t end = sp - sz;

        return ((end < sp) && (end >= stack_bottom));
}

/*PRINTFLIKE1*/
void
lx_unsupported(char *msg, ...)
{
        va_list ap;

        assert(msg != NULL);

        /* send the msg to the error stream */
        va_start(ap, msg);
        i_lx_msg(STDERR_FILENO, msg, ap);
        va_end(ap);

        /*
         * If the user doesn't trust the application to responsibly
         * handle ENOTSUP, we kill the application.
         */
        if (lx_strict)
                (void) kill(getpid(), SIGSYS);
}

extern void lx_runexe(void *argv, int32_t entry);
int lx_init(int argc, char *argv[], char *envp[]);

static int
lx_emulate_args(lx_regs_t *rp, struct lx_sysent *s, uintptr_t *args)
{
        /*
         * If the system call takes 6 args, then libc has stashed them in
         * memory at the address contained in %ebx. Except for some syscalls
         * which store the 6th argument in %ebp.
         */
        if (s->sy_narg == 6 && !(s->sy_flags & EBP_HAS_ARG6)) {
                if (uucopy((void *)rp->lxr_ebx, args,
                    sizeof (args[0]) * 6) != 0)
                        return (-stol_errno[errno]);
        } else {
                args[0] = rp->lxr_ebx;
                args[1] = rp->lxr_ecx;
                args[2] = rp->lxr_edx;
                args[3] = rp->lxr_esi;
                args[4] = rp->lxr_edi;
                args[5] = rp->lxr_ebp;
        }

        return (0);
}

void
```

```
391 lx_emulate(lx_regs_t *rp)
392 {
393         struct lx_sysent *s;
394         uintptr_t args[6];
395         uintptr_t gs = rp->lxr_gs & 0xffff;     /* %gs is only 16 bits */
396         int syscall_num, ret;

398         syscall_num = rp->lxr_eax;

400         /*
401          * lx_brand_int80_callback() ensures that the syscall_num is sane;
402          * Use it as is.
403          */
404         assert(syscall_num >= 0);
405         assert(syscall_num < (sizeof (sysents) / sizeof (sysents[0])));
406         s = &sysents[syscall_num];

408         if ((ret = lx_emulate_args(rp, s, args)) != 0)
409                 goto out;

411         /*
412          * If the tracing flag is enabled we call into the brand-specific
413          * kernel module to handle the tracing activity (DTrace or ptrace).
414          * It would be tempting to perform DTrace activity in the brand
415          * module's syscall trap callback, rather than having to return
416          * to the kernel here, but -- since argument encoding can vary
417          * according to the specific system call -- that would require
418          * replicating the knowledge of argument decoding in the kernel
419          * module as well as here in the brand library.
420          */
421         if (lx_traceflag != 0) {
422                 /*
423                  * Part of the ptrace "interface" is that on syscall entry
424                  * %eax should be reported as -ENOSYS while the orig_eax
425                  * field of the user structure needs to contain the actual
426                  * system call number. If we end up stopping here, the
427                  * controlling process will dig the lx_regs_t structure out of
428                  * our stack.
429                  */
430                 rp->lxr_orig_eax = syscall_num;
431                 rp->lxr_eax = -stol_errno[ENOSYS];

433                 (void) syscall(SYS_brand, B_SYSENTRY, syscall_num, args);

435                 /*
436                  * The external tracer may have modified the arguments to this
437                  * system call. Refresh the argument cache to account for this.
438                  */
439                 if ((ret = lx_emulate_args(rp, s, args)) != 0)
440                         goto out;
441         }

443         if (s->sy_callc == NULL) {
444                 lx_unsupported(gettext("unimplemented syscall #%d (%s): %s\n"),
445                     syscall_num, s->sy_name, nosys_msgs[(int)s->sy_flags]);
446                 ret = -stol_errno[ENOTSUP];
447                 goto out;
448         }

450         if (lx_debug_enabled != 0) {
451                 const char *fmt = NULL;

453                 switch (s->sy_narg) {
454                 case 0:
455                         fmt = "calling %s()";
456                         break;
```



```
457                 case 1:
458                         fmt = "calling %s(0x%p)";
459                         break;
460                 case 2:
461                         fmt = "calling %s(0x%p, 0x%p)";
462                         break;
463                 case 3:
464                         fmt = "calling %s(0x%p, 0x%p, 0x%p)";
465                         break;
466                 case 4:
467                         fmt = "calling %s(0x%p, 0x%p, 0x%p, 0x%p)";
468                         break;
469                 case 5:
470                         fmt = "calling %s(0x%p, 0x%p, 0x%p, 0x%p, 0x%p)";
471                         break;
472                 case 6:
473                         fmt = "calling %s(0x%p, 0x%p, 0x%p, 0x%p, 0x%p, 0x%p)";
474                         break;
475                 }

477                 lx_debug(fmt, s->sy_name, args[0], args[1], args[2], args[3],
478                     args[4], args[5]);
479         }

481         if (gs != LWPGS_SEL) {
482                 lx_tsd_t *lx_tsd;

484                 /*
485                  * While a %gs of 0 is technically legal (as long as the
486                  * application never dereferences memory using %gs), Solaris
487                  * has its own ideas as to how a zero %gs should be handled in
488                  * _update_sregs(), such that any 32-bit user process with a
489                  * %gs of zero running on a system with a 64-bit kernel will
490                  * have its %gs hidden base register stomped on on return from
491                  * a system call, leaving an incorrect base address in place
492                  * until the next time %gs is actually reloaded (forcing a
493                  * reload of the base address from the appropriate descriptor
494                  * table.)
495                  *
496                  * Of course the kernel will once again stomp on THAT base
497                  * address when returning from a system call, resulting in an
498                  * an application segmentation fault.
499                  *
500                  * To avoid this situation, disallow a save of a zero %gs
501                  * here in order to try and capture any Linux process that
502                  * attempts to make a syscall with a zero %gs installed.
503                  */
504                 assert(gs != 0);

506                 if ((ret = thr_getspecific(lx_tsd_key,
507                     (void **)&lx_tsd)) != 0)
508                         lx_err_fatal(gettext(
509                             "%s: unable to read thread-specific data: %s"),
510                             "lx_emulate", strerror(ret));

512                 assert(lx_tsd != 0);

514                 lx_tsd->lxtsd_gs = gs;

516                 lx_debug("lx_emulate(): gsp 0x%p, saved gs: 0x%x", lx_tsd, gs);
517         }

519         if (s->sy_flags == SYS_PASSTHRU)
520                 lx_debug("\tCalling Solaris %s()", s->sy_name);

522         ret = s->sy_callc(args[0], args[1], args[2], args[3], args[4], args[5]);
```

```
524         if (ret > -65536 && ret < 65536)
525                 lx_debug("\t= %d", ret);
526         else
527                 lx_debug("\t= 0x%x", ret);

529         if ((s->sy_flags == SYS_PASSTHRU) && (ret == -1)) {
530                 ret = -stol_errno[errno];
531         } else {
532                 /*
533                  * If the return value is between -4096 and 0 we assume it's an
534                  * error, so we translate the Solaris error number into the
535                  * Linux equivalent.
536                  */
537                 if (ret < 0 && ret > -4096) {
538                         if (-ret >=
539                             sizeof (stol_errno) / sizeof (stol_errno[0])) {
540                                 lx_debug("Invalid return value from emulated "
541                                     "syscall %d (%s): %d\n",
542                                     syscall_num, s->sy_name, ret);
543                                 assert(0);
544                         }

546                         ret = -stol_errno[-ret];
547                 }
548         }

550 out:
551         /*
552          * %eax holds the return code from the system call.
553          */
554         rp->lxr_eax = ret;

556         /*
557          * If the trace flag is set, bounce into the kernel to let it do
558          * any necessary tracing (DTrace or ptrace).
559          */
560         if (lx_traceflag != 0) {
561                 rp->lxr_orig_eax = syscall_num;
562                 (void) syscall(SYS_brand, B_SYSRETURN, syscall_num, ret);
563         }
564 }

566 /* Transform the Linux locale name to make it look like a Solaris locale name */
567 /* static const char *
568 lx_translate_locale(char *translated_name_mem, int mem_size)
569 {
570         char *loc;
571         int i;
572         size_t len;

574         if ((loc = getenv("LC_ALL")) == NULL)
575                 if ((loc = getenv("LANG")) == NULL)
576                         return ("C");

578         if (strlcpy(translated_name_mem, loc, mem_size) >= mem_size)
579                 return ("");

581         len = strlen(loc);

583         // replace the end of the locale name if it's a known pattern
584         for (i = 0; i < sizeof (lx_locales) / sizeof (struct lx_locale_ending);
585             i++) {
586                 if (len <= lx_locales[i].le_size)
587                         continue;
```



```
589                 if (strncmp(loc + len - lx_locales[i].le_size,
590                     lx_locales[i].linux_end, lx_locales[i].le_size))
591                         continue; // don't match

593                 if (len - lx_locales[i].le_size + lx_locales[i].se_size
594                     >= mem_size)
595                         return ("C"); // size too small for the new name

597                 (void) strlcpy(translated_name_mem + len -
598                     lx_locales[i].le_size, lx_locales[i].solaris_end,
599                     lx_locales[i].se_size + 1);

601                 return ((const char *)translated_name_mem);
602         }

604         // no match
605         return ("");
606 } */

608 static void
609 lx_close_fh(FILE *file)
610 {
611         int fd, fd_new;

613         if (file == NULL)
614                 return;

616         if ((fd = fileno(file)) < 0)
617                 return;

619         fd_new = dup(fd);
620         if (fd_new == -1)
621                 return;

623         (void) fclose(file);
624         (void) dup2(fd_new, fd);
625         (void) close(fd_new);
626 }


629 extern int set_l10n_alternate_root(char *path);

631 /*ARGSUSED*/
632 int
633 lx_init(int argc, char *argv[], char *envp[])
634 {
635         char            *r;
636         auxv_t          *ap;
637         int             *p, err;
638         lx_elf_data_t   edp;
639         lx_brand_registration_t reg;
640 /*      char            locale_translated_name[MAXLOCALENAMELEN]; */
641         static lx_tsd_t lx_tsd;


644         /* Look up the PID that serves as init for this zone */
645         if ((err = lx_lpid_to_spid(1, &zoneinit_pid)) < 0)
646                 lx_err_fatal(gettext(
647                     "Unable to find PID for zone init process: %s"),
648                     strerror(err));

650         /*
651          * Ubuntu init will fail if its TERM environment variable is not set
652          * so if we are running init, and TERM is not set, we set term and
653          * reexec so that the new environment variable is propagated to the
654          * linux application stack.
```

```
655          */
656         if ((getpid() == zoneinit_pid) && (getenv("TERM") == NULL)) {
657                 if (setenv("TERM", "vt100", 1) < 0 || execv(argv[0], argv) < 0)
658                         lx_err_fatal(gettext("failed to set TERM"));
659         }
660 /*
661         if ((set_l10n_alternate_root("/native") == 0) &&
662             (setlocale(LC_ALL, lx_translate_locale(locale_translated_name,
663             sizeof (locale_translated_name))) != NULL) &&
664             (bindtextdomain(TEXT_DOMAIN, "/native/usr/lib/locale") != NULL)) {
665                 (void) textdomain(TEXT_DOMAIN);
666         }
667 */
668         stack_bottom = 2 * sysconf(_SC_PAGESIZE);

670         /*
671          * We need to shutdown all libc stdio.  libc stdio normally goes to
672          * file descriptors, but since we're actually part of a linux
673          * process we don't own these file descriptors and we can't make
674          * any assumptions about their state.
675          */
676         lx_close_fh(stdin);
677         lx_close_fh(stdout);
678         lx_close_fh(stderr);

680         lx_debug_init();

682         r = getenv("LX_RELEASE");
683         if (r == NULL) {
684                 if (lx_get_kern_version() == LX_KERN_2_6)
685                         (void) strlcpy(lx_release, LX_UNAME_RELEASE_2_6,
686                             sizeof (lx_release));
687                 else
688                         (void) strlcpy(lx_release, LX_UNAME_RELEASE_2_4,
689                             sizeof (lx_release));
690         } else {
691                 (void) strlcpy(lx_release, r, 128);
692         }

694         lx_debug("lx_release: %s\n", lx_release);

696         /*
697          * Should we kill an application that attempts an unimplemented
698          * system call?
699          */
700         if (getenv("LX_STRICT") != NULL) {
701                 lx_strict = 1;
702                 lx_debug("STRICT mode enabled.\n");
703         }

705         /*
706          * Are we in install mode?
707          */
708         if (getenv("LX_INSTALL") != NULL) {
709                 lx_install = 1;
710                 lx_debug("INSTALL mode enabled.\n");
711         }

713         /*
714          * Should we attempt to send messages to the screen?
715          */
716         if (getenv("LX_VERBOSE") != NULL) {
717                 lx_verbose = 1;
718                 lx_debug("VERBOSE mode enabled.\n");
719         }
```



```
721         lx_debug("executing linux process: %s", argv[0]);
722         lx_debug("branding myself and setting handler to 0x%p",
723             (void *)lx_handler_table);

725         /*
726          * The version of rpm that ships with CentOS/RHEL 3.x has a race
727          * condition in it.  If it creates a child process to run a
728          * post-install script, and that child process completes too
729          * quickly, it will disappear before the parent notices.  This
730          * causes the parent to hang forever waiting for the already dead
731          * child to die.  I'm sure there's a Lazarus joke buried in here
732          * somewhere.
733          *
734          * Anyway, as a workaround, we make every child of an 'rpm' process
735          * sleep for 1 second, giving the parent a chance to enter its
736          * wait-for-the-child-to-die loop.  Thay may be the hackiest trick
737          * in all of our Linux emulation code - and that's saying
738          * something.
739          */
740         if (strcmp("rpm", basename(argv[0])) == NULL)
741                 lx_is_rpm = B_TRUE;

743         reg.lxbr_version = LX_VERSION;
744         reg.lxbr_handler = (void *)&lx_handler_table;
745         reg.lxbr_tracehandler = (void *)&lx_handler_trace_table;
746         reg.lxbr_traceflag = &lx_traceflag;

748         /*
749          * Register the address of the user-space handler with the lx
750          * brand module.
751          */
752         if (syscall(SYS_brand, B_REGISTER, &reg))
753                 lx_err_fatal(gettext("failed to brand the process"));

755         /*
756          * Download data about the lx executable from the kernel.
757          */
758         if (syscall(SYS_brand, B_ELFDATA, (void *)&edp))
759                 lx_err_fatal(gettext(
760                     "failed to get required ELF data from the kernel"));

762         if (lx_ioctl_init() != 0)
763                 lx_err_fatal(gettext("failed to setup the %s translator"),
764                     "ioctl");

766         if (lx_stat_init() != 0)
767                 lx_err_fatal(gettext("failed to setup the %s translator"),
768                     "stat");

770         if (lx_statfs_init() != 0)
771                 lx_err_fatal(gettext("failed to setup the %s translator"),
772                     "statfs");

774         /*
775          * Find the aux vector on the stack.
776          */
777         p = (int *)envp;
778         while (*p != NULL)
779                 p++;
780         /*
781          * p is now pointing at the 0 word after the environ pointers. After
782          * that is the aux vectors.
783          */
784         p++;
785         for (ap = (auxv_t *)p; ap->a_type != 0; ap++) {
786                 switch (ap->a_type) {
```

```
787                         case AT_BASE:
788                                 ap->a_un.a_val = edp.ed_base;
789                                 break;
790                         case AT_ENTRY:
791                                 ap->a_un.a_val = edp.ed_entry;
792                                 break;
793                         case AT_PHDR:
794                                 ap->a_un.a_val = edp.ed_phdr;
795                                 break;
796                         case AT_PHENT:
797                                 ap->a_un.a_val = edp.ed_phent;
798                                 break;
799                         case AT_PHNUM:
800                                 ap->a_un.a_val = edp.ed_phnum;
801                                 break;
802                         default:
803                                 break;
804                 }
805         }

807         /* Do any thunk server initalization. */
808         lxt_server_init(argc, argv);

810         /* Setup signal handler information. */
811         if (lx_siginit())
812                 lx_err_fatal(gettext(
813                     "failed to initialize lx signals for the branded process"));

815         /* Setup thread-specific data area for managing linux threads. */
816         if ((err = thr_keycreate(&lx_tsd_key, NULL)) != 0)
817                 lx_err_fatal(
818                     gettext("%s failed: %s"), "thr_keycreate(lx_tsd_key)",
819                     strerror(err));

821         lx_debug("thr_keycreate created lx_tsd_key (%d)", lx_tsd_key);

823         /* Initialize the thread specific data for this thread. */
824         bzero(&lx_tsd, sizeof (lx_tsd));
825         lx_tsd.lxtsd_gs = LWPGS_SEL;

827         if ((err = thr_setspecific(lx_tsd_key, &lx_tsd)) != 0)
828                 lx_err_fatal(gettext(
829                     "Unable to initialize thread-specific data: %s"),
830                     strerror(err));

832         /*
833          * Save the current context of this thread.
834          * We'll restore this context when this thread attempts to exit.
835          */
836         if (getcontext(&lx_tsd.lxtsd_exit_context) != 0)
837                 lx_err_fatal(gettext(
838                     "Unable to initialize thread-specific exit context: %s"),
839                     strerror(errno));

841         if (lx_tsd.lxtsd_exit == 0) {
842                 lx_runexe(argv, edp.ed_ldentry);
843                 /* lx_runexe() never returns. */
844                 assert(0);
845         }

847         /*
848          * We are here because the Linux application called the exit() or
849          * exit_group() system call.  In turn the brand library did a
850          * setcontext() to jump to the thread context state we saved above.
851          */
852         if (lx_tsd.lxtsd_exit == 1)
```



```
853                 thr_exit((void *)lx_tsd.lxtsd_exit_status);
854         else
855                 exit(lx_tsd.lxtsd_exit_status);

857         assert(0);

859         /*NOTREACHED*/
860         return (0);
861 }

863 /*
864  * Walk back through the stack until we find the lx_emulate() frame.
865  */
866 lx_regs_t *
867 lx_syscall_regs(void)
868 {
869         /* LINTED - alignment */
870         struct frame *fr = (struct frame *)_getfp();

872         while (fr->fr_savpc != (uintptr_t)&lx_emulate_done) {
873                 fr = (struct frame *)fr->fr_savfp;
874                 assert(fr->fr_savpc != NULL);
875         }

877         return ((lx_regs_t *)((uintptr_t *)fr)[2]);
878 }

880 int
881 lx_lpid_to_spair(pid_t lpid, pid_t *spid, lwpid_t *slwp)
882 {
883         pid_t pid;
884         lwpid_t tid;

886         if (lpid == 0) {
887                 pid = getpid();
888                 tid = thr_self();
889         } else {
890                 if (syscall(SYS_brand, B_LPID_TO_SPAIR, lpid, &pid, &tid) < 0)
891                         return (-errno);

893                 /*
894                  * If the returned pid is -1, that indicates we tried to
895                  * look up the PID for init, but that process no longer
896                  * exists.
897                  */
898                 if (pid == -1)
899                         return (-ESRCH);
900         }

902         if (uucopy(&pid, spid, sizeof (pid_t)) != 0)
903                 return (-errno);

905         if (uucopy(&tid, slwp, sizeof (lwpid_t)) != 0)
906                 return (-errno);

908         return (0);
909 }

911 int
912 lx_lpid_to_spid(pid_t lpid, pid_t *spid)
913 {
914         lwpid_t slwp;

916         return (lx_lpid_to_spair(lpid, spid, &slwp));
917 }
```

```
char *
lx_fd_to_path(int fd, char *buf, int buf_size)
{
        char    path_proc[MAXPATHLEN];
        pid_t   pid;
        int     n;

        assert((buf != NULL) && (buf_size >= 0));

        if (fd < 0)
                return (NULL);

        if ((pid = getpid()) == -1)
                return (NULL);

        (void) snprintf(path_proc, MAXPATHLEN,
            "/native/proc/%d/path/%d", pid, fd);

        if ((n = readlink(path_proc, buf, buf_size - 1)) == -1)
                return (NULL);
        buf[n] = '\0';

        return (buf);
}

/*
 * Create a translation routine that jumps to a particular emulation
 * module syscall.
 */
#define IN_KERNEL_SYSCALL(name, num)            \
int                                             \
lx_##name(uintptr_t p1, uintptr_t p2, uintptr_t p3, uintptr_t p4,       \
        uintptr_t p5, uintptr_t p6)             \
{                                               \
        int r;                                  \
        lx_debug("\tsyscall %d re-vectoring to lx kernel module "       \
            "for " #name "()", num);            \
        r = syscall(SYS_brand, B_EMULATE_SYSCALL + num, p1, p2,         \
            p3, p4, p5, p6);                    \
        return ((r == -1) ? -errno : r);                \
}

IN_KERNEL_SYSCALL(kill, 37)
IN_KERNEL_SYSCALL(brk, 45)
IN_KERNEL_SYSCALL(ustat, 62)
IN_KERNEL_SYSCALL(getppid, 64)
IN_KERNEL_SYSCALL(sysinfo, 116)
IN_KERNEL_SYSCALL(modify_ldt, 123)
IN_KERNEL_SYSCALL(adjtimex, 124)
IN_KERNEL_SYSCALL(setresuid16, 164)
IN_KERNEL_SYSCALL(setresgid16, 170)
IN_KERNEL_SYSCALL(setresuid, 208)
IN_KERNEL_SYSCALL(setresgid, 210)
IN_KERNEL_SYSCALL(gettid, 224)
IN_KERNEL_SYSCALL(tkill, 238)
IN_KERNEL_SYSCALL(futex, 240)
IN_KERNEL_SYSCALL(set_thread_area, 243)
IN_KERNEL_SYSCALL(get_thread_area, 244)
IN_KERNEL_SYSCALL(set_tid_address, 258)

static struct lx_sysent sysents[] = {
        {"nosys",       NULL,           NOSYS_NULL,     0},     /*  0 */
        {"exit",        lx_exit,        0,              1},     /*  1 */
        {"fork",        lx_fork,        0,              0},     /*  2 */
        {"read",        lx_read,        0,              3},     /*  3 */
        {"write",       write,          SYS_PASSTHRU,   3},     /*  4 */
```



```
        {"open",        lx_open,        0,              3},     /*  5 */
        {"close",       close,          SYS_PASSTHRU,   1},     /*  6 */
        {"waitpid",     lx_waitpid,     0,              3},     /*  7 */
        {"creat",       creat,          SYS_PASSTHRU,   2},     /*  8 */
        {"link",        lx_link,        0,              2},     /*  9 */
        {"unlink",      lx_unlink,      0,              1},     /* 10 */
        {"execve",      lx_execve,      0,              3},     /* 11 */
        {"chdir",       chdir,          SYS_PASSTHRU,   1},     /* 12 */
        {"time",        lx_time,        0,              1},     /* 13 */
        {"mknod",       lx_mknod,       0,              3},     /* 14 */
        {"chmod",       lx_chmod,       0,              2},     /* 15 */
        {"lchown16",    lx_lchown16,    0,              3},     /* 16 */
        {"break",       NULL,           NOSYS_OBSOLETE, 0},     /* 17 */
        {"stat",        NULL,           NOSYS_OBSOLETE, 0},     /* 18 */
        {"lseek",       lx_lseek,       0,              3},     /* 19 */
        {"getpid",      lx_getpid,      0,              0},     /* 20 */
        {"mount",       lx_mount,       0,              5},     /* 21 */
        {"umount",      lx_umount,      0,              1},     /* 22 */
        {"setuid16",    lx_setuid16,    0,              1},     /* 23 */
        {"getuid16",    lx_getuid16,    0,              0},     /* 24 */
        {"stime",       stime,          SYS_PASSTHRU,   1},     /* 25 */
        {"ptrace",      lx_ptrace,      0,              4},     /* 26 */
        {"alarm",       (int (*)())alarm, SYS_PASSTHRU, 1},     /* 27 */
        {"fstat",       NULL,           NOSYS_OBSOLETE, 0},     /* 28 */
        {"pause",       pause,          SYS_PASSTHRU,   0},     /* 29 */
        {"utime",       lx_utime,       0,              2},     /* 30 */
        {"stty",        NULL,           NOSYS_OBSOLETE, 0},     /* 31 */
        {"gtty",        NULL,           NOSYS_OBSOLETE, 0},     /* 32 */
        {"access",      access,         SYS_PASSTHRU,   2},     /* 33 */
        {"nice",        nice,           SYS_PASSTHRU,   1},     /* 34 */
        {"ftime",       NULL,           NOSYS_OBSOLETE, 0},     /* 35 */
        {"sync",        lx_sync,        0,              0},     /* 36 */
        {"kill",        lx_kill,        0,              2},     /* 37 */
        {"rename",      lx_rename,      0,              2},     /* 38 */
        {"mkdir",       mkdir,          SYS_PASSTHRU,   2},     /* 39 */
        {"rmdir",       lx_rmdir,       0,              1},     /* 40 */
        {"dup",         dup,            SYS_PASSTHRU,   1},     /* 41 */
        {"pipe",        lx_pipe,        0,              1},     /* 42 */
        {"times",       lx_times,       0,              1},     /* 43 */
        {"prof",        NULL,           NOSYS_OBSOLETE, 0},     /* 44 */
        {"brk",         lx_brk,         0,              1},     /* 45 */
        {"setgid16",    lx_setgid16,    0,              1},     /* 46 */
        {"getgid16",    lx_getgid16,    0,              0},     /* 47 */
        {"signal",      lx_signal,      0,              2},     /* 48 */
        {"geteuid16",   lx_geteuid16,   0,              0},     /* 49 */
        {"getegid16",   lx_getegid16,   0,              0},     /* 50 */
        {"acct",        NULL,           NOSYS_NO_EQUIV, 0},     /* 51 */
        {"umount2",     lx_umount2,     0,              2},     /* 52 */
        {"lock",        NULL,           NOSYS_OBSOLETE, 0},     /* 53 */
        {"ioctl",       lx_ioctl,       0,              3},     /* 54 */
        {"fcntl",       lx_fcntl,       0,              3},     /* 55 */
        {"mpx",         NULL,           NOSYS_OBSOLETE, 0},     /* 56 */
        {"setpgid",     lx_setpgid,     0,              2},     /* 57 */
        {"ulimit",      NULL,           NOSYS_OBSOLETE, 0},     /* 58 */
        {"olduname",    NULL,           NOSYS_OBSOLETE, 0},     /* 59 */
        {"umask",       (int (*)())umask, SYS_PASSTHRU, 1},     /* 60 */
        {"chroot",      chroot,         SYS_PASSTHRU,   1},     /* 61 */
        {"ustat",       lx_ustat,       0,              2},     /* 62 */
        {"dup2",        lx_dup2,        0,              2},     /* 63 */
        {"getppid",     lx_getppid,     0,              0},     /* 64 */
        {"getpgrp",     lx_getpgrp,     0,              0},     /* 65 */
        {"setsid",      lx_setsid,      0,              0},     /* 66 */
        {"sigaction",   lx_sigaction,   0,              3},     /* 67 */
        {"sgetmask",    NULL,           NOSYS_OBSOLETE, 0},     /* 68 */
        {"ssetmask",    NULL,           NOSYS_OBSOLETE, 0},     /* 69 */
        {"setreuid16",  lx_setreuid16,  0,              2},     /* 70 */
```

```
1051        {"setregid16",  lx_setregid16,  0,              2},     /* 71 */
1052        {"sigsuspend",  lx_sigsuspend,  0,              1},     /* 72 */
1053        {"sigpending",  lx_sigpending,  0,              1},     /* 73 */
1054        {"sethostname", lx_sethostname, 0,              2},     /* 74 */
1055        {"setrlimit",   lx_setrlimit,   0,              2},     /* 75 */
1056        {"getrlimit",   lx_oldgetrlimit, 0,             2},     /* 76 */
1057        {"getrusage",   lx_getrusage,   0,              2},     /* 77 */
1058        {"gettimeofday", lx_gettimeofday, 0,            2},     /* 78 */
1059        {"settimeofday", lx_settimeofday, 0,            2},     /* 79 */
1060        {"getgroups16", lx_getgroups16, 0,              2},     /* 80 */
1061        {"setgroups16", lx_setgroups16, 0,              2},     /* 81 */
1062        {"select",      NULL,           NOSYS_OBSOLETE, 0},     /* 82 */
1063        {"symlink",     symlink,        SYS_PASSTHRU,   2},     /* 83 */
1064        {"oldlstat",    NULL,           NOSYS_OBSOLETE, 0},     /* 84 */
1065        {"readlink",    readlink,       SYS_PASSTHRU,   3},     /* 85 */
1066        {"uselib",      NULL,           NOSYS_KERNEL,   0},     /* 86 */
1067        {"swapon",      NULL,           NOSYS_KERNEL,   0},     /* 87 */
1068        {"reboot",      lx_reboot,      0,              4},     /* 88 */
1069        {"readdir",     lx_readdir,     0,              3},     /* 89 */
1070        {"mmap",        lx_mmap,        0,              6},     /* 90 */
1071        {"munmap",      munmap,         SYS_PASSTHRU,   2},     /* 91 */
1072        {"truncate",    lx_truncate,    0,              2},     /* 92 */
1073        {"ftruncate",   lx_ftruncate,   0,              2},     /* 93 */
1074        {"fchmod",      fchmod,         SYS_PASSTHRU,   2},     /* 94 */
1075        {"fchown16",    lx_fchown16,    0,              3},     /* 95 */
1076        {"getpriority", lx_getpriority, 0,              2},     /* 96 */
1077        {"setpriority", lx_setpriority, 0,              3},     /* 97 */
1078        {"profil",      NULL,           NOSYS_NO_EQUIV, 0},     /* 98 */
1079        {"statfs",      lx_statfs,      0,              2},     /* 99 */
1080        {"fstatfs",     lx_fstatfs,     0,              2},     /* 100 */
1081        {"ioperm",      NULL,           NOSYS_NO_EQUIV, 0},     /* 101 */
1082        {"socketcall",  lx_socketcall,  0,              2},     /* 102 */
1083        {"syslog",      NULL,           NOSYS_KERNEL,   0},     /* 103 */
1084        {"setitimer",   lx_setitimer,   0,              3},     /* 104 */
1085        {"getitimer",   getitimer,      SYS_PASSTHRU,   2},     /* 105 */
1086        {"stat",        lx_stat,        0,              2},     /* 106 */
1087        {"lstat",       lx_lstat,       0,              2},     /* 107 */
1088        {"fstat",       lx_fstat,       0,              2},     /* 108 */
1089        {"uname",       NULL,           NOSYS_OBSOLETE, 0},     /* 109 */
1090        {"oldiopl",     NULL,           NOSYS_NO_EQUIV, 0},     /* 110 */
1091        {"vhangup",     lx_vhangup,     0,              0},     /* 111 */
1092        {"idle",        NULL,           NOSYS_NO_EQUIV, 0},     /* 112 */
1093        {"vm86old",     NULL,           NOSYS_OBSOLETE, 0},     /* 113 */
1094        {"wait4",       lx_wait4,       0,              4},     /* 114 */
1095        {"swapoff",     NULL,           NOSYS_KERNEL,   0},     /* 115 */
1096        {"sysinfo",     lx_sysinfo,     0,              1},     /* 116 */
1097        {"ipc",         lx_ipc,         0,              5},     /* 117 */
1098        {"fsync",       lx_fsync,       0,              1},     /* 118 */
1099        {"sigreturn",   lx_sigreturn,   0,              1},     /* 119 */
1100        {"clone",       lx_clone,       0,              5},     /* 120 */
1101        {"setdomainname", lx_setdomainname, 0,          2},     /* 121 */
1102        {"uname",       lx_uname,       0,              1},     /* 122 */
1103        {"modify_ldt",  lx_modify_ldt,  0,              3},     /* 123 */
1104        {"adjtimex",    lx_adjtimex,    0,              1},     /* 124 */
1105        {"mprotect",    lx_mprotect,    0,              3},     /* 125 */
1106        {"sigprocmask", lx_sigprocmask, 0,              3},     /* 126 */
1107        {"create_module", NULL,         NOSYS_KERNEL,   0},     /* 127 */
1108        {"init_module", NULL,           NOSYS_KERNEL,   0},     /* 128 */
1109        {"delete_module", NULL,         NOSYS_KERNEL,   0},     /* 129 */
1110        {"get_kernel_syms", NULL,       NOSYS_KERNEL,   0},     /* 130 */
1111        {"quotactl",    NULL,           NOSYS_KERNEL,   0},     /* 131 */
1112        {"getpgid",     lx_getpgid,     0,              1},     /* 132 */
1113        {"fchdir",      fchdir,         SYS_PASSTHRU,   1},     /* 133 */
1114        {"bdflush",     NULL,           NOSYS_KERNEL,   0},     /* 134 */
1115        {"sysfs",       lx_sysfs,       0,              3},     /* 135 */
1116        {"personality", lx_personality, 0,              1},     /* 136 */
```



```
1117        {"afs_syscall", NULL,           NOSYS_KERNEL,   0},     /* 137 */
1118        {"setfsuid16",  lx_setfsuid16,  0,              1},     /* 138 */
1119        {"setfsgid16",  lx_setfsgid16,  0,              1},     /* 139 */
1120        {"llseek",      lx_llseek,      0,              5},     /* 140 */
1121        {"getdents",    getdents,       SYS_PASSTHRU,   3},     /* 141 */
1122        {"select",      lx_select,      0,              5},     /* 142 */
1123        {"flock",       lx_flock,       0,              2},     /* 143 */
1124        {"msync",       lx_msync,       0,              3},     /* 144 */
1125        {"readv",       lx_readv,       0,              3},     /* 145 */
1126        {"writev",      lx_writev,      0,              3},     /* 146 */
1127        {"getsid",      lx_getsid,      0,              1},     /* 147 */
1128        {"fdatasync",   lx_fdatasync,   0,              1},     /* 148 */
1129        {"sysctl",      lx_sysctl,      0,              1},     /* 149 */
1130        {"mlock",       lx_mlock,       0,              2},     /* 150 */
1131        {"munlock",     lx_munlock,     0,              2},     /* 151 */
1132        {"mlockall",    lx_mlockall,    0,              1},     /* 152 */
1133        {"munlockall",  lx_munlockall,  0,              0},     /* 153 */
1134        {"sched_setparam", lx_sched_setparam,   0,      2},     /* 154 */
1135        {"sched_getparam", lx_sched_getparam,   0,      2},     /* 155 */
1136        {"sched_setscheduler", lx_sched_setscheduler, 0, 3},    /* 156 */
1137        {"sched_getscheduler", lx_sched_getscheduler, 0, 1},    /* 157 */
1138        {"sched_yield", (int (*)())yield, SYS_PASSTHRU, 0},     /* 158 */
1139        {"sched_get_priority_max", lx_sched_get_priority_max, 0, 1}, /* 159 */
1140        {"sched_get_priority_min", lx_sched_get_priority_min, 0, 1}, /* 160 */
1141        {"sched_rr_get_interval", lx_sched_rr_get_interval, 0,  2},  /* 161 */
1142        {"nanosleep",   nanosleep,      SYS_PASSTHRU,   2},     /* 162 */
1143        {"mremap",      NULL,           NOSYS_NO_EQUIV, 0},     /* 163 */
1144        {"setresuid16", lx_setresuid16, 0,              3},     /* 164 */
1145        {"getresuid16", lx_getresuid16, 0,              3},     /* 165 */
1146        {"vm86",        NULL,           NOSYS_NO_EQUIV, 0},     /* 166 */
1147        {"query_module", lx_query_module, NOSYS_KERNEL, 5},     /* 167 */
1148        {"poll",        lx_poll,        0,              3},     /* 168 */
1149        {"nfsservctl",  NULL,           NOSYS_KERNEL,   0},     /* 169 */
1150        {"setresgid16", lx_setresgid16, 0,              3},     /* 170 */
1151        {"getresgid16", lx_getresgid16, 0,              3},     /* 171 */
1152        {"prctl",       NULL,           NOSYS_UNDOC,    0},     /* 172 */
1153        {"rt_sigreturn", lx_rt_sigreturn, 0,            0},     /* 173 */
1154        {"rt_sigaction", lx_rt_sigaction, 0,            4},     /* 174 */
1155        {"rt_sigprocmask", lx_rt_sigprocmask, 0,        4},     /* 175 */
1156        {"rt_sigpending", lx_rt_sigpending, 0,          2},     /* 176 */
1157        {"rt_sigtimedwait", lx_rt_sigtimedwait, 0,      4},     /* 177 */
1158        {"sigqueueinfo", NULL,          NOSYS_UNDOC,    0},     /* 178 */
1159        {"rt_sigsuspend", lx_rt_sigsuspend, 0,          2},     /* 179 */
1160        {"pread64",     lx_pread64,     0,              5},     /* 180 */
1161        {"pwrite64",    lx_pwrite64,    0,              5},     /* 181 */
1162        {"chown16",     lx_chown16,     0,              3},     /* 182 */
1163        {"getcwd",      lx_getcwd,      0,              2},     /* 183 */
1164        {"capget",      NULL,           NOSYS_NO_EQUIV, 0},     /* 184 */
1165        {"capset",      NULL,           NOSYS_NO_EQUIV, 0},     /* 185 */
1166        {"sigaltstack", lx_sigaltstack, 0,              2},     /* 186 */
1167        {"sendfile",    lx_sendfile,    0,              4},     /* 187 */
1168        {"getpmsg",     NULL,           NOSYS_OBSOLETE, 0},     /* 188 */
1169        {"putpmsg",     NULL,           NOSYS_OBSOLETE, 0},     /* 189 */
1170        {"vfork",       lx_vfork,       0,              0},     /* 190 */
1171        {"getrlimit",   lx_getrlimit,   0,              2},     /* 191 */
1172        {"mmap2",       lx_mmap2,       EBP_HAS_ARG6,   6},     /* 192 */
1173        {"truncate64",  lx_truncate64,  0,              3},     /* 193 */
1174        {"ftruncate64", lx_ftruncate64, 0,              3},     /* 194 */
1175        {"stat64",      lx_stat64,      0,              2},     /* 195 */
1176        {"lstat64",     lx_lstat64,     0,              2},     /* 196 */
1177        {"fstat64",     lx_fstat64,     0,              2},     /* 197 */
1178        {"lchown",      lchown,         SYS_PASSTHRU,   3},     /* 198 */
1179        {"getuid",      (int (*)())getuid, SYS_PASSTHRU, 0},    /* 199 */
1180        {"getgid",      (int (*)())getgid, SYS_PASSTHRU, 0},    /* 200 */
1181        {"geteuid",     lx_geteuid,     0,              0},     /* 201 */
1182        {"getegid",     lx_getegid,     0,              0},     /* 202 */
```

```
1183         {"setreuid",    setreuid,       SYS_PASSTHRU,   0},     /* 203 */
1184         {"setregid",    setregid,       SYS_PASSTHRU,   0},     /* 204 */
1185         {"getgroups",   getgroups,      SYS_PASSTHRU,   2},     /* 205 */
1186         {"setgroups",   lx_setgroups,   0,              2},     /* 206 */
1187         {"fchown",      lx_fchown,      0,              3},     /* 207 */
1188         {"setresuid",   lx_setresuid,   0,              3},     /* 208 */
1189         {"getresuid",   lx_getresuid,   0,              3},     /* 209 */
1190         {"setresgid",   lx_setresgid,   0,              3},     /* 210 */
1191         {"getresgid",   lx_getresgid,   0,              3},     /* 211 */
1192         {"chown",       lx_chown,       0,              3},     /* 212 */
1193         {"setuid",      setuid,         SYS_PASSTHRU,   1},     /* 213 */
1194         {"setgid",      setgid,         SYS_PASSTHRU,   1},     /* 214 */
1195         {"setfsuid",    lx_setfsuid,    0,              1},     /* 215 */
1196         {"setfsgid",    lx_setfsgid,    0,              1},     /* 216 */
1197         {"pivot_root",  NULL,           NOSYS_KERNEL,   0},     /* 217 */
1198         {"mincore",     mincore,        SYS_PASSTHRU,   3},     /* 218 */
1199         {"madvise",     lx_madvise,     0,              3},     /* 219 */
1200         {"getdents64",  lx_getdents64,  0,              3},     /* 220 */
1201         {"fcntl64",     lx_fcntl64,     0,              3},     /* 221 */
1202         {"tux",         NULL,           NOSYS_NO_EQUIV, 0},     /* 222 */
1203         {"security",    NULL,           NOSYS_NO_EQUIV, 0},     /* 223 */
1204         {"gettid",      lx_gettid,      0,              0},     /* 224 */
1205         {"readahead",   NULL,           NOSYS_NO_EQUIV, 0},     /* 225 */
1206         {"setxattr",    NULL,           NOSYS_NO_EQUIV, 0},     /* 226 */
1207         {"lsetxattr",   NULL,           NOSYS_NO_EQUIV, 0},     /* 227 */
1208         {"fsetxattr",   NULL,           NOSYS_NO_EQUIV, 0},     /* 228 */
1209         {"getxattr",    NULL,           NOSYS_NO_EQUIV, 0},     /* 229 */
1210         {"lgetxattr",   NULL,           NOSYS_NO_EQUIV, 0},     /* 230 */
1211         {"fgetxattr",   NULL,           NOSYS_NO_EQUIV, 0},     /* 231 */
1212         {"listxattr",   NULL,           NOSYS_NO_EQUIV, 0},     /* 232 */
1213         {"llistxattr",  NULL,           NOSYS_NO_EQUIV, 0},     /* 233 */
1214         {"flistxattr",  NULL,           NOSYS_NO_EQUIV, 0},     /* 234 */
1215         {"removexattr", NULL,           NOSYS_NO_EQUIV, 0},     /* 235 */
1216         {"lremovexattr", NULL,          NOSYS_NO_EQUIV, 0},     /* 236 */
1217         {"fremovexattr", NULL,          NOSYS_NO_EQUIV, 0},     /* 237 */
1218         {"tkill",       lx_tkill,       0,              2},     /* 238 */
1219         {"sendfile64",  lx_sendfile64,  0,              4},     /* 239 */
1220         {"futex",       lx_futex,       EBP_HAS_ARG6,   6},     /* 240 */
1221         {"sched_setaffinity",   lx_sched_setaffinity,   0, 3},  /* 241 */
1222         {"sched_getaffinity",   lx_sched_getaffinity,   0, 3},  /* 242 */
1223         {"set_thread_area", lx_set_thread_area, 0,      1},     /* 243 */
1224         {"get_thread_area", lx_get_thread_area, 0,      1},     /* 244 */
1225         {"io_setup",    NULL,           NOSYS_NO_EQUIV, 0},     /* 245 */
1226         {"io_destroy",  NULL,           NOSYS_NO_EQUIV, 0},     /* 246 */
1227         {"io_getevents", NULL,          NOSYS_NO_EQUIV, 0},     /* 247 */
1228         {"io_submit",   NULL,           NOSYS_NO_EQUIV, 0},     /* 248 */
1229         {"io_cancel",   NULL,           NOSYS_NO_EQUIV, 0},     /* 249 */
1230         {"fadvise64",   NULL,           NOSYS_UNDOC,    0},     /* 250 */
1231         {"nosys",       NULL,           0,              0},     /* 251 */
1232         {"group_exit",  lx_group_exit,  0,              1},     /* 252 */
1233         {"lookup_dcookie", NULL,        NOSYS_NO_EQUIV, 0},     /* 253 */
1234         {"epoll_create", NULL,          NOSYS_NO_EQUIV, 0},     /* 254 */
1235         {"epoll_ctl",   NULL,           NOSYS_NO_EQUIV, 0},     /* 255 */
1236         {"epoll_wait",  NULL,           NOSYS_NO_EQUIV, 0},     /* 256 */
1237         {"remap_file_pages", NULL,      NOSYS_NO_EQUIV, 0},     /* 257 */
1238         {"set_tid_address", lx_set_tid_address, 0,      1},     /* 258 */
1239         {"timer_create", NULL,          NOSYS_UNDOC,    0},     /* 259 */
1240         {"timer_settime", NULL,         NOSYS_UNDOC,    0},     /* 260 */
1241         {"timer_gettime", NULL,         NOSYS_UNDOC,    0},     /* 261 */
1242         {"timer_getoverrun", NULL,      NOSYS_UNDOC,    0},     /* 262 */
1243         {"timer_delete", NULL,          NOSYS_UNDOC,    0},     /* 263 */
1244         {"clock_settime", lx_clock_settime,     0,      2},     /* 264 */
1245         {"clock_gettime", lx_clock_gettime,     0,      2},     /* 265 */
1246         {"clock_getres", lx_clock_getres,       0,      2},     /* 266 */
1247         {"clock_nanosleep", lx_clock_nanosleep, 0,      4},     /* 267 */
1248         {"statfs64",    lx_statfs64,    0,              2},     /* 268 */
```



```
1249         {"fstatfs64",   lx_fstatfs64,   0,              2},     /* 269 */
1250         {"tgkill",      lx_tgkill,      0,              3},     /* 270 */

1252         /* The following system calls only exist in kernel 2.6 and greater */
1253         {"utimes",      utimes,         SYS_PASSTHRU,   2},     /* 271 */
1254         {"fadvise64_64", NULL,          NOSYS_NULL,     0},     /* 272 */
1255         {"vserver",     NULL,           NOSYS_NULL,     0},     /* 273 */
1256         {"mbind",       NULL,           NOSYS_NULL,     0},     /* 274 */
1257         {"get_mempolicy", NULL,         NOSYS_NULL,     0},     /* 275 */
1258         {"set_mempolicy", NULL,         NOSYS_NULL,     0},     /* 276 */
1259         {"mq_open",     NULL,           NOSYS_NULL,     0},     /* 277 */
1260         {"mq_unlink",   NULL,           NOSYS_NULL,     0},     /* 278 */
1261         {"mq_timedsend", NULL,          NOSYS_NULL,     0},     /* 279 */
1262         {"mq_timedreceive", NULL,       NOSYS_NULL,     0},     /* 280 */
1263         {"mq_notify",   NULL,           NOSYS_NULL,     0},     /* 281 */
1264         {"mq_getsetattr", NULL,         NOSYS_NULL,     0},     /* 282 */
1265         {"kexec_load",  NULL,           NOSYS_NULL,     0},     /* 283 */
1266         {"waitid",      lx_waitid,      0,              4},     /* 284 */
1267         {"sys_setaltroot", NULL,        NOSYS_NULL,     0},     /* 285 */
1268         {"add_key",     NULL,           NOSYS_NULL,     0},     /* 286 */
1269         {"request_key", NULL,           NOSYS_NULL,     0},     /* 287 */
1270         {"keyctl",      NULL,           NOSYS_NULL,     0},     /* 288 */
1271         {"ioprio_set",  NULL,           NOSYS_NULL,     0},     /* 289 */
1272         {"ioprio_get",  NULL,           NOSYS_NULL,     0},     /* 290 */
1273         {"inotify_init", NULL,          NOSYS_NULL,     0},     /* 291 */
1274         {"inotify_add_watch", NULL,     NOSYS_NULL,     0},     /* 292 */
1275         {"inotify_rm_watch", NULL,      NOSYS_NULL,     0},     /* 293 */
1276         {"migrate_pages", NULL,         NOSYS_NULL,     0},     /* 294 */
1277         {"openat",      lx_openat,      0,              4},     /* 295 */
1278         {"mkdirat",     lx_mkdirat,     0,              3},     /* 296 */
1279         {"mknodat",     lx_mknodat,     0,              4},     /* 297 */
1280         {"fchownat",    lx_fchownat,    0,              5},     /* 298 */
1281         {"futimesat",   lx_futimesat,   0,              3},     /* 299 */
1282         {"fstatat64",   lx_fstatat64,   0,              4},     /* 300 */
1283         {"unlinkat",    lx_unlinkat,    0,              3},     /* 301 */
1284         {"renameat",    lx_renameat,    0,              4},     /* 302 */
1285         {"linkat",      lx_linkat,      0,              5},     /* 303 */
1286         {"symlinkat",   lx_symlinkat,   0,              3},     /* 304 */
1287         {"readlinkat",  lx_readlinkat,  0,              4},     /* 305 */
1288         {"fchmodat",    lx_fchmodat,    0,              4},     /* 306 */
1289         {"faccessat",   lx_faccessat,   0,              4},     /* 307 */
1290         {"pselect6",    NULL,           NOSYS_NULL,     0},     /* 308 */
1291         {"ppoll",       NULL,           NOSYS_NULL,     0},     /* 309 */
1292         {"unshare",     NULL,           NOSYS_NULL,     0},     /* 310 */
1293         {"set_robust_list", NULL,       NOSYS_NULL,     0},     /* 311 */
1294         {"get_robust_list", NULL,       NOSYS_NULL,     0},     /* 312 */
1295         {"splice",      NULL,           NOSYS_NULL,     0},     /* 313 */
1296         {"sync_file_range", NULL,       NOSYS_NULL,     0},     /* 314 */
1297         {"tee",         NULL,           NOSYS_NULL,     0},     /* 315 */
1298         {"vmsplice",    NULL,           NOSYS_NULL,     0},     /* 316 */
1299         {"move_pages",  NULL,           NOSYS_NULL,     0},     /* 317 */
1300 };
1301 #endif /* ! codereview */
```

```
**********************************************************
   31875 Tue Jan 14 16:17:02 2014
new/usr/src/lib/brand/lx/lx_brand/common/lx_thunk_server.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

/*
 * The BrandZ Linux thunking server.
 *
 * The interfaces defined in this file form the server side of a bridge
 * to allow native solaris process to access Linux services.  Currently
 * the Linux services that is made accessible by these interfaces here
 * are:
 *      - Linux host <-> address naming services
 *      - Linux service <-> port naming services
 *      - Linux syslog
 *
 * Access to all these services is provided through a doors server.
 * Currently the only client of these interfaces and the process that
 * initially starts up the doors server is lx_thunk.so.
 *
 * lx_thunk.so is a native solaris library that is loaded into native
 * solaris process that need to run inside a Linux zone and have access
 * to Linux services.  When lx_thunk.so receives a request that requires
 * accessing Linux services it creates a "thunk server" process by
 * forking and executing the following shell script (which runs as
 * a native /bin/sh Linux process):
 *      /native/usr/lib/brand/lx/lx_thunk
 *
 * The first and only thing this shell script attempts to do is re-exec
 * itself.  The brand library will detect when this script attempts to
 * re-exec itself and take control of the process.  The exec() system
 * call made by the Linux shell will never return.
 *
 * At this point the process becomes a "thunk server" process.
 * The first thing it does is a bunch of initialization:
 *
 * - Sanity check that a file descriptor based communication mechanism
 *   needed talk to the parent process is correctly initialized.
```



```
 *
 * - Verify that two predetermined file descriptors are FIFOs.
 *   These FIFOs will be used to establish communications with
 *   the client program that spawned us and which will be sending
 *   us requests.
 *
 * - Use existing debugging libraries (libproc.so, librtld_db.so,
 *   and the BrandZ lx plug-in to librtld_db.so) and /native/proc to
 *   walk the Linux link maps in our own address space to determine
 *   the address of the Linux dlsym() function.
 *
 * - Use the native Linux dlsym() function to look up other symbols
 *   (for both functions and variables) that we will need access
 *   to service thunking requests.
 *
 * - Create a doors server and notify the parent process that we
 *   are ready to service requests.
 *
 * - Enter a service loop and wait for requests.
 *
 * At this point the lx_thunk process is ready to service door
 * based requests.  When door service request is received the
 * following happens inside the lx_thunk process:
 *
 * - The doors server function is is invoked on a new solaris thread
 *   that the kernel injects into the lx_thunk process.  We sanity
 *   check the incoming request, place it on a service queue, and
 *   wait for notification that the request has been completed.
 *
 * - A Linux thread takes this request off the service queue
 *   and dispatches it to a service function that will:
 *      - Decode the request.
 *      - Handle the request by invoking native Linux interfaces.
 *      - Encode the results for the request.
 *
 * - The Linux thread then notifies the requesting doors server
 *   thread that the  request has been completed and goes to sleep
 *   until it receives another request.
 *
 * - the solaris door server thread returns the results of the
 *   operation to the caller.
 *
 * Notes:
 *
 * - The service request hand off operation from the solaris doors thread to
 *   the "Linux thread" is required because only "Linux threads" can call
 *   into Linux code.  In this context a "Linux thread" is a thread that
 *   is either the initial thread of a Linux process or a thread that was
 *   created by calling the Linux version of thread_create().  The reason
 *   for this restriction is that any thread that invokes Linux code needs
 *   to have been initialized in the Linux threading libraries and have
 *   things like Linux thread local storage properly setup.
 *
 *   But under solaris all door server threads are created and destroyed
 *   dynamically.  This means that when a doors server function is invoked,
 *   it is invoked via a thread that hasn't been initialized in the Linux
 *   environment and there for can't call directly into Linux code.
 *
 * - Currently when a thunk server process is starting up, it communicated
 *   with it's parent via two FIFOs.  These FIFOs are setup by the
 *   lx_thunk.so library.  After creating the FIFOs and starting the lx_thunk
 *   server, lx_thunk.so writes the name of the file that the door should
 *   be attached to to the first pipe.  The lx_thunk server reads in this
 *   value, initialized the server, fattach()s it to the file request by
 *   lx_thunk.so and does a write to the second FIFO to let lx_thunk.so
 *   know that the server is ready to take requests.
```

```
 *
 *   This negotiation could be simplified to use only use one FIFO.
 *   lx_thunk.so would attempt to read from the FIFO and the lx_thunk
 *   server process could send the new door server file descriptor
 *   to this process via an I_SENDFD ioctl (see streamio.7I).
 *
 * - The lx_thunk server process will exit when the client process
 *   that it's handling requests for exists.  (ie, when there are no
 *   more open file handles to the doors server.)
 */

#include <assert.h>
#include <door.h>
#include <errno.h>
#include <libproc.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <sys/lx_debug.h>
#include <sys/lx_misc.h>
#include <sys/lx_thread.h>
#include <sys/lx_thunk_server.h>
#include <sys/varargs.h>
#include <thread.h>
#include <unistd.h>

/*
 * Generic interfaces used for looking up and calling Linux functions.
 */
typedef struct __lx_handle_dlsym        *lx_handle_dlsym_t;
typedef struct __lx_handle_sym          *lx_handle_sym_t;

uintptr_t lx_call0(lx_handle_sym_t);
uintptr_t lx_call1(lx_handle_sym_t, uintptr_t);
uintptr_t lx_call2(lx_handle_sym_t, uintptr_t, uintptr_t);
uintptr_t lx_call3(lx_handle_sym_t, uintptr_t, uintptr_t, uintptr_t);
uintptr_t lx_call4(lx_handle_sym_t, uintptr_t, uintptr_t, uintptr_t,
    uintptr_t);
uintptr_t lx_call5(lx_handle_sym_t, uintptr_t, uintptr_t, uintptr_t,
    uintptr_t, uintptr_t);
uintptr_t lx_call6(lx_handle_sym_t, uintptr_t, uintptr_t, uintptr_t,
    uintptr_t, uintptr_t, uintptr_t);
uintptr_t lx_call7(lx_handle_sym_t, uintptr_t, uintptr_t, uintptr_t,
    uintptr_t, uintptr_t, uintptr_t, uintptr_t);
uintptr_t lx_call8(lx_handle_sym_t, uintptr_t, uintptr_t, uintptr_t,
    uintptr_t, uintptr_t, uintptr_t, uintptr_t, uintptr_t);

/*
 * Flag indicating if this process is destined to become a thunking
 * server process.
 */
static int lxt_server_processes = 0;

/*
 * Linux function call defines and handles.
 */
static lx_handle_dlsym_t        lxh_init = NULL;

#define LXTH_GETHOSTBYNAME_R    0
#define LXTH_GETHOSTBYADDR_R    1
#define LXTH_GETSERVBYNAME_R    2
#define LXTH_GETSERVBYPORT_R    3
#define LXTH_OPENLOG            4
#define LXTH_SYSLOG             5
#define LXTH_CLOSELOG           6
#define LXTH_PROGNAME           7
```



```
static struct lxt_handles {
        int             lxth_index;
        char            *lxth_name;
        lx_handle_sym_t lxth_handle;
} lxt_handles[] = {
        { LXTH_GETHOSTBYNAME_R, "gethostbyname_r",      NULL },
        { LXTH_GETHOSTBYADDR_R, "gethostbyaddr_r",      NULL },
        { LXTH_GETSERVBYNAME_R, "getservbyname_r",      NULL },
        { LXTH_GETSERVBYPORT_R, "getservbyport_r",      NULL },
        { LXTH_OPENLOG,         "openlog",              NULL },
        { LXTH_SYSLOG,          "syslog",               NULL },
        { LXTH_CLOSELOG,        "closelog",             NULL },
        { LXTH_PROGNAME,        "__progname",           NULL },
        { -1,                   NULL,                   NULL },
};

/*
 * Door server operations dispatch functions and table.
 *
 * When the doors server get's a request for a particlar operation
 * this dispatch table controls what function will be invoked to
 * service the request.  The function is invoked via Linux thread
 * so that it can call into native Linux code if necessary.
 */
static void lxt_server_gethost(lxt_server_arg_t *request, size_t request_size,
    char **door_result, size_t *door_result_size);
static void lxt_server_getserv(lxt_server_arg_t *request, size_t request_size,
    char **door_result, size_t *door_result_size);
static void lxt_server_openlog(lxt_server_arg_t *request, size_t request_size,
    char **door_result, size_t *door_result_size);
static void lxt_server_syslog(lxt_server_arg_t *request, size_t request_size,
    char **door_result, size_t *door_result_size);
static void lxt_server_closelog(lxt_server_arg_t *request, size_t request_size,
    char **door_result, size_t *door_result_size);

typedef void (*lxt_op_func_t)(lxt_server_arg_t *request, size_t request_size,
    char **door_result, size_t *door_result_size);

static struct lxt_operations {
        int             lxto_index;
        lxt_op_func_t   lxto_fp;
} lxt_operations[] = {
        { LXT_SERVER_OP_PING,           NULL },
        { LXT_SERVER_OP_NAME2HOST,      lxt_server_gethost },
        { LXT_SERVER_OP_ADDR2HOST,      lxt_server_gethost },
        { LXT_SERVER_OP_NAME2SERV,      lxt_server_getserv },
        { LXT_SERVER_OP_PORT2SERV,      lxt_server_getserv },
        { LXT_SERVER_OP_OPENLOG,        lxt_server_openlog },
        { LXT_SERVER_OP_SYSLOG,         lxt_server_syslog },
        { LXT_SERVER_OP_CLOSELOG,       lxt_server_closelog },
};

/*
 * Structures for passing off requests from doors threads (which are
 * solaris threads) to a Linux thread that that can handle them.
 */
typedef struct lxt_req {
        lxt_server_arg_t        *lxtr_request;
        size_t                  lxtr_request_size;
        char                    *lxtr_result;
        size_t                  lxtr_result_size;
        int                     lxtr_complete;
        cond_t                  lxtr_complete_cv;
} lxt_req_t;
```

```
260 static mutex_t          lxt_req_lock = DEFAULTMUTEX;
261 static cond_t           lxt_req_cv = DEFAULTCV;
262 static lxt_req_t        *lxt_req_ptr = NULL;

264 static mutex_t          lxt_pid_lock = DEFAULTMUTEX;
265 static pid_t            lxt_pid = NULL;

267 /*
268  * Interfaces used to call from lx_brand.so into Linux code.
269  */
270 typedef struct lookup_cb_arg {
271         struct ps_prochandle    *lca_ph;
272         caddr_t                 lca_ptr;
273 } lookup_cb_arg_t;

275 static int
276 /*ARGSUSED*/
277 lookup_cb(void *data, const prmap_t *pmp, const char *object)
278 {
279         lookup_cb_arg_t         *lcap = (lookup_cb_arg_t *)data;
280         prsyminfo_t             si;
281         GElf_Sym                sym;

283         if (Pxlookup_by_name(lcap->lca_ph,
284             LM_ID_BASE, object, "dlsym", &sym, &si) != 0)
285                 return (0);

287         if (sym.st_shndx == SHN_UNDEF)
288                 return (0);

290         /*
291          * XXX: we should be more paranoid and verify that the symbol
292          * we just looked up is libdl.so.2‘dlsym
293          */
294         lcap->lca_ptr = (caddr_t)(uintptr_t)sym.st_value;
295         return (1);
296 }

298 lx_handle_dlsym_t
299 lx_call_init(void)
300 {
301         struct ps_prochandle    *ph;
302         lookup_cb_arg_t         lca;
303         extern int              __libc_threaded;
304         int                     err;

306         lx_debug("lx_call_init(): looking up Linux dlsym");

308         /*
309          * The handle is really the address of the Linux "dlsym" function.
310          * Once we have this address we can call into the Linux "dlsym"
311          * function to lookup other functions.  It’s the initial lookup
312          * of "dlsym" that’s difficult.  To do this we’ll leverage the
313          * brand support that we added to librtld_db.  We’re going
314          * to fire up a seperate native solaris process that will
315          * attach to us via libproc/librtld_db and lookup the symbol
316          * for us.
317          */

319         /* Make sure we’re single threaded. */
320         if (__libc_threaded) {
321                 lx_debug("lx_call_init() fail: "
322                     "process must be single threaded");
323                 return (NULL);
324         }
```



```
326         /* Tell libproc.so where the real procfs is mounted. */
327         Pset_procfs_path("/native/proc");

329         /* Tell librtld_db.so where the real /native is */
330         (void) rd_ctl(RD_CTL_SET_HELPPATH, "/native");

332         /* Grab ourselves but don’t stop ourselves. */
333         if ((ph = Pgrab(getpid(),
334             PGRAB_FORCE | PGRAB_RDONLY | PGRAB_NOSTOP, &err)) == NULL) {
335                 lx_debug("lx_call_init() fail: Pgrab failed: %s",
336                     Pgrab_error(err));
337                 return (NULL);
338         }

340         lca.lca_ph = ph;
341         if (Pobject_iter(ph, lookup_cb, &lca) == -1) {
342                 lx_debug("lx_call_init() fail: couldn’t find Linux dlsym");
343                 return (NULL);
344         }

346         lx_debug("lx_call_init(): Linux dlsym = 0x%p", lca.lca_ptr);
347         return ((lx_handle_dlsym_t)lca.lca_ptr);
348 }

350 #define LX_RTLD_DEFAULT         ((void *)0)
351 #define LX_RTLD_NEXT            ((void *) -1l)

353 lx_handle_sym_t
354 lx_call_dlsym(lx_handle_dlsym_t lxh_dlsym, const char *str)
355 {
356         lx_handle_sym_t result;
357         lx_debug("lx_call_dlsym: calling Linux dlsym for: %s", str);
358         result = (lx_handle_sym_t)lx_call2((lx_handle_sym_t)lxh_dlsym,
359             (uintptr_t)LX_RTLD_DEFAULT, (uintptr_t)str);
360         lx_debug("lx_call_dlsym: Linux sym: \"%s\" = 0x%p", str, result);
361         return (result);
362 }

364 static uintptr_t
365 /*ARGSUSED*/
366 lx_call(lx_handle_sym_t lx_ch, uintptr_t p1, uintptr_t p2,
367     uintptr_t p3, uintptr_t p4, uintptr_t p5, uintptr_t p6, uintptr_t p7,
368     uintptr_t p8)
369 {
370         typedef uintptr_t       (*fp8_t)(uintptr_t, uintptr_t, uintptr_t,
371             uintptr_t, uintptr_t, uintptr_t, uintptr_t, uintptr_t);
372         lx_regs_t               *rp;
373         uintptr_t               ret;
374         fp8_t                   lx_funcp = (fp8_t)lx_ch;
375         long                    cur_gs;

377         rp = lx_syscall_regs();

379         lx_debug("lx_call: calling to Linux code at 0x%p", lx_ch);
380         lx_debug("lx_call: loading Linux gs, rp = 0x%p, gs = 0x%p",
381             rp, rp->lxr_gs);

383         lx_swap_gs(rp->lxr_gs, &cur_gs);
384         ret = lx_funcp(p1, p2, p3, p4, p5, p6, p7, p8);
385         lx_swap_gs(cur_gs, &rp->lxr_gs);

387         lx_debug("lx_call: returned from Linux code at 0x%p (%p)", lx_ch, ret);
388         lx_debug("lx_call: restored solaris gs 0x%p", cur_gs);
389         return (ret);
390 }
```

```
392 uintptr_t
393 lx_call0(lx_handle_sym_t lx_ch)
394 {
395         return (lx_call(lx_ch, 0, 0, 0, 0, 0, 0, 0, 0));
396 }

398 uintptr_t
399 lx_call1(lx_handle_sym_t lx_ch, uintptr_t p1)
400 {
401         return (lx_call(lx_ch, p1, 0, 0, 0, 0, 0, 0, 0));
402 }

404 uintptr_t
405 lx_call2(lx_handle_sym_t lx_ch, uintptr_t p1, uintptr_t p2)
406 {
407         return (lx_call(lx_ch, p1, p2, 0, 0, 0, 0, 0, 0));
408 }

410 uintptr_t
411 lx_call3(lx_handle_sym_t lx_ch, uintptr_t p1, uintptr_t p2, uintptr_t p3)
412 {
413         return (lx_call(lx_ch, p1, p2, p3, 0, 0, 0, 0, 0));
414 }

416 uintptr_t
417 lx_call4(lx_handle_sym_t lx_ch, uintptr_t p1, uintptr_t p2, uintptr_t p3,
418     uintptr_t p4)
419 {
420         return (lx_call(lx_ch, p1, p2, p3, p4, 0, 0, 0, 0));
421 }

423 uintptr_t
424 lx_call5(lx_handle_sym_t lx_ch, uintptr_t p1, uintptr_t p2, uintptr_t p3,
425     uintptr_t p4, uintptr_t p5)
426 {
427         return (lx_call(lx_ch, p1, p2, p3, p4, p5, 0, 0, 0));
428 }

430 uintptr_t
431 lx_call6(lx_handle_sym_t lx_ch, uintptr_t p1, uintptr_t p2, uintptr_t p3,
432     uintptr_t p4, uintptr_t p5, uintptr_t p6)
433 {
434         return (lx_call(lx_ch, p1, p2, p3, p4, p5, p6, 0, 0));
435 }

437 uintptr_t
438 lx_call7(lx_handle_sym_t lx_ch, uintptr_t p1, uintptr_t p2, uintptr_t p3,
439     uintptr_t p4, uintptr_t p5, uintptr_t p6, uintptr_t p7)
440 {
441         return (lx_call(lx_ch, p1, p2, p3, p4, p5, p6, p7, 0));
442 }

444 uintptr_t
445 lx_call8(lx_handle_sym_t lx_ch, uintptr_t p1, uintptr_t p2, uintptr_t p3,
446     uintptr_t p4, uintptr_t p5, uintptr_t p6, uintptr_t p7, uintptr_t p8)
447 {
448         return (lx_call(lx_ch, p1, p2, p3, p4, p5, p6, p7, p8));
449 }

451 /*
452  * Linux Thunking Interfaces - Server Side
453  */
454 static int
455 lxt_gethost_arg_check(lxt_gethost_arg_t *x, int x_size)
456 {
457         if (x_size != sizeof (*x) + x->lxt_gh_buf_len - 1)
```



```
458                 return (-1);

460         if ((x->lxt_gh_token_len < 0) || (x->lxt_gh_buf_len < 0))
461                 return (-1);

463         /* Token and buf should use up all the storage. */
464         if ((x->lxt_gh_token_len + x->lxt_gh_buf_len) != x->lxt_gh_storage_len)
465                 return (-1);

467         return (0);
468 }

470 static void
471 lxt_server_gethost(lxt_server_arg_t *request, size_t request_size,
472     char **door_result, size_t *door_result_size)
473 {
474         lxt_gethost_arg_t       *data;
475         struct hostent          *result, *rv;
476         int                     token_len, buf_len, type, data_size, i;
477         char                    *token, *buf;
478         int                     h_errnop;

480         assert((request->lxt_sa_op == LXT_SERVER_OP_NAME2HOST) ||
481             (request->lxt_sa_op == LXT_SERVER_OP_ADDR2HOST));

483         /*LINTED*/
484         data = (lxt_gethost_arg_t *)&request->lxt_sa_data[0];
485         data_size = request_size - sizeof (*request) - 1;

487         if (!lxt_gethost_arg_check(data, data_size)) {
488                 lx_debug("lxt_server_gethost: invalid request");
489                 *door_result = NULL;
490                 *door_result_size = 0;
491                 return;
492         }

494         /* Unpack the arguments. */
495         type = data->lxt_gh_type;
496         token = &data->lxt_gh_storage[0];
497         token_len = data->lxt_gh_token_len;
498         result = &data->lxt_gh_result;
499         buf = &data->lxt_gh_storage[data->lxt_gh_token_len];
500         buf_len = data->lxt_gh_buf_len - data->lxt_gh_token_len;

502         if (request->lxt_sa_op == LXT_SERVER_OP_NAME2HOST) {
503                 (void) lx_call6(lxt_handles[LXTH_GETHOSTBYNAME_R].lxth_handle,
504                     (uintptr_t)token, (uintptr_t)result,
505                     (uintptr_t)buf, buf_len, (uintptr_t)&rv,
506                     (uintptr_t)&h_errnop);
507         } else {
508                 (void) lx_call8(lxt_handles[LXTH_GETHOSTBYADDR_R].lxth_handle,
509                     (uintptr_t)token, token_len, type, (uintptr_t)result,
510                     (uintptr_t)buf, buf_len, (uintptr_t)&rv,
511                     (uintptr_t)&h_errnop);
512         }

514         if (rv == NULL) {
515                 /* the lookup failed */
516                 request->lxt_sa_success = 0;
517                 request->lxt_sa_errno = errno;
518                 data->lxt_gh_h_errno = h_errnop;
519                 *door_result = (char *)request;
520                 *door_result_size = request_size;
521                 return;
522         }
523         request->lxt_sa_success = 1;
```

```
524         request->lxt_sa_errno = 0;
525         data->lxt_gh_h_errno = 0;

527         /*
528          * The result structure that we would normally return contains a
529          * bunch of pointers, but those pointers are useless to our caller
530          * since they are in a different address space.  So before returning
531          * we'll convert all the result pointers into offsets.  The caller
532          * can then map the offsets back into pointers.
533          */
534         for (i = 0; result->h_aliases[i] != NULL; i++) {
535                 result->h_aliases[i] =
536                     LXT_PTR_TO_OFFSET(result->h_aliases[i], buf);
537         }
538         for (i = 0; result->h_addr_list[i] != NULL; i++) {
539                 result->h_addr_list[i] =
540                     LXT_PTR_TO_OFFSET(result->h_addr_list[i], buf);
541         }
542         result->h_name = LXT_PTR_TO_OFFSET(result->h_name, buf);
543         result->h_aliases = LXT_PTR_TO_OFFSET(result->h_aliases, buf);
544         result->h_addr_list = LXT_PTR_TO_OFFSET(result->h_addr_list, buf);

546         *door_result = (char *)request;
547         *door_result_size = request_size;
548 }

550 static int
551 lxt_getserv_arg_check(lxt_getserv_arg_t *x, int x_size)
552 {
553         if (x_size != sizeof (*x) + x->lxt_gs_buf_len - 1)
554                 return (-1);

556         if ((x->lxt_gs_token_len < 0) || (x->lxt_gs_buf_len < 0))
557                 return (-1);

559         /* Token and buf should use up all the storage. */
560         if ((x->lxt_gs_token_len + x->lxt_gs_buf_len) != x->lxt_gs_storage_len)
561                 return (-1);

563         return (0);
564 }

566 static void
567 lxt_server_getserv(lxt_server_arg_t *request, size_t request_size,
568     char **door_result, size_t *door_result_size)
569 {
570         lxt_getserv_arg_t       *data;
571         struct servent          *result, *rv;
572         int                     token_len, buf_len, data_size, i, port;
573         char                    *token, *buf, *proto = NULL;

575         assert((request->lxt_sa_op == LXT_SERVER_OP_NAME2SERV) ||
576             (request->lxt_sa_op == LXT_SERVER_OP_PORT2SERV));

578         /*LINTED*/
579         data = (lxt_getserv_arg_t *)&request->lxt_sa_data[0];
580         data_size = request_size - sizeof (*request) - 1;

582         if (!lxt_getserv_arg_check(data, data_size)) {
583                 lx_debug("lxt_server_getserv: invalid request");
584                 *door_result = NULL;
585                 *door_result_size = 0;
586                 return;
587         }

589         /* Unpack the arguments. */
```



```
590         token = &data->lxt_gs_storage[0];
591         token_len = data->lxt_gs_token_len;
592         result = &data->lxt_gs_result;
593         buf = &data->lxt_gs_storage[data->lxt_gs_token_len];
594         buf_len = data->lxt_gs_buf_len - data->lxt_gs_token_len;
595         if (strlen(data->lxt_gs_proto) > 0)
596                 proto = data->lxt_gs_proto;

598         /* Do more sanity checks */
599         if ((request->lxt_sa_op == LXT_SERVER_OP_PORT2SERV) &&
600             (token_len != sizeof (int))) {
601                 lx_debug("lxt_server_getserv: invalid request");
602                 *door_result = NULL;
603                 *door_result_size = 0;
604                 return;
605         }

607         if (request->lxt_sa_op == LXT_SERVER_OP_NAME2SERV) {
608                 (void) lx_call6(lxt_handles[LXTH_GETSERVBYNAME_R].lxth_handle,
609                     (uintptr_t)token, (uintptr_t)proto, (uintptr_t)result,
610                     (uintptr_t)buf, buf_len, (uintptr_t)&rv);
611         } else {
612                 bcopy(token, &port, sizeof (int));
613                 (void) lx_call6(lxt_handles[LXTH_GETSERVBYPORT_R].lxth_handle,
614                     port, (uintptr_t)proto, (uintptr_t)result,
615                     (uintptr_t)buf, buf_len, (uintptr_t)&rv);
616         }

618         if (rv == NULL) {
619                 /* the lookup failed */
620                 request->lxt_sa_success = 0;
621                 request->lxt_sa_errno = errno;
622                 *door_result = (char *)request;
623                 *door_result_size = request_size;
624                 return;
625         }
626         request->lxt_sa_success = 1;
627         request->lxt_sa_errno = 0;

629         /*
630          * The result structure that we would normally return contains a
631          * bunch of pointers, but those pointers are useless to our caller
632          * since they are in a different address space.  So before returning
633          * we'll convert all the result pointers into offsets.  The caller
634          * can then map the offsets back into pointers.
635          */
636         for (i = 0; result->s_aliases[i] != NULL; i++) {
637                 result->s_aliases[i] =
638                     LXT_PTR_TO_OFFSET(result->s_aliases[i], buf);
639         }
640         result->s_proto = LXT_PTR_TO_OFFSET(result->s_proto, buf);
641         result->s_aliases = LXT_PTR_TO_OFFSET(result->s_aliases, buf);
642         result->s_name = LXT_PTR_TO_OFFSET(result->s_name, buf);

644         *door_result = (char *)request;
645         *door_result_size = request_size;
646 }

648 static void
649 /*ARGSUSED*/
650 lxt_server_openlog(lxt_server_arg_t *request, size_t request_size,
651     char **door_result, size_t *door_result_size)
652 {
653         lxt_openlog_arg_t       *data;
654         int                     data_size;
655         static char             ident[128];
```

```
657         assert(request->lxt_sa_op == LXT_SERVER_OP_OPENLOG);

659         /*LINTED*/
660         data = (lxt_openlog_arg_t *)&request->lxt_sa_data[0];
661         data_size = request_size - sizeof (*request);

663         if (data_size != sizeof (*data)) {
664                 lx_debug("lxt_server_openlog: invalid request");
665                 *door_result = NULL;
666                 *door_result_size = 0;
667                 return;
668         }

670         /*
671          * Linux expects that the ident pointer passed to openlog()
672          * points to a static string that won't go away.  Linux
673          * saves the pointer and references with syslog() is called.
674          * Hence we'll make a local copy of the ident string here.
675          */
676         (void) mutex_lock(&lxt_pid_lock);
677         (void) strlcpy(ident, data->lxt_ol_ident, sizeof (ident));
678         (void) mutex_unlock(&lxt_pid_lock);

680         /* Call Linx openlog(). */
681         (void) lx_call3(lxt_handles[LXTH_OPENLOG].lxth_handle,
682             (uintptr_t)ident, data->lxt_ol_logopt, data->lxt_ol_facility);

684         request->lxt_sa_success = 1;
685         request->lxt_sa_errno = 0;
686         *door_result = (char *)request;
687         *door_result_size = request_size;
688 }

690 static void
691 /*ARGSUSED*/
692 lxt_server_syslog(lxt_server_arg_t *request, size_t request_size,
693     char **door_result, size_t *door_result_size)
694 {
695         lxt_syslog_arg_t        *data;
696         int                     data_size;
697         char                    *progname_ptr_new;
698         char                    *progname_ptr_old;

700         assert(request->lxt_sa_op == LXT_SERVER_OP_SYSLOG);

702         /*LINTED*/
703         data = (lxt_syslog_arg_t *)&request->lxt_sa_data[0];
704         data_size = request_size - sizeof (*request);

706         if (data_size != sizeof (*data)) {
707                 lx_debug("lxt_server_openlog: invalid request");
708                 *door_result = NULL;
709                 *door_result_size = 0;
710                 return;
711         }
712         progname_ptr_new = data->lxt_sl_progname;

714         (void) mutex_lock(&lxt_pid_lock);

716         /*
717          * Ensure the message has the correct pid.
718          * We do this by telling our getpid() system call to return a
719          * different value.
720          */
721         lxt_pid = data->lxt_sl_pid;
```



```
723         /*
724          * Ensure the message has the correct program name.
725          * Normally instead of a program name an "ident" string is
726          * used, this is the string passed to openlog().  But if
727          * openlog() wasn't called before syslog() then Linux
728          * syslog() will attempt to use the program name as
729          * the ident string, and the program name is determined
730          * by looking at the __progname variable.  So we'll just
731          * update the Linux __progname variable while we do the
732          * call.
733          */
734         (void) uucopy(lxt_handles[LXTH_PROGNAME].lxth_handle,
735             &progname_ptr_old, sizeof (char *));
736         (void) uucopy(&progname_ptr_new,
737             lxt_handles[LXTH_PROGNAME].lxth_handle, sizeof (char *));

739         /* Call Linux syslog(). */
740         (void) lx_call2(lxt_handles[LXTH_SYSLOG].lxth_handle,
741             data->lxt_sl_priority, (uintptr_t)data->lxt_sl_message);

743         /* Restore pid and program name. */
744         (void) uucopy(&progname_ptr_old,
745             lxt_handles[LXTH_PROGNAME].lxth_handle, sizeof (char *));
746         lxt_pid = NULL;

748         (void) mutex_unlock(&lxt_pid_lock);

750         request->lxt_sa_success = 1;
751         request->lxt_sa_errno = 0;
752         *door_result = (char *)request;
753         *door_result_size = request_size;
754 }

756 static void
757 /*ARGSUSED*/
758 lxt_server_closelog(lxt_server_arg_t *request, size_t request_size,
759     char **door_result, size_t *door_result_size)
760 {
761         int                     data_size;

763         assert(request->lxt_sa_op == LXT_SERVER_OP_CLOSELOG);

765         data_size = request_size - sizeof (*request);
766         if (data_size != 0) {
767                 lx_debug("lxt_server_closelog: invalid request");
768                 *door_result = NULL;
769                 *door_result_size = 0;
770                 return;
771         }

773         /* Call Linux closelog(). */
774         (void) lx_call0(lxt_handles[LXTH_CLOSELOG].lxth_handle);

776         request->lxt_sa_success = 1;
777         request->lxt_sa_errno = 0;
778         *door_result = (char *)request;
779         *door_result_size = request_size;
780 }

782 static void
783 /*ARGSUSED*/
784 lxt_server(void *cookie, char *argp, size_t request_size,
785     door_desc_t *dp, uint_t n_desc)
786 {
787         /*LINTED*/
```

```
788         lxt_server_arg_t        *request = (lxt_server_arg_t *)argp;
789         lxt_req_t               lxt_req;
790         char                    *door_path = cookie;

792         /* Check if there's no callers left */
793         if (argp == DOOR_UNREF_DATA) {
794                 (void) fdetach(door_path);
795                 (void) unlink(door_path);
796                 lx_debug("lxt_thunk_server: no clients, exiting");
797                 exit(0);
798         }

800         /* Sanity check the incomming request. */
801         if (request_size < sizeof (*request)) {
802                 /* the lookup failed */
803                 lx_debug("lxt_thunk_server: invalid request size");
804                 (void) door_return(NULL, 0, NULL, 0);
805                 return;
806         }

808         if ((request->lxt_sa_op < LXT_SERVER_OP_MIN) ||
809             (request->lxt_sa_op > LXT_SERVER_OP_MAX)) {
810                 lx_debug("lxt_thunk_server: invalid request op");
811                 (void) door_return(NULL, 0, NULL, 0);
812                 return;
813         }

815         /* Handle ping requests immediatly, return here. */
816         if (request->lxt_sa_op == LXT_SERVER_OP_PING) {
817                 lx_debug("lxt_thunk_server: handling ping request");
818                 request->lxt_sa_success = 1;
819                 (void) door_return((char *)request, request_size, NULL, 0);
820                 return;
821         }

823         lx_debug("lxt_thunk_server: hand off request to Linux thread, "
824             "request = 0x%p", request);

826         /* Pack the request up so we can pass it to a Linux thread. */
827         lxt_req.lxtr_request = request;
828         lxt_req.lxtr_request_size = request_size;
829         lxt_req.lxtr_result = NULL;
830         lxt_req.lxtr_result_size = 0;
831         lxt_req.lxtr_complete = 0;
832         (void) cond_init(&lxt_req.lxtr_complete_cv, USYNC_THREAD, NULL);

834         /* Pass the request onto a Linux thread. */
835         (void) mutex_lock(&lxt_req_lock);
836         while (lxt_req_ptr != NULL)
837                 (void) cond_wait(&lxt_req_cv, &lxt_req_lock);
838         lxt_req_ptr = &lxt_req;
839         (void) cond_broadcast(&lxt_req_cv);

841         /* Wait for the request to be completed. */
842         while (lxt_req.lxtr_complete == 0)
843                 (void) cond_wait(&lxt_req.lxtr_complete_cv, &lxt_req_lock);
844         assert(lxt_req_ptr != &lxt_req);
845         (void) mutex_unlock(&lxt_req_lock);

847         lx_debug("lxt_thunk_server: hand off request completed, "
848             "request = 0x%p", request);

850         /*
851          * If door_return() is successfull it never returns, so if we made
852          * it here there was some kind of error, but there's nothing we can
853          * really do about it.
```



```
854          */
855         (void) door_return(
856             lxt_req.lxtr_result, lxt_req.lxtr_result_size, NULL, 0);
857 }

859 static void
860 lxt_server_loop(void)
861 {
862         lxt_req_t               *lxt_req;
863         lxt_server_arg_t        *request;
864         size_t                  request_size;
865         char                    *door_result;
866         size_t                  door_result_size;

868         for (;;) {
869                 /* Wait for a request from a doors server thread. */
870                 (void) mutex_lock(&lxt_req_lock);
871                 while (lxt_req_ptr == NULL)
872                         (void) cond_wait(&lxt_req_cv, &lxt_req_lock);

874                 /* We got a request, get a local pointer to it. */
875                 lxt_req = lxt_req_ptr;
876                 lxt_req_ptr = NULL;
877                 (void) cond_broadcast(&lxt_req_cv);
878                 (void) mutex_unlock(&lxt_req_lock);

880                 /* Get a pointer to the request. */
881                 request = lxt_req->lxtr_request;
882                 request_size = lxt_req->lxtr_request_size;

884                 lx_debug("lxt_server_loop: Linux thread request recieved, "
885                     "request = %p", request);

887                 /* Dispatch the request. */
888                 assert((request->lxt_sa_op > LXT_SERVER_OP_PING) ||
889                     (request->lxt_sa_op < LXT_SERVER_OP_MAX));
890                 lxt_operations[request->lxt_sa_op].lxto_fp(
891                     request, request_size, &door_result, &door_result_size);

893                 lx_debug("lxt_server_loop: Linux thread request completed, "
894                     "request = %p", request);

896                 (void) mutex_lock(&lxt_req_lock);

898                 /* Set the result pointers for the calling door thread. */
899                 lxt_req->lxtr_result = door_result;
900                 lxt_req->lxtr_result_size = door_result_size;

902                 /* Let the door thread know we're done. */
903                 lxt_req->lxtr_complete = 1;
904                 (void) cond_signal(&lxt_req->lxtr_complete_cv);

906                 (void) mutex_unlock(&lxt_req_lock);
907         }
908         /*NOTREACHED*/
909 }

911 static void
912 lxt_server_enter(int fifo1_wr, int fifo2_rd)
913 {
914         struct stat     stat;
915         char            door_path[MAXPATHLEN];
916         int             i, dfd, junk = 0;

918         /*
919          * Do some sanity checks.  Make sure we've got the fifos
```

```
920          * we need passed to us on the correct file descriptors.
921          */
922         if ((fstat(fifo1_wr, &stat) != 0) ||
923             ((stat.st_mode & S_IFMT) != S_IFIFO) ||
924             (fstat(fifo2_rd, &stat) != 0) ||
925             ((stat.st_mode & S_IFMT) != S_IFIFO)) {
926                 lx_err("lx_thunk server aborting, can't contact parent");
927                 exit(-1);
928         }
929
930         /*
931          * Get the initial Linux call handle so we can invoke other
932          * Linux calls.
933          */
934         lxh_init = lx_call_init();
935         if (lxh_init == NULL) {
936                 lx_err("lx_thunk server aborting, failed Linux call init");
937                 exit(-1);
938         }
939
940         /* Now lookup other Linux symbols we'll need access to. */
941         for (i = 0; lxt_handles[i].lxth_name != NULL; i++) {
942                 assert(lxt_handles[i].lxth_index == i);
943                 if ((lxt_handles[i].lxth_handle = lx_call_dlsym(lxh_init,
944                     lxt_handles[i].lxth_name)) == NULL) {
945                         lx_err("lx_thunk server aborting, "
946                             "failed Linux symbol lookup: %s",
947                             lxt_handles[i].lxth_name);
948                         exit(-1);
949                 }
950         }
951
952         /* get the path to the door server */
953         if (read(fifo2_rd, door_path, sizeof (door_path)) < 0) {
954                 lx_err("lxt_server_enter: failed to get door path");
955                 exit(-1);
956         }
957         (void) close(fifo2_rd);
958
959         /* Create the door server. */
960         if ((dfd = door_create(lxt_server, door_path,
961             DOOR_UNREF | DOOR_REFUSE_DESC | DOOR_NO_CANCEL)) < 0) {
962                 lx_err("lxt_server_enter: door_create() failed");
963                 exit(-1);
964         }
965
966         /* Attach the door to a file system path. */
967         (void) fdetach(door_path);
968         if (fattach(dfd, door_path) < 0) {
969                 lx_err("lxt_server_enter: fattach() failed");
970                 exit(-1);
971         }
972
973         /* The door server is ready, signal this via a fifo write */
974         (void) write(fifo1_wr, &junk, 1);
975         (void) close(fifo1_wr);
976
977         lx_debug("lxt_server_enter: doors server initialized");
978         lxt_server_loop();
979         /*NOTREACHED*/
980 }
981
982 void
983 lxt_server_exec_check(void)
984 {
985         if (lxt_server_processes == 0)
```



```
986                 return;
987
988         /*
989          * We're a thunk server process, so we take over control of
990          * the current Linux process here.
991          */
992         lx_debug("lx_thunk server initalization starting");
993         lxt_server_enter(LXT_SERVER_FIFO_WR_FD, LXT_SERVER_FIFO_RD_FD);
994         /*NOTREACHED*/
995 }
996
997 void
998 lxt_server_init(int argc, char *argv[])
999 {
1000         /*
1001          * The thunk server process is a shell script named LXT_SERVER_BINARY.
1002          * It is executed without any parameters.  Since it's a shell script
1003          * the arguments passed to the shell's main entry point are:
1004          *      1) the name of the shell
1005          *      2) the name of the script to execute
1006          *
1007          * So to check if we're the thunk server process we first check
1008          * for the expected number of arduments and then we'll look at
1009          * the second parameter to see if it's LXT_SERVER_BINARY.
1010          */
1011         if ((argc != 2) ||
1012             (strcmp(argv[1], LXT_SERVER_BINARY) != 0))
1013                 return;
1014
1015         lxt_server_processes = 1;
1016         lx_debug("lx_thunk server detected, delaying initalization");
1017 }
1018
1019 int
1020 lxt_server_pid(int *pid)
1021 {
1022         if (lxt_server_processes == 0)
1023                 return (0);
1024         *pid = lxt_pid;
1025         return (1);
1026 }
1027 #endif /* ! codereview */
```

```
**********************************************************
    1349 Tue Jan 14 16:17:02 2014
new/usr/src/lib/brand/lx/lx_brand/common/mapfile
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

#
# MAPFILE HEADER START
#
# WARNING:  STOP NOW.  DO NOT MODIFY THIS FILE.
# Object versioning must comply with the rules detailed in
#
#       usr/src/lib/README.mapfiles
#
# You should not be making modifications here until you've read the most current
# copy of that file. If you need help, contact a gatekeeper for guidance.
#
# MAPFILE HEADER END
#

#
# Scope everything local -- our .init section is our only public interface.
#
{
        local:
                *;
};
#endif /* ! codereview */
```

****
1349 Tue Jan 14 16:17:02 2014
new/usr/src/lib/brand/lx/lx_brand/common/mapfile-vers
Bring back LX zones.
****

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

#
# MAPFILE HEADER START
#
# WARNING:  STOP NOW.  DO NOT MODIFY THIS FILE.
# Object versioning must comply with the rules detailed in
#
#       usr/src/lib/README.mapfiles
#
# You should not be making modifications here until you've read the most current
# copy of that file. If you need help, contact a gatekeeper for guidance.
#
# MAPFILE HEADER END
#

#
# Scope everything local -- our .init section is our only public interface.
#
{
        local:
                *;
};
#endif /* ! codereview */
```

```
**********************************************************
    5360 Tue Jan 14 16:17:02 2014
new/usr/src/lib/brand/lx/lx_brand/common/mem.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <errno.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/param.h>
#include <sys/lx_debug.h>
#include <sys/lx_misc.h>

/*
 * There are two forms of mmap, mmap() and mmap2().  The only difference is that
 * the final argument to mmap2() specifies the number of pages, not bytes.
 * Linux has a number of additional flags, but they are all deprecated.  We also
 * ignore the MAP_GROWSDOWN flag, which has no equivalent on Solaris.
 *
 * The Linux mmap() returns ENOMEM in some cases where Solaris returns
 * EOVERFLOW, so we translate the errno as necessary.
 */

int pagesize;   /* needed for mmap2() */

#define LX_MAP_ANONYMOUS        0x00020
#define LX_MAP_NORESERVE        0x04000

static int
ltos_mmap_flags(int flags)
{
        int new_flags;

        new_flags = flags & (MAP_TYPE | MAP_FIXED);
        if (flags & LX_MAP_ANONYMOUS)
                new_flags |= MAP_ANONYMOUS;
        if (flags & LX_MAP_NORESERVE)
                new_flags |= MAP_NORESERVE;

        return (new_flags);
```



```
}

static int
mmap_common(uintptr_t p1, uintptr_t p2, uintptr_t p3, uintptr_t p4,
    uintptr_t p5, off64_t p6)
{
        void *addr = (void *)p1;
        size_t len = p2;
        int prot = p3;
        int flags = p4;
        int fd = p5;
        off64_t off = p6;
        void *ret;

        if (lx_debug_enabled != 0) {
                char *path, path_buf[MAXPATHLEN];

                path = lx_fd_to_path(fd, path_buf, sizeof (path_buf));
                if (path == NULL)
                        path = "?";

                lx_debug("\tmmap_common(): fd = %d - %s", fd, path);
        }

        /*
         * Under Linux, the file descriptor is ignored when mapping zfod
         * anonymous memory,  On Solaris, we want the fd set to -1 for the
         * same functionality.
         */
        if (flags & LX_MAP_ANONYMOUS)
                fd = -1;

        /*
         * This is totally insane. The NOTES section in the linux mmap(2) man
         * page claims that on some architectures, read protection may
         * automatically include exec protection. It has been observed on a
         * native linux system that the /proc/<pid>/maps file does indeed
         * show that segments mmap'd from userland (such as libraries mapped in
         * by the dynamic linker) all have exec the permission set, even for
         * data segments.
         */
        if (prot & PROT_READ)
                prot |= PROT_EXEC;

        ret = mmap64(addr, len, prot, ltos_mmap_flags(flags), fd, off);

        if (ret == MAP_FAILED)
                return (errno == EOVERFLOW ? -ENOMEM : -errno);
        else
                return ((int)ret);
}

int
lx_mmap(uintptr_t p1, uintptr_t p2, uintptr_t p3, uintptr_t p4,
    uintptr_t p5, uintptr_t p6)
{
        return (mmap_common(p1, p2, p3, p4, p5, (off64_t)p6));
}

int
lx_mmap2(uintptr_t p1, uintptr_t p2, uintptr_t p3, uintptr_t p4,
    uintptr_t p5, uintptr_t p6)
{
        if (pagesize == 0)
                pagesize = sysconf(_SC_PAGESIZE);
```

```
128         return (mmap_common(p1, p2, p3, p4, p5, (off64_t)p6 * pagesize));
129 }


132 /*
133  * The locking family of system calls, as well as msync(), are identical.  On
134  * Solaris, they are layered on top of the memcntl syscall, so they cannot be
135  * pass-thru.
136  */
137 int
138 lx_mlock(uintptr_t addr, uintptr_t len)
139 {
140         uintptr_t addr1 = addr & PAGEMASK;
141         uintptr_t len1 = len + (addr & PAGEOFFSET);

143         return (mlock((void *)addr1, (size_t)len1) ? -errno : 0);
144 }

146 int
147 lx_mlockall(uintptr_t flags)
148 {
149         return (mlockall(flags) ? -errno : 0);
150 }

152 int
153 lx_munlock(uintptr_t addr, uintptr_t len)
154 {
155         uintptr_t addr1 = addr & PAGEMASK;
156         uintptr_t len1 = len + (addr & PAGEOFFSET);

158         return (munlock((void *)addr1, (size_t)len1) ? -errno : 0);
159 }

161 int
162 lx_munlockall(void)
163 {
164         return (munlockall() ? -errno : 0);
165 }

167 int
168 lx_msync(uintptr_t addr, uintptr_t len, uintptr_t flags)
169 {
170         return (msync((void *)addr, (size_t)len, flags) ? -errno : 0);
171 }

173 /*
174  * Solaris recognizes more flags than Linux, so we don't want to inadvertently
175  * use what would be an invalid flag on Linux.  Linux also allows the length to
176  * be zero, while Solaris does not.
177  */
178 int
179 lx_madvise(uintptr_t start, uintptr_t len, uintptr_t advice)
180 {
181         if (len == 0)
182                 return (0);

184         switch (advice) {
185         case MADV_NORMAL:
186         case MADV_RANDOM:
187         case MADV_SEQUENTIAL:
188         case MADV_WILLNEED:
189         case MADV_DONTNEED:
190                 return (madvise((void *)start, len, advice) ? -errno : 0);

192         default:
193                 return (-EINVAL);
```



```
194         }
195 }

197 /*
198  * mprotect() is identical except that we ignore the Linux flags PROT_GROWSDOWN
199  * and PROT_GROWSUP, which have no equivalent on Solaris.
200  */
201 #define LX_PROT_GROWSDOWN       0x01000000
202 #define LX_PROT_GROWSUP         0x02000000

204 int
205 lx_mprotect(uintptr_t start, uintptr_t len, uintptr_t prot)
206 {
207         prot &= ~(LX_PROT_GROWSUP | LX_PROT_GROWSDOWN);

209         return (mprotect((void *)start, len, prot) ? -errno : 0);
210 }
211 #endif /* ! codereview */
```

```
**********************************************************
   12520 Tue Jan 14 16:17:02 2014
new/usr/src/lib/brand/lx/lx_brand/common/misc.c
LX zone support should now build and packages of relevance produced.
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <assert.h>
#include <alloca.h>
#include <errno.h>
#include <fcntl.h>
#include <strings.h>
#include <macros.h>
#include <sys/brand.h>
#include <sys/reboot.h>
#include <sys/stat.h>
#include <sys/syscall.h>
#include <sys/sysmacros.h>
#include <sys/systeminfo.h>
#include <sys/types.h>
#include <sys/lx_types.h>
#include <sys/lx_debug.h>
#include <sys/lx_misc.h>
#include <sys/lx_stat.h>
#include <sys/lx_syscall.h>
#include <sys/lx_thunk_server.h>
#include <sys/lx_fcntl.h>
#include <unistd.h>
#include <libintl.h>
#include <zone.h>

extern int sethostname(char *, int);

/* ARGUSED */
int
lx_rename(uintptr_t p1, uintptr_t p2)
{
        int ret;

        ret = rename((const char *)p1, (const char *)p2);

        if (ret < 0) {
```



```
                /*
                 * If rename(2) failed and we're in install mode, return
                 * success if the the reason we failed was either because the
                 * source file didn't actually exist or if it was because we
                 * tried to rename it to be the name of a device currently in
                 * use (resulting in an EBUSY.)
                 *
                 * To help install along further, if the failure was due
                 * to an EBUSY, delete the original file so we don't leave
                 * extra files lying around.
                 */
                if (lx_install != 0) {
                        if (errno == ENOENT)
                                return (0);

                        if (errno == EBUSY) {
                                (void) unlink((const char *)p1);
                                return (0);
                        }
                }

                return (-errno);
        }

        return (0);
}

int
lx_renameat(uintptr_t ext1, uintptr_t p1, uintptr_t ext2, uintptr_t p2)
{
        int ret;
        int atfd1 = (int)ext1;
        int atfd2 = (int)ext2;

        if (atfd1 == LX_AT_FDCWD)
                atfd1 = AT_FDCWD;

        if (atfd2 == LX_AT_FDCWD)
                atfd2 = AT_FDCWD;

        ret = renameat(atfd1, (const char *)p1, atfd2, (const char *)p2);

        if (ret < 0) {
                /* see lx_rename() for why we check lx_install */
                if (lx_install != 0) {
                        if (errno == ENOENT)
                                return (0);

                        if (errno == EBUSY) {
                                (void) unlinkat(ext1, (const char *)p1, 0);
                                return (0);
                        }
                }

                return (-errno);
        }

        return (0);
}

/*ARGSUSED*/
int
lx_reboot(uintptr_t p1, uintptr_t p2, uintptr_t p3, uintptr_t p4)
{
        int magic = (int)p1;
        int magic2 = (int)p2;
```

```
127         uint_t flag = (int)p3;
128         int rc;

130         if (magic != LINUX_REBOOT_MAGIC1)
131                 return (-EINVAL);
132         if (magic2 != LINUX_REBOOT_MAGIC2 && magic2 != LINUX_REBOOT_MAGIC2A &&
133             magic2 != LINUX_REBOOT_MAGIC2B && magic2 != LINUX_REBOOT_MAGIC2C &&
134             magic2 != LINUX_REBOOT_MAGIC2D)
135                 return (-EINVAL);

137         if (geteuid() != 0)
138                 return (-EPERM);

140         switch (flag) {
141         case LINUX_REBOOT_CMD_CAD_ON:
142         case LINUX_REBOOT_CMD_CAD_OFF:
143                 /* ignored */
144                 rc = 0;
145                 break;
146         case LINUX_REBOOT_CMD_POWER_OFF:
147         case LINUX_REBOOT_CMD_HALT:
148                 rc = reboot(RB_HALT, NULL);
149                 break;
150         case LINUX_REBOOT_CMD_RESTART:
151         case LINUX_REBOOT_CMD_RESTART2:
152                 /* RESTART2 may need more work */
153                 lx_msg(gettext("Restarting system.\n"));
154                 rc = reboot(RB_AUTOBOOT, NULL);
155                 break;
156         default:
157                 return (-EINVAL);
158         }

160         return ((rc == -1) ? -errno : rc);
161 }

163 /*
164  * getcwd() - Linux syscall semantics are slightly different; we need to return
165  * the length of the pathname copied (+ 1 for the terminating NULL byte.)
166  */
167 int
168 lx_getcwd(uintptr_t p1, uintptr_t p2)
169 {
170         char *buf;
171         size_t buflen = (size_t)p2;
172         size_t copylen, local_len;
173         size_t len = 0;

175         if ((getcwd((char *)p1, (size_t)p2)) == NULL)
176                 return (-errno);

178         /*
179          * We need the length of the pathname getcwd() copied but we never want
180          * to dereference a Linux pointer for any reason.
181          *
182          * Thus, to get the string length we will uucopy() up to copylen bytes
183          * at a time into a local buffer and will walk each chunk looking for
184          * the string-terminating NULL byte.
185          *
186          * We can use strlen() to find the length of the string in the
187          * local buffer by delimiting the buffer with a NULL byte in the
188          * last element that will never be overwritten.
189          */
190         copylen = min(buflen, MAXPATHLEN + 1);
191         buf = SAFE_ALLOCA(copylen + 1);
192         if (buf == NULL)
```



```
193                 return (-ENOMEM);
194         buf[copylen] = '\0';

196         for (;;) {
197                 if (uucopy((char *)p1 + len, buf, copylen) != 0)
198                         return (-errno);

200                 local_len = strlen(buf);
201                 len += local_len;

203                 /*
204                  * If the strlen() is less than copylen, we found the
205                  * real end of the string -- not the NULL byte used to
206                  * delimit the end of our buffer.
207                  */
208                 if (local_len != copylen)
209                         break;

211                 /* prepare to check the next chunk of the string */
212                 buflen -= copylen;
213                 copylen = min(buflen, copylen);
214         }

216         return (len + 1);
217 }

219 int
220 lx_get_kern_version(void)
221 {
222         /*
223          * Since this function is called quite often, and zone_getattr is slow,
224          * we cache the kernel version in kvers_cache. -1 signifies that no
225          * value has yet been cached.
226          */
227         static int kvers_cache = -1;
228         /* dummy variable for use in zone_getattr */
229         int kvers;

231         if (kvers_cache != -1)
232                 return (kvers_cache);
233         if (zone_getattr(getzoneid(), LX_KERN_VERSION_NUM, &kvers, sizeof (int))
234             != sizeof (int))
235                 return (kvers_cache = LX_KERN_2_4);
236         else
237                 return (kvers_cache = kvers);
238 }

240 int
241 lx_uname(uintptr_t p1)
242 {
243         struct lx_utsname *un = (struct lx_utsname *)p1;
244         char buf[LX_SYS_UTS_LN + 1];

246         if (gethostname(un->nodename, sizeof (un->nodename)) == -1)
247                 return (-errno);

249         (void) strlcpy(un->sysname, LX_UNAME_SYSNAME, LX_SYS_UTS_LN);
250         (void) strlcpy(un->release, lx_release, LX_SYS_UTS_LN);
251         (void) strlcpy(un->version, LX_UNAME_VERSION, LX_SYS_UTS_LN);
252         (void) strlcpy(un->machine, LX_UNAME_MACHINE, LX_SYS_UTS_LN);
253         if ((sysinfo(SI_SRPC_DOMAIN, buf, LX_SYS_UTS_LN) < 0))
254                 un->domainname[0] = '\0';
255         else
256                 (void) strlcpy(un->domainname, buf, LX_SYS_UTS_LN);

258         return (0);
```

```
259 }

261 /*
262  * {get,set}groups16() - Handle the conversion between 16-bit Linux gids and
263  * 32-bit Solaris gids.
264  */
265 int
266 lx_getgroups16(uintptr_t p1, uintptr_t p2)
267 {
268         int count = (int)p1;
269         lx_gid16_t *grouplist = (lx_gid16_t *)p2;
270         gid_t *grouplist32;
271         int ret;
272         int i;

274         grouplist32 = SAFE_ALLOCA(count * sizeof (gid_t));
275         if (grouplist32 == NULL)
276                 return (-ENOMEM);
277         if ((ret = getgroups(count, grouplist32)) < 0)
278                 return (-errno);

280         for (i = 0; i < ret; i++)
281                 grouplist[i] = LX_GID32_TO_GID16(grouplist32[i]);

283         return (ret);
284 }

286 int
287 lx_setgroups16(uintptr_t p1, uintptr_t p2)
288 {
289         int count = (int)p1;
290         lx_gid16_t *grouplist = (lx_gid16_t *)p2;
291         gid_t *grouplist32;
292         int i;

294         grouplist32 = SAFE_ALLOCA(count * sizeof (gid_t));
295         if (grouplist32 == NULL)
296                 return (-ENOMEM);
297         for (i = 0; i < count; i++)
298                 grouplist32[i] = LX_GID16_TO_GID32(grouplist[i]);

300         return (setgroups(count, grouplist32) ? -errno : 0);
301 }

303 /*
304  * personality() - Solaris doesn't support Linux personalities, but we have to
305  * emulate enough to show that we support the basic personality.
306  */
307 #define LX_PER_LINUX    0x0

309 int
310 lx_personality(uintptr_t p1)
311 {
312         int per = (int)p1;

314         switch (per) {
315         case -1:
316                 /* Request current personality */
317                 return (LX_PER_LINUX);
318         case LX_PER_LINUX:
319                 return (0);
320         default:
321                 return (-EINVAL);
322         }
323 }
```



```
325 /*
326  * mknod() - Since we don't have the SYS_CONFIG privilege within a zone, the
327  * only mode we have to support is S_IFIFO.  We also have to distinguish between
328  * an invalid type and insufficient privileges.
329  */
330 #define LX_S_IFMT       0170000
331 #define LX_S_IFDIR      0040000
332 #define LX_S_IFCHR      0020000
333 #define LX_S_IFBLK      0060000
334 #define LX_S_IFREG      0100000
335 #define LX_S_IFIFO      0010000
336 #define LX_S_IFLNK      0120000
337 #define LX_S_IFSOCK     0140000

339 /*ARGSUSED*/
340 int
341 lx_mknod(uintptr_t p1, uintptr_t p2, uintptr_t p3)
342 {
343         char *path = (char *)p1;
344         lx_dev_t lx_dev = (lx_dev_t)p3;
345         struct sockaddr_un sockaddr;
346         struct stat statbuf;
347         mode_t mode, type;
348         dev_t dev;
349         int fd;

351         type = ((mode_t)p2 & LX_S_IFMT);
352         mode = ((mode_t)p2 & 07777);

354         switch (type) {
355         case 0:
356         case LX_S_IFREG:
357                 /* create a regular file */
358                 if (stat(path, &statbuf) == 0)
359                         return (-EEXIST);

361                 if (errno != ENOENT)
362                         return (-errno);

364                 if ((fd = creat(path, mode)) < 0)
365                         return (-errno);

367                 (void) close(fd);
368                 return (0);

370         case LX_S_IFSOCK:
371                 /*
372                  * Create a UNIX domain socket.
373                  *
374                  * Most programmers aren't even aware you can do this.
375                  *
376                  * Note you can also do this via Solaris' mknod(2), but
377                  * Linux allows anyone who can create a UNIX domain
378                  * socket via bind(2) to create one via mknod(2);
379                  * Solaris requires the caller to be privileged.
380                  */
381                 if ((fd = socket(AF_UNIX, SOCK_STREAM, 0)) < 0)
382                         return (-errno);

384                 if (stat(path, &statbuf) == 0)
385                         return (-EEXIST);

387                 if (errno != ENOENT)
388                         return (-errno);

390                 if (uucopy(path, &sockaddr.sun_path,
```

```
391                     sizeof (sockaddr.sun_path)) < 0)
392                         return (-errno);
393
394                 /* assure NULL termination of sockaddr.sun_path */
395                 sockaddr.sun_path[sizeof (sockaddr.sun_path) - 1] = '\0';
396                 sockaddr.sun_family = AF_UNIX;
397
398                 if (bind(fd, (struct sockaddr *)&sockaddr,
399                     strlen(sockaddr.sun_path) +
400                     sizeof (sockaddr.sun_family)) < 0)
401                         return (-errno);
402
403                 (void) close(fd);
404                 return (0);
405
406         case LX_S_IFIFO:
407                 dev = 0;
408                 break;
409
410         case LX_S_IFCHR:
411         case LX_S_IFBLK:
412                 /*
413                  * The "dev" RPM package wants to create all possible Linux
414                  * device nodes, so just report its mknod()s as having
415                  * succeeded if we're in install mode.
416                  */
417                 if (lx_install != 0) {
418                         lx_debug("lx_mknod: install mode spoofed creation of "
419                             "Linux device [%lld, %lld]\n",
420                             LX_GETMAJOR(lx_dev), LX_GETMINOR(lx_dev));
421
422                         return (0);
423                 }
424
425                 dev = makedevice(LX_GETMAJOR(lx_dev), LX_GETMINOR(lx_dev));
426                 break;
427
428         default:
429                 return (-EINVAL);
430         }
431
432         return (mknod(path, mode | type, dev) ? -errno : 0);
433 }
434
435 int
436 lx_sethostname(uintptr_t p1, uintptr_t p2)
437 {
438         char *name = (char *)p1;
439         int len = (size_t)p2;
440
441         return (sethostname(name, len) ? -errno : 0);
442 }
443
444 int
445 lx_setdomainname(uintptr_t p1, uintptr_t p2)
446 {
447         char *name = (char *)p1;
448         int len = (size_t)p2;
449         long rval;
450
451         if (len < 0 || len >= LX_SYS_UTS_LN)
452                 return (-EINVAL);
453
454         rval = sysinfo(SI_SET_SRPC_DOMAIN, name, len);
455
456         return ((rval < 0) ? -errno : 0);
```



```
457 }
458
459 int
460 lx_getpid(void)
461 {
462         int pid;
463
464         /* First call the thunk server hook. */
465         if (lxt_server_pid(&pid) != 0)
466                 return (pid);
467
468         pid = syscall(SYS_brand, B_EMULATE_SYSCALL + 20);
469         return ((pid == -1) ? -errno : pid);
470 }
471
472 int
473 lx_execve(uintptr_t p1, uintptr_t p2, uintptr_t p3)
474 {
475         char *filename = (char *)p1;
476         char **argv = (char **)p2;
477         char **envp = (char **)p3;
478         char *nullist[] = { NULL };
479         char path[64];
480
481         /* First call the thunk server hook. */
482         lxt_server_exec_check();
483
484         /* Get a copy of the executable we're trying to run */
485         path[0] = '\0';
486         (void) uucopystr(filename, path, sizeof (path));
487
488         /* Check if we're trying to run a native binary */
489         if (strncmp(path, "/native/usr/lib/brand/lx/lx_native",
490             sizeof (path)) == 0) {
491                 /* Skip the first element in the argv array */
492                 argv++;
493
494                 /*
495                  * The name of the new program to execute was the first
496                  * parameter passed to lx_native.
497                  */
498                 if (uucopy(argv, &filename, sizeof (char *)) != 0)
499                         return (-errno);
500
501                 (void) syscall(SYS_brand, B_EXEC_NATIVE, filename, argv, envp,
502                     NULL, NULL, NULL);
503                 return (-errno);
504         }
505
506         if (argv == NULL)
507                 argv = nullist;
508
509         /* This is a normal exec call. */
510         (void) execve(filename, argv, envp);
511
512         return (-errno);
513 }
514
515 int
516 lx_setgroups(uintptr_t p1, uintptr_t p2)
517 {
518         int ng = (int)p1;
519         gid_t *glist = NULL;
520         int i, r;
521
522         lx_debug("\tlx_setgroups(%d, 0x%p", ng, p2);
```

```
524         if (ng > 0) {
525                 if ((glist = (gid_t *)SAFE_ALLOCA(ng * sizeof (gid_t))) == NULL)
526                         return (-ENOMEM);

528                 if (uucopy((void *)p2, glist, ng * sizeof (gid_t)) != 0)
529                         return (-errno);

531                 /*
532                  * Linux doesn't check the validity of the group IDs, but
533                  * Solaris does. Change any invalid group IDs to a known, valid
534                  * value (yuck).
535                  */
536                 for (i = 0; i < ng; i++) {
537                         if (glist[i] > MAXUID)
538                                 glist[i] = MAXUID;
539                 }
540         }

542         r = syscall(SYS_brand, B_EMULATE_SYSCALL + LX_SYS_setgroups32,
543             ng, glist);

545         return ((r == -1) ? -errno : r);
546 }
547 #endif /* ! codereview */
```

```
**********************************************************
    2128 Tue Jan 14 16:17:03 2014
new/usr/src/lib/brand/lx/lx_brand/common/module.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

/*
 * We don't support Linux modules, but we have to emulate enough of the system
 * calls to show that we don't have any modules installed.
 */

#include <errno.h>
#include <sys/types.h>
#include <sys/lx_misc.h>

/*
 * For query_module(), we provide an empty list of modules, and return ENOENT
 * on any request for a specific module.
 */
#define LX_QM_MODULES   1
#define LX_QM_DEPS      2
#define LX_QM_REFS      3
#define LX_QM_SYMBOLS   4
#define LX_QM_INFO      5

/*ARGSUSED*/
int
lx_query_module(uintptr_t p1, uintptr_t p2, uintptr_t p3, uintptr_t p4,
    uintptr_t p5)
{
        /*
         * parameter p1 is the 'name' argument.
         */
        int which = (int)p2;
        char *buf = (char *)p3;
        size_t bufsize = (size_t)p4;
        size_t *ret = (size_t *)p5;

        switch (which) {
```



```
        case 0:
                /*
                 * Special case: always return 0
                 */
                return (0);

        case LX_QM_MODULES:
                /*
                 * Generate an empty list of modules.
                 */
                if (bufsize && buf)
                        buf[0] = '\0';
                if (ret)
                        *ret = 0;
                return (0);

        case LX_QM_DEPS:
        case LX_QM_REFS:
        case LX_QM_SYMBOLS:
        case LX_QM_INFO:
                /*
                 * Any requests for specific module information return ENOENT.
                 */
                return (-ENOENT);

        default:
                return (-EINVAL);
        }
}
#endif /* ! codereview */
```

```
**********************************************************
   20423 Tue Jan 14 16:17:03 2014
new/usr/src/lib/brand/lx/lx_brand/common/mount.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <alloca.h>
#include <assert.h>
#include <ctype.h>
#include <fcntl.h>
#include <errno.h>
#include <signal.h>
#include <string.h>
#include <strings.h>
#include <nfs/mount.h>
#include <sys/types.h>
#include <sys/mount.h>
#include <sys/param.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

#include <sys/lx_autofs.h>
#include <sys/lx_debug.h>
#include <sys/lx_misc.h>
#include <sys/lx_mount.h>

/*
 * support definitions
 */
union fh_buffer {
        struct nfs_fid  fh2;
        struct nfs_fh3  fh3;
        char            fh_data[NFS3_FHSIZE + 2];
};

typedef enum mount_opt_type {
        MOUNT_OPT_INVALID       = 0,
        MOUNT_OPT_NORMAL        = 1,    /* option value: none */
```



```
        MOUNT_OPT_UINT          = 2     /* option value: unsigned int */
} mount_opt_type_t;

typedef struct mount_opt {
        char                    *mo_name;
        mount_opt_type_t        mo_type;
} mount_opt_t;


/*
 * Globals
 */
mount_opt_t lofs_options[] = {
        { NULL, MOUNT_OPT_INVALID }
};

mount_opt_t lx_proc_options[] = {
        { NULL, MOUNT_OPT_INVALID }
};

mount_opt_t lx_autofs_options[] = {
        { LX_MNTOPT_FD,         MOUNT_OPT_UINT },
        { LX_MNTOPT_PGRP,       MOUNT_OPT_UINT },
        { LX_MNTOPT_MINPROTO,   MOUNT_OPT_UINT },
        { LX_MNTOPT_MAXPROTO,   MOUNT_OPT_UINT },
};


/*
 * i_lx_opt_verify() - Check the mount options.
 *
 * You might wonder why we're being so strict about the mount options
 * we allow.  The reason is that normally all mount option verification
 * is done by the Solaris userland mount command.  Once mount options
 * are passed to the kernel, invalid options are simply ignored.  So
 * if we actually want to catch requests for functionality that we
 * don't support, or if we want to make sure that we don't randomly
 * enable options that we haven't check to make sure they have the
 * same syntax on Linux and Solaris, we need to reject any options
 * we don't know to be ok here.
 */
static int
i_lx_opt_verify(char *opts, mount_opt_t *mop)
{
        int     opts_len = strlen(opts);
        char    *opts_tmp, *opt;
        int     opt_len, i;

        assert((opts != NULL) && (mop != NULL));

        /* If no options were specified, there's no problem. */
        if (opts_len == 0)
                return (1);

        /* If no options are allowed, fail. */
        if (mop[0].mo_name == NULL)
                return (0);

        /* Don't accept leading or trailing ','. */
        if ((opts[0] == ',') || (opts[opts_len] == ','))
                return (0);

        /* Don't accept sequential ','. */
        for (i = 1; i < opts_len; i++)
                if ((opts[i - 1] ==  ',') && (opts[i] ==  ','))
                        return (0);
```

```
129         /*
130          * We're going to use strtok() which modifies the target
131          * string so make a temporary copy.
132          */
133         opts_tmp = SAFE_ALLOCA(opts_len);
134         if (opts_tmp == NULL)
135                 return (-1);
136         bcopy(opts, opts_tmp, opts_len + 1);
137
138         /* Verify each prop one at a time. */
139         opt = strtok(opts_tmp, ",");
140         opt_len = strlen(opt);
141         for (;;) {
142
143                 /* Check for matching option/value pair. */
144                 for (i = 0; mop[i].mo_name != NULL; i++) {
145                         char    *ovalue;
146                         int     ovalue_len, mo_len;
147
148                         /* If the options is too short don't bother comparing */
149                         mo_len = strlen(mop[i].mo_name);
150                         if (opt_len < mo_len) {
151                                 /* Keep trying to find a match. */
152                                 continue;
153                         }
154
155                         /* Compare the option to an allowed option. */
156                         if (strncmp(mop[i].mo_name, opt, mo_len) != 0) {
157                                 /* Keep trying to find a match. */
158                                 continue;
159                         }
160
161                         if (mop[i].mo_type == MOUNT_OPT_NORMAL) {
162                                 /* The option doesn't take a value. */
163                                 if (opt_len == mo_len) {
164                                         /* This option is ok. */
165                                         break;
166                                 } else {
167                                         /* Keep trying to find a match. */
168                                         continue;
169                                 }
170                         }
171
172                         /* This options takes a value. */
173                         if ((opt_len == mo_len) || (opt[mo_len] != '=')) {
174                                 /* Keep trying to find a match. */
175                                 continue;
176                         }
177
178                         /* We have an option match.  Verify option value. */
179                         ovalue = &opt[mo_len] + 1;
180                         ovalue_len = strlen(ovalue);
181
182                         /* Value can't be zero length string. */
183                         if (ovalue_len == 0)
184                                 return (0);
185
186                         if (mop[i].mo_type == MOUNT_OPT_UINT) {
187                                 int j;
188                                 /* Verify that value is an unsigned int. */
189                                 for (j = 0; j < ovalue_len; j++)
190                                         if (!isdigit(ovalue[j]))
191                                                 return (0);
192                         } else {
193                                 /* Unknown option type specified. */
```



```
194                                 assert(0);
195                         }
196
197                         /* The option is ok. */
198                         break;
199                 }
200
201                 /* If there were no matches this is an unsupported option. */
202                 if (mop[i].mo_name == NULL)
203                         return (0);
204
205                 /* This option is ok, move onto the next option. */
206                 if ((opt = strtok(NULL, ",")) == NULL)
207                         break;
208                 opt_len = strlen(opt);
209         };
210
211         /* We verified all the options. */
212         return (1);
213 }
214
215 static int
216 i_add_option(char *option, char *buf, size_t buf_size)
217 {
218         char *fmt_str = NULL;
219
220         assert((option != NULL) && (strlen(option) > 0));
221         assert((buf != NULL) && (buf_size > 0));
222
223         if (buf[0] == '\0') {
224                 fmt_str = "%s";
225         } else {
226                 fmt_str = ",%s";
227         }
228
229         buf_size -= strlen(buf);
230         buf += strlen(buf);
231
232         /*LINTED*/
233         if (snprintf(buf, buf_size, fmt_str, option) > (buf_size - 1))
234                 return (-EOVERFLOW);
235         return (0);
236 }
237
238 static int
239 i_add_option_int(char *option, int val, char *buf, size_t buf_size)
240 {
241         char *fmt_str = NULL;
242
243         assert((option != NULL) && (strlen(option) > 0));
244         assert((buf != NULL) && (buf_size > 0));
245
246         if (buf[0] == '\0') {
247                 fmt_str = "%s=%d";
248         } else {
249                 fmt_str = ",%s=%d";
250         }
251
252         buf_size -= strlen(buf);
253         buf += strlen(buf);
254
255         /*LINTED*/
256         if (snprintf(buf, buf_size, fmt_str, option, val) > (buf_size - 1))
257                 return (-EOVERFLOW);
258         return (0);
259 }
```

```
261 static int
262 i_make_nfs_args(lx_nfs_mount_data_t *lx_nmd, struct nfs_args *nfs_args,
263     struct netbuf *nfs_args_addr, struct knetconfig *nfs_args_knconf,
264     union fh_buffer *nfs_args_fh, struct sec_data *nfs_args_secdata,
265     char *fstype, char *options, int options_size)
266 {
267         struct stat     statbuf;
268         int             i, rv, use_tcp;

270         /* Sanity check the incomming Linux request. */
271         if ((lx_nmd->nmd_rsize < 0) || (lx_nmd->nmd_wsize < 0) ||
272             (lx_nmd->nmd_timeo < 0) || (lx_nmd->nmd_retrans < 0) ||
273             (lx_nmd->nmd_acregmin < 0) || (lx_nmd->nmd_acregmax < 0) ||
274             (lx_nmd->nmd_acdirmax < 0)) {
275                 return (-EINVAL);
276         }

278         /*
279          * Additional sanity checks of incomming request.
280          *
281          * Some of the sanity checks below should probably return
282          * EINVAL (or some other error code) instead or ENOTSUP,
283          * but without experiminting on Linux to see how it
284          * deals with certain strange values there is no way
285          * to really know what we should return, hence we return
286          * ENOTSUP to tell us that eventually if we see some
287          * application hitting the problem we can go to a real
288          * Linux system, figure out how it deals with the situation
289          * and update our code to handle it in the same fashion.
290          */
291         if (lx_nmd->nmd_version != 4) {
292                 lx_unsupported("unsupported nfs mount request, "
293                     "unrecognized NFS mount structure: %d\n",
294                     lx_nmd->nmd_version);
295                 return (-ENOTSUP);
296         }
297         if ((lx_nmd->nmd_flags & ~LX_NFS_MOUNT_SUPPORTED) != 0) {
298                 lx_unsupported("unsupported nfs mount request, "
299                     "flags: 0x%x\n", lx_nmd->nmd_flags);
300                 return (-ENOTSUP);
301         }
302         if (lx_nmd->nmd_addr.sin_family != AF_INET) {
303                 lx_unsupported("unsupported nfs mount request, "
304                     "transport address family: 0x%x\n",
305                     lx_nmd->nmd_addr.sin_family);
306                 return (-ENOTSUP);
307         }
308         for (i = 0; i < LX_NMD_MAXHOSTNAMELEN; i++) {
309                 if (lx_nmd->nmd_hostname[i] == '\0')
310                         break;
311         }
312         if (i == 0) {
313                 lx_unsupported("unsupported nfs mount request, "
314                     "no hostname specified\n");
315                 return (-ENOTSUP);
316         }
317         if (i == LX_NMD_MAXHOSTNAMELEN) {
318                 lx_unsupported("unsupported nfs mount request, "
319                     "hostname not terminated\n");
320                 return (-ENOTSUP);
321         }
322         if (lx_nmd->nmd_namlen < i) {
323                 lx_unsupported("unsupported nfs mount request, "
324                     "invalid namlen value: 0x%x\n", lx_nmd->nmd_namlen);
325                 return (-ENOTSUP);
```



```
326         }
327         if (lx_nmd->nmd_bsize != 0) {
328                 lx_unsupported("unsupported nfs mount request, "
329                     "bsize value: 0x%x\n", lx_nmd->nmd_bsize);
330                 return (-ENOTSUP);
331         }

333         /* Initialize and clear the output structure pointers passed in. */
334         bzero(nfs_args, sizeof (*nfs_args));
335         bzero(nfs_args_addr, sizeof (*nfs_args_addr));
336         bzero(nfs_args_knconf, sizeof (*nfs_args_knconf));
337         bzero(nfs_args_fh, sizeof (*nfs_args_fh));
338         bzero(nfs_args_secdata, sizeof (*nfs_args_secdata));
339         nfs_args->addr = nfs_args_addr;
340         nfs_args->knconf = nfs_args_knconf;
341         nfs_args->fh = (caddr_t)nfs_args_fh;
342         nfs_args->nfs_ext_u.nfs_extB.secdata = nfs_args_secdata;

344         /* Check if we're using tcp. */
345         use_tcp = (lx_nmd->nmd_flags & LX_NFS_MOUNT_TCP) ? 1 : 0;

347         /*
348          * These seem to be the default flags used by Solaris for v2 and v3
349          * nfs mounts.
350          *
351          * Don't bother with NFSMNT_TRYRDMA since we always specify a
352          * transport (either udp or tcp).
353          */
354         nfs_args->flags = NFSMNT_NEWARGS | NFSMNT_KNCONF | NFSMNT_INT |
355             NFSMNT_HOSTNAME;

357         /* Translate some Linux mount flags into Solaris mount flags. */
358         if (lx_nmd->nmd_flags & LX_NFS_MOUNT_SOFT)
359                 nfs_args->flags |= NFSMNT_SOFT;
360         if (lx_nmd->nmd_flags & LX_NFS_MOUNT_INTR)
361                 nfs_args->flags |= NFSMNT_INT;
362         if (lx_nmd->nmd_flags & LX_NFS_MOUNT_POSIX)
363                 nfs_args->flags |= NFSMNT_POSIX;
364         if (lx_nmd->nmd_flags & LX_NFS_MOUNT_NOCTO)
365                 nfs_args->flags |= NFSMNT_NOCTO;
366         if (lx_nmd->nmd_flags & LX_NFS_MOUNT_NOAC)
367                 nfs_args->flags |= NFSMNT_NOAC;
368         if (lx_nmd->nmd_flags & LX_NFS_MOUNT_NONLM)
369                 nfs_args->flags |= NFSMNT_LLOCK;

371         if ((lx_nmd->nmd_flags & LX_NFS_MOUNT_VER3) != 0) {
372                 (void) strcpy(fstype, "nfs3");
373                 if ((rv = i_add_option_int("vers", 3,
374                             options, options_size)) != 0)
375                         return (rv);

377                 if (lx_nmd->nmd_root.lx_fh3_length >
378                     sizeof (nfs_args_fh->fh3.fh3_u.data)) {
379                         lx_unsupported("unsupported nfs mount request, "
380                             "nfs file handle length: 0x%x\n",
381                             lx_nmd->nmd_root.lx_fh3_length);
382                         return (-ENOTSUP);
383                 }

385                 /* Set the v3 file handle info. */
386                 nfs_args_fh->fh3.fh3_length = lx_nmd->nmd_root.lx_fh3_length;
387                 bcopy(&lx_nmd->nmd_root.lx_fh3_data,
388                     nfs_args_fh->fh3.fh3_u.data,
389                     lx_nmd->nmd_root.lx_fh3_length);
390         } else {
391                 /*
```

```
392                  * Assume nfs v2.  Note that this could also be a v1
393                  * mount request but there doesn't seem to be any difference
394                  * in the parameters passed to the Linux mount system
395                  * call for v1 or v2 mounts so there is no way of really
396                  * knowing.
397                  */
398                 (void) strcpy(fstype, "nfs");
399                 if ((rv = i_add_option_int("vers", 2,
400                         options, options_size)) != 0)
401                         return (rv);
402
403                 /* Solaris seems to add this flag when using v2. */
404                 nfs_args->flags |= NFSMNT_SECDEFAULT;
405
406                 /* Set the v2 file handle info. */
407                 bcopy(&lx_nmd->nmd_old_root,
408                     nfs_args_fh, sizeof (nfs_args_fh->fh2));
409         }
410
411         /*
412          * We can't use getnetconfig() here because there is no netconfig
413          * database in linux.
414          */
415         nfs_args_knconf->knc_protofmly = "inet";
416         if (use_tcp) {
417                 /*
418                  * TCP uses NC_TPI_COTS_ORD semantics.
419                  * See /etc/netconfig.
420                  */
421                 nfs_args_knconf->knc_semantics = NC_TPI_COTS_ORD;
422                 nfs_args_knconf->knc_proto = "tcp";
423                 if ((rv = i_add_option("proto=tcp",
424                         options, options_size)) != 0)
425                         return (rv);
426                 if (stat("/dev/tcp", &statbuf) != 0)
427                         return (-errno);
428                 nfs_args_knconf->knc_rdev = statbuf.st_rdev;
429         } else {
430                 /*
431                  * Assume UDP.  UDP uses NC_TPI_CLTS semantics.
432                  * See /etc/netconfig.
433                  */
434                 nfs_args_knconf->knc_semantics = NC_TPI_CLTS;
435                 nfs_args_knconf->knc_proto = "udp";
436                 if ((rv = i_add_option("proto=udp",
437                         options, options_size)) != 0)
438                         return (rv);
439                 if (stat("/dev/udp", &statbuf) != 0)
440                         return (-errno);
441                 nfs_args_knconf->knc_rdev = statbuf.st_rdev;
442         }
443
444         /* Set the server address. */
445         nfs_args_addr->maxlen = nfs_args_addr->len =
446             sizeof (struct sockaddr_in);
447         nfs_args_addr->buf = (char *)&lx_nmd->nmd_addr;
448
449         /* Set the server hostname string. */
450         nfs_args->hostname = lx_nmd->nmd_hostname;
451
452         /* Translate Linux nfs mount parameters into Solaris mount options. */
453         if (lx_nmd->nmd_rsize != LX_NMD_DEFAULT_RSIZE) {
454                 if ((rv = i_add_option_int("rsize", lx_nmd->nmd_rsize,
455                         options, options_size)) != 0)
456                         return (rv);
457                 nfs_args->rsize = lx_nmd->nmd_rsize;
```



```
458                 nfs_args->flags |= NFSMNT_RSIZE;
459         }
460         if (lx_nmd->nmd_wsize != LX_NMD_DEFAULT_WSIZE) {
461                 if ((rv = i_add_option_int("wsize", lx_nmd->nmd_wsize,
462                         options, options_size)) != 0)
463                         return (rv);
464                 nfs_args->wsize = lx_nmd->nmd_wsize;
465                 nfs_args->flags |= NFSMNT_WSIZE;
466         }
467         if ((rv = i_add_option_int("timeo", lx_nmd->nmd_timeo,
468                 options, options_size)) != 0)
469                 return (rv);
470         nfs_args->timeo = lx_nmd->nmd_timeo;
471         nfs_args->flags |= NFSMNT_TIMEO;
472         if ((rv = i_add_option_int("retrans", lx_nmd->nmd_retrans,
473                 options, options_size)) != 0)
474                 return (rv);
475         nfs_args->retrans = lx_nmd->nmd_retrans;
476         nfs_args->flags |= NFSMNT_RETRANS;
477         if ((rv = i_add_option_int("acregmin", lx_nmd->nmd_acregmin,
478                 options, options_size)) != 0)
479                 return (rv);
480         nfs_args->acregmin = lx_nmd->nmd_acregmin;
481         nfs_args->flags |= NFSMNT_ACREGMIN;
482         if ((rv = i_add_option_int("acregmax", lx_nmd->nmd_acregmax,
483                 options, options_size)) != 0)
484                 return (rv);
485         nfs_args->acregmax = lx_nmd->nmd_acregmax;
486         nfs_args->flags |= NFSMNT_ACREGMAX;
487         if ((rv = i_add_option_int("acdirmin", lx_nmd->nmd_acdirmin,
488                 options, options_size)) != 0)
489                 return (rv);
490         nfs_args->acdirmin = lx_nmd->nmd_acdirmin;
491         nfs_args->flags |= NFSMNT_ACDIRMIN;
492         if ((rv = i_add_option_int("acdirmax", lx_nmd->nmd_acdirmax,
493                 options, options_size)) != 0)
494                 return (rv);
495         nfs_args->acdirmax = lx_nmd->nmd_acdirmax;
496         nfs_args->flags |= NFSMNT_ACDIRMAX;
497
498         /* We only support nfs with a security type of AUTH_SYS. */
499         nfs_args->nfs_args_ext = NFS_ARGS_EXTB;
500         nfs_args_secdata->secmod = AUTH_SYS;
501         nfs_args_secdata->rpcflavor = AUTH_SYS;
502         nfs_args_secdata->flags = 0;
503         nfs_args_secdata->uid = 0;
504         nfs_args_secdata->data = NULL;
505         nfs_args->nfs_ext_u.nfs_extB.next = NULL;
506
507         /*
508          * The Linux nfs mount command seems to pass an open socket fd
509          * to the kernel during the mount system call.  We don't need
510          * this fd on Solaris so just close it.
511          */
512         (void) close(lx_nmd->nmd_fd);
513
514         return (0);
515 }
516
517 int
518 lx_mount(uintptr_t p1, uintptr_t p2, uintptr_t p3, uintptr_t p4,
519     uintptr_t p5)
520 {
521         /* Linux input arguments. */
522         const char              *sourcep = (const char *)p1;
523         const char              *targetp = (const char *)p2;
```

```
524         const char              *fstypep = (const char *)p3;
525         unsigned int            flags = (unsigned int)p4;
526         const void              *datap = (const void *)p5;

528         /* Variables needed for all mounts. */
529         char                    source[MAXPATHLEN], target[MAXPATHLEN];
530         char                    fstype[MAXPATHLEN], options[MAXPATHLEN];
531         int                     sflags, rv;

533         /* Variables needed for nfs mounts. */
534         lx_nfs_mount_data_t     lx_nmd;
535         struct nfs_args         nfs_args;
536         struct netbuf           nfs_args_addr;
537         struct knetconfig       nfs_args_knconf;
538         union fh_buffer         nfs_args_fh;
539         struct sec_data         nfs_args_secdata;
540         char                    *sdataptr = NULL;
541         int                     sdatalen = 0;

543         /* Initialize Solaris mount arguments. */
544         sflags = MS_OPTIONSTR;
545         options[0] = '\0';
546         sdatalen = 0;

548         /* Copy in parameters that are always present. */
549         rv = uucopystr((void *)sourcep, &source, sizeof (source));
550         if ((rv == -1) || (rv == sizeof (source)))
551                 return (-EFAULT);

553         rv = uucopystr((void *)targetp, &target, sizeof (target));
554         if ((rv == -1) || (rv == sizeof (target)))
555                 return (-EFAULT);

557         rv = uucopystr((void *)fstypep, &fstype, sizeof (fstype));
558         if ((rv == -1) || (rv == sizeof (fstype)))
559                 return (-EFAULT);

561         lx_debug("\tlinux mount source: %s", source);
562         lx_debug("\tlinux mount target: %s", target);
563         lx_debug("\tlinux mount fstype: %s", fstype);

565         /* Make sure we support the requested mount flags. */
566         if ((flags & ~LX_MS_SUPPORTED) != 0) {
567                 lx_unsupported(
568                     "unsupported mount flags: 0x%x", flags);
569                 return (-ENOTSUP);
570         }

572         /* Do filesystem specific mount work. */
573         if (flags & LX_MS_BIND) {

575                 /* If MS_BIND is set, we turn this into a lofs mount.  */
576                 (void) strcpy(fstype, "lofs");

578                 /* Copy in Linux mount options. */
579                 if (datap != NULL) {
580                         rv = uucopystr((void *)datap,
581                             options, sizeof (options));
582                         if ((rv == -1) || (rv == sizeof (options)))
583                                 return (-EFAULT);
584                 }
585                 lx_debug("\tlinux mount options: \"%s\"", options);

587                 /* Verify Linux mount options. */
588                 if (i_lx_opt_verify(options, lofs_options) == 0) {
589                         lx_unsupported("unsupported lofs mount options");
```



```
590                         return (-ENOTSUP);
591                 }
592         } else if (strcmp(fstype, "proc") == 0) {

594                 /* Translate proc mount requests to lx_proc requests. */
595                 (void) strcpy(fstype, "lx_proc");

597                 /* Copy in Linux mount options. */
598                 if (datap != NULL) {
599                         rv = uucopystr((void *)datap,
600                             options, sizeof (options));
601                         if ((rv == -1) || (rv == sizeof (options)))
602                                 return (-EFAULT);
603                 }
604                 lx_debug("\tlinux mount options: \"%s\"", options);

606                 /* Verify Linux mount options. */
607                 if (i_lx_opt_verify(options, lx_proc_options) == 0) {
608                         lx_unsupported("unsupported lx_proc mount options");
609                         return (-ENOTSUP);
610                 }
611         } else if (strcmp(fstype, "autofs") == 0) {

613                 /* Translate proc mount requests to lx_afs requests. */
614                 (void) strcpy(fstype, LX_AUTOFS_NAME);

616                 /* Copy in Linux mount options. */
617                 if (datap != NULL) {
618                         rv = uucopystr((void *)datap,
619                             options, sizeof (options));
620                         if ((rv == -1) || (rv == sizeof (options)))
621                                 return (-EFAULT);
622                 }
623                 lx_debug("\tlinux mount options: \"%s\"", options);

625                 /* Verify Linux mount options. */
626                 if (i_lx_opt_verify(options, lx_autofs_options) == 0) {
627                         lx_unsupported("unsupported lx_autofs mount options");
628                         return (-ENOTSUP);
629                 }
630         } else if (strcmp(fstype, "nfs") == 0) {

632                 /*
633                  * Copy in Linux mount options.  Note that for Linux
634                  * nfs mounts the mount options pointer (which normally
635                  * points to a string) points to a structure.
636                  */
637                 if (uucopy((void *)datap, &lx_nmd, sizeof (lx_nmd)) < 0)
638                         return (-errno);

640                 /*
641                  * For Solaris nfs mounts, the kernel expects a special
642                  * strucutre, but a pointer to this structure is passed
643                  * in via an extra parameter (sdataptr below.)
644                  */
645                 if ((rv = i_make_nfs_args(&lx_nmd, &nfs_args,
646                     &nfs_args_addr, &nfs_args_knconf, &nfs_args_fh,
647                     &nfs_args_secdata, fstype,
648                     options, sizeof (options))) != 0)
649                         return (rv);

651                 /*
652                  * For nfs mounts we need to tell the mount system call
653                  * to expect extra parameters.
654                  */
655                 sflags |= MS_DATA;
```

```
656                 sdataptr = (char *)&nfs_args;
657                 sdatalen = sizeof (nfs_args);
658         } else {
659                 lx_unsupported(
660                     "unsupported mount filesystem type: %s", fstype);
661                 return (-ENOTSUP);
662         }

664         /* Convert some Linux flags to Solaris flags. */
665         if (flags & LX_MS_RDONLY)
666                 sflags |= MS_RDONLY;
667         if (flags & LX_MS_NOSUID)
668                 sflags |= MS_NOSUID;
669         if (flags & LX_MS_REMOUNT)
670                 sflags |= MS_REMOUNT;

672         /* Convert some Linux flags to Solaris option strings. */
673         if ((flags & LX_MS_NODEV) &&
674             ((rv = i_add_option("nodev", options, sizeof (options))) != 0))
675                 return (rv);
676         if ((flags & LX_MS_NOEXEC) &&
677             ((rv = i_add_option("noexec", options, sizeof (options))) != 0))
678                 return (rv);
679         if ((flags & LX_MS_NOATIME) &&
680             ((rv = i_add_option("noatime", options, sizeof (options))) != 0))
681                 return (rv);

683         lx_debug("\tsolaris mount fstype: %s", fstype);
684         lx_debug("\tsolaris mount options: \"%s\"", options);

686         return (mount(source, target, sflags, fstype, sdataptr, sdatalen,
687             options, sizeof (options)) ? -errno : 0);
688 }

690 /*
691  * umount() is identical, though it is implemented on top of umount2() in
692  * Solaris so it cannot be a pass-thru system call.
693  */
694 int
695 lx_umount(uintptr_t p1)
696 {
697         return (umount((char *)p1) ? -errno : 0);
698 }

700 /*
701  * The Linux umount2() system call is identical but has a different value for
702  * MNT_FORCE (the logical equivalent to MS_FORCE).
703  */
704 #define LX_MNT_FORCE    0x1

706 int
707 lx_umount2(uintptr_t p1, uintptr_t p2)
708 {
709         char *path = (char *)p1;
710         int flags = 0;

712         if (p2 & ~LX_MNT_FORCE)
713                 return (-EINVAL);

715         if (p2 & LX_MNT_FORCE)
716                 flags |= MS_FORCE;

718         return (umount2(path, flags) ? -errno : 0);
719 }
720 #endif /* ! codereview */
```

```
**********************************************************
    4355 Tue Jan 14 16:17:03 2014
new/usr/src/lib/brand/lx/lx_brand/common/open.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/types.h>
#include <sys/stat.h>
#include <sys/inttypes.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <libintl.h>
#include <stdio.h>

#include <sys/lx_types.h>
#include <sys/lx_debug.h>
#include <sys/lx_syscall.h>
#include <sys/lx_fcntl.h>
#include <sys/lx_misc.h>

static int
ltos_open_flags(uintptr_t p2)
{
        int flags;

        if ((p2 & O_ACCMODE) == LX_O_RDONLY)
                flags = O_RDONLY;
        else if ((p2 & O_ACCMODE) == LX_O_WRONLY)
                flags = O_WRONLY;
        else
                flags = O_RDWR;

        if (p2 & LX_O_CREAT) {
                flags |= O_CREAT;
        }

        if (p2 & LX_O_EXCL)
                flags |= O_EXCL;
        if (p2 & LX_O_NOCTTY)
                flags |= O_NOCTTY;
```



```
        if (p2 & LX_O_TRUNC)
                flags |= O_TRUNC;
        if (p2 & LX_O_APPEND)
                flags |= O_APPEND;
        if (p2 & LX_O_NONBLOCK)
                flags |= O_NONBLOCK;
        if (p2 & LX_O_SYNC)
                flags |= O_SYNC;
        if (p2 & LX_O_LARGEFILE)
                flags |= O_LARGEFILE;
        if (p2 & LX_O_NOFOLLOW)
                flags |= O_NOFOLLOW;

        /*
         * Linux uses the LX_O_DIRECT flag to do raw, synchronous I/O to the
         * device backing the fd in question.  Solaris doesn't have similar
         * functionality, but we can attempt to simulate it using the flags
         * (O_RSYNC|O_SYNC) and directio(3C).
         *
         * The LX_O_DIRECT flag also requires that the transfer size and
         * alignment of I/O buffers be a multiple of the logical block size for
         * the underlying file system, but frankly there isn't an easy way to
         * support that functionality without doing something like adding an
         * fcntl(2) flag to denote LX_O_DIRECT mode.
         *
         * Since LX_O_DIRECT is merely a performance advisory, we'll just
         * emulate what we can and trust that the only applications expecting
         * an error when performing I/O from a misaligned buffer or when
         * passing a transfer size is not a multiple of the underlying file
         * system block size will be test suites.
         */
        if (p2 & LX_O_DIRECT)
                flags |= (O_RSYNC|O_SYNC);

        return (flags);
}

static int
lx_open_postprocess(int fd, uintptr_t p2)
{
        struct stat64 statbuf;

        /*
         * Check the file type AFTER opening the file to avoid a race condition
         * where the file we want to open could change types between a stat64()
         * and an open().
         */
        if (p2 & LX_O_DIRECTORY) {
                if (fstat64(fd, &statbuf) < 0) {
                        int ret = -errno;

                        (void) close(fd);
                        return (ret);
                } else if (!S_ISDIR(statbuf.st_mode)) {
                        (void) close(fd);
                        return (-ENOTDIR);
                }
        }

        if (p2 & LX_O_DIRECT)
                (void) directio(fd, DIRECTIO_ON);

        /*
         * Set the ASYNC flag if passsed.
         */
        if (p2 & LX_O_ASYNC) {
```

```
128                 if (fcntl(fd, F_SETFL, FASYNC) < 0) {
129                         int ret = -errno;

131                         (void) close(fd);
132                         return (ret);
133                 }
134         }

136         return (fd);
137 }

139 int
140 lx_openat(uintptr_t ext1, uintptr_t p1, uintptr_t p2, uintptr_t p3)
141 {
142         int atfd = (int)ext1;
143         int flags, fd;
144         mode_t mode = 0;
145         char *path = (char *)p1;

147         if (atfd == LX_AT_FDCWD)
148                 atfd = AT_FDCWD;

150         flags = ltos_open_flags(p2);

152         if (flags & O_CREAT) {
153                 mode = (mode_t)p3;
154         }

156         lx_debug("\topenat(%d, %s, 0%o, 0%o)", atfd, path, flags, mode);

158         if ((fd = openat(atfd, path, flags, mode)) < 0)
159                 return (-errno);

161         return (lx_open_postprocess(fd, p2));
162 }

164 int
165 lx_open(uintptr_t p1, uintptr_t p2, uintptr_t p3)
166 {
167         int flags, fd;
168         mode_t mode = 0;
169         char *path = (char *)p1;

171         flags = ltos_open_flags(p2);

173         if (flags & O_CREAT) {
174                 mode = (mode_t)p3;
175         }

177         lx_debug("\topen(%s, 0%o, 0%o)", path, flags, mode);

179         if ((fd = open(path, flags, mode)) < 0)
180                 return (-errno);

182         return (lx_open_postprocess(fd, p2));
183 }
184 #endif /* ! codereview */
```

```
**********************************************************
    3004 Tue Jan 14 16:17:03 2014
new/usr/src/lib/brand/lx/lx_brand/common/pgrp.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/types.h>
#include <unistd.h>
#include <errno.h>
#include <sys/lx_misc.h>

int
lx_getpgrp(void)
{
        int ret;

        ret = getpgrp();

        /*
         * If the pgrp is that of the init process, return the value Linux
         * expects.
         */
        if (ret == zoneinit_pid)
                return (LX_INIT_PGID);

        return ((ret == -1) ? -errno : ret);
}

int
lx_getpgid(uintptr_t p1)
{
        pid_t spid;
        int pid = (int)p1;
        int ret;

        if (pid < 0)
                return (-ESRCH);

        /*
```



```
         * If the supplied pid matches that of the init process, return
         * the pgid Linux expects.
         */
        if (pid == zoneinit_pid)
                return (LX_INIT_PGID);

        if ((ret = lx_lpid_to_spid(pid, &spid)) < 0)
                return (ret);

        ret = getpgid(spid);

        /*
         * If the pgid is that of the init process, return the value Linux
         * expects.
         */
        if (ret == zoneinit_pid)
                return (LX_INIT_PGID);

        return ((ret == -1) ? -errno : ret);
}

int
lx_setpgid(uintptr_t p1, uintptr_t p2)
{
        pid_t pid = (pid_t)p1;
        pid_t pgid = (pid_t)p2;
        pid_t spid, spgid;
        int ret;

        if (pid < 0)
                return (-ESRCH);

        if (pgid < 0)
                return (-EINVAL);

        if ((ret = lx_lpid_to_spid(pid, &spid)) < 0)
                return (ret);

        if (pgid == 0)
                spgid = spid;
        else if ((ret = lx_lpid_to_spid(pgid, &spgid)) < 0)
                return (ret);

        ret = setpgid(spid, spgid);

        return ((ret == 0) ? 0 : -errno);
}

int
lx_getsid(uintptr_t p1)
{
        pid_t spid;
        int pid = (int)p1;
        int ret;

        if (pid < 0)
                return (-ESRCH);

        /*
         * If the supplied matches that of the init process, return the value
         * Linux expects.
         */
        if (pid == zoneinit_pid)
                return (LX_INIT_SID);

        if ((ret = lx_lpid_to_spid(pid, &spid)) < 0)
```

```
128                 return (ret);

130         ret = getsid(spid);

132         /*
133          * If the sid is that of the init process, return the value Linux
134          * expects.
135          */
136         if (ret == zoneinit_pid)
137                 return (LX_INIT_SID);

139         return ((ret == -1) ? -errno : ret);
140 }

142 int
143 lx_setsid(void)
144 {
145         int ret;

147         ret = setsid();

149         /*
150          * If the pgid is that of the init process, return the value Linux
151          * expects.
152          */
153         if (ret == zoneinit_pid)
154                 return (LX_INIT_SID);

156         return ((ret == -1) ? -errno : ret);
157 }
158 #endif /* ! codereview */
```

```
**********************************************************
    5629 Tue Jan 14 16:17:03 2014
new/usr/src/lib/brand/lx/lx_brand/common/poll_select.c
LX zone support should now build and packages of relevance produced.
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <assert.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloca.h>
#include <signal.h>
#include <strings.h>
#include <sys/param.h>
#include <sys/brand.h>
#include <sys/poll.h>
#include <sys/syscall.h>
#include <sys/lx_debug.h>
#include <sys/lx_poll.h>
#include <sys/lx_syscall.h>
#include <sys/lx_brand.h>
#include <sys/lx_misc.h>

extern int select_large_fdset(int nfds, fd_set *in0, fd_set *out0, fd_set *ex0,
        struct timeval *tv);

int
lx_select(uintptr_t p1, uintptr_t p2, uintptr_t p3, uintptr_t p4,
        uintptr_t p5)
{
        int nfds = (int)p1;
        fd_set *rfdsp = NULL;
        fd_set *wfdsp = NULL;
        fd_set *efdsp = NULL;
        struct timeval tv, *tvp = NULL;
        int fd_set_len = howmany(nfds, 8);
```



```
        int r;
        hrtime_t start = NULL, end;

        lx_debug("\tselect(%d, 0x%p, x%p, 0x%p. 0x%p, 0x%p)",
            nfds, rfdsp, wfdsp, efdsp, tvp);

        if (nfds > 0) {
                if (p2 != NULL) {
                        rfdsp = SAFE_ALLOCA(fd_set_len);
                        if (rfdsp == NULL)
                                return (-ENOMEM);
                        if (uucopy((void *)p2, rfdsp, fd_set_len) != 0)
                                return (-errno);
                }
                if (p3 != NULL) {
                        wfdsp = SAFE_ALLOCA(fd_set_len);
                        if (wfdsp == NULL)
                                return (-ENOMEM);
                        if (uucopy((void *)p3, wfdsp, fd_set_len) != 0)
                                return (-errno);
                }
                if (p4 != NULL) {
                        efdsp = SAFE_ALLOCA(fd_set_len);
                        if (efdsp == NULL)
                                return (-ENOMEM);
                        if (uucopy((void *)p4, efdsp, fd_set_len) != 0)
                                return (-errno);
                }
        }
        if (p5 != NULL) {
                tvp = &tv;
                if (uucopy((void *)p5, &tv, sizeof (tv)) != 0)
                        return (-errno);
                start = gethrtime();
        }

        if (nfds >= FD_SETSIZE)
                r = select_large_fdset(nfds, rfdsp, wfdsp, efdsp, tvp);
        else
                r = select(nfds, rfdsp, wfdsp, efdsp, tvp);
        if (r < 0)
                return (-errno);

        if (tvp != NULL) {
                long long tv_total;

                /*
                 * Linux updates the timeval parameter for select() calls
                 * with the amount of time that left before the select
                 * would have timed out.
                 */
                end = gethrtime();
                tv_total = (tv.tv_sec * MICROSEC) + tv.tv_usec;
                tv_total -= ((end - start) / (NANOSEC / MICROSEC));
                if (tv_total < 0) {
                        tv.tv_sec = 0;
                        tv.tv_usec = 0;
                } else {
                        tv.tv_sec = tv_total / MICROSEC;
                        tv.tv_usec = tv_total % MICROSEC;
                }

                if (uucopy(&tv, (void *)p5, sizeof (tv)) != 0)
                        return (-errno);
        }
```

```
127         if ((rfdsp != NULL) && (uucopy(rfdsp, (void *)p2, fd_set_len) != 0))
128                 return (-errno);
129         if ((wfdsp != NULL) && (uucopy(wfdsp, (void *)p3, fd_set_len) != 0))
130                 return (-errno);
131         if ((efdsp != NULL) && (uucopy(efdsp, (void *)p4, fd_set_len) != 0))
132                 return (-errno);

134         return (r);
135 }

137 int
138 lx_poll(uintptr_t p1, uintptr_t p2, uintptr_t p3)
139 {
140         struct pollfd   *lfds, *sfds;
141         nfds_t          nfds = (nfds_t)p2;
142         int             fds_size, i, rval, revents;

144         /*
145          * Note: we are assuming that the Linux and Solaris pollfd
146          * structures are identical.  Copy in the linux poll structure.
147          */
148         fds_size = sizeof (struct pollfd) * nfds;
149         lfds = (struct pollfd *)SAFE_ALLOCA(fds_size);
150         if (lfds == NULL)
151                 return (-ENOMEM);
152         if (uucopy((void *)p1, lfds, fds_size) != 0)
153                 return (-errno);

155         /*
156          * The poll system call modifies the poll structures passed in
157          * so we'll need to make an exra copy of them.
158          */
159         sfds = (struct pollfd *)SAFE_ALLOCA(fds_size);
160         if (sfds == NULL)
161                 return (-ENOMEM);

163         /* Convert the Linux events bitmask into the Solaris equivalent. */
164         for (i = 0; i < nfds; i++) {
165                 /*
166                  * If the caller is polling for an unsupported event, we
167                  * have to bail out.
168                  */
169                 if (lfds[i].events & ~LX_POLL_SUPPORTED_EVENTS) {
170                         lx_unsupported("unsupported poll events requested: "
171                             "events=0x%x", lfds[i].events);
172                         return (-ENOTSUP);
173                 }

175                 sfds[i].fd = lfds[i].fd;
176                 sfds[i].events = lfds[i].events & LX_POLL_COMMON_EVENTS;
177                 if (lfds[i].events & LX_POLLWRNORM)
178                         sfds[i].events |= POLLWRNORM;
179                 if (lfds[i].events & LX_POLLWRBAND)
180                         sfds[i].events |= POLLWRBAND;
181                 sfds[i].revents = 0;
182         }

184         lx_debug("\tpoll(0x%p, %u, %d)", sfds, nfds, (int)p3);

186         if ((rval = poll(sfds, nfds, (int)p3)) < 0)
187                 return (-errno);

189         /* Convert the Solaris revents bitmask into the Linux equivalent */
190         for (i = 0; i < nfds; i++) {
191                 revents = sfds[i].revents & LX_POLL_COMMON_EVENTS;
192                 if (sfds[i].revents & POLLWRBAND)
```



```
193                         revents |= LX_POLLWRBAND;

195                 /*
196                  * Be carefull because on solaris POLLOUT and POLLWRNORM
197                  * are defined to the same values but on linux they
198                  * are not.
199                  */
200                 if (sfds[i].revents & POLLOUT) {
201                         if ((lfds[i].events & LX_POLLOUT) == 0)
202                                 revents &= ~LX_POLLOUT;
203                         if (lfds[i].events & LX_POLLWRNORM)
204                                 revents |= LX_POLLWRNORM;
205                 }

207                 lfds[i].revents = revents;
208         }

210         /* Copy out the results */
211         if (uucopy(lfds, (void *)p1, fds_size) != 0)
212                 return (-errno);

214         return (rval);
215 }
216 #endif /* ! codereview */
```

```
**********************************************************
    2215 Tue Jan 14 16:17:03 2014
new/usr/src/lib/brand/lx/lx_brand/common/priority.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <errno.h>
#include <sys/types.h>
#include <sys/lx_debug.h>
#include <sys/lx_misc.h>
#include <sys/lx_syscall.h>
#include <sys/lx_types.h>
#include <sys/resource.h>
#include <sys/lx_misc.h>

int
lx_getpriority(uintptr_t p1, uintptr_t p2)
{
        uint_t  which = (int)p1;
        id_t    who  = (id_t)p2;
        int     ret;

        /*
         * The only valid values for 'which' are positive integers, and unlike
         * Solaris, linux doesn't support anything past PRIO_USER.
         */
        if (which > PRIO_USER)
                return (-EINVAL);

        lx_debug("\tgetpriority(%d, %d)", which, who);

        errno = 0;

        if ((which == PRIO_PROCESS) && (who == 1))
                who = zoneinit_pid;

        ret = getpriority(which, who);
        if (ret == -1 && errno != 0)
                return (-errno);
```



```
        /*
         * The return value of the getpriority syscall is biased by 20 to avoid
         * returning negative values when successful.
         */
        return (20 - ret);
}

int
lx_setpriority(uintptr_t p1, uintptr_t p2, uintptr_t p3)
{
        int which = (int)p1;
        id_t who  = (id_t)p2;
        int prio  = (int)p3;
        int rval;

        if (which > PRIO_USER)
                return (-EINVAL);

        lx_debug("\tsetpriority(%d, %d, %d)", which, who, prio);

        if ((which == PRIO_PROCESS) && (who == 1))
                who = zoneinit_pid;

        rval = setpriority(which, who, prio);

        return ((rval == -1) ? -errno : rval);
}
#endif /* ! codereview */
```

```
**********************************************************
   50402 Tue Jan 14 16:17:04 2014
new/usr/src/lib/brand/lx/lx_brand/common/ptrace.c
LX zone support should now build and packages of relevance produced.
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <errno.h>
#include <sys/types.h>
#include <sys/param.h>
#include <sys/lx_misc.h>
#include <sys/lx_debug.h>
#include <sys/lx_syscall.h>
#include <sys/lx_signal.h>
#include <sys/lx_thread.h>
#include <sys/lwp.h>
#include <unistd.h>
#include <fcntl.h>
#include <procfs.h>
#include <sys/frame.h>
#include <strings.h>
#include <signal.h>
#include <stddef.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <sys/auxv.h>
#include <thread.h>
#include <pthread.h>
#include <synch.h>
#include <elf.h>
#include <ieeefp.h>
#include <assert.h>
#include <libintl.h>

/*
 * Linux ptrace compatibility.
 *
 * The brand support for ptrace(2) is built on top of the Solaris /proc
 * interfaces, mounted at /native/proc in the zone.  This gets quite
 * complicated due to the way ptrace works and the Solaris realization of the
 * Linux threading model.
```



```
 *
 * ptrace can only interact with a process if we are tracing it, and it is
 * currently stopped. There are two ways a process can begin tracing another
 * process:
 *
 *   PTRACE_TRACEME
 *
 *   A child process can use PTRACE_TRACEME to indicate that it wants to be
 *   traced by the parent. This sets the ptrace compatibility flag in /proc
 *   which causes ths ptrace consumer to be notified through the wait(2)
 *   system call of events of interest. PTRACE_TRACEME is typically used by
 *   the debugger by forking a process, using PTRACE_TRACEME, and finally
 *   doing an exec of the specified program.
 *
 *
 *   PTRACE_ATTACH
 *
 *   We can attach to a process using PTRACE_ATTACH. This is considerably
 *   more complicated than the previous case. On Linux, the traced process is
 *   effectively reparented to the ptrace consumer so that event notification
 *   can go through the normal wait(2) system call. Solaris has no such
 *   ability to reparent a process (nor should it) so some trickery was
 *   required.
 *
 *   When the ptrace consumer uses PTRACE_ATTACH it forks a monitor child
 *   process. The monitor enables the /proc ptrace flag for itself and uses
 *   the native /proc mechanisms to observe the traced process and wait for
 *   events of interest. When the traced process stops, the monitor process
 *   sends itself a SIGTRAP thus rousting its parent process (the ptrace
 *   consumer) out of wait(2). We then translate the process id and status
 *   code from wait(2) to those of the traced process.
 *
 *   To detach from the process we just have to clean up tracing flags and
 *   clean up the monitor.
 *
 * ptrace can only interact with a process if we have traced it, and it is
 * currently stopped (see is_traced()). For threads, there's no way to
 * distinguish whether ptrace() has been called for all threads or some
 * subset. Since most clients will be tracing all threads, and erroneously
 * allowing ptrace to access a non-traced thread is non-fatal (or at least
 * would be fatal on linux), we ignore this aspect of the problem.
 */

#define LX_PTRACE_TRACEME       0
#define LX_PTRACE_PEEKTEXT      1
#define LX_PTRACE_PEEKDATA      2
#define LX_PTRACE_PEEKUSER      3
#define LX_PTRACE_POKETEXT      4
#define LX_PTRACE_POKEDATA      5
#define LX_PTRACE_POKEUSER      6
#define LX_PTRACE_CONT          7
#define LX_PTRACE_KILL          8
#define LX_PTRACE_SINGLESTEP    9
#define LX_PTRACE_GETREGS       12
#define LX_PTRACE_SETREGS       13
#define LX_PTRACE_GETFPREGS     14
#define LX_PTRACE_SETFPREGS     15
#define LX_PTRACE_ATTACH        16
#define LX_PTRACE_DETACH        17
#define LX_PTRACE_GETFPXREGS    18
#define LX_PTRACE_SETFPXREGS    19
#define LX_PTRACE_SYSCALL       24

/*
 * This corresponds to the user_i387_struct Linux structure.
 */
```

```
127 typedef struct lx_user_fpregs {
128         long lxuf_cwd;
129         long lxuf_swd;
130         long lxuf_twd;
131         long lxuf_fip;
132         long lxuf_fcs;
133         long lxuf_foo;
134         long lxuf_fos;
135         long lxuf_st_space[20];
136 } lx_user_fpregs_t;

138 /*
139  * This corresponds to the user_fxsr_struct Linux structure.
140  */
141 typedef struct lx_user_fpxregs {
142         uint16_t lxux_cwd;
143         uint16_t lxux_swd;
144         uint16_t lxux_twd;
145         uint16_t lxux_fop;
146         long lxux_fip;
147         long lxux_fcs;
148         long lxux_foo;
149         long lxux_fos;
150         long lxux_mxcsr;
151         long lxux_reserved;
152         long lxux_st_space[32];
153         long lxux_xmm_space[32];
154         long lxux_padding[56];
155 } lx_user_fpxregs_t;

157 /*
158  * This corresponds to the user_regs_struct Linux structure.
159  */
160 typedef struct lx_user_regs {
161         long lxur_ebx;
162         long lxur_ecx;
163         long lxur_edx;
164         long lxur_esi;
165         long lxur_edi;
166         long lxur_ebp;
167         long lxur_eax;
168         long lxur_xds;
169         long lxur_xes;
170         long lxur_xfs;
171         long lxur_xgs;
172         long lxur_orig_eax;
173         long lxur_eip;
174         long lxur_xcs;
175         long lxur_eflags;
176         long lxur_esp;
177         long lxur_xss;
178 } lx_user_regs_t;

180 typedef struct lx_user {
181         lx_user_regs_t lxu_regs;
182         int lxu_fpvalid;
183         lx_user_fpregs_t lxu_i387;
184         ulong_t lxu_tsize;
185         ulong_t lxu_dsize;
186         ulong_t lxu_ssize;
187         ulong_t lxu_start_code;
188         ulong_t lxu_start_stack;
189         long lxu_signal;
190         int lxu_reserved;
191         lx_user_regs_t *lxu_ar0;
192         lx_user_fpregs_t *lxu_fpstate;
```



```
193         ulong_t lxu_magic;
194         char lxu_comm[32];
195         int lxu_debugreg[8];
196 } lx_user_t;

198 typedef struct ptrace_monitor_map {
199         struct ptrace_monitor_map *pmm_next;    /* next pointer */
200         pid_t pmm_monitor;                      /* monitor child process */
201         pid_t pmm_target;                       /* traced Linux pid */
202         pid_t pmm_pid;                          /* Solaris pid */
203         lwpid_t pmm_lwpid;                      /* Solaris lwpid */
204         uint_t pmm_exiting;                     /* detached */
205 } ptrace_monitor_map_t;

207 typedef struct ptrace_state_map {
208         struct ptrace_state_map *psm_next;      /* next pointer */
209         pid_t           psm_pid;                /* Solaris pid */
210         uintptr_t       psm_debugreg[8];        /* debug registers */
211 } ptrace_state_map_t;

213 static ptrace_monitor_map_t *ptrace_monitor_map = NULL;
214 static ptrace_state_map_t *ptrace_state_map = NULL;
215 static mutex_t ptrace_map_mtx = DEFAULTMUTEX;

217 extern void *_START_;

219 static sigset_t blockable_sigs;

221 #pragma init(ptrace_init)
222 void
223 ptrace_init(void)
224 {
225         (void) sigfillset(&blockable_sigs);
226         (void) sigdelset(&blockable_sigs, SIGKILL);
227         (void) sigdelset(&blockable_sigs, SIGSTOP);
228 }

230 /*
231  * Given a pid, open the named file under /native/proc/<pid>/name using the
232  * given mode.
233  */
234 static int
235 open_procfile(pid_t pid, int mode, const char *name)
236 {
237         char path[MAXPATHLEN];

239         (void) snprintf(path, sizeof (path), "/native/proc/%d/%s", pid, name);

241         return (open(path, mode));
242 }

244 /*
245  * Given a pid and lwpid, open the named file under
246  * /native/proc/<pid>/<lwpid>/name using the given mode.
247  */
248 static int
249 open_lwpfile(pid_t pid, lwpid_t lwpid, int mode, const char *name)
250 {
251         char path[MAXPATHLEN];

253         (void) snprintf(path, sizeof (path), "/native/proc/%d/lwp/%d/%s",
254             pid, lwpid, name);

256         return (open(path, mode));
257 }
```

```
259 static int
260 get_status(pid_t pid, pstatus_t *psp)
261 {
262         int fd;

264         if ((fd = open_procfile(pid, O_RDONLY, "status")) < 0)
265                 return (-ESRCH);

267         if (read(fd, psp, sizeof (pstatus_t)) != sizeof (pstatus_t)) {
268                 (void) close(fd);
269                 return (-EIO);
270         }

272         (void) close(fd);

274         return (0);
275 }

277 static int
278 get_lwpstatus(pid_t pid, lwpid_t lwpid, lwpstatus_t *lsp)
279 {
280         int fd;

282         if ((fd = open_lwpfile(pid, lwpid, O_RDONLY, "lwpstatus")) < 0)
283                 return (-ESRCH);

285         if (read(fd, lsp, sizeof (lwpstatus_t)) != sizeof (lwpstatus_t)) {
286                 (void) close(fd);
287                 return (-EIO);
288         }

290         (void) close(fd);

292         return (0);
293 }

295 static uintptr_t
296 syscall_regs(int fd, uintptr_t fp, pid_t pid)
297 {
298         uintptr_t addr, done;
299         struct frame fr;
300         auxv_t auxv;
301         int afd;
302         Elf32_Phdr phdr;

304         /*
305          * Try to walk the stack looking for a return address that corresponds
306          * to the traced process's lx_emulate_done symbol. This relies on the
307          * fact that the brand library in the traced process is the same as the
308          * brand library in this process (indeed, this is true of all processes
309          * in a given branded zone).
310          */

312         /*
313          * Find the base address for the brand library in the traced process
314          * by grabbing the AT_PHDR auxv entry, reading in the program header
315          * at that location and subtracting off the p_vaddr member. We use
316          * this to compute the location of lx_emulate done in the traced
317          * process.
318          */
319         if ((afd = open_procfile(pid, O_RDONLY, "auxv")) < 0)
320                 return (0);

322         do {
323                 if (read(afd, &auxv, sizeof (auxv)) != sizeof (auxv)) {
324                         (void) close(afd);
```



```
325                         return (0);
326                 }
327         } while (auxv.a_type != AT_PHDR);

329         (void) close(afd);

331         if (pread(fd, &phdr, sizeof (phdr), auxv.a_un.a_val) != sizeof (phdr)) {
332                 lx_debug("failed to read brand library's phdr");
333                 return (0);
334         }

336         addr = auxv.a_un.a_val - phdr.p_vaddr;
337         done = (uintptr_t)&lx_emulate_done - (uintptr_t)&_START_ + addr;

339         fr.fr_savfp = fp;

341         do {
342                 addr = fr.fr_savfp;
343                 if (pread(fd, &fr, sizeof (fr), addr) != sizeof (fr)) {
344                         lx_debug("ptrace read failed for stack walk");
345                         return (0);
346                 }

348                 if (addr >= fr.fr_savfp) {
349                         lx_debug("ptrace stack not monotonically increasing "
350                             "%p %p (%p)", addr, fr.fr_savfp, done);
351                         return (0);
352                 }
353         } while (fr.fr_savpc != done);

355         /*
356          * The first argument to lx_emulate is known to be an lx_regs_t
357          * structure and the ABI specifies that it will be placed on the stack
358          * immediately preceeding the return address.
359          */
360         addr += sizeof (fr);
361         if (pread(fd, &addr, sizeof (addr), addr) != sizeof (addr)) {
362                 lx_debug("ptrace stack failed to read register set address");
363                 return (0);
364         }

366         return (addr);
367 }

369 static int
370 getregs(pid_t pid, lwpid_t lwpid, lx_user_regs_t *rp)
371 {
372         lwpstatus_t status;
373         uintptr_t addr;
374         int fd, ret;

376         if ((ret = get_lwpstatus(pid, lwpid, &status)) != 0)
377                 return (ret);

379         if ((fd = open_procfile(pid, O_RDONLY, "as")) < 0)
380                 return (-ESRCH);

382         /*
383          * If we find the syscall regs (and are therefore in an emulated
384          * syscall, use the register set at given address. Otherwise, use the
385          * registers as reported by /proc.
386          */
387         if ((addr = syscall_regs(fd, status.pr_reg[EBP], pid)) != 0) {
388                 lx_regs_t regs;

390                 if (pread(fd, &regs, sizeof (regs), addr) != sizeof (regs)) {
```

```
391                         (void) close(fd);
392                         lx_debug("ptrace failed to read register set");
393                         return (-EIO);
394                 }

396                 (void) close(fd);

398                 rp->lxur_ebx = regs.lxr_ebx;
399                 rp->lxur_ecx = regs.lxr_ecx;
400                 rp->lxur_edx = regs.lxr_edx;
401                 rp->lxur_esi = regs.lxr_esi;
402                 rp->lxur_edi = regs.lxr_edi;
403                 rp->lxur_ebp = regs.lxr_ebp;
404                 rp->lxur_eax = regs.lxr_eax;
405                 rp->lxur_xds = status.pr_reg[DS];
406                 rp->lxur_xes = status.pr_reg[ES];
407                 rp->lxur_xfs = status.pr_reg[FS];
408                 rp->lxur_xgs = regs.lxr_gs;
409                 rp->lxur_orig_eax = regs.lxr_orig_eax;
410                 rp->lxur_eip = regs.lxr_eip;
411                 rp->lxur_xcs = status.pr_reg[CS];
412                 rp->lxur_eflags = status.pr_reg[EFL];
413                 rp->lxur_esp = regs.lxr_esp;
414                 rp->lxur_xss = status.pr_reg[SS];

416         } else {
417                 (void) close(fd);

419                 rp->lxur_ebx = status.pr_reg[EBX];
420                 rp->lxur_ecx = status.pr_reg[ECX];
421                 rp->lxur_edx = status.pr_reg[EDX];
422                 rp->lxur_esi = status.pr_reg[ESI];
423                 rp->lxur_edi = status.pr_reg[EDI];
424                 rp->lxur_ebp = status.pr_reg[EBP];
425                 rp->lxur_eax = status.pr_reg[EAX];
426                 rp->lxur_xds = status.pr_reg[DS];
427                 rp->lxur_xes = status.pr_reg[ES];
428                 rp->lxur_xfs = status.pr_reg[FS];
429                 rp->lxur_xgs = status.pr_reg[GS];
430                 rp->lxur_orig_eax = 0;
431                 rp->lxur_eip = status.pr_reg[EIP];
432                 rp->lxur_xcs = status.pr_reg[CS];
433                 rp->lxur_eflags = status.pr_reg[EFL];
434                 rp->lxur_esp = status.pr_reg[UESP];
435                 rp->lxur_xss = status.pr_reg[SS];

437                 /*
438                  * If the target process has just returned from exec, it's not
439                  * going to be sitting in the emulation function. In that case
440                  * we need to manually fake up the values for %eax and orig_eax
441                  * to indicate a successful return and that the traced process
442                  * had called execve (respectively).
443                  */
444                 if (status.pr_why == PR_SYSEXIT &&
445                     status.pr_what == SYS_execve) {
446                         rp->lxur_eax = 0;
447                         rp->lxur_orig_eax = LX_SYS_execve;
448                 }
449         }

451         return (0);
452 }

454 static int
455 setregs(pid_t pid, lwpid_t lwpid, const lx_user_regs_t *rp)
456 {
```



```
457         long ctl[1 + sizeof (prgregset_t) / sizeof (long)];
458         lwpstatus_t status;
459         uintptr_t addr;
460         int fd, ret;

462         if ((ret = get_lwpstatus(pid, lwpid, &status)) != 0)
463                 return (ret);

465         if ((fd = open_procfile(pid, O_RDWR, "as")) < 0)
466                 return (-ESRCH);

468         /*
469          * If we find the syscall regs (and are therefore in an emulated
470          * syscall, modify the register set at given address and set the
471          * remaining registers through the /proc interface. Otherwise just use
472          * the /proc interface to set register values;
473          */
474         if ((addr = syscall_regs(fd, status.pr_reg[EBP], pid)) != 0) {
475                 lx_regs_t regs;

477                 regs.lxr_ebx = rp->lxur_ebx;
478                 regs.lxr_ecx = rp->lxur_ecx;
479                 regs.lxr_edx = rp->lxur_edx;
480                 regs.lxr_esi = rp->lxur_esi;
481                 regs.lxr_edi = rp->lxur_edi;
482                 regs.lxr_ebp = rp->lxur_ebp;
483                 regs.lxr_eax = rp->lxur_eax;
484                 regs.lxr_gs = rp->lxur_xgs;
485                 regs.lxr_orig_eax = rp->lxur_orig_eax;
486                 regs.lxr_eip = rp->lxur_eip;
487                 regs.lxr_esp = rp->lxur_esp;

489                 if (pwrite(fd, &regs, sizeof (regs), addr) != sizeof (regs)) {
490                         (void) close(fd);
491                         lx_debug("ptrace failed to write register set");
492                         return (-EIO);
493                 }

495                 (void) close(fd);

497                 status.pr_reg[DS] = rp->lxur_xds;
498                 status.pr_reg[ES] = rp->lxur_xes;
499                 status.pr_reg[FS] = rp->lxur_xfs;
500                 status.pr_reg[CS] = rp->lxur_xcs;
501                 status.pr_reg[EFL] = rp->lxur_eflags;
502                 status.pr_reg[SS] = rp->lxur_xss;

504         } else {
505                 (void) close(fd);

507                 status.pr_reg[EBX] = rp->lxur_ebx;
508                 status.pr_reg[ECX] = rp->lxur_ecx;
509                 status.pr_reg[EDX] = rp->lxur_edx;
510                 status.pr_reg[ESI] = rp->lxur_esi;
511                 status.pr_reg[EDI] = rp->lxur_edi;
512                 status.pr_reg[EBP] = rp->lxur_ebp;
513                 status.pr_reg[EAX] = rp->lxur_eax;
514                 status.pr_reg[DS] = rp->lxur_xds;
515                 status.pr_reg[ES] = rp->lxur_xes;
516                 status.pr_reg[FS] = rp->lxur_xfs;
517                 status.pr_reg[GS] = rp->lxur_xgs;
518                 status.pr_reg[EIP] = rp->lxur_eip;
519                 status.pr_reg[CS] = rp->lxur_xcs;
520                 status.pr_reg[EFL] = rp->lxur_eflags;
521                 status.pr_reg[UESP] = rp->lxur_esp;
522                 status.pr_reg[SS] = rp->lxur_xss;
```

```
523                 status.pr_reg[SS] = rp->lxur_xss;
524         }

526         if ((fd = open_lwpfile(pid, lwpid, O_WRONLY, "lwpctl")) < 0)
527                 return (-ESRCH);

529         ctl[0] = PCSREG;
530         bcopy(status.pr_reg, &ctl[1], sizeof (prgregset_t));

532         if (write(fd, &ctl, sizeof (ctl)) != sizeof (ctl)) {
533                 (void) close(fd);
534                 return (-EIO);
535         }

537         (void) close(fd);

539         return (0);
540 }

542 static int
543 getfpregs(pid_t pid, lwpid_t lwpid, lx_user_fpregs_t *rp)
544 {
545         lwpstatus_t status;
546         struct _fpstate *fp;
547         char *data;
548         int ret, i;

550         if ((ret = get_lwpstatus(pid, lwpid, &status)) != 0)
551                 return (ret);

553         fp = (struct _fpstate *)&status.pr_fpreg.fp_reg_set.fpchip_state;

555         rp->lxuf_cwd = fp->cw;
556         rp->lxuf_swd = fp->sw;
557         rp->lxuf_twd = fp->tag;
558         rp->lxuf_fip = fp->ipoff;
559         rp->lxuf_fcs = fp->cssel;
560         rp->lxuf_foo = fp->dataoff;
561         rp->lxuf_fos = fp->datasel;

563         /*
564          * The Linux structure uses 10 bytes per floating-point register.
565          */
566         data = (char *)&rp->lxuf_st_space[0];
567         for (i = 0; i < 8; i++) {
568                 bcopy(&fp->_st[i], data, 10);
569                 data += 10;
570         }

572         return (0);
573 }

575 static int
576 setfpregs(pid_t pid, lwpid_t lwpid, const lx_user_fpregs_t *rp)
577 {
578         lwpstatus_t status;
579         struct {
580                 long cmd;
581                 prfpregset_t regs;
582         } ctl;
583         struct _fpstate *fp = (struct _fpstate *)&ctl.regs;
584         char *data;
585         int ret, i, fd;

587         if ((ret = get_lwpstatus(pid, lwpid, &status)) != 0)
588                 return (ret);
```



```
590         bcopy(&status.pr_fpreg, &ctl.regs, sizeof (ctl.regs));

592         fp->cw = rp->lxuf_cwd;
593         fp->sw = rp->lxuf_swd;
594         fp->tag = rp->lxuf_twd;
595         fp->ipoff = rp->lxuf_fip;
596         fp->cssel = rp->lxuf_fcs;
597         fp->dataoff = rp->lxuf_foo;
598         fp->datasel = rp->lxuf_fos;

600         /*
601          * The Linux structure uses 10 bytes per floating-point register.
602          */
603         data = (char *)&rp->lxuf_st_space[0];
604         for (i = 0; i < 8; i++) {
605                 bcopy(data, &fp->_st[i], 10);
606                 data += 10;
607         }

609         if ((fd = open_lwpfile(pid, lwpid, O_WRONLY, "lwpctl")) < 0)
610                 return (-ESRCH);

612         ctl.cmd = PCSFPREG;
613         if (write(fd, &ctl, sizeof (ctl)) != sizeof (ctl)) {
614                 (void) close(fd);
615                 return (-EIO);
616         }

618         (void) close(fd);

620         return (0);
621 }


624 static int
625 getfpxregs(pid_t pid, lwpid_t lwpid, lx_user_fpxregs_t *rp)
626 {
627         lwpstatus_t status;
628         struct _fpstate *fp;
629         int ret, i;

631         if ((ret = get_lwpstatus(pid, lwpid, &status)) != 0)
632                 return (ret);

634         fp = (struct _fpstate *)&status.pr_fpreg.fp_reg_set.fpchip_state;

636         rp->lxux_cwd = (uint16_t)fp->cw;
637         rp->lxux_swd = (uint16_t)fp->sw;
638         rp->lxux_twd = (uint16_t)fp->tag;
639         rp->lxux_fop = (uint16_t)(fp->cssel >> 16);
640         rp->lxux_fip = fp->ipoff;
641         rp->lxux_fcs = (uint16_t)fp->cssel;
642         rp->lxux_foo = fp->dataoff;
643         rp->lxux_fos = fp->datasel;
644         rp->lxux_mxcsr = status.pr_fpreg.fp_reg_set.fpchip_state.mxcsr;

646         bcopy(fp->xmm, rp->lxux_xmm_space, sizeof (rp->lxux_xmm_space));
647         bzero(rp->lxux_st_space, sizeof (rp->lxux_st_space));
648         for (i = 0; i < 8; i++) {
649                 bcopy(&fp->_st[i], &rp->lxux_st_space[i * 4],
650                     sizeof (fp->_st[i]));
651         }

653         return (0);
654 }
```

```
656 static int
657 setfpxregs(pid_t pid, lwpid_t lwpid, const lx_user_fpxregs_t *rp)
658 {
659         lwpstatus_t status;
660         struct {
661                 long cmd;
662                 prfpregset_t regs;
663         } ctl;
664         struct _fpstate *fp = (struct _fpstate *)&ctl.regs;
665         int ret, i, fd;

667         if ((ret = get_lwpstatus(pid, lwpid, &status)) != 0)
668                 return (ret);

670         bcopy(&status.pr_fpreg, &ctl.regs, sizeof (ctl.regs));

672         fp->cw = rp->lxux_cwd;
673         fp->sw = rp->lxux_swd;
674         fp->tag = rp->lxux_twd;
675         fp->ipoff = rp->lxux_fip;
676         fp->cssel = rp->lxux_fcs | (rp->lxux_fop << 16);
677         fp->dataoff = rp->lxux_foo;
678         fp->datasel = rp->lxux_fos;

680         bcopy(rp->lxux_xmm_space, fp->xmm, sizeof (rp->lxux_xmm_space));
681         for (i = 0; i < 8; i++) {
682                 bcopy(&rp->lxux_st_space[i * 4], &fp->_st[i],
683                     sizeof (fp->_st[i]));
684         }

686         if ((fd = open_lwpfile(pid, lwpid, O_WRONLY, "lwpctl")) < 0)
687                 return (-ESRCH);

689         ctl.cmd = PCSFPREG;
690         if (write(fd, &ctl, sizeof (ctl)) != sizeof (ctl)) {
691                 (void) close(fd);
692                 return (-EIO);
693         }

695         (void) close(fd);

697         return (0);
698 }

700 /*
701  * Solaris does not allow a process to manipulate its own or some
702  * other process's debug registers.  Linux ptrace(2) allows this
703  * and gdb manipulates them for its watchpoint implementation.
704  *
705  * We keep a pseudo set of debug registers for each traced process
706  * and map their contents into the appropriate PCWATCH /proc
707  * operations when they are activated by gdb.
708  *
709  * To understand how the debug registers work on x86 machines,
710  * see section 13.1 of the AMD x86-64 Architecture Programmer's
711  * Manual, Volume 2, System Programming.
712  */
713 static uintptr_t *
714 debug_registers(pid_t pid)
715 {
716         ptrace_state_map_t *p;

718         (void) mutex_lock(&ptrace_map_mtx);
719         for (p = ptrace_state_map; p != NULL; p = p->psm_next) {
720                 if (p->psm_pid == pid)
```



```
721                         break;
722         }
723         if (p == NULL && (p = malloc(sizeof (*p))) != NULL) {
724                 bzero(p, sizeof (*p));
725                 p->psm_pid = pid;
726                 p->psm_next = ptrace_state_map;
727                 p->psm_debugreg[6] = 0xffff0ff0;        /* read as ones */
728                 ptrace_state_map = p;
729         }
730         (void) mutex_unlock(&ptrace_map_mtx);
731         return (p != NULL? p->psm_debugreg : NULL);
732 }

734 static void
735 free_debug_registers(pid_t pid)
736 {
737         ptrace_state_map_t **pp;
738         ptrace_state_map_t *p;

740         /* ASSERT(MUTEX_HELD(&ptrace_map_mtx) */
741         for (pp = &ptrace_state_map; (p = *pp) != NULL; pp = &p->psm_next) {
742                 if (p->psm_pid == pid) {
743                         *pp = p->psm_next;
744                         free(p);
745                         break;
746                 }
747         }
748 }

750 static int
751 setup_watchpoints(pid_t pid, uintptr_t *debugreg)
752 {
753         int dr7 = debugreg[7];
754         int lrw;
755         int fd;
756         size_t size = NULL;
757         prwatch_t prwatch[4];
758         int nwatch;
759         int i;
760         int wflags = NULL;
761         int error;
762         struct {
763                 long req;
764                 prwatch_t prwatch;
765         } ctl;

767         /* find all watched areas */
768         if ((fd = open_procfile(pid, O_RDONLY, "watch")) < 0)
769                 return (-ESRCH);
770         nwatch = read(fd, prwatch, sizeof (prwatch)) / sizeof (prwatch_t);
771         (void) close(fd);
772         if ((fd = open_procfile(pid, O_WRONLY, "ctl")) < 0)
773                 return (-ESRCH);
774         /* clear all watched areas */
775         for (i = 0; i < nwatch; i++) {
776                 ctl.req = PCWATCH;
777                 ctl.prwatch = prwatch[i];
778                 ctl.prwatch.pr_wflags = 0;
779                 if (write(fd, &ctl, sizeof (ctl)) != sizeof (ctl)) {
780                         error = -errno;
781                         (void) close(fd);
782                         return (error);
783                 }
784         }
785         /* establish all new watched areas */
786         for (i = 0; i < 4; i++) {
```

```
787                 if ((dr7 & (1 << (2 * i))) == 0)        /* enabled? */
788                         continue;
789                 lrw = (dr7 >> (16 + (4 * i))) & 0xf;
790                 switch (lrw >> 2) {     /* length */
791                 case 0: size = 1; break;
792                 case 1: size = 2; break;
793                 case 2: size = 8; break;
794                 case 3: size = 4; break;
795                 }
796                 switch (lrw & 0x3) {    /* mode */
797                 case 0: wflags = WA_EXEC; break;
798                 case 1: wflags = WA_WRITE; break;
799                 case 2: continue;
800                 case 3: wflags = WA_READ | WA_WRITE; break;
801                 }
802                 ctl.req = PCWATCH;
803                 ctl.prwatch.pr_vaddr = debugreg[i];
804                 ctl.prwatch.pr_size = size;
805                 ctl.prwatch.pr_wflags = wflags | WA_TRAPAFTER;
806                 if (write(fd, &ctl, sizeof (ctl)) != sizeof (ctl)) {
807                         error = -errno;
808                         (void) close(fd);
809                         return (error);
810                 }
811         }
812         (void) close(fd);
813         return (0);
814 }

816 /*
817  * Returns TRUE if the process is traced, FALSE otherwise.  This is only true
818  * if the process is currently stopped, and has been traced using PTRACE_TRACEME
819  * or PTRACE_ATTACH.
820  */
821 static int
822 is_traced(pid_t pid)
823 {
824         ptrace_monitor_map_t *p;
825         pstatus_t status;

827         if (get_status(pid, &status) != 0)
828                 return (0);

830         if ((status.pr_flags & PR_PTRACE) &&
831             (status.pr_ppid == getpid()) &&
832             (status.pr_lwp.pr_flags & PR_ISTOP))
833                 return (1);

835         (void) mutex_lock(&ptrace_map_mtx);
836         for (p = ptrace_monitor_map; p != NULL; p = p->pmm_next) {
837                 if (p->pmm_target == pid) {
838                         (void) mutex_unlock(&ptrace_map_mtx);
839                         return (1);
840                 }
841         }
842         (void) mutex_unlock(&ptrace_map_mtx);

844         return (0);
845 }

847 static int
848 ptrace_trace_common(int fd)
849 {
850         struct {
851                 long cmd;
852                 union {
```



```
853                         long flags;
854                         sigset_t signals;
855                         fltset_t faults;
856                 } arg;
857         } ctl;
858         size_t size;

860         ctl.cmd = PCSTRACE;
861         prfillset(&ctl.arg.signals);
862         size = sizeof (long) + sizeof (sigset_t);
863         if (write(fd, &ctl, size) != size)
864                 return (-1);

866         ctl.cmd = PCSFAULT;
867         premptyset(&ctl.arg.faults);
868         size = sizeof (long) + sizeof (fltset_t);
869         if (write(fd, &ctl, size) != size)
870                 return (-1);

872         ctl.cmd = PCUNSET;
873         ctl.arg.flags = PR_FORK;
874         size = sizeof (long) + sizeof (long);
875         if (write(fd, &ctl, size) != size)
876                 return (-1);

878         return (0);
879 }

881 /*
882  * Notify that parent that we wish to be traced.  This is the equivalent of:
883  *
884  *      1. Stop on all signals, and nothing else
885  *      2. Turn off inherit-on-fork flag
886  *      3. Set ptrace compatible flag
887  *
888  * If we are not the main thread, then the client is trying to request behavior
889  * by which one of its own thread is to be traced.  We don't support this mode
890  * of operation.
891  */
892 static int
893 ptrace_traceme(void)
894 {
895         int fd, ret;
896         int error;
897         long ctl[2];
898         pstatus_t status;
899         pid_t pid = getpid();

901         if (_lwp_self() != 1) {
902                 lx_unsupported(gettext(
903                     "thread %d calling PTRACE_TRACEME is unsupported"),
904                     _lwp_self());
905                 return (-ENOTSUP);
906         }

908         if ((ret = get_status(pid, &status)) != 0)
909                 return (ret);

911         /*
912          * Why would a process try to do this twice? I'm not sure, but there's
913          * a conformance test which wants this to fail just so.
914          */
915         if (status.pr_flags & PR_PTRACE)
916                 return (-EPERM);

918         if ((fd = open_procfile(pid, O_WRONLY, "ctl")) < 0)
```

```
919                 return (-errno);

921         ctl[0] = PCSET;
922         ctl[1] = PR_PTRACE;
923         error = 0;
924         if (write(fd, ctl, sizeof (ctl)) != sizeof (ctl) ||
925             ptrace_trace_common(fd) != 0)
926                 error = -errno;

928         (void) close(fd);
929         return (error);
930 }

932 /*
933  * Read a word of data from the given address.  Because this is a process-wide
934  * action, we don't need the lwpid.
935  */
936 static int
937 ptrace_peek(pid_t pid, uintptr_t addr, int *ret)
938 {
939         int fd, data;

941         if (!is_traced(pid))
942                 return (-ESRCH);

944         if ((fd = open_procfile(pid, O_RDONLY, "as")) < 0)
945                 return (-ESRCH);

947         if (pread(fd, &data, sizeof (data), addr) != sizeof (data)) {
948                 (void) close(fd);
949                 return (-EIO);
950         }

952         (void) close(fd);

954         if (uucopy(&data, ret, sizeof (data)) != 0)
955                 return (-errno);

957         return (0);
958 }

960 #define LX_USER_BOUND(m)        \
961 (offsetof(lx_user_t, m) + sizeof (((lx_user_t *)NULL)->m))

963 static int
964 ptrace_peek_user(pid_t pid, lwpid_t lwpid, uintptr_t off, int *ret)
965 {
966         int err, data;
967         uintptr_t *debugreg;
968         int dreg;

970         if (!is_traced(pid))
971                 return (-ESRCH);

973         /*
974          * The offset specified by the user is an offset into the Linux
975          * user structure (seriously). Rather than constructing a full
976          * user structure, we figure out which part of the user structure
977          * the offset is in, and fill in just that component.
978          */
979         if (off < LX_USER_BOUND(lxu_regs)) {
980                 lx_user_regs_t regs;

982                 if ((err = getregs(pid, lwpid, &regs)) != 0)
983                         return (err);
```



```
985                 data = *(int *)((uintptr_t)&regs + off -
986                     offsetof(lx_user_t, lxu_regs));

988         } else if (off < LX_USER_BOUND(lxu_fpvalid)) {
989                 lx_err(gettext("offset = %lu\n"), off);
990                 assert(0);
991         } else if (off < LX_USER_BOUND(lxu_i387)) {
992                 lx_user_fpregs_t regs;

994                 if ((err = getfpregs(pid, lwpid, &regs)) != 0)
995                         return (err);

997                 data = *(int *)((uintptr_t)&regs + off -
998                     offsetof(lx_user_t, lxu_i387));

1000        } else if (off < LX_USER_BOUND(lxu_tsize)) {
1001                lx_err(gettext("offset = %lu\n"), off);
1002                assert(0);
1003        } else if (off < LX_USER_BOUND(lxu_dsize)) {
1004                lx_err(gettext("offset = %lu\n"), off);
1005                assert(0);
1006        } else if (off < LX_USER_BOUND(lxu_ssize)) {
1007                lx_err(gettext("offset = %lu\n"), off);
1008                assert(0);
1009        } else if (off < LX_USER_BOUND(lxu_start_code)) {
1010                lx_err(gettext("offset = %lu\n"), off);
1011                assert(0);
1012        } else if (off < LX_USER_BOUND(lxu_start_stack)) {
1013                lx_err(gettext("offset = %lu\n"), off);
1014                assert(0);
1015        } else if (off < LX_USER_BOUND(lxu_signal)) {
1016                lx_err(gettext("offset = %lu\n"), off);
1017                assert(0);
1018        } else if (off < LX_USER_BOUND(lxu_reserved)) {
1019                lx_err(gettext("offset = %lu\n"), off);
1020                assert(0);
1021        } else if (off < LX_USER_BOUND(lxu_ar0)) {
1022                lx_err(gettext("offset = %lu\n"), off);
1023                assert(0);
1024        } else if (off < LX_USER_BOUND(lxu_fpstate)) {
1025                lx_err(gettext("offset = %lu\n"), off);
1026                assert(0);
1027        } else if (off < LX_USER_BOUND(lxu_magic)) {
1028                lx_err(gettext("offset = %lu\n"), off);
1029                assert(0);
1030        } else if (off < LX_USER_BOUND(lxu_comm)) {
1031                lx_err(gettext("offset = %lu\n"), off);
1032                assert(0);
1033        } else if (off < LX_USER_BOUND(lxu_debugreg)) {
1034                dreg = (off - offsetof(lx_user_t, lxu_debugreg)) / sizeof (int);
1035                if (dreg == 4)          /* aliased */
1036                        dreg = 6;
1037                else if (dreg == 5)     /* aliased */
1038                        dreg = 7;
1039                if ((debugreg = debug_registers(pid)) != NULL)
1040                        data = debugreg[dreg];
1041                else
1042                        data = 0;
1043        } else {
1044                lx_unsupported(gettext(
1045                    "unsupported ptrace %s user offset: 0x%x\n"), "peek", off);
1046                assert(0);
1047                return (-ENOTSUP);
1048        }

1050        if (uucopy(&data, ret, sizeof (data)) != 0)
```

```
1051                return (-errno);

1053        return (0);
1054 }

1056 /*
1057  * Write a word of data to the given address.  Because this is a process-wide
1058  * action, we don't need the lwpid.  Returns EINVAL if the address is not
1059  * word-aligned.
1060  */
1061 static int
1062 ptrace_poke(pid_t pid, uintptr_t addr, int data)
1063 {
1064         int fd;

1066         if (!is_traced(pid))
1067                 return (-ESRCH);

1069         if (addr & 0x3)
1070                 return (-EINVAL);

1072         if ((fd = open_procfile(pid, O_WRONLY, "as")) < 0)
1073                 return (-ESRCH);

1075         if (pwrite(fd, &data, sizeof (data), addr) != sizeof (data)) {
1076                 (void) close(fd);
1077                 return (-EIO);
1078         }

1080         (void) close(fd);
1081         return (0);
1082 }

1084 static int
1085 ptrace_poke_user(pid_t pid, lwpid_t lwpid, uintptr_t off, int data)
1086 {
1087         lx_user_regs_t regs;
1088         int err = 0;
1089         uintptr_t *debugreg;
1090         int dreg;

1092         if (!is_traced(pid))
1093                 return (-ESRCH);

1095         if (off & 0x3)
1096                 return (-EINVAL);

1098         if (off < offsetof(lx_user_t, lxu_regs) + sizeof (lx_user_regs_t)) {
1099                 if ((err = getregs(pid, lwpid, &regs)) != 0)
1100                         return (err);
1101                 *(int *)((uintptr_t)&regs + off -
1102                     offsetof(lx_user_t, lxu_regs)) = data;
1103                 return (setregs(pid, lwpid, &regs));
1104         }

1106         if (off >= offsetof(lx_user_t, lxu_debugreg) &&
1107             off < offsetof(lx_user_t, lxu_debugreg) + 8 * sizeof (int)) {
1108                 dreg = (off - offsetof(lx_user_t, lxu_debugreg)) / sizeof (int);
1109                 if (dreg == 4)          /* aliased */
1110                         dreg = 6;
1111                 else if (dreg == 5)     /* aliased */
1112                         dreg = 7;
1113                 if ((debugreg = debug_registers(pid)) != NULL) {
1114                         debugreg[dreg] = data;
1115                         if (dreg == 7)
1116                                 err = setup_watchpoints(pid, debugreg);
```



```
1117                 }
1118                 return (err);
1119         }

1121         lx_unsupported(gettext("unsupported ptrace %s user offset: 0x%x\n"),
1122             "poke", off);
1123         assert(0);
1124         return (-ENOTSUP);
1125 }

1127 static int
1128 ptrace_cont_common(int fd, int sig, int run, int step)
1129 {
1130         long ctl[1 + 1 + sizeof (siginfo_t) / sizeof (long) + 2];
1131         long *ctlp = ctl;
1132         size_t size;

1134         assert(0 <= sig && sig < LX_NSIG);
1135         assert(!step || run);

1137         /*
1138          * Clear the current signal.
1139          */
1140         *ctlp++ = PCCSIG;

1142         /*
1143          * Send a signal if one was specified.
1144          */
1145         if (sig != 0 && sig != LX_SIGSTOP) {
1146                 siginfo_t *infop;

1148                 *ctlp++ = PCSSIG;
1149                 infop = (siginfo_t *)ctlp;
1150                 bzero(infop, sizeof (siginfo_t));
1151                 infop->si_signo = ltos_signo[sig];

1153                 ctlp += sizeof (siginfo_t) / sizeof (long);
1154         }

1156         /*
1157          * If run is true, set the lwp running.
1158          */
1159         if (run) {
1160                 *ctlp++ = PCRUN;
1161                 *ctlp++ = step ? PRSTEP : 0;
1162         }

1164         size = (char *)ctlp - (char *)&ctl[0];
1165         assert(size <= sizeof (ctl));

1167         if (write(fd, ctl, size) != size) {
1168                 lx_debug("failed to continue %s", strerror(errno));
1169                 return (-EIO);
1170         }

1172         return (0);
1173 }

1175 static int
1176 ptrace_cont_monitor(ptrace_monitor_map_t *p)
1177 {
1178         long ctl[2];
1179         int fd;

1181         fd = open_procfile(p->pmm_monitor, O_WRONLY, "ctl");
1182         if (fd < 0) {
```

```
1183                 lx_debug("failed to open monitor ctl %d",
1184                     errno);
1185                 return (-EIO);
1186         }

1188         ctl[0] = PCRUN;
1189         ctl[1] = PRCSIG;
1190         if (write(fd, ctl, sizeof (ctl)) != sizeof (ctl)) {
1191                 (void) close(fd);
1192                 return (-EIO);
1193         }

1195         (void) close(fd);

1197         return (0);
1198 }

1200 static int
1201 ptrace_cont(pid_t lxpid, pid_t pid, lwpid_t lwpid, int sig, int step)
1202 {
1203         ptrace_monitor_map_t *p;
1204         uintptr_t *debugreg;
1205         int fd, ret;

1207         if (!is_traced(pid))
1208                 return (-ESRCH);

1210         if (sig < 0 || sig >= LX_NSIG)
1211                 return (-EINVAL);

1213         if ((fd = open_lwpfile(pid, lwpid, O_WRONLY, "lwpctl")) < 0)
1214                 return (-ESRCH);

1216         if ((ret = ptrace_cont_common(fd, sig, 1, step)) != 0) {
1217                 (void) close(fd);
1218                 return (ret);
1219         }

1221         (void) close(fd);

1223         /* kludge: use debugreg[4] to remember the single-step flag */
1224         if ((debugreg = debug_registers(pid)) != NULL)
1225                 debugreg[4] = step;

1227         /*
1228          * Check for a monitor and get it moving if we find it. If any of the
1229          * /proc operations fail, we're kind of sunk so just return an error.
1230          */
1231         (void) mutex_lock(&ptrace_map_mtx);
1232         for (p = ptrace_monitor_map; p != NULL; p = p->pmm_next) {
1233                 if (p->pmm_target == lxpid) {
1234                         if ((ret = ptrace_cont_monitor(p)) != 0)
1235                                 return (ret);
1236                         break;
1237                 }
1238         }
1239         (void) mutex_unlock(&ptrace_map_mtx);

1241         return (0);
1242 }

1244 /*
1245  * If a monitor exists for this traced process, dispose of it.
1246  * First turn off its ptrace flag so we won't be notified of its
1247  * impending demise.  We ignore errors for this step since they
1248  * indicate only that the monitor has been damaged due to pilot
```



```
1249  * error.  Then kill the monitor, and wait for it.  If the wait
1250  * succeeds we can dispose of the corpse, otherwise another thread's
1251  * wait call has collected it and we need to set a flag in the
1252  * structure so that if can be picked up in wait.
1253  */
1254 static void
1255 monitor_kill(pid_t lxpid, pid_t pid)
1256 {
1257         ptrace_monitor_map_t *p, **pp;
1258         pid_t mpid;
1259         int fd;
1260         long ctl[2];

1262         (void) mutex_lock(&ptrace_map_mtx);
1263         free_debug_registers(pid);
1264         for (pp = &ptrace_monitor_map; (p = *pp) != NULL; pp = &p->pmm_next) {
1265                 if (p->pmm_target == lxpid) {
1266                         mpid = p->pmm_monitor;
1267                         if ((fd = open_procfile(mpid, O_WRONLY, "ctl")) >= 0) {
1268                                 ctl[0] = PCUNSET;
1269                                 ctl[1] = PR_PTRACE;
1270                                 (void) write(fd, ctl, sizeof (ctl));
1271                                 (void) close(fd);
1272                         }

1274                         (void) kill(mpid, SIGKILL);

1276                         if (waitpid(mpid, NULL, 0) == mpid) {
1277                                 *pp = p->pmm_next;
1278                                 free(p);
1279                         } else {
1280                                 p->pmm_exiting = 1;
1281                         }

1283                         break;
1284                 }
1285         }
1286         (void) mutex_unlock(&ptrace_map_mtx);
1287 }

1289 static int
1290 ptrace_kill(pid_t lxpid, pid_t pid)
1291 {
1292         int ret;

1294         if (!is_traced(pid))
1295                 return (-ESRCH);

1297         ret = kill(pid, SIGKILL);

1299         /* kill off the monitor process, if any */
1300         monitor_kill(lxpid, pid);

1302         return (ret);
1303 }

1305 static int
1306 ptrace_step(pid_t lxpid, pid_t pid, lwpid_t lwpid, int sig)
1307 {
1308         return (ptrace_cont(lxpid, pid, lwpid, sig, 1));
1309 }

1311 static int
1312 ptrace_getregs(pid_t pid, lwpid_t lwpid, uintptr_t addr)
1313 {
1314         lx_user_regs_t regs;
```

```
1315         int ret;

1317         if (!is_traced(pid))
1318                 return (-ESRCH);

1320         if ((ret = getregs(pid, lwpid, &regs)) != 0)
1321                 return (ret);

1323         if (uucopy(&regs, (void *)addr, sizeof (regs)) != 0)
1324                 return (-errno);

1326         return (0);
1327 }

1329 static int
1330 ptrace_setregs(pid_t pid, lwpid_t lwpid, uintptr_t addr)
1331 {
1332         lx_user_regs_t regs;

1334         if (!is_traced(pid))
1335                 return (-ESRCH);

1337         if (uucopy((void *)addr, &regs, sizeof (regs)) != 0)
1338                 return (-errno);

1340         return (setregs(pid, lwpid, &regs));
1341 }

1343 static int
1344 ptrace_getfpregs(pid_t pid, lwpid_t lwpid, uintptr_t addr)
1345 {
1346         lx_user_fpregs_t regs;
1347         int ret;

1349         if (!is_traced(pid))
1350                 return (-ESRCH);

1352         if ((ret = getfpregs(pid, lwpid, &regs)) != 0)
1353                 return (ret);

1355         if (uucopy(&regs, (void *)addr, sizeof (regs)) != 0)
1356                 return (-errno);

1358         return (0);
1359 }

1361 static int
1362 ptrace_setfpregs(pid_t pid, lwpid_t lwpid, uintptr_t addr)
1363 {
1364         lx_user_fpregs_t regs;

1366         if (!is_traced(pid))
1367                 return (-ESRCH);

1369         if (uucopy((void *)addr, &regs, sizeof (regs)) != 0)
1370                 return (-errno);

1372         return (setfpregs(pid, lwpid, &regs));
1373 }

1375 static int
1376 ptrace_getfpxregs(pid_t pid, lwpid_t lwpid, uintptr_t addr)
1377 {
1378         lx_user_fpxregs_t regs;
1379         int ret;
```



```
1381         if (!is_traced(pid))
1382                 return (-ESRCH);

1384         if ((ret = getfpxregs(pid, lwpid, &regs)) != 0)
1385                 return (ret);

1387         if (uucopy(&regs, (void *)addr, sizeof (regs)) != 0)
1388                 return (-errno);

1390         return (0);
1391 }

1393 static int
1394 ptrace_setfpxregs(pid_t pid, lwpid_t lwpid, uintptr_t addr)
1395 {
1396         lx_user_fpxregs_t regs;

1398         if (!is_traced(pid))
1399                 return (-ESRCH);

1401         if (uucopy((void *)addr, &regs, sizeof (regs)) != 0)
1402                 return (-errno);

1404         return (setfpxregs(pid, lwpid, &regs));
1405 }

1407 static void __NORETURN
1408 ptrace_monitor(int fd)
1409 {
1410         struct {
1411                 long cmd;
1412                 union {
1413                         long flags;
1414                         sigset_t signals;
1415                         fltset_t faults;
1416                 } arg;
1417         } ctl;
1418         size_t size;
1419         int monfd;
1420         int rv;

1422         monfd = open_procfile(getpid(), O_WRONLY, "ctl");

1424         ctl.cmd = PCSTRACE;     /* trace only SIGTRAP */
1425         premptyset(&ctl.arg.signals);
1426         praddset(&ctl.arg.signals, SIGTRAP);
1427         size = sizeof (long) + sizeof (sigset_t);
1428         (void) write(monfd, &ctl, size);        /* can't fail */

1430         ctl.cmd = PCSFAULT;
1431         premptyset(&ctl.arg.faults);
1432         size = sizeof (long) + sizeof (fltset_t);
1433         (void) write(monfd, &ctl, size);        /* can't fail */

1435         ctl.cmd = PCUNSET;
1436         ctl.arg.flags = PR_FORK;
1437         size = sizeof (long) + sizeof (long);
1438         (void) write(monfd, &ctl, size);        /* can't fail */

1440         ctl.cmd = PCSET;        /* wait()able by the parent */
1441         ctl.arg.flags = PR_PTRACE;
1442         size = sizeof (long) + sizeof (long);
1443         (void) write(monfd, &ctl, size);        /* can't fail */

1445         (void) close(monfd);
```

```
1447         ctl.cmd = PCWSTOP;
1448         size = sizeof (long);

1450         for (;;) {
1451                 /*
1452                  * Wait for the traced process to stop.
1453                  */
1454                 if (write(fd, &ctl, size) != size) {
1455                         rv = (errno == ENOENT)? 0 : 1;
1456                         lx_debug("monitor failed to wait for LWP to stop: %s",
1457                             strerror(errno));
1458                         _exit(rv);
1459                 }

1461                 lx_debug("monitor caught traced LWP");

1463                 /*
1464                  * Pull the ptrace trigger by sending ourself a SIGTRAP. This
1465                  * will cause this, the monitor process, to stop which will
1466                  * cause the parent's waitid(2) call to return this process
1467                  * id. In lx_wait(), we remap the monitor process's pid and
1468                  * status to those of the traced LWP. When the parent process
1469                  * uses ptrace to resume the traced LWP, it will additionally
1470                  * restart this process.
1471                  */
1472                 (void) _lwp_kill(_lwp_self(), SIGTRAP);

1474                 lx_debug("monitor was resumed");
1475         }
1476 }

1478 static int
1479 ptrace_attach_common(int fd, pid_t lxpid, pid_t pid, lwpid_t lwpid, int run)
1480 {
1481         pid_t child;
1482         ptrace_monitor_map_t *p;
1483         sigset_t unblock;
1484         pstatus_t status;
1485         long ctl[1 + sizeof (sysset_t) / sizeof (long) + 2];
1486         long *ctlp = ctl;
1487         size_t size;
1488         sysset_t *sysp;
1489         int ret;

1491         /*
1492          * We're going to need this structure so better to fail now before its
1493          * too late to turn back.
1494          */
1495         if ((p = malloc(sizeof (ptrace_monitor_map_t))) == NULL)
1496                 return (-EIO);

1498         if ((ret = get_status(pid, &status)) != 0) {
1499                 free(p);
1500                 return (ret);
1501         }

1503         /*
1504          * If this process is already traced, bail.
1505          */
1506         if (status.pr_flags & PR_PTRACE) {
1507                 free(p);
1508                 return (-EPERM);
1509         }

1511         /*
1512          * Turn on the appropriate tracing flags. It's exceedingly unlikely
```



```
1513          * that this operation will fail; any failure would probably be due
1514          * to another /proc consumer mucking around.
1515          */
1516         if (ptrace_trace_common(fd) != 0) {
1517                 free(p);
1518                 return (-EIO);
1519         }

1521         /*
1522          * Native ptrace automatically catches processes when they exec so we
1523          * have to do that explicitly here.
1524          */
1525         *ctlp++ = PCSEXIT;
1526         sysp = (sysset_t *)ctlp;
1527         ctlp += sizeof (sysset_t) / sizeof (long);
1528         premptyset(sysp);
1529         praddset(sysp, SYS_execve);
1530         if (run) {
1531                 *ctlp++ = PCRUN;
1532                 *ctlp++ = 0;
1533         }

1535         size = (char *)ctlp - (char *)&ctl[0];

1537         if (write(fd, ctl, size) != size) {
1538                 free(p);
1539                 return (-EIO);
1540         }

1542         /*
1543          * Spawn the monitor proceses to notify this process of events of
1544          * interest in the traced process. We block signals here both so
1545          * we're not interrupted during this operation and so that the
1546          * monitor process doesn't accept signals.
1547          */
1548         (void) sigprocmask(SIG_BLOCK, &blockable_sigs, &unblock);
1549         if ((child = fork1()) == 0)
1550                 ptrace_monitor(fd);
1551         (void) sigprocmask(SIG_SETMASK, &unblock, NULL);

1553         if (child == -1) {
1554                 lx_debug("failed to fork monitor process\n");
1555                 free(p);
1556                 return (-EIO);
1557         }

1559         p->pmm_monitor = child;
1560         p->pmm_target = lxpid;
1561         p->pmm_pid = pid;
1562         p->pmm_lwpid = lwpid;
1563         p->pmm_exiting = 0;

1565         (void) mutex_lock(&ptrace_map_mtx);
1566         p->pmm_next = ptrace_monitor_map;
1567         ptrace_monitor_map = p;
1568         (void) mutex_unlock(&ptrace_map_mtx);

1570         return (0);
1571 }

1573 static int
1574 ptrace_attach(pid_t lxpid, pid_t pid, lwpid_t lwpid)
1575 {
1576         int fd, ret;
1577         long ctl;
```

```
1579         /*
1580          * Linux doesn't let you trace process 1 -- go figure.
1581          */
1582         if (lxpid == 1)
1583                 return (-EPERM);

1585         if ((fd = open_lwpfile(pid, lwpid, O_WRONLY | O_EXCL, "lwpctl")) < 0)
1586                 return (errno == EBUSY ? -EPERM : -ESRCH);

1588         ctl = PCSTOP;
1589         if (write(fd, &ctl, sizeof (ctl)) != sizeof (ctl)) {
1590                 lx_err(gettext("failed to stop %d/%d\n"), (int)pid, (int)lwpid);
1591                 assert(0);
1592         }

1594         ret = ptrace_attach_common(fd, lxpid, pid, lwpid, 0);

1596         (void) close(fd);

1598         return (ret);
1599 }

1601 static int
1602 ptrace_detach(pid_t lxpid, pid_t pid, lwpid_t lwpid, int sig)
1603 {
1604         long ctl[2];
1605         int fd, ret;

1607         if (!is_traced(pid))
1608                 return (-ESRCH);

1610         if (sig < 0 || sig >= LX_NSIG)
1611                 return (-EINVAL);

1613         if ((fd = open_lwpfile(pid, lwpid, O_WRONLY, "lwpctl")) < 0)
1614                 return (-ESRCH);

1616         /*
1617          * The /proc ptrace flag may not be set, but we clear it
1618          * unconditionally since doing so doesn't hurt anything.
1619          */
1620         ctl[0] = PCUNSET;
1621         ctl[1] = PR_PTRACE;
1622         if (write(fd, ctl, sizeof (ctl)) != sizeof (ctl)) {
1623                 (void) close(fd);
1624                 return (-EIO);
1625         }

1627         /*
1628          * Clear the brand-specific system call tracing flag to ensure that
1629          * the target doesn't stop unexpectedly some time in the future.
1630          */
1631         if ((ret = syscall(SYS_brand, B_PTRACE_SYSCALL, pid, lwpid, 0)) != 0) {
1632                 (void) close(fd);
1633                 return (-ret);
1634         }

1636         /* kill off the monitor process, if any */
1637         monitor_kill(lxpid, pid);

1639         /*
1640          * Turn on the run-on-last-close flag so that all tracing flags will be
1641          * cleared when we close the control file descriptor.
1642          */
1643         ctl[0] = PCSET;
1644         ctl[1] = PR_RLC;
```



```
1645         if (write(fd, ctl, sizeof (ctl)) != sizeof (ctl)) {
1646                 (void) close(fd);
1647                 return (-EIO);
1648         }

1650         /*
1651          * Clear the current signal (if any) and possibly send the traced
1652          * process a new signal.
1653          */
1654         ret = ptrace_cont_common(fd, sig, 0, 0);

1656         (void) close(fd);

1658         return (ret);
1659 }

1661 static int
1662 ptrace_syscall(pid_t lxpid, pid_t pid, lwpid_t lwpid, int sig)
1663 {
1664         int ret;

1666         if (!is_traced(pid))
1667                 return (-ESRCH);

1669         if ((ret = syscall(SYS_brand, B_PTRACE_SYSCALL, pid, lwpid, 1)) != 0)
1670                 return (-ret);

1672         return (ptrace_cont(lxpid, pid, lwpid, sig, 0));
1673 }

1675 int
1676 lx_ptrace(uintptr_t p1, uintptr_t p2, uintptr_t p3, uintptr_t p4)
1677 {
1678         pid_t pid, lxpid = (pid_t)p2;
1679         lwpid_t lwpid;

1681         if ((p1 != LX_PTRACE_TRACEME) &&
1682             (lx_lpid_to_spair(lxpid, &pid, &lwpid) < 0))
1683                 return (-ESRCH);

1685         switch (p1) {
1686         case LX_PTRACE_TRACEME:
1687                 return (ptrace_traceme());

1689         case LX_PTRACE_PEEKTEXT:
1690         case LX_PTRACE_PEEKDATA:
1691                 return (ptrace_peek(pid, p3, (int *)p4));

1693         case LX_PTRACE_PEEKUSER:
1694                 return (ptrace_peek_user(pid, lwpid, p3, (int *)p4));

1696         case LX_PTRACE_POKETEXT:
1697         case LX_PTRACE_POKEDATA:
1698                 return (ptrace_poke(pid, p3, (int)p4));

1700         case LX_PTRACE_POKEUSER:
1701                 return (ptrace_poke_user(pid, lwpid, p3, (int)p4));

1703         case LX_PTRACE_CONT:
1704                 return (ptrace_cont(lxpid, pid, lwpid, (int)p4, 0));

1706         case LX_PTRACE_KILL:
1707                 return (ptrace_kill(lxpid, pid));

1709         case LX_PTRACE_SINGLESTEP:
1710                 return (ptrace_step(lxpid, pid, lwpid, (int)p4));
```

```
1712         case LX_PTRACE_GETREGS:
1713                 return (ptrace_getregs(pid, lwpid, p4));

1715         case LX_PTRACE_SETREGS:
1716                 return (ptrace_setregs(pid, lwpid, p4));

1718         case LX_PTRACE_GETFPREGS:
1719                 return (ptrace_getfpregs(pid, lwpid, p4));

1721         case LX_PTRACE_SETFPREGS:
1722                 return (ptrace_setfpregs(pid, lwpid, p4));

1724         case LX_PTRACE_ATTACH:
1725                 return (ptrace_attach(lxpid, pid, lwpid));

1727         case LX_PTRACE_DETACH:
1728                 return (ptrace_detach(lxpid, pid, lwpid, (int)p4));

1730         case LX_PTRACE_GETFPXREGS:
1731                 return (ptrace_getfpxregs(pid, lwpid, p4));

1733         case LX_PTRACE_SETFPXREGS:
1734                 return (ptrace_setfpxregs(pid, lwpid, p4));

1736         case LX_PTRACE_SYSCALL:
1737                 return (ptrace_syscall(lxpid, pid, lwpid, (int)p4));

1739         default:
1740                 return (-EINVAL);
1741         }
1742 }

1744 void
1745 lx_ptrace_fork(void)
1746 {
1747         /*
1748          * Send a special signal (that has no Linux equivalent) to indicate
1749          * that we're in this particularly special case. The signal will be
1750          * ignored by this process, but noticed by /proc consumers tracing
1751          * this process.
1752          */
1753         (void) _lwp_kill(_lwp_self(), SIGWAITING);
1754 }

1756 static void
1757 ptrace_catch_fork(pid_t pid, int monitor)
1758 {
1759         long ctl[14 + 2 * sizeof (sysset_t) / sizeof (long)];
1760         long *ctlp;
1761         sysset_t *sysp;
1762         size_t size;
1763         pstatus_t ps;
1764         pid_t child;
1765         int fd, err;

1767         /*
1768          * If any of this fails, we're really sunk since the child
1769          * will be stuck in the middle of lx_ptrace_fork().
1770          * Fortunately it's practically assured to succeed unless
1771          * something is seriously wrong on the system.
1772          */
1773         if ((fd = open_procfile(pid, O_WRONLY, "ctl")) < 0) {
1774                 lx_debug("lx_catch_fork: failed to control %d",
1775                     (int)pid);
1776                 return;
```



```
1777         }

1779         /*
1780          * Turn off the /proc PR_PTRACE flag so the parent doesn't get
1781          * spurious wake ups while we're working our dark magic. Arrange to
1782          * catch the process when it exits from fork, and turn on the /proc
1783          * inherit-on-fork flag so we catcht the child as well. We then run
1784          * the process, wait for it to stop on the fork1(2) call and reset
1785          * the tracing flags to their original state.
1786          */
1787         ctlp = ctl;
1788         *ctlp++ = PCCSIG;
1789         if (!monitor) {
1790                 *ctlp++ = PCUNSET;
1791                 *ctlp++ = PR_PTRACE;
1792         }
1793         *ctlp++ = PCSET;
1794         *ctlp++ = PR_FORK;
1795         *ctlp++ = PCSEXIT;
1796         sysp = (sysset_t *)ctlp;
1797         ctlp += sizeof (sysset_t) / sizeof (long);
1798         premptyset(sysp);
1799         praddset(sysp, SYS_forksys);    /* fork1() is forksys(0, 0) */
1800         *ctlp++ = PCRUN;
1801         *ctlp++ = 0;
1802         *ctlp++ = PCWSTOP;
1803         if (!monitor) {
1804                 *ctlp++ = PCSET;
1805                 *ctlp++ = PR_PTRACE;
1806         }
1807         *ctlp++ = PCUNSET;
1808         *ctlp++ = PR_FORK;
1809         *ctlp++ = PCSEXIT;
1810         sysp = (sysset_t *)ctlp;
1811         ctlp += sizeof (sysset_t) / sizeof (long);
1812         premptyset(sysp);
1813         if (monitor)
1814                 praddset(sysp, SYS_execve);

1816         size = (char *)ctlp - (char *)&ctl[0];
1817         assert(size <= sizeof (ctl));

1819         if (write(fd, ctl, size) != size) {
1820                 (void) close(fd);
1821                 lx_debug("lx_catch_fork: failed to set %d running",
1822                     (int)pid);
1823                 return;
1824         }

1826         /*
1827          * Get the status so we can find the value returned from fork1() --
1828          * the child process's pid.
1829          */
1830         if (get_status(pid, &ps) != 0) {
1831                 (void) close(fd);
1832                 lx_debug("lx_catch_fork: failed to get status for %d",
1833                     (int)pid);
1834                 return;
1835         }

1837         child = (pid_t)ps.pr_lwp.pr_reg[R_R0];

1839         /*
1840          * We're done with the parent -- off you go.
1841          */
1842         ctl[0] = PCRUN;
```

```
1843         ctl[1] = 0;
1844         size = 2 * sizeof (long);

1846         if (write(fd, ctl, size) != size) {
1847                 (void) close(fd);
1848                 lx_debug("lx_catch_fork: failed to set %d running",
1849                     (int)pid);
1850                 return;
1851         }

1853         (void) close(fd);

1855         /*
1856          * If fork1(2) failed, we're done.
1857          */
1858         if (child < 0) {
1859                 lx_debug("lx_catch_fork: fork1 failed");
1860                 return;
1861         }

1863         /*
1864          * Now we need to screw with the child process.
1865          */
1866         if ((fd = open_lwpfile(child, 1, O_WRONLY, "lwpctl")) < 0) {
1867                 lx_debug("lx_catch_fork: failed to control %d",
1868                     (int)child);
1869                 return;
1870         }

1872         ctlp = ctl;
1873         *ctlp++ = PCUNSET;
1874         *ctlp++ = PR_FORK;
1875         *ctlp++ = PCSEXIT;
1876         sysp = (sysset_t *)ctlp;
1877         ctlp += sizeof (sysset_t) / sizeof (long);
1878         premptyset(sysp);
1879         size = (char *)ctlp - (char *)&ctl[0];

1881         if (write(fd, ctl, size) != size) {
1882                 (void) close(fd);
1883                 lx_debug("lx_catch_fork: failed to clear trace flags for  %d",
1884                     (int)child);
1885                 return;
1886         }

1888         /*
1889          * Now treat the child as though we had attached to it explicitly.
1890          */
1891         err = ptrace_attach_common(fd, child, child, 1, 1);
1892         assert(err == 0);

1894         (void) close(fd);
1895 }

1897 static void
1898 set_dr6(pid_t pid, siginfo_t *infop)
1899 {
1900         uintptr_t *debugreg;
1901         uintptr_t addr;
1902         uintptr_t base;
1903         size_t size = NULL;
1904         int dr7;
1905         int lrw;
1906         int i;

1908         if ((debugreg = debug_registers(pid)) == NULL)
```



```
1909                 return;

1911         debugreg[6] = 0xffff0ff0;       /* read as ones */
1912         switch (infop->si_code) {
1913         case TRAP_TRACE:
1914                 debugreg[6] |= 0x4000;  /* single-step */
1915                 break;
1916         case TRAP_RWATCH:
1917         case TRAP_WWATCH:
1918         case TRAP_XWATCH:
1919                 dr7 = debugreg[7];
1920                 addr = (uintptr_t)infop->si_addr;
1921                 for (i = 0; i < 4; i++) {
1922                         if ((dr7 & (1 << (2 * i))) == 0)        /* enabled? */
1923                                 continue;
1924                         lrw = (dr7 >> (16 + (4 * i))) & 0xf;
1925                         switch (lrw >> 2) {     /* length */
1926                         case 0: size = 1; break;
1927                         case 1: size = 2; break;
1928                         case 2: size = 8; break;
1929                         case 3: size = 4; break;
1930                         }
1931                         base = debugreg[i];
1932                         if (addr >= base && addr < base + size)
1933                                 debugreg[6] |= (1 << i);
1934                 }
1935                 /*
1936                  * Were we also attempting a single-step?
1937                  * (kludge: we use debugreg[4] for this flag.)
1938                  */
1939                 if (debugreg[4])
1940                         debugreg[6] |= 0x4000;
1941                 break;
1942         default:
1943                 break;
1944         }
1945 }

1947 /*
1948  * This is called from the emulation of the wait4 and waitpid system call to
1949  * take into account the monitor processes which we spawn to observe other
1950  * processes from ptrace_attach().
1951  */
1952 int
1953 lx_ptrace_wait(siginfo_t *infop)
1954 {
1955         ptrace_monitor_map_t *p, **pp;
1956         pid_t lxpid, pid = infop->si_pid;
1957         lwpid_t lwpid;
1958         int fd;
1959         pstatus_t status;

1961         /*
1962          * If the process observed by waitid(2) corresponds to the monitor
1963          * process for a traced thread, we need to rewhack the siginfo_t to
1964          * look like it came from the traced thread with the flags set
1965          * according to the current state.
1966          */
1967         (void) mutex_lock(&ptrace_map_mtx);
1968         for (pp = &ptrace_monitor_map; (p = *pp) != NULL; pp = &p->pmm_next) {
1969                 if (p->pmm_monitor == pid) {
1970                         assert(infop->si_code == CLD_EXITED ||
1971                             infop->si_code == CLD_KILLED ||
1972                             infop->si_code == CLD_DUMPED ||
1973                             infop->si_code == CLD_TRAPPED);
1974                         goto found;
```

```
1975                }
1976        }
1977        (void) mutex_unlock(&ptrace_map_mtx);

1979        /*
1980         * If the traced process got a SIGWAITING, we must be in the middle
1981         * of a clone(2) with CLONE_PTRACE set.
1982         */
1983        if (infop->si_code == CLD_TRAPPED && infop->si_status == SIGWAITING) {
1984                ptrace_catch_fork(pid, 0);
1985                return (-1);
1986        }

1988        if (get_status(pid, &status) == 0 &&
1989            (status.pr_lwp.pr_flags & PR_STOPPED) &&
1990            status.pr_lwp.pr_why == PR_SIGNALLED &&
1991            status.pr_lwp.pr_info.si_signo == SIGTRAP)
1992                set_dr6(pid, &status.pr_lwp.pr_info);

1994        return (0);

1996 found:
1997        /*
1998         * If the monitor is in the exiting state, ignore the event and free
1999         * the monitor structure if the monitor has exited. By returning -1 we
2000         * indicate to the caller that this was a spurious return from
2001         * waitid(2) and that it should ignore the result and try again.
2002         */
2003        if (p->pmm_exiting) {
2004                if (infop->si_code == CLD_EXITED ||
2005                    infop->si_code == CLD_KILLED ||
2006                    infop->si_code == CLD_DUMPED) {
2007                        *pp = p->pmm_next;
2008                        (void) mutex_unlock(&ptrace_map_mtx);
2009                        free(p);
2010                }
2011                return (-1);
2012        }

2014        lxpid = p->pmm_target;
2015        pid = p->pmm_pid;
2016        lwpid = p->pmm_lwpid;
2017        (void) mutex_unlock(&ptrace_map_mtx);

2019        /*
2020         * If we can't find the traced process, kill off its monitor.
2021         */
2022        if ((fd = open_lwpfile(pid, lwpid, O_RDONLY, "lwpstatus")) < 0) {
2023                assert(errno == ENOENT);
2024                monitor_kill(lxpid, pid);
2025                infop->si_code = CLD_EXITED;
2026                infop->si_status = 0;
2027                infop->si_pid = lxpid;
2028                return (0);
2029        }

2031        if (read(fd, &status.pr_lwp, sizeof (status.pr_lwp)) !=
2032            sizeof (status.pr_lwp)) {
2033                lx_err(gettext("read lwpstatus failed %d %s"),
2034                    fd, strerror(errno));
2035                assert(0);
2036        }

2038        (void) close(fd);

2040        /*
```



```
2041         * If the traced process isn't stopped, this is a truly spurious
2042         * event probably caused by another /proc consumer tracing the
2043         * monitor.
2044         */
2045        if (!(status.pr_lwp.pr_flags & PR_STOPPED)) {
2046                (void) ptrace_cont_monitor(p);
2047                return (-1);
2048        }

2050        switch (status.pr_lwp.pr_why) {
2051        case PR_SIGNALLED:
2052                /*
2053                 * If the traced process got a SIGWAITING, we must be in the
2054                 * middle of a clone(2) with CLONE_PTRACE set.
2055                 */
2056                if (status.pr_lwp.pr_what == SIGWAITING) {
2057                        ptrace_catch_fork(lxpid, 1);
2058                        (void) ptrace_cont_monitor(p);
2059                        return (-1);
2060                }
2061                infop->si_code = CLD_TRAPPED;
2062                infop->si_status = status.pr_lwp.pr_what;
2063                if (status.pr_lwp.pr_info.si_signo == SIGTRAP)
2064                        set_dr6(pid, &status.pr_lwp.pr_info);
2065                break;

2067        case PR_REQUESTED:
2068                /*
2069                 * Make it look like the traced process stopped on an
2070                 * event of interest.
2071                 */
2072                infop->si_code = CLD_TRAPPED;
2073                infop->si_status = SIGTRAP;
2074                break;

2076        case PR_JOBCONTROL:
2077                /*
2078                 * Ignore this as it was probably caused by another /proc
2079                 * consumer tracing the monitor.
2080                 */
2081                (void) ptrace_cont_monitor(p);
2082                return (-1);

2084        case PR_SYSEXIT:
2085                /*
2086                 * Processes traced via a monitor (rather than using the
2087                 * native Solaris ptrace support) explicitly trace returns
2088                 * from exec system calls since it's an implicit ptrace
2089                 * trace point. Accordingly we need to present a process
2090                 * in that state as though it had reached the ptrace trace
2091                 * point.
2092                 */
2093                if (status.pr_lwp.pr_what == SYS_execve) {
2094                        infop->si_code = CLD_TRAPPED;
2095                        infop->si_status = SIGTRAP;
2096                        break;
2097                }

2099                /*FALLTHROUGH*/

2101        case PR_SYSENTRY:
2102        case PR_FAULTED:
2103        case PR_SUSPENDED:
2104        default:
2105                lx_err(gettext("didn't expect %d (%d %d)"),
2106                    status.pr_lwp.pr_why,
```

```
2107                     status.pr_lwp.pr_what, status.pr_lwp.pr_flags);
2108                 assert(0);
2109         }

2111         infop->si_pid = lxpid;

2113         return (0);
2114 }
2115 #endif /* ! codereview */
```

```
**********************************************************
    5835 Tue Jan 14 16:17:04 2014
new/usr/src/lib/brand/lx/lx_brand/common/rlimit.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <errno.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/systm.h>
#include <sys/resource.h>
#include <sys/sysconfig.h>
#include <sys/lx_types.h>
#include <sys/lx_misc.h>

#define LX_RLIMIT_RSS           5
#define LX_RLIMIT_NPROC         6
#define LX_RLIMIT_MEMLOCK       8
#define LX_RLIMIT_LOCKS         10
#define LX_RLIMIT_NLIMITS       11

/*
 * Linux supports many of the same resources that we do, but the numbering
 * is slightly different.  This table is used to translate Linux resource
 * limit keys into their Solaris equivalents.
 */
static int ltos_resource[LX_RLIMIT_NLIMITS] = {
        RLIMIT_CPU,
        RLIMIT_FSIZE,
        RLIMIT_DATA,
        RLIMIT_STACK,
        RLIMIT_CORE,
        -1,                     /* RSS */
        -1,                     /* NPROC */
        RLIMIT_NOFILE,
        -1,                     /* MEMLOCK */
        RLIMIT_AS,
        -1                      /* LOCKS */
};
```



```
#define NLIMITS (sizeof (ltos_resource) / sizeof (int))

/*
 * Magic values Linux uses to indicate infinity
 */
#define LX_RLIM_INFINITY_O      (0x7fffffffUL)
#define LX_RLIM_INFINITY_N      (0xffffffffUL)

/*
 * Array to store the rlimits that we track but do not enforce.
 */
static struct rlimit fake_limits[NLIMITS] = {
        0, 0,
        0, 0,
        0, 0,
        0, 0,
        0, 0,
        RLIM_INFINITY, RLIM_INFINITY,   /* LX_RLIM_RSS */
        RLIM_INFINITY, RLIM_INFINITY,   /* LX_RLIM_NPROC */
        0, 0,
        RLIM_INFINITY, RLIM_INFINITY,   /* LX_RLIM_MEMLOCK */
        0, 0,
        RLIM_INFINITY, RLIM_INFINITY    /* LX_RLIM_LOCKS */
};

static int
lx_getrlimit_common(int resource, struct rlimit *rlp, int inf)
{
        int rv;
        int sresource;
        struct rlimit rl;

        if (resource < 0 || resource >= LX_RLIMIT_NLIMITS)
                return (-EINVAL);

        sresource = ltos_resource[resource];

        if (sresource == -1) {
                switch (resource) {
                case LX_RLIMIT_MEMLOCK:
                case LX_RLIMIT_RSS:
                case LX_RLIMIT_LOCKS:
                case LX_RLIMIT_NPROC:
                        rl.rlim_max = fake_limits[resource].rlim_max;
                        rl.rlim_cur = fake_limits[resource].rlim_cur;
                        if (rl.rlim_cur == RLIM_INFINITY)
                                rl.rlim_cur = inf;
                        if (rl.rlim_max == RLIM_INFINITY)
                                rl.rlim_max = inf;
                        if ((uucopy(&rl, rlp, sizeof (rl))) != 0)
                                return (-errno);
                        return (0);
                default:
                        lx_unsupported("Unsupported resource type %d\n",
                            resource);
                        return (-ENOTSUP);
                }
        } else {
                rv = getrlimit(sresource, rlp);
        }

        if (rv < 0)
                return (-errno);

        if (rlp->rlim_cur == RLIM_INFINITY)
                rlp->rlim_cur = inf;
```

```
129         if (rlp->rlim_max == RLIM_INFINITY)
130                 rlp->rlim_max = inf;
131
132         return (0);
133 }
134
135 /*
136  * This is the 'new' getrlimit, variously called getrlimit or ugetrlimit
137  * in Linux headers and code.  The only difference between this and the old
138  * getrlimit (variously called getrlimit or old_getrlimit) is the value of
139  * RLIM_INFINITY, which is smaller for the older version.  Modern code will
140  * use this version by default.
141  */
142 int
143 lx_getrlimit(uintptr_t p1, uintptr_t p2)
144 {
145         int resource = (int)p1;
146         struct rlimit *rlp = (struct rlimit *)p2;
147
148         return (lx_getrlimit_common(resource, rlp, LX_RLIM_INFINITY_N));
149 }
150
151 /*
152  * This is the 'old' getrlimit, variously called getrlimit or old_getrlimit
153  * in Linux headers and code.  The only difference between this and the new
154  * getrlimit (variously called getrlimit or ugetrlimit) is the value of
155  * RLIM_INFINITY, which is smaller for the older version.
156  */
157 int
158 lx_oldgetrlimit(uintptr_t p1, uintptr_t p2)
159 {
160         int resource = (int)p1;
161         struct rlimit *rlp = (struct rlimit *)p2;
162
163         return (lx_getrlimit_common(resource, rlp, LX_RLIM_INFINITY_O));
164 }
165
166 int
167 lx_setrlimit(uintptr_t p1, uintptr_t p2)
168 {
169         int resource = (int)p1;
170         struct rlimit *rlp = (struct rlimit *)p2;
171         struct rlimit rl;
172         int rv, sresource;
173
174         if (resource < 0 || resource >= LX_RLIMIT_NLIMITS)
175                 return (-EINVAL);
176
177         sresource = ltos_resource[resource];
178
179         if (sresource == -1) {
180                 if (uucopy((void *)p2, &rl, sizeof (rl)) != 0)
181                         return (-errno);
182
183                 switch (resource) {
184                 case LX_RLIMIT_MEMLOCK:
185                 case LX_RLIMIT_RSS:
186                 case LX_RLIMIT_LOCKS:
187                 case LX_RLIMIT_NPROC:
188                         if (rl.rlim_max != LX_RLIM_INFINITY_N &&
189                             (rl.rlim_cur == LX_RLIM_INFINITY_N ||
190                             rl.rlim_cur > rl.rlim_max))
191                                 return (-EINVAL);
192                         if (rl.rlim_max == LX_RLIM_INFINITY_N)
193                                 fake_limits[resource].rlim_max = RLIM_INFINITY;
```



```
194                         else
195                                 fake_limits[resource].rlim_max = rl.rlim_max;
196                         if (rl.rlim_cur == LX_RLIM_INFINITY_N)
197                                 fake_limits[resource].rlim_cur = RLIM_INFINITY;
198                         else
199                                 fake_limits[resource].rlim_cur = rl.rlim_cur;
200                         return (0);
201                 }
202
203                 lx_unsupported("Unsupported resource type %d\n", resource);
204                 return (-ENOTSUP);
205         }
206
207         rv = setrlimit(sresource, rlp);
208
209         return (rv < 0 ? -errno : 0);
210 }
211
212 /*
213  * We lucked out here.  Linux and Solaris have exactly the same
214  * rusage structures.
215  */
216 int
217 lx_getrusage(uintptr_t p1, uintptr_t p2)
218 {
219         int who = (int)p1;
220         struct rusage *rup = (struct rusage *)p2;
221         int rv, swho;
222
223         if (who == LX_RUSAGE_SELF)
224                 swho = _RUSAGESYS_GETRUSAGE;
225         else if (who == LX_RUSAGE_CHILDREN)
226                 swho = _RUSAGESYS_GETRUSAGE_CHLD;
227         else
228                 return (-EINVAL);
229
230         rv = getrusage(swho, rup);
231
232         return (rv < 0 ? -errno : 0);
233 }
234 #endif /* ! codereview */
```

```
**********************************************************
   14761 Tue Jan 14 16:17:04 2014
new/usr/src/lib/brand/lx/lx_brand/common/sched.c
LX zone support should now build and packages of relevance produced.
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/types.h>
#include <sys/cred_impl.h>
#include <sys/ucred.h>
#include <ucred.h>
#include <stdlib.h>
#include <signal.h>
#include <errno.h>
#include <sched.h>
#include <strings.h>
#include <pthread.h>
#include <time.h>
#include <thread.h>
#include <alloca.h>
#include <unistd.h>
#include <sys/syscall.h>
#include <sys/lx_syscall.h>
#include <sys/lx_debug.h>
#include <sys/lx_brand.h>
#include <sys/lx_misc.h>
#include <sys/lx_sched.h>

/* Linux only has three valid policies, SCHED_FIFO, SCHED_RR and SCHED_OTHER */
static int
validate_policy(int policy)
{
        switch (policy) {
                case LX_SCHED_FIFO:
                        return (SCHED_FIFO);

                case LX_SCHED_RR:
                        return (SCHED_RR);
```



```
                case LX_SCHED_OTHER:
                        return (SCHED_OTHER);

                default:
                        lx_debug("validate_policy: illegal policy: %d", policy);
                        return (-EINVAL);
        }
}

/*
 * Check to see if we have the permissions to set scheduler parameters and
 * policy, based on Linux' demand that such commands fail with errno set to
 * EPERM if the current euid is not the euid or ruid of the process in
 * question.
 */
static int
check_schedperms(pid_t pid)
{
        size_t sz;
        ucred_t *cr;
        uid_t euid;

        euid = geteuid();

        if (pid == getpid()) {
                /*
                 * If we're the process to be checked, simply check the euid
                 * against our ruid.
                 */
                if (euid != getuid())
                        return (-EPERM);

                return (0);
        }

        /*
         * We allocate a ucred_t ourselves rather than call ucred_get(3C)
         * because ucred_get() calls malloc(3C), which the brand library cannot
         * use.  Because we allocate the space with SAFE_ALLOCA(), there's
         * no need to free it when we're done.
         */
        sz = ucred_size();
        cr = (ucred_t *)SAFE_ALLOCA(sz);

        if (cr == NULL)
                return (-ENOMEM);

        /*
         * If we can't access the process' credentials, fail with errno EPERM
         * as the call would not have succeeded anyway.
         */
        if (syscall(SYS_ucredsys, UCREDSYS_UCREDGET, pid, cr) != 0)
                return ((errno == EACCES) ? -EPERM : -errno);

        if ((euid != ucred_geteuid(cr)) && (euid != ucred_getruid(cr)))
                return (-EPERM);

        return (0);
}

static int
ltos_sparam(int policy, struct lx_sched_param *lsp, struct sched_param *sp)
{
        struct lx_sched_param ls;
        int smin = sched_get_priority_min(policy);
        int smax = sched_get_priority_max(policy);
```

```
128         if (uucopy(lsp, &ls, sizeof (struct lx_sched_param)) != 0)
129                 return (-errno);

131         bzero(sp, sizeof (struct sched_param));

133         /*
134          * Linux has a fixed priority range, 0 - 99, which we need to convert to
135          * Solaris's dynamic range. Linux considers lower numbers to be
136          * higher priority, so we'll invert the priority within Solaris's range.
137          *
138          * The formula to convert between ranges is:
139          *
140          *      L * (smax - smin)
141          * S =  -----------------  + smin
142          *        (lmax - lmin)
143          *
144          * where S is the Solaris equivalent of the linux priority L.
145          *
146          * To invert the priority, we use:
147          * S' = smax - S + smin
148          *
149          * Together, these two formulas become:
150          *
151          *              L * (smax - smin)
152          *   S = smax - -----------------  + 2smin
153          *                     99
154          */
155         sp->sched_priority = smax -
156             ((ls.lx_sched_prio * (smax - smin)) / LX_PRI_MAX) + 2*smin;

158         lx_debug("ltos_sparam: linux prio %d = Solaris prio %d "
159             "(Solaris range %d,%d)\n", ls.lx_sched_prio, sp->sched_priority,
160             smin, smax);

162         return (0);
163 }

165 static int
166 stol_sparam(int policy, struct sched_param *sp, struct lx_sched_param *lsp)
167 {
168         struct lx_sched_param ls;
169         int smin = sched_get_priority_min(policy);
170         int smax = sched_get_priority_max(policy);

172         if (policy == SCHED_OTHER) {
173                 /*
174                  * In Linux, the only valid SCHED_OTHER scheduler priority is 0
175                  */
176                 ls.lx_sched_prio = 0;
177         } else {
178                 /*
179                  * Convert Solaris's dynamic, inverted priority range to the
180                  * fixed Linux range of 1 - 99.
181                  *
182                  * The formula is (see above):
183                  *
184                  *      (smax - s + 2smin) * 99
185                  *  l = -----------------------
186                  *              smax - smin
187                  */
188                 ls.lx_sched_prio = ((smax - sp->sched_priority + 2*smin) *
189                     LX_PRI_MAX) / (smax - smin);
190         }

192         lx_debug("stol_sparam: Solaris prio %d = linux prio %d "
```



```
193             "(Solaris range %d,%d)\n", sp->sched_priority, ls.lx_sched_prio,
194             smin, smax);

196         return ((uucopy(&ls, lsp, sizeof (struct lx_sched_param)) != 0)
197             ? -errno : 0);
198 }

200 #define BITINDEX(ind)   (ind / (sizeof (ulong_t) * 8))
201 #define BITSHIFT(ind)   (1 << (ind % (sizeof (ulong_t) * 8)))

203 /* ARGSUSED */
204 int
205 lx_sched_getaffinity(uintptr_t pid, uintptr_t len, uintptr_t maskp)
206 {
207         int     sz;
208         ulong_t *lmask, *zmask;
209         int     i;

211         sz = syscall(SYS_brand, B_GET_AFFINITY_MASK, pid, len, maskp);
212         if (sz == -1)
213                 return (-errno);

215         /*
216          * If the target LWP hasn't ever had an affinity mask set, the kernel
217          * will return a mask of all 0's. If that is the case we must build a
218          * default mask that has all valid bits turned on.
219          */
220         lmask = SAFE_ALLOCA(sz);
221         zmask = SAFE_ALLOCA(sz);
222         if (lmask == NULL || zmask == NULL)
223                 return (-ENOMEM);

225         bzero(zmask, sz);

227         if (uucopy((void *)maskp, lmask, sz) != 0)
228                 return (-EFAULT);

230         if (bcmp(lmask, zmask, sz) != 0)
231                 return (sz);

233         for (i = 0; i < sz * 8; i++) {
234                 if (p_online(i, P_STATUS) != -1) {
235                         lmask[BITINDEX(i)] |= BITSHIFT(i);
236                 }
237         }

239         if (uucopy(lmask, (void *)maskp, sz) != 0)
240                 return (-EFAULT);

242         return (sz);
243 }

245 /* ARGSUSED */
246 int
247 lx_sched_setaffinity(uintptr_t pid, uintptr_t len, uintptr_t maskp)
248 {
249         int             ret;
250         int             sz;
251         int             i;
252         int             found;
253         ulong_t         *lmask;
254         pid_t           s_pid;
255         lwpid_t         s_tid;
256         processorid_t   cpuid = NULL;

258         if ((pid_t)pid < 0)
```

```
259                 return (-EINVAL);

261         if (lx_lpid_to_spair(pid, &s_pid, &s_tid) < 0)
262                 return (-ESRCH);

264         /*
265          * We only support setting affinity masks for threads in
266          * the calling process.
267          */
268         if (s_pid != getpid())
269                 return (-EPERM);

271         /*
272          * First, get the minimum bitmask size from the kernel.
273          */
274         sz = syscall(SYS_brand, B_GET_AFFINITY_MASK, 0, 0, 0);
275         if (sz == -1)
276                 return (-errno);

278         lmask = SAFE_ALLOCA(sz);
279         if (lmask == NULL)
280                 return (-ENOMEM);

282         if (uucopy((void *)maskp, lmask, sz) != 0)
283                 return (-EFAULT);

285         /*
286          * Make sure the mask contains at least one processor that is
287          * physically on the system. Reduce the user's mask to the set of
288          * physically present CPUs. Keep track of how many valid
289          * bits are set in the user's mask.
290          */

292         for (found = 0, i = 0; i < sz * 8; i++) {
293                 if (p_online(i, P_STATUS) == -1) {
294                         /*
295                          * This CPU doesn't exist, so clear this bit from
296                          * the user's mask.
297                          */
298                         lmask[BITINDEX(i)] &= ~BITSHIFT(i);
299                         continue;
300                 }

302                 if ((lmask[BITINDEX(i)] & BITSHIFT(i)) == BITSHIFT(i)) {
303                         found++;
304                         cpuid = i;
305                 }
306         }

308         if (found == 0) {
309                 lx_debug("\tlx_sched_setaffinity: mask has no present CPUs\n");
310                 return (-EINVAL);
311         }

313         /*
314          * If only one bit is set, bind the thread to that procesor;
315          * otherwise, clear the binding.
316          */
317         if (found == 1) {
318                 lx_debug("\tlx_sched_setaffinity: binding thread %d to cpu%d\n",
319                     s_tid, cpuid);
320                 if (processor_bind(P_LWPID, s_tid, cpuid, NULL) != 0)
321                         /*
322                          * It could be that the requested processor is offline,
323                          * so we'll just abandon our good-natured attempt to
324                          * bind to it.
```



```
325                          */
326                         lx_debug("couldn't bind LWP %d to cpu %d: %s\n", s_tid,
327                             cpuid, strerror(errno));
328         } else {
329                 lx_debug("\tlx_sched_setaffinity: clearing thr %d binding\n",
330                     s_tid);
331                 if (processor_bind(P_LWPID, s_tid, PBIND_NONE, NULL) != 0) {
332                         lx_debug("couldn't clear CPU binding for LWP %d: %s\n",
333                             s_tid, strerror(errno));
334                 }
335         }

337         /*
338          * Finally, ask the kernel to make a note of our current (though fairly
339          * meaningless) affinity mask.
340          */
341         ret = syscall(SYS_brand, B_SET_AFFINITY_MASK, pid, sz, lmask);

343         return ((ret == 0) ? 0 : -errno);
344 }

346 int
347 lx_sched_getparam(uintptr_t pid, uintptr_t param)
348 {
349         int     policy, ret;
350         pid_t   s_pid;
351         lwpid_t s_tid;

353         struct sched_param sp;

355         if (((pid_t)pid < 0) || (param == NULL))
356                 return (-EINVAL);

358         if (lx_lpid_to_spair((pid_t)pid, &s_pid, &s_tid) < 0)
359                 return (-ESRCH);

361         /*
362          * If we're attempting to get information on our own process, we can
363          * get data on a per-thread basis; if not, punt and use the specified
364          * pid.
365          */
366         if (s_pid == getpid()) {
367                 if ((ret = pthread_getschedparam(s_tid, &policy, &sp)) != 0)
368                         return (-ret);
369         } else {
370                 if (sched_getparam(s_pid, &sp) == -1)
371                         return (-errno);

373                 if ((policy = sched_getscheduler(s_pid)) < 0)
374                         return (-errno);
375         }

377         return (stol_sparam(policy, &sp, (struct lx_sched_param *)param));
378 }

380 int
381 lx_sched_setparam(uintptr_t pid, uintptr_t param)
382 {
383         int     err, policy;
384         pid_t   s_pid;
385         lwpid_t s_tid;
386         struct lx_sched_param lp;
387         struct sched_param sp;

389         if (((pid_t)pid < 0) || (param == NULL))
390                 return (-EINVAL);
```

```
392         if (lx_lpid_to_spair((pid_t)pid, &s_pid, &s_tid) < 0)
393                 return (-ESRCH);

395         if (s_pid == getpid()) {
396                 struct sched_param dummy;

398                 if ((err = pthread_getschedparam(s_tid, &policy, &dummy)) != 0)
399                         return (-err);
400         } else
401                 if ((policy = sched_getscheduler(s_pid)) < 0)
402                         return (-errno);

404         lx_debug("sched_setparam(): current policy %d", policy);

406         if (uucopy((void *)param, &lp, sizeof (lp)) != 0)
407                 return (-errno);

409         /*
410          * In Linux, the only valid SCHED_OTHER scheduler priority is 0
411          */
412         if ((policy == SCHED_OTHER) && (lp.lx_sched_prio != 0))
413                 return (-EINVAL);

415         if ((err = ltos_sparam(policy, (struct lx_sched_param *)&lp,
416             &sp)) != 0)
417                 return (err);

419         /*
420          * Check if we're allowed to change the scheduler for the process.
421          *
422          * If we're operating on a thread, we can't just call
423          * pthread_setschedparam() because as all threads reside within a
424          * single Solaris process, Solaris will allow the modification
425          *
426          * If we're operating on a process, we can't just call sched_setparam()
427          * because Solaris will allow the call to succeed if the scheduler
428          * parameters do not differ from those being installed, but Linux wants
429          * the call to fail.
430          */
431         if ((err = check_schedperms(s_pid)) != 0)
432                 return (err);

434         if (s_pid == getpid())
435                 return (((err = pthread_setschedparam(s_tid, policy, &sp)) != 0)
436                     ? -err : 0);

438         return ((sched_setparam(s_pid, &sp) == -1) ? -errno : 0);
439 }

441 int
442 lx_sched_rr_get_interval(uintptr_t pid, uintptr_t timespec)
443 {
444         struct timespec ts;
445         pid_t   s_pid;

447         if ((pid_t)pid < 0)
448                 return (-EINVAL);

450         if (lx_lpid_to_spid((pid_t)pid, &s_pid) < 0)
451                 return (-ESRCH);

453         if (uucopy((struct timespec *)timespec, &ts,
454             sizeof (struct timespec)) != 0)
455                 return (-errno);
```



```
457         return ((sched_rr_get_interval(s_pid, &ts) == -1) ? -errno : 0);
458 }

460 int
461 lx_sched_getscheduler(uintptr_t pid)
462 {
463         int     policy, rv;
464         pid_t   s_pid;
465         lwpid_t s_tid;

467         if ((pid_t)pid < 0)
468                 return (-EINVAL);

470         if (lx_lpid_to_spair((pid_t)pid, &s_pid, &s_tid) < 0)
471                 return (-ESRCH);

473         if (s_pid == getpid()) {
474                 struct sched_param dummy;

476                 if ((rv = pthread_getschedparam(s_tid, &policy, &dummy)) != 0)
477                         return (-rv);
478         } else
479                 if ((policy = sched_getscheduler(s_pid)) < 0)
480                         return (-errno);

482         /*
483          * Linux only supports certain policies; avoid confusing apps with
484          * alien policies.
485          */
486         switch (policy) {
487         case SCHED_FIFO:
488                 return (LX_SCHED_FIFO);
489         case SCHED_OTHER:
490                 return (LX_SCHED_OTHER);
491         case SCHED_RR:
492                 return (LX_SCHED_RR);
493         default:
494                 break;
495         }

497         return (LX_SCHED_OTHER);
498 }

500 int
501 lx_sched_setscheduler(uintptr_t pid, uintptr_t policy, uintptr_t param)
502 {
503         int     rt_pol;
504         int     rv;
505         pid_t   s_pid;
506         lwpid_t s_tid;
507         struct lx_sched_param lp;

509         struct sched_param sp;

511         if (((pid_t)pid < 0) || (param == NULL))
512                 return (-EINVAL);

514         if ((rt_pol = validate_policy((int)policy)) < 0)
515                 return (rt_pol);

517         if ((rv = ltos_sparam(policy, (struct lx_sched_param *)param,
518             &sp)) != 0)
519                 return (rv);

521         if (uucopy((void *)param, &lp, sizeof (lp)) != 0)
522                 return (-errno);
```

```
524         /*
525          * In Linux, the only valid SCHED_OTHER scheduler priority is 0
526          */
527         if ((rt_pol == LX_SCHED_OTHER) && (lp.lx_sched_prio != 0))
528                 return (-EINVAL);

530         if (lx_lpid_to_spair((pid_t)pid, &s_pid, &s_tid) < 0)
531                 return (-ESRCH);

533         /*
534          * Check if we're allowed to change the scheduler for the process.
535          *
536          * If we're operating on a thread, we can't just call
537          * pthread_setschedparam() because as all threads reside within a
538          * single Solaris process, Solaris will allow the modification.
539          *
540          * If we're operating on a process, we can't just call
541          * sched_setscheduler() because Solaris will allow the call to succeed
542          * if the scheduler and scheduler parameters do not differ from those
543          * being installed, but Linux wants the call to fail.
544          */
545         if ((rv = check_schedperms(s_pid)) != 0)
546                 return (rv);

548         if (s_pid == getpid()) {
549                 struct sched_param param;
550                 int pol;

552                 if ((pol = sched_getscheduler(s_pid)) != 0)
553                         return (-errno);

555                 /*
556                  * sched_setscheduler() returns the previous scheduling policy
557                  * on success, so call pthread_getschedparam() to get the
558                  * current thread's scheduling policy and return that if the
559                  * call to pthread_setschedparam() succeeds.
560                  */
561                 if ((rv = pthread_getschedparam(s_tid, &pol, &param)) != 0)
562                         return (-rv);

564                 return (((rv = pthread_setschedparam(s_tid, rt_pol, &sp)) != 0)
565                     ? -rv : pol);
566         }

568         return (((rv = sched_setscheduler(s_pid, rt_pol, &sp)) == -1)
569             ? -errno : rv);
570 }

572 int
573 lx_sched_get_priority_min(uintptr_t policy)
574 {
575         /*
576          * In Linux, the only valid SCHED_OTHER scheduler priority is 0.
577          * Linux scheduling priorities are not alterable, so there is no
578          * Solaris translation necessary.
579          */
580         switch (policy) {
581         case LX_SCHED_FIFO:
582         case LX_SCHED_RR:
583                 return (LX_SCHED_PRIORITY_MIN_RRFIFO);
584         case LX_SCHED_OTHER:
585                 return (LX_SCHED_PRIORITY_MIN_OTHER);
586         default:
587                 break;
588         }
```



```
589         return (-EINVAL);
590 }

592 int
593 lx_sched_get_priority_max(uintptr_t policy)
594 {
595         /*
596          * In Linux, the only valid SCHED_OTHER scheduler priority is 0
597          * Linux scheduling priorities are not alterable, so there is no
598          * Solaris translation necessary.
599          */
600         switch (policy) {
601         case LX_SCHED_FIFO:
602         case LX_SCHED_RR:
603                 return (LX_SCHED_PRIORITY_MAX_RRFIFO);
604         case LX_SCHED_OTHER:
605                 return (LX_SCHED_PRIORITY_MAX_OTHER);
606         default:
607                 break;
608         }
609         return (-EINVAL);
610 }
611 #endif /* ! codereview */
```

```
**********************************************************
    2468 Tue Jan 14 16:17:04 2014
new/usr/src/lib/brand/lx/lx_brand/common/sendfile.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

/*
 * lx_sendfile() and lx_sendfile64() are just branded versions of the
 * library calls available in the Solaris libsendfile (see sendfile(3EXT)).
 */

#include <sys/types.h>
#include <sys/syscall.h>
#include <sys/sendfile.h>
#include <string.h>
#include <errno.h>
#include <sys/lx_misc.h>

int
lx_sendfile(uintptr_t p1, uintptr_t p2, uintptr_t p3, uintptr_t p4)
{
        sysret_t rval;
        off_t off = 0;
        off_t *offp = (off_t *)p3;
        int error;
        struct sendfilevec sfv;
        size_t xferred;
        size_t sz = (size_t)p4;

        if (sz > 0 && uucopy(offp, &off, sizeof (off)) != 0)
                return (-errno);

        sfv.sfv_fd = p2;
        sfv.sfv_flag = 0;
        sfv.sfv_off = off;
        sfv.sfv_len = sz;
        error = __systemcall(&rval, SYS_sendfilev, SENDFILEV, p1, &sfv,
            1, &xferred);
```



```
        if (error == 0 && xferred > 0) {
                off += xferred;
                error = uucopy(&off, offp, sizeof (off));
        }

        return (error ? -error : (int)rval.sys_rval1);
}

int
lx_sendfile64(uintptr_t p1, uintptr_t p2, uintptr_t p3, uintptr_t p4)
{
        sysret_t rval;
        off64_t off = 0;
        off64_t *offp = (off64_t *)p3;
        size_t sz = (size_t)p4;
        int error;
        struct sendfilevec64 sfv;
        size_t xferred;

        if (sz > 0 && uucopy(offp, &off, sizeof (off)) != 0)
                return (-errno);

        sfv.sfv_fd = p2;
        sfv.sfv_flag = 0;
        sfv.sfv_off = off;
        sfv.sfv_len = sz;
        error = __systemcall(&rval, SYS_sendfilev, SENDFILEV64, p1, &sfv,
            1, &xferred);

        if (error == 0 && xferred > 0) {
                off += xferred;
                error = uucopy(&off, offp, sizeof (off));
        }

        return (error ? -error : (int)rval.sys_rval1);
}
#endif /* ! codereview */
```

```
**********************************************************
   51074 Tue Jan 14 16:17:04 2014
new/usr/src/lib/brand/lx/lx_brand/common/signal.c
LX zone support should now build and packages of relevance produced.
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/types.h>
#include <sys/param.h>
#include <sys/segments.h>
#include <sys/lx_types.h>
#include <sys/lx_brand.h>
#include <sys/lx_misc.h>
#include <sys/lx_debug.h>
#include <sys/lx_signal.h>
#include <sys/lx_syscall.h>
#include <sys/lx_thread.h>
#include <assert.h>
#include <errno.h>
#include <signal.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <thread.h>
#include <ucontext.h>
#include <unistd.h>
#include <stdio.h>
#include <libintl.h>
#include <ieeefp.h>

/*
 * Delivering signals to a Linux process is complicated by differences in
 * signal numbering, stack structure and contents, and the action taken when a
 * signal handler exits.  In addition, many signal-related structures, such as
 * sigset_ts, vary between Solaris and Linux.
 *
 * To support user-level signal handlers, the brand uses a double layer of
 * indirection to process and deliver signals to branded threads.
 *
```



```
 * When a Linux process sends a signal using the kill(2) system call, we must
 * translate the signal into the Solaris equivalent before handing control off
 * to the standard signalling mechanism.  When a signal is delivered to a Linux
 * process, we translate the signal number from Solaris to back to Linux.
 * Translating signals both at generation and delivery time ensures both that
 * Solaris signals are sent properly to Linux applications and that signals'
 * default behavior works as expected.
 *
 * In a normal Solaris process, signal delivery is interposed on for any thread
 * registering a signal handler by libc. Libc needs to do various bits of magic
 * to provide thread-safe critical regions, so it registers its own handler,
 * named sigacthandler(), using the sigaction(2) system call. When a signal is
 * received, sigacthandler() is called, and after some processing, libc turns
 * around and calls the user's signal handler via a routine named
 * call_user_handler().
 *
 * Adding a Linux branded thread to the mix complicates things somewhat.
 *
 * First, when a thread receives a signal, it may be running with a Linux value
 * in the x86 %gs segment register as opposed to the value Solaris threads
 * expect; if control were passed directly to Solaris code, such as libc's
 * sigacthandler(), that code would experience a segmentation fault the first
 * time it tried to dereference a memory location using %gs.
 *
 * Second, the signal number translation referenced above must take place.
 * Further, as was the case with Solaris libc, before the Linux signal handler
 * is called, the value of the %gs segment register MUST be restored to the
 * value Linux code expects.
 *
 * This need to translate signal numbers and manipulate the %gs register means
 * that while with standard Solaris libc, following a signal from generation to
 * delivery looks something like:
 *
 *      kernel ->
 *          sigacthandler() ->
 *              call_user_handler() ->
 *                  user signal handler
 *
 * while for the brand's Linux threads, this would look like:
 *
 *      kernel ->
 *          lx_sigacthandler() ->
 *              sigacthandler() ->
 *                  call_user_handler() ->
 *                      lx_call_user_handler() ->
 *                          Linux user signal handler
 *
 * The new addtions are:
 *
 *      lx_sigacthandler
 *      ================
 *      This routine is responsible for setting the %gs segment register to the
 *      value Solaris code expects, and jumping to Solaris' libc signal
 *      interposition handler, sigacthandler().
 *
 *      lx_call_user_handler
 *      ====================
 *      This routine is responsible for translating Solaris signal numbers to
 *      their Linux equivalents, building a Linux signal stack based on the
 *      information Solaris has provided, and passing the stack to the
 *      registered Linux signal handler. It is, in effect, the Linux thread
 *      equivalent to libc's call_user_handler().
 *
 * Installing lx_sigacthandler() is a bit tricky, as normally libc's
 * sigacthandler() routine is hidden from user programs. To facilitate this, a
 * new private function was added to libc, setsigaction():
```

```
 *
 *      void setsigacthandler(void (*new_handler)(int, siginfo_t *, void *),
 *          void (**old_handler)(int, siginfo_t *, void *))
 *
 * The routine works by modifying the per-thread data structure libc already
 * keeps that keeps track of the address of its own interposition handler with
 * the address passed in; the old handler's address is set in the pointer
 * pointed to by the second argument, if it is non-NULL, mimicking the behavior
 * of sigaction() itself.  Once setsigacthandler() has been executed, all
 * future branded threads this thread may create will automatically have the
 * proper interposition handler installed as the result of a normal
 * sigaction() call.
 *
 * Note that none of this interposition is necessary unless a Linux thread
 * registers a user signal handler, as the default action for all signals is the
 * same between Solaris and Linux save for one signal, SIGPWR.  For this reason,
 * the brand ALWAYS installs its own internal signal handler for SIGPWR that
 * translates the action to the Linux default, to terminate the process.
 * (Solaris' default action is to ignore SIGPWR.)
 *
 * It is also important to note that when signals are not translated, the brand
 * relies upon code interposing upon the wait(2) system call to translate
 * signals to their proper values for any Linux threads retrieving the status
 * of others.  So while the Solaris signal number for a particular signal is set
 * in a process' data structures (and would be returned as the result of say,
 * WTERMSIG()), the brand's interposiiton upon wait(2) is responsible for
 * translating the value WTERMSIG() would return from a Solaris signal number
 * to the appropriate Linux value.
 *
 * The process of returning to an interrupted thread of execution from a user
 * signal handler is entirely different between Solaris and Linux.  While
 * Solaris generally expects to set the context to the interrupted one on a
 * normal return from a signal handler, in the normal case Linux instead calls
 * code that calls a specific Linux system call, sigreturn(2).  Thus when a
 * Linux signal handler completes execution, instead of returning through what
 * would in libc be a call to setcontext(2), the sigreturn(2) Linux system call
 * is responsible for accomplishing much the same thing.
 *
 * This trampoline code looks something like this:
 *
 *      pop     %eax
 *      mov     LX_SYS_rt_sigreturn, %eax
 *      int     $0x80
 *
 * so when the Linux user signal handler is eventually called, the stack looks
 * like this (in the case of an "lx_sigstack" stack:
 *
 *      ------------------------------------------------------
 *      | Pointer to actual trampoline code (in code segment)  |
 *      ------------------------------------------------------
 *      | Linux signal number                                  |
 *      ------------------------------------------------------
 *      | Pointer to Linux siginfo_t (or NULL)                 |
 *      ------------------------------------------------------
 *      | Pointer to Linux ucontext_t (or NULL)                |
 *      ------------------------------------------------------
 *      | Linux siginfo_t                                      |
 *      ------------------------------------------------------
 *      | Linux ucontext_t                                     |
 *      ------------------------------------------------------
 *      | Linux struct _fpstate                                |
 *      ------------------------------------------------------
 *      | Trampoline code (marker for gdb, not really executed) |
 *      ------------------------------------------------------
 *
 * The brand takes the approach of intercepting the Linux sigreturn(2) system
```



```
 * call in order to turn it into the return through the libc call stack that
 * Solaris expects. This is done by the lx_sigreturn() and lx_rt_sigreturn()
 * routines, which remove the Linux signal frame from the stack and pass the
 * resulting stack pointer to another routine, lx_sigreturn_tolibc(), which
 * makes libc believe the user signal handler it had called returned.
 *
 * (Note that the trampoline code actually lives in a proper executable segment
 * and not on the stack, but gdb checks for the exact code sequence of the
 * trampoline code on the stack to determine whether it is in a signal stack
 * frame or not.  Really.)
 *
 * When control then returns to libc's call_user_handler() routine, a
 * setcontext(2) will be done that (in most cases) returns the thread executing
 * the code back to the location originally interrupted by receipt of the
 * signal.
 */

/*
 * Two flavors of Linux signal stacks:
 *
 * lx_sigstack - used for "modern" signal handlers, in practice those
 *               that have the sigaction(2) flag SA_SIGINFO set
 *
 * lx_oldsigstack - used for legacy signal handlers, those that do not have
 *                  the sigaction(2) flag SA_SIGINFO set or that were setup via
 *                  the signal(2) call.
 *
 * NOTE: Since these structures will be placed on the stack and stack math will
 *       be done with their sizes, they must be word aligned in size (32 bits)
 *       so the stack remains word aligned per the i386 ABI.
 */
struct lx_sigstack {
        void (*retaddr)();      /* address of real lx_rt_sigreturn code */
        int sig;                /* signal number */
        lx_siginfo_t *sip;      /* points to "si" if valid, NULL if not */
        lx_ucontext_t *ucp;     /* points to "uc" if valid, NULL if not */
        lx_siginfo_t si;        /* saved signal information */
        lx_ucontext_t uc;       /* saved user context */
        lx_fpstate_t fpstate;   /* saved FP state */
        char trampoline[8];     /* code for trampoline to lx_rt_sigreturn() */
};

struct lx_oldsigstack {
        void (*retaddr)();      /* address of real lx_sigreturn code */
        int sig;                /* signal number */
        lx_sigcontext_t sigc;   /* saved user context */
        lx_fpstate_t fpstate;   /* saved FP state */
        int sig_extra;          /* signal mask for signals [32 .. NSIG - 1] */
        char trampoline[8];     /* code for trampoline to lx_sigreturn() */
};

/*
 * libc_sigacthandler is set to the address of the libc signal interposition
 * routine, sigacthandler().
 */
void (*libc_sigacthandler)(int, siginfo_t *, void*);

/*
 * The lx_sighandlers structure needs to be a global due to the semantics of
 * clone().
 *
 * If CLONE_SIGHAND is set, the calling process and child share signal
 * handlers, and if either calls sigaction(2) it should change the behavior
 * in the other thread.  Each thread does, however, have its own signal mask
 * and set of pending signals.
 *
```

```
 * If CLONE_SIGHAND is not set, the child process should inherit a copy of
 * the signal handlers at the time of the clone() but later calls to
 * sigaction(2) should only affect the individual thread calling it.
 *
 * This maps perfectly to a thr_create(3C) thread semantic in the first
 * case and a fork(2)-type semantic in the second case.  By making
 * lx_sighandlers global, we automatically get the correct behavior.
 */
static lx_sighandlers_t lx_sighandlers;

/*
 * stol_stack() and ltos_stack() convert between Solaris and Linux stack_t
 * structures.
 *
 * These routines are needed because although the two structures have the same
 * contents, their contents are declared in a different order, so the content
 * of the structures cannot be copied with a simple bcopy().
 */
static void
stol_stack(stack_t *fr, lx_stack_t *to)
{
        to->ss_sp = fr->ss_sp;
        to->ss_flags = fr->ss_flags;
        to->ss_size = fr->ss_size;
}

static void
ltos_stack(lx_stack_t *fr, stack_t *to)
{
        to->ss_sp = fr->ss_sp;
        to->ss_flags = fr->ss_flags;
        to->ss_size = fr->ss_size;
}

static int
ltos_sigset(lx_sigset_t *lx_sigsetp, sigset_t *s_sigsetp)
{
        lx_sigset_t l;
        int lx_sig, sig;

        if (uucopy(lx_sigsetp, &l, sizeof (lx_sigset_t)) != 0)
                return (-errno);

        (void) sigemptyset(s_sigsetp);

        for (lx_sig = 1; lx_sig < LX_NSIG; lx_sig++) {
                if (lx_sigismember(&l, lx_sig) &&
                    ((sig = ltos_signo[lx_sig]) > 0))
                        (void) sigaddset(s_sigsetp, sig);
        }

        return (0);
}

static int
stol_sigset(sigset_t *s_sigsetp, lx_sigset_t *lx_sigsetp)
{
        lx_sigset_t l;
        int sig, lx_sig;

        bzero(&l, sizeof (lx_sigset_t));

        for (sig = 1; sig < NSIG; sig++) {
                if (sigismember(s_sigsetp, sig) &&
                    ((lx_sig = stol_signo[sig]) > 0))
                        lx_sigaddset(&l, lx_sig);
```



```
        }

        return ((uucopy(&l, lx_sigsetp, sizeof (lx_sigset_t)) != 0)
            ? -errno : 0);
}

static int
ltos_osigset(lx_osigset_t *lx_osigsetp, sigset_t *s_sigsetp)
{
        lx_osigset_t lo;
        int lx_sig, sig;

        if (uucopy(lx_osigsetp, &lo, sizeof (lx_osigset_t)) != 0)
                return (-errno);

        (void) sigemptyset(s_sigsetp);

        for (lx_sig = 1; lx_sig <= OSIGSET_NBITS; lx_sig++)
                if ((lo & OSIGSET_BITSET(lx_sig)) &&
                    ((sig = ltos_signo[lx_sig]) > 0))
                        (void) sigaddset(s_sigsetp, sig);

        return (0);
}

static int
stol_osigset(sigset_t *s_sigsetp, lx_osigset_t *lx_osigsetp)
{
        lx_osigset_t lo = 0;
        int lx_sig, sig;

        /*
         * Note that an lx_osigset_t can only represent the signals from
         * [1 .. OSIGSET_NBITS], so even though a signal may be present in the
         * Solaris sigset_t, it may not be representable as a bit in the
         * lx_osigset_t.
         */
        for (sig = 1; sig < NSIG; sig++)
                if (sigismember(s_sigsetp, sig) &&
                    ((lx_sig = stol_signo[sig]) > 0) &&
                    (lx_sig <= OSIGSET_NBITS))
                        lo |= OSIGSET_BITSET(lx_sig);

        return ((uucopy(&lo, lx_osigsetp, sizeof (lx_osigset_t)) != 0)
            ? -errno : 0);
}

static int
stol_sigcode(int si_code)
{
        switch (si_code) {
                case SI_USER:
                        return (LX_SI_USER);
                case SI_LWP:
                        return (LX_SI_TKILL);
                case SI_QUEUE:
                        return (LX_SI_QUEUE);
                case SI_TIMER:
                        return (LX_SI_TIMER);
                case SI_ASYNCIO:
                        return (LX_SI_ASYNCIO);
                case SI_MESGQ:
                        return (LX_SI_MESGQ);
                default:
                        return (si_code);
        }
```

```
391 }

393 int
394 stol_siginfo(siginfo_t *siginfop, lx_siginfo_t *lx_siginfop)
395 {
396         lx_siginfo_t lx_siginfo;

398         bzero(&lx_siginfo, sizeof (*lx_siginfop));

400         if ((lx_siginfo.lsi_signo = stol_signo[siginfop->si_signo]) <= 0) {
401                 errno = EINVAL;
402                 return (-1);
403         }

405         lx_siginfo.lsi_code = stol_sigcode(siginfop->si_code);
406         lx_siginfo.lsi_errno = siginfop->si_errno;

408         switch (lx_siginfo.lsi_signo) {
409                 /*
410                  * Semantics ARE defined for SIGKILL, but since
411                  * we can't catch it, we can't translate it. :-(
412                  */
413                 case LX_SIGPOLL:
414                         lx_siginfo.lsi_band = siginfop->si_band;
415                         lx_siginfo.lsi_fd = siginfop->si_fd;
416                         break;

418                 case LX_SIGCHLD:
419                         lx_siginfo.lsi_pid = siginfop->si_pid;
420                         lx_siginfo.lsi_status = siginfop->si_status;
421                         lx_siginfo.lsi_utime = siginfop->si_utime;
422                         lx_siginfo.lsi_stime = siginfop->si_stime;

424                         break;

426                 case LX_SIGILL:
427                 case LX_SIGBUS:
428                 case LX_SIGFPE:
429                         lx_siginfo.lsi_addr = siginfop->si_addr;
430                         break;

432                 default:
433                         lx_siginfo.lsi_pid = siginfop->si_pid;
434                         lx_siginfo.lsi_uid =
435                             LX_UID32_TO_UID16(siginfop->si_uid);
436                         break;
437         }

439         return ((uucopy(&lx_siginfo, lx_siginfop, sizeof (lx_siginfo_t)) != 0)
440             ? -errno : 0);
441 }

443 static void
444 stol_fpstate(fpregset_t *fpr, lx_fpstate_t *lfpr)
445 {
446         struct _fpstate *fpsp = (struct _fpstate *)fpr;
447         size_t copy_len;

449         /*
450          * The Solaris struct _fpstate and lx_fpstate_t are identical from the
451          * beginning of the structure to the lx_fpstate_t "magic" field, so
452          * just bcopy() those entries.
453          */
454         copy_len = (size_t)&(((lx_fpstate_t *)0)->magic);
455         bcopy(fpsp, lfpr, copy_len);
```



```
457         /*
458          * These fields are all only significant for the first 16 bits.
459          */
460         lfpr->cw &= 0xffff;             /* x87 control word */
461         lfpr->tag &= 0xffff;            /* x87 tag word */
462         lfpr->cssel &= 0xffff;          /* cs selector */
463         lfpr->datasel &= 0xffff;        /* ds selector */

465         /*
466          * Linux wants the x87 status word field to contain the value of the
467          * x87 saved exception status word.
468          */
469         lfpr->sw = lfpr->status & 0xffff;       /* x87 status word */

471         lfpr->mxcsr = fpsp->mxcsr;

473         if (fpsp->mxcsr != 0) {
474                 /*
475                  * Linux uses the "magic" field to denote whether the XMM
476                  * registers contain legal data or not.  Since we can't get to
477                  * %cr4 from userland to check the status of the OSFXSR bit,
478                  * check the mxcsr field to see if it's 0, which it should
479                  * never be on a system with the OXFXSR bit enabled.
480                  */
481                 lfpr->magic = LX_X86_FXSR_MAGIC;
482                 bcopy(fpsp->xmm, lfpr->_xmm, sizeof (lfpr->_xmm));
483         } else {
484                 lfpr->magic = LX_X86_FXSR_NONE;
485         }
486 }

488 static void
489 ltos_fpstate(lx_fpstate_t *lfpr, fpregset_t *fpr)
490 {
491         struct _fpstate *fpsp = (struct _fpstate *)fpr;
492         size_t copy_len;

494         /*
495          * The lx_fpstate_t and Solaris struct _fpstate are identical from the
496          * beginning of the structure to the struct _fpstate "mxcsr" field, so
497          * just bcopy() those entries.
498          *
499          * Note that we do NOT have to propogate changes the user may have made
500          * to the "status" word back to the "sw" word, unlike the way we have
501          * to deal with processing the ESP and UESP register values on return
502          * from a signal handler.
503          */
504         copy_len = (size_t)&(((struct _fpstate *)0)->mxcsr);
505         bcopy(lfpr, fpsp, copy_len);

507         /*
508          * These fields are all only significant for the first 16 bits.
509          */
510         fpsp->cw &= 0xffff;             /* x87 control word */
511         fpsp->sw &= 0xffff;             /* x87 status word */
512         fpsp->tag &= 0xffff;            /* x87 tag word */
513         fpsp->cssel &= 0xffff;          /* cs selector */
514         fpsp->datasel &= 0xffff;        /* ds selector */
515         fpsp->status &= 0xffff;         /* saved status */

517         fpsp->mxcsr = lfpr->mxcsr;

519         if (lfpr->magic == LX_X86_FXSR_MAGIC)
520                 bcopy(lfpr->_xmm, fpsp->xmm, sizeof (fpsp->xmm));
521 }
```

```
523 /*
524  * The brand needs a lx version of this because the format of the lx stack_t
525  * differs from the Solaris stack_t not really in content but in ORDER,
526  * so we can't simply pass pointers and expect things to work (sigh...)
527  */
528 int
529 lx_sigaltstack(uintptr_t nsp, uintptr_t osp)
530 {
531         lx_stack_t ls;
532         stack_t newsstack, oldsstack;
533         stack_t *nssp = (nsp ? &newsstack : NULL);
534         stack_t *ossp = (osp ? &oldsstack : NULL);
535
536         if (nsp) {
537                 if (uucopy((void *)nsp, &ls, sizeof (lx_stack_t)) != 0)
538                         return (-errno);
539
540                 if ((ls.ss_flags & LX_SS_DISABLE) == 0 &&
541                     ls.ss_size < LX_MINSIGSTKSZ)
542                         return (-ENOMEM);
543
544                 newsstack.ss_sp = (int *)ls.ss_sp;
545                 newsstack.ss_size = (long)ls.ss_size;
546                 newsstack.ss_flags = ls.ss_flags;
547         }
548
549         if (sigaltstack(nssp, ossp) != 0)
550                 return (-errno);
551
552         if (osp) {
553                 ls.ss_sp = (void *)oldsstack.ss_sp;
554                 ls.ss_size = (size_t)oldsstack.ss_size;
555                 ls.ss_flags = oldsstack.ss_flags;
556
557                 if (uucopy(&ls, (void *)osp, sizeof (lx_stack_t)) != 0)
558                         return (-errno);
559         }
560
561         return (0);
562 }
563
564 /*
565  * The following routines are needed because sigset_ts and siginfo_ts are
566  * different in format between Linux and Solaris.
567  *
568  * Note that there are two different lx_sigset structures, lx_sigset_ts and
569  * lx_osigset_ts:
570  *
571  *    + An lx_sigset_t is the equivalent of a Solaris sigset_t and supports
572  *      more than 32 signals.
573  *
574  *    + An lx_osigset_t is simply a uint32_t, so it by definition only supports
575  *      32 signals.
576  *
577  * When there are two versions of a routine, one prefixed with lx_rt_ and
578  * one prefixed with lx_ alone, in GENERAL the lx_rt_ routines deal with
579  * lx_sigset_ts while the lx_ routines deal with lx_osigset_ts.  Unfortunately,
580  * this is not always the case (e.g. lx_sigreturn() vs. lx_rt_sigreturn())
581  */
582 int
583 lx_sigpending(uintptr_t sigpend)
584 {
585         sigset_t sigpendset;
586
587         if (sigpending(&sigpendset) != 0)
588                 return (-errno);
```



```
589
590         return (stol_osigset(&sigpendset, (lx_osigset_t *)sigpend));
591 }
592
593 int
594 lx_rt_sigpending(uintptr_t sigpend, uintptr_t setsize)
595 {
596         sigset_t sigpendset;
597
598         if ((size_t)setsize != sizeof (lx_sigset_t))
599                 return (-EINVAL);
600
601         if (sigpending(&sigpendset) != 0)
602                 return (-errno);
603
604         return (stol_sigset(&sigpendset, (lx_sigset_t *)sigpend));
605 }
606
607 /*
608  * Create a common routine to encapsulate all of the sigprocmask code,
609  * as the only difference between lx_sigprocmask() and lx_rt_sigprocmask()
610  * is the usage of lx_osigset_ts vs. lx_sigset_ts, as toggled in the code by
611  * the setting of the "sigset_type" flag.
612  */
613 static int
614 lx_sigprocmask_common(uintptr_t how, uintptr_t l_setp, uintptr_t l_osetp,
615     uintptr_t sigset_type)
616 {
617         int err;
618         sigset_t set, oset;
619         sigset_t *s_setp = NULL;
620         sigset_t *s_osetp;
621
622         if (l_setp) {
623                 switch (how) {
624                         case LX_SIG_BLOCK:
625                                 how = SIG_BLOCK;
626                                 break;
627
628                         case LX_SIG_UNBLOCK:
629                                 how = SIG_UNBLOCK;
630                                 break;
631
632                         case LX_SIG_SETMASK:
633                                 how = SIG_SETMASK;
634                                 break;
635
636                         default:
637                                 return (-EINVAL);
638                 }
639
640                 s_setp = &set;
641
642                 if (sigset_type == USE_SIGSET)
643                         err = ltos_sigset((lx_sigset_t *)l_setp, s_setp);
644                 else
645                         err = ltos_osigset((lx_osigset_t *)l_setp, s_setp);
646
647                 if (err != 0)
648                         return (err);
649         }
650
651         s_osetp = (l_osetp ? &oset : NULL);
652
653         /*
654          * In a multithreaded environment, a call to sigprocmask(2) should
```

```
655          * only affect the current thread's signal mask so we don't need to
656          * explicitly call thr_sigsetmask(3C) here.
657          */
658         if (sigprocmask(how, s_setp, s_osetp) != 0)
659                 return (-errno);

661         if (l_osetp) {
662                 if (sigset_type == USE_SIGSET)
663                         err = stol_sigset(s_osetp, (lx_sigset_t *)l_osetp);
664                 else
665                         err = stol_osigset(s_osetp, (lx_osigset_t *)l_osetp);

667                 if (err != 0) {
668                         /*
669                          * Encountered a fault while writing to the old signal
670                          * mask buffer, so unwind the signal mask change made
671                          * above.
672                          */
673                         (void) sigprocmask(how, s_osetp, (sigset_t *)NULL);
674                         return (err);
675                 }
676         }

678         return (0);
679 }

681 int
682 lx_sigprocmask(uintptr_t how, uintptr_t setp, uintptr_t osetp)
683 {
684         return (lx_sigprocmask_common(how, setp, osetp, USE_OSIGSET));
685 }

687 int
688 lx_sgetmask(void)
689 {
690         lx_osigset_t oldmask;

692         return ((lx_sigprocmask_common(SIG_SETMASK, NULL, (uintptr_t)&oldmask,
693             USE_OSIGSET) != 0) ? -errno : (int)oldmask);
694 }

696 int
697 lx_ssetmask(uintptr_t sigmask)
698 {
699         lx_osigset_t newmask, oldmask;

701         newmask = (lx_osigset_t)sigmask;

703         return ((lx_sigprocmask_common(SIG_SETMASK, (uintptr_t)&newmask,
704             (uintptr_t)&oldmask, USE_OSIGSET) != 0) ? -errno : (int)oldmask);
705 }

707 int
708 lx_rt_sigprocmask(uintptr_t how, uintptr_t setp, uintptr_t osetp,
709     uintptr_t setsize)
710 {
711         if ((size_t)setsize != sizeof (lx_sigset_t))
712                 return (-EINVAL);

714         return (lx_sigprocmask_common(how, setp, osetp, USE_SIGSET));
715 }

717 int
718 lx_sigsuspend(uintptr_t set)
719 {
720         sigset_t s_set;
```



```
722         if (ltos_osigset((lx_osigset_t *)set, &s_set) != 0)
723                 return (-errno);

725         return ((sigsuspend(&s_set) == -1) ? -errno : 0);
726 }

728 int
729 lx_rt_sigsuspend(uintptr_t set, uintptr_t setsize)
730 {
731         sigset_t s_set;

733         if ((size_t)setsize != sizeof (lx_sigset_t))
734                 return (-EINVAL);

736         if (ltos_sigset((lx_sigset_t *)set, &s_set) != 0)
737                 return (-errno);

739         return ((sigsuspend(&s_set) == -1) ? -errno : 0);
740 }

742 int
743 lx_sigwaitinfo(uintptr_t set, uintptr_t sinfo)
744 {
745         lx_osigset_t *setp = (lx_osigset_t *)set;
746         lx_siginfo_t *sinfop = (lx_siginfo_t *)sinfo;

748         sigset_t s_set;
749         siginfo_t s_sinfo, *s_sinfop;
750         int rc;

752         if (ltos_osigset(setp, &s_set) != 0)
753                 return (-errno);

755         s_sinfop = (sinfop == NULL) ? NULL : &s_sinfo;

757         if ((rc = sigwaitinfo(&s_set, s_sinfop)) == -1)
758                 return (-errno);

760         if (s_sinfop == NULL)
761                 return (rc);

763         return ((stol_siginfo(s_sinfop, sinfop) != 0) ? -errno : rc);
764 }

766 int
767 lx_rt_sigwaitinfo(uintptr_t set, uintptr_t sinfo, uintptr_t setsize)
768 {
769         sigset_t s_set;
770         siginfo_t s_sinfo, *s_sinfop;
771         int rc;

773         lx_sigset_t *setp = (lx_sigset_t *)set;
774         lx_siginfo_t *sinfop = (lx_siginfo_t *)sinfo;

776         if ((size_t)setsize != sizeof (lx_sigset_t))
777                 return (-EINVAL);

779         if (ltos_sigset(setp, &s_set) != 0)
780                 return (-errno);

782         s_sinfop = (sinfop == NULL) ? NULL : &s_sinfo;

784         if ((rc = sigwaitinfo(&s_set, s_sinfop)) == -1)
785                 return (-errno);
```

```
787         if (s_sinfop == NULL)
788                 return (rc);
789
790         return ((stol_siginfo(s_sinfop, sinfop) != 0) ? -errno : rc);
791 }
792
793 int
794 lx_sigtimedwait(uintptr_t set, uintptr_t sinfo, uintptr_t toutp)
795 {
796         sigset_t s_set;
797         siginfo_t s_sinfo, *s_sinfop;
798         int rc;
799
800         lx_osigset_t *setp = (lx_osigset_t *)set;
801         lx_siginfo_t *sinfop = (lx_siginfo_t *)sinfo;
802
803         if (ltos_osigset(setp, &s_set) != 0)
804                 return (-errno);
805
806         s_sinfop = (sinfop == NULL) ? NULL : &s_sinfo;
807
808         if ((rc = sigtimedwait(&s_set, s_sinfop,
809             (struct timespec *)toutp)) == -1)
810                 return (-errno);
811
812         if (s_sinfop == NULL)
813                 return (rc);
814
815         return ((stol_siginfo(s_sinfop, sinfop) != 0) ? -errno : rc);
816 }
817
818 int
819 lx_rt_sigtimedwait(uintptr_t set, uintptr_t sinfo, uintptr_t toutp,
820     uintptr_t setsize)
821 {
822         sigset_t s_set;
823         siginfo_t s_sinfo, *s_sinfop;
824         int rc;
825
826         lx_sigset_t *setp = (lx_sigset_t *)set;
827         lx_siginfo_t *sinfop = (lx_siginfo_t *)sinfo;
828
829         if ((size_t)setsize != sizeof (lx_sigset_t))
830                 return (-EINVAL);
831
832         if (ltos_sigset(setp, &s_set) != 0)
833                 return (-errno);
834
835         s_sinfop = (sinfop == NULL) ? NULL : &s_sinfo;
836
837         if ((rc = sigtimedwait(&s_set, s_sinfop,
838             (struct timespec *)toutp)) == -1)
839                 return (-errno);
840
841         if (s_sinfop == NULL)
842                 return (rc);
843
844         return ((stol_siginfo(s_sinfop, sinfop) != 0) ? -errno : rc);
845 }
846
847 /*
848  * Intercept the Linux sigreturn() syscall to turn it into the return through
849  * the libc call stack that Solaris expects.
850  *
851  * When control returns to libc's call_user_handler() routine, a setcontext(2)
852  * will be done that returns thread execution to the point originally
```



```
853  * interrupted by receipt of the signal.
854  */
855 int
856 lx_sigreturn(void)
857 {
858         struct lx_oldsigstack *lx_ossp;
859         lx_sigset_t lx_sigset;
860         lx_regs_t *rp;
861         ucontext_t *ucp;
862         uintptr_t sp;
863
864         rp = lx_syscall_regs();
865
866         /*
867          * NOTE:  The sp saved in the context is eight bytes off of where we
868          *        need it to be.
869          */
870         sp = (uintptr_t)rp->lxr_esp - 8;
871
872         /*
873          * At this point, the stack pointer should point to the struct
874          * lx_oldsigstack that lx_build_old_signal_frame() constructed and
875          * placed on the stack.  We need to reference it a bit later, so
876          * save a pointer to it before incrementing our copy of the sp.
877          */
878         lx_ossp = (struct lx_oldsigstack *)sp;
879         sp += sizeof (struct lx_oldsigstack);
880
881         /*
882          * lx_sigdeliver() pushes LX_SIGRT_MAGIC on the stack before it
883          * creates the struct lx_oldsigstack.
884          *
885          * If we don't find it here, the stack's been corrupted and we need to
886          * kill ourselves.
887          */
888         if (*(uint32_t *)sp != LX_SIGRT_MAGIC)
889                 lx_err_fatal(gettext(
890                     "sp @ 0x%p, expected 0x%x, found 0x%x!"),
891                     sp, LX_SIGRT_MAGIC, *(uint32_t *)sp);
892
893         sp += sizeof (uint32_t);
894
895         /*
896          * For signal mask handling to be done properly, this call needs to
897          * return to the libc routine that originally called the signal handler
898          * rather than directly set the context back to the place the signal
899          * interrupted execution as the original Linux code would do.
900          *
901          * Here *sp points to the Solaris ucontext_t, so we need to copy
902          * machine registers the Linux signal handler may have modified
903          * back to the Solaris version.
904          */
905         ucp = (ucontext_t *)(*(uint32_t *)sp);
906
907         /*
908          * General registers copy across as-is, except Linux expects that
909          * changes made to uc_mcontext.gregs[ESP] will be reflected when the
910          * interrupted thread resumes execution after the signal handler. To
911          * emulate this behavior, we must modify uc_mcontext.gregs[UESP] to
912          * match uc_mcontext.gregs[ESP] as Solaris will restore the UESP
913          * value to ESP.
914          */
915         lx_ossp->sigc.sc_esp_at_signal = lx_ossp->sigc.sc_esp;
916         bcopy(&lx_ossp->sigc, &ucp->uc_mcontext, sizeof (gregset_t));
917
918         /* copy back FP regs if present */
```

```
        if (lx_ossp->sigc.sc_fpstate != NULL)
                ltos_fpstate(&lx_ossp->fpstate, &ucp->uc_mcontext.fpregs);

        /* convert Linux signal mask back to its Solaris equivalent */
        bzero(&lx_sigset, sizeof (lx_sigset_t));
        lx_sigset.__bits[0] = lx_ossp->sigc.sc_mask;
        lx_sigset.__bits[1] = lx_ossp->sig_extra;
        (void) ltos_sigset(&lx_sigset, &ucp->uc_sigmask);

        /*
         * At this point sp contains the value of the stack pointer when
         * lx_call_user_handler() was called.
         *
         * Pop one more value off the stack and pass the new sp to
         * lx_sigreturn_tolibc(), which will in turn manipulate the x86
         * registers to make it appear to libc's call_user_handler() as if the
         * handler it had called returned.
         */
        sp += sizeof (uint32_t);
        lx_debug("calling lx_sigreturn_tolibc(0x%p)", sp);
        lx_sigreturn_tolibc(sp);

        /*NOTREACHED*/
        return (0);
}

int
lx_rt_sigreturn(void)
{
        struct lx_sigstack *lx_ssp;
        lx_regs_t *rp;
        lx_ucontext_t *lx_ucp;
        ucontext_t *ucp;
        uintptr_t sp;

        rp = lx_syscall_regs();

        /*
         * NOTE:  Because of some silly compatibility measures done in the
         *        signal trampoline code to make sure it uses the _exact same_
         *        instruction sequence Linux does, we have to manually "pop"
         *        one extra four byte instruction off the stack here before
         *        passing the stack address to the syscall because the
         *        trampoline code isn't allowed to do it.
         *
         *        No, I'm not kidding.
         *
         *        The sp saved in the context is eight bytes off of where we
         *        need it to be, so the need to pop the extra four byte
         *        instruction means we need to subtract a net four bytes from
         *        the sp before "popping" the struct lx_sigstack off the stack.
         *        This will yield the value the stack pointer had before
         *        lx_sigdeliver() created the stack frame for the Linux signal
         *        handler.
         */
        sp = (uintptr_t)rp->lxr_esp - 4;

        /*
         * At this point, the stack pointer should point to the struct
         * lx_sigstack that lx_build_signal_frame() constructed and
         * placed on the stack.  We need to reference it a bit later, so
         * save a pointer to it before incrementing our copy of the sp.
         */
        lx_ssp = (struct lx_sigstack *)sp;
        sp += sizeof (struct lx_sigstack);
```



```
        /*
         * lx_sigdeliver() pushes LX_SIGRT_MAGIC on the stack before it
         * creates the struct lx_sigstack (and possibly struct lx_fpstate_t).
         *
         * If we don't find it here, the stack's been corrupted and we need to
         * kill ourselves.
         */
        if (*(uint32_t *)sp != LX_SIGRT_MAGIC)
                lx_err_fatal(gettext("sp @ 0x%p, expected 0x%x, found 0x%x!"),
                    sp, LX_SIGRT_MAGIC, *(uint32_t *)sp);

        sp += sizeof (uint32_t);

        /*
         * For signal mask handling to be done properly, this call needs to
         * return to the libc routine that originally called the signal handler
         * rather than directly set the context back to the place the signal
         * interrupted execution as the original Linux code would do.
         *
         * Here *sp points to the Solaris ucontext_t, so we need to copy
         * machine registers the Linux signal handler may have modified
         * back to the Solaris version.
         */
        ucp = (ucontext_t *)(*(uint32_t *)sp);

        lx_ucp = lx_ssp->ucp;

        if (lx_ucp != NULL) {
                /*
                 * General registers copy across as-is, except Linux expects
                 * that changes made to uc_mcontext.gregs[ESP] will be reflected
                 * when the interrupted thread resumes execution after the
                 * signal handler. To emulate this behavior, we must modify
                 * uc_mcontext.gregs[UESP] to match uc_mcontext.gregs[ESP] as
                 * Solaris will restore the UESP value to ESP.
                 */
                lx_ucp->uc_sigcontext.sc_esp_at_signal =
                    lx_ucp->uc_sigcontext.sc_esp;
                bcopy(&lx_ucp->uc_sigcontext, &ucp->uc_mcontext.gregs,
                    sizeof (gregset_t));

                if (lx_ucp->uc_sigcontext.sc_fpstate != NULL)
                        ltos_fpstate(lx_ucp->uc_sigcontext.sc_fpstate,
                            &ucp->uc_mcontext.fpregs);

                /*
                 * Convert the Linux signal mask and stack back to their
                 * Solaris equivalents.
                 */
                (void) ltos_sigset(&lx_ucp->uc_sigmask, &ucp->uc_sigmask);
                ltos_stack(&lx_ucp->uc_stack, &ucp->uc_stack);
        }

        /*
         * At this point sp contains the value of the stack pointer when
         * lx_call_user_handler() was called.
         *
         * Pop one more value off the stack and pass the new sp to
         * lx_sigreturn_tolibc(), which will in turn manipulate the x86
         * registers to make it appear to libc's call_user_handler() as if the
         * handler it had called returned.
         */
        sp += sizeof (uint32_t);
        lx_debug("calling lx_sigreturn_tolibc(0x%p)", sp);
        lx_sigreturn_tolibc(sp);
```

```
1051         /*NOTREACHED*/
1052         return (0);
1053 }

1055 /*
1056  * Build signal frame for processing for "old" (legacy) Linux signals
1057  */
1058 static void
1059 lx_build_old_signal_frame(int lx_sig, siginfo_t *sip, void *p, void *sp)
1060 {
1061         extern void lx_sigreturn_tramp();

1063         lx_sigset_t lx_sigset;
1064         ucontext_t *ucp = (ucontext_t *)p;
1065         struct lx_sigaction *lxsap;
1066         struct lx_oldsigstack *lx_ossp = sp;

1068         lx_debug("building old signal frame for lx sig %d at 0x%p", lx_sig, sp);

1070         lx_ossp->sig = lx_sig;
1071         lxsap = &lx_sighandlers.lx_sa[lx_sig];
1072         lx_debug("lxsap @ 0x%p", lxsap);

1074         if (lxsap && (lxsap->lxsa_flags & LX_SA_RESTORER) &&
1075             lxsap->lxsa_restorer) {
1076                 lx_ossp->retaddr = lxsap->lxsa_restorer;
1077                 lx_debug("lxsa_restorer exists @ 0x%p", lx_ossp->retaddr);
1078         } else {
1079                 lx_ossp->retaddr = lx_sigreturn_tramp;
1080                 lx_debug("lx_ossp->retaddr set to 0x%p", lx_sigreturn_tramp);
1081         }

1083         lx_debug("osf retaddr = 0x%p", lx_ossp->retaddr);

1085         /* convert Solaris signal mask and stack to their Linux equivalents */
1086         (void) stol_sigset(&ucp->uc_sigmask, &lx_sigset);
1087         lx_ossp->sigc.sc_mask = lx_sigset.__bits[0];
1088         lx_ossp->sig_extra = lx_sigset.__bits[1];

1090         /*
1091          * General registers copy across as-is, except Linux expects that
1092          * uc_mcontext.gregs[ESP] == uc_mcontext.gregs[UESP] on receipt of a
1093          * signal.
1094          */
1095         bcopy(&ucp->uc_mcontext, &lx_ossp->sigc, sizeof (gregset_t));
1096         lx_ossp->sigc.sc_esp = lx_ossp->sigc.sc_esp_at_signal;

1098         /*
1099          * cr2 contains the faulting address, and Linux only sets cr2 for a
1100          * a segmentation fault.
1101          */
1102         lx_ossp->sigc.sc_cr2 = (((lx_sig == LX_SIGSEGV) && (sip)) ?
1103             (uintptr_t)sip->si_addr : 0);

1105         /* convert FP regs if present */
1106         if (ucp->uc_flags & UC_FPU) {
1107                 stol_fpstate(&ucp->uc_mcontext.fpregs, &lx_ossp->fpstate);
1108                 lx_ossp->sigc.sc_fpstate = &lx_ossp->fpstate;
1109         } else {
1110                 lx_ossp->sigc.sc_fpstate = NULL;
1111         }

1113         /*
1114          * Believe it or not, gdb wants to SEE the trampoline code on the
1115          * bottom of the stack to determine whether the stack frame belongs to
1116          * a signal handler, even though this code is no longer actually
```



```
1117          * called.
1118          *
1119          * You can't make this stuff up.
1120          */
1121         bcopy((void *)lx_sigreturn_tramp, lx_ossp->trampoline,
1122             sizeof (lx_ossp->trampoline));
1123 }

1125 /*
1126  * Build signal frame for processing for modern Linux signals
1127  */
1128 static void
1129 lx_build_signal_frame(int lx_sig, siginfo_t *sip, void *p, void *sp)
1130 {
1131         extern void lx_rt_sigreturn_tramp();

1133         lx_ucontext_t *lx_ucp;
1134         ucontext_t *ucp = (ucontext_t *)p;
1135         struct lx_sigstack *lx_ssp = sp;
1136         struct lx_sigaction *lxsap;

1138         lx_debug("building signal frame for lx sig %d at 0x%p", lx_sig, sp);

1140         lx_ucp = &lx_ssp->uc;
1141         lx_ssp->ucp = lx_ucp;
1142         lx_ssp->sig = lx_sig;

1144         lxsap = &lx_sighandlers.lx_sa[lx_sig];
1145         lx_debug("lxsap @ 0x%p", lxsap);

1147         if (lxsap && (lxsap->lxsa_flags & LX_SA_RESTORER) &&
1148             lxsap->lxsa_restorer) {
1149                 lx_ssp->retaddr = lxsap->lxsa_restorer;
1150                 lx_debug("lxsa_restorer exists @ 0x%p", lx_ssp->retaddr);
1151         } else {
1152                 lx_ssp->retaddr = lx_rt_sigreturn_tramp;
1153                 lx_debug("lx_ssp->retaddr set to 0x%p", lx_rt_sigreturn_tramp);
1154         }

1156         /* Linux has these fields but always clears them to 0 */
1157         lx_ucp->uc_flags = 0;
1158         lx_ucp->uc_link = NULL;

1160         /* convert Solaris signal mask and stack to their Linux equivalents */
1161         (void) stol_sigset(&ucp->uc_sigmask, &lx_ucp->uc_sigmask);
1162         stol_stack(&ucp->uc_stack, &lx_ucp->uc_stack);

1164         /*
1165          * General registers copy across as-is, except Linux expects that
1166          * uc_mcontext.gregs[ESP] == uc_mcontext.gregs[UESP] on receipt of a
1167          * signal.
1168          */
1169         bcopy(&ucp->uc_mcontext, &lx_ucp->uc_sigcontext, sizeof (gregset_t));
1170         lx_ucp->uc_sigcontext.sc_esp = lx_ucp->uc_sigcontext.sc_esp_at_signal;

1172         /*
1173          * cr2 contains the faulting address, which Linux only sets for a
1174          * a segmentation fault.
1175          */
1176         lx_ucp->uc_sigcontext.sc_cr2 = ((lx_sig == LX_SIGSEGV) && (sip)) ?
1177             (uintptr_t)sip->si_addr : 0;

1179         /*
1180          * Point the lx_siginfo_t pointer to the signal stack's lx_siginfo_t
1181          * if there was a Solaris siginfo_t to convert, otherwise set it to
1182          * NULL.
```

```
1183          */
1184         if ((sip) && (stol_siginfo(sip, &lx_ssp->si) == 0))
1185                 lx_ssp->sip = &lx_ssp->si;
1186         else
1187                 lx_ssp->sip = NULL;

1189         /* convert FP regs if present */
1190         if (ucp->uc_flags & UC_FPU) {
1191                 /*
1192                  * Copy FP regs to the appropriate place in the the lx_sigstack
1193                  * structure.
1194                  */
1195                 stol_fpstate(&ucp->uc_mcontext.fpregs, &lx_ssp->fpstate);
1196                 lx_ucp->uc_sigcontext.sc_fpstate = &lx_ssp->fpstate;
1197         } else
1198                 lx_ucp->uc_sigcontext.sc_fpstate = NULL;

1200         /*
1201          * Believe it or not, gdb wants to SEE the trampoline code on the
1202          * bottom of the stack to determine whether the stack frame belongs to
1203          * a signal handler, even though this code is no longer actually
1204          * called.
1205          *
1206          * You can't make this stuff up.
1207          */
1208         bcopy((void *)lx_rt_sigreturn_tramp, lx_ssp->trampoline,
1209             sizeof (lx_ssp->trampoline));
1210 }

1212 /*
1213  * This is the second level interposition handler for Linux signals.
1214  */
1215 static void
1216 lx_call_user_handler(int sig, siginfo_t *sip, void *p)
1217 {
1218         void (*user_handler)();
1219         void (*stk_builder)();

1221         lx_tsd_t *lx_tsd;
1222         struct lx_sigaction *lxsap;
1223         ucontext_t *ucp = (ucontext_t *)p;
1224         uintptr_t gs;
1225         size_t stksize;
1226         int err, lx_sig;

1228         /*
1229          * If Solaris signal has no Linux equivalent, effectively
1230          * ignore it.
1231          */
1232         if ((lx_sig = stol_signo[sig]) == -1) {
1233                 lx_debug("caught solaris signal %d, no Linux equivalent", sig);
1234                 return;
1235         }

1237         lx_debug("interpose caught solaris signal %d, translating to Linux "
1238             "signal %d", sig, lx_sig);

1240         lxsap = &lx_sighandlers.lx_sa[lx_sig];
1241         lx_debug("lxsap @ 0x%p", lxsap);

1243         if ((sig == SIGPWR) && (lxsap->lxsa_handler == SIG_DFL)) {
1244                 /* Linux SIG_DFL for SIGPWR is to terminate */
1245                 exit(LX_SIGPWR | 0x80);
1246         }

1248         if ((lxsap->lxsa_handler == SIG_DFL) ||
```



```
1249             (lxsap->lxsa_handler == SIG_IGN))
1250                 lx_err_fatal(gettext("%s set to %s?  How?!?!?"),
1251                     "lxsa_handler",
1252                     ((lxsap->lxsa_handler == SIG_DFL) ? "SIG_DFL" : "SIG_IGN"),
1253                     lxsap->lxsa_handler);

1255         if ((err = thr_getspecific(lx_tsd_key, (void **)&lx_tsd)) != 0)
1256                 lx_err_fatal(gettext(
1257                     "%s: unable to read thread-specific data: %s"),
1258                     "lx_call_user_handler", strerror(err));

1260         assert(lx_tsd != 0);

1262         gs = lx_tsd->lxtsd_gs & 0xffff;         /* gs is only 16 bits */

1264         /*
1265          * Any zero %gs value should be caught when a save is attempted in
1266          * lx_emulate(), but this extra check will catch any zero values due to
1267          * bugs in the library.
1268          */
1269         assert(gs != 0);

1271         if (lxsap->lxsa_flags & LX_SA_SIGINFO) {
1272                 stksize = sizeof (struct lx_sigstack);
1273                 stk_builder = lx_build_signal_frame;
1274         } else  {
1275                 stksize = sizeof (struct lx_oldsigstack);
1276                 stk_builder = lx_build_old_signal_frame;
1277         }

1279         user_handler = lxsap->lxsa_handler;

1281         lx_debug("delivering %d (lx %d) to handler at 0x%p with gs 0x%x", sig,
1282             lx_sig, lxsap->lxsa_handler, gs);

1284         if (lxsap->lxsa_flags & LX_SA_RESETHAND)
1285                 lxsap->lxsa_handler = SIG_DFL;

1287         /*
1288          * lx_sigdeliver() doesn't return, so it relies on the Linux
1289          * signal handlers to clean up the stack, reset the current
1290          * signal mask and return to the code interrupted by the signal.
1291          */
1292         lx_sigdeliver(lx_sig, sip, ucp, stksize, stk_builder, user_handler, gs);
1293 }

1295 /*
1296  * Common routine to modify sigaction characteristics of a thread.
1297  *
1298  * We shouldn't need any special locking code here as we actually use
1299  * libc's sigaction() to do all the real work, so its thread locking should
1300  * take care of any issues for us.
1301  */
1302 static int
1303 lx_sigaction_common(int lx_sig, struct lx_sigaction *lxsp,
1304     struct lx_sigaction *olxsp)
1305 {
1306         struct lx_sigaction *lxsap;
1307         struct sigaction sa;

1309         if (lx_sig <= 0 || lx_sig >= LX_NSIG)
1310                 return (-EINVAL);

1312         lxsap = &lx_sighandlers.lx_sa[lx_sig];
1313         lx_debug("&lx_sighandlers.lx_sa[%d] = 0x%p", lx_sig, lxsap);
```

```
1315        if ((olxsp != NULL) &&
1316            ((uucopy(lxsap, olxsp, sizeof (struct lx_sigaction))) != 0))
1317                return (-errno);

1319        if (lxsp != NULL) {
1320                int err, sig;
1321                struct lx_sigaction lxsa;
1322                sigset_t new_set, oset;

1324                if (uucopy(lxsp, &lxsa, sizeof (struct lx_sigaction)) != 0)
1325                        return (-errno);

1327                if ((sig = ltos_signo[lx_sig]) != -1) {
1328                        /*
1329                         * Block this signal while messing with its dispostion
1330                         */
1331                        (void) sigemptyset(&new_set);
1332                        (void) sigaddset(&new_set, sig);

1334                        if (sigprocmask(SIG_BLOCK, &new_set, &oset) < 0) {
1335                                err = errno;
1336                                lx_debug("unable to block signal %d: %s", sig,
1337                                    strerror(err));
1338                                return (-err);
1339                        }

1341                        /*
1342                         * We don't really need the old signal disposition at
1343                         * this point, but this weeds out signals that would
1344                         * cause sigaction() to return an error before we change
1345                         * anything other than the current signal mask.
1346                         */
1347                        if (sigaction(sig, NULL, &sa) < 0) {
1348                                err = errno;
1349                                lx_debug("sigaction() to get old "
1350                                    "disposition for signal %d failed: "
1351                                    "%s", sig, strerror(err));
1352                                (void) sigprocmask(SIG_SETMASK, &oset, NULL);
1353                                return (-err);
1354                        }

1356                        if ((lxsa.lxsa_handler != SIG_DFL) &&
1357                            (lxsa.lxsa_handler != SIG_IGN)) {
1358                                sa.sa_handler = lx_call_user_handler;

1360                                /*
1361                                 * The interposition signal handler needs the
1362                                 * information provided via the SA_SIGINFO flag.
1363                                 */
1364                                sa.sa_flags = SA_SIGINFO;

1366                                if (lxsa.lxsa_flags & LX_SA_NOCLDSTOP)
1367                                        sa.sa_flags |= SA_NOCLDSTOP;
1368                                if (lxsa.lxsa_flags & LX_SA_NOCLDWAIT)
1369                                        sa.sa_flags |= SA_NOCLDWAIT;
1370                                if (lxsa.lxsa_flags & LX_SA_ONSTACK)
1371                                        sa.sa_flags |= SA_ONSTACK;
1372                                if (lxsa.lxsa_flags & LX_SA_RESTART)
1373                                        sa.sa_flags |= SA_RESTART;
1374                                if (lxsa.lxsa_flags & LX_SA_NODEFER)
1375                                        sa.sa_flags |= SA_NODEFER;

1377                                /*
1378                                 * Can't use RESETHAND with SIGPWR due to
1379                                 * different default actions between Linux
1380                                 * and Solaris.
```



```
1381                                 */
1382                                if ((sig != SIGPWR) &&
1383                                    (lxsa.lxsa_flags & LX_SA_RESETHAND))
1384                                        sa.sa_flags |= SA_RESETHAND;

1386                                if (ltos_sigset(&lxsa.lxsa_mask,
1387                                    &sa.sa_mask) != 0) {
1388                                        err = errno;
1389                                        (void) sigprocmask(SIG_SETMASK, &oset,
1390                                            NULL);
1391                                        return (-err);
1392                                }

1394                                lx_debug("interposing handler @ 0x%p for "
1395                                    "signal %d (lx %d), flags 0x%x",
1396                                    lxsa.lxsa_handler, sig, lx_sig,
1397                                    lxsa.lxsa_flags);

1399                                if (sigaction(sig, &sa, NULL) < 0) {
1400                                        err = errno;
1401                                        lx_debug("sigaction() to set new "
1402                                            "disposition for signal %d failed: "
1403                                            "%s", sig, strerror(err));
1404                                        (void) sigprocmask(SIG_SETMASK, &oset,
1405                                            NULL);
1406                                        return (-err);
1407                                }
1408                        } else if ((sig != SIGPWR) ||
1409                            ((sig == SIGPWR) &&
1410                            (lxsa.lxsa_handler == SIG_IGN))) {
1411                                /*
1412                                 * There's no need to interpose for SIG_DFL or
1413                                 * SIG_IGN so just call libc's sigaction(), but
1414                                 * don't allow SIG_DFL for SIGPWR due to
1415                                 * differing default actions between Linux and
1416                                 * Solaris.
1417                                 *
1418                                 * Get the previous disposition first so things
1419                                 * like sa_mask and sa_flags are preserved over
1420                                 * a transition to SIG_DFL or SIG_IGN, which is
1421                                 * what Linux expects.
1422                                 */

1424                                sa.sa_handler = lxsa.lxsa_handler;

1426                                if (sigaction(sig, &sa, NULL) < 0) {
1427                                        err = errno;
1428                                        lx_debug("sigaction(%d, %s) failed: %s",
1429                                            sig, ((sa.sa_handler == SIG_DFL) ?
1430                                            "SIG_DFL" : "SIG_IGN"),
1431                                            strerror(err));
1432                                        (void) sigprocmask(SIG_SETMASK, &oset,
1433                                            NULL);
1434                                        return (-err);
1435                                }
1436                        }
1437                } else {
1438                        lx_debug("Linux signal with no kill support "
1439                            "specified: %d", lx_sig);
1440                }

1442                /*
1443                 * Save the new disposition for the signal in the global
1444                 * lx_sighandlers structure.
1445                 */
1446                bcopy(&lxsa, lxsap, sizeof (struct lx_sigaction));
```

```
1448                 /*
1449                  * Reset the signal mask to what we came in with if
1450                  * we were modifying a kill-supported signal.
1451                  */
1452                 if (sig != -1)
1453                         (void) sigprocmask(SIG_SETMASK, &oset, NULL);
1454         }

1456         return (0);
1457 }

1459 int
1460 lx_sigaction(uintptr_t lx_sig, uintptr_t actp, uintptr_t oactp)
1461 {
1462         int val;
1463         struct lx_sigaction sa, osa;
1464         struct lx_sigaction *sap, *osap;
1465         struct lx_osigaction *osp;

1467         sap = (actp ? &sa : NULL);
1468         osap = (oactp ? &osa : NULL);

1470         /*
1471          * If we have a source pointer, convert source lxsa_mask from
1472          * lx_osigset_t to lx_sigset_t format.
1473          */
1474         if (sap) {
1475                 osp = (struct lx_osigaction *)actp;
1476                 sap->lxsa_handler = osp->lxsa_handler;

1478                 bzero(&sap->lxsa_mask, sizeof (lx_sigset_t));

1480                 for (val = 1; val <= OSIGSET_NBITS; val++)
1481                         if (osp->lxsa_mask & OSIGSET_BITSET(val))
1482                                 (void) lx_sigaddset(&sap->lxsa_mask, val);

1484                 sap->lxsa_flags = osp->lxsa_flags;
1485                 sap->lxsa_restorer = osp->lxsa_restorer;
1486         }

1488         if ((val = lx_sigaction_common(lx_sig, sap, osap)))
1489                 return (val);

1491         /*
1492          * If we have a save pointer, convert the old lxsa_mask from
1493          * lx_sigset_t to lx_osigset_t format.
1494          */
1495         if (osap) {
1496                 osp = (struct lx_osigaction *)oactp;

1498                 osp->lxsa_handler = osap->lxsa_handler;

1500                 bzero(&osp->lxsa_mask, sizeof (osp->lxsa_mask));
1501                 for (val = 1; val <= OSIGSET_NBITS; val++)
1502                         if (lx_sigismember(&osap->lxsa_mask, val))
1503                                 osp->lxsa_mask |= OSIGSET_BITSET(val);

1505                 osp->lxsa_flags = osap->lxsa_flags;
1506                 osp->lxsa_restorer = osap->lxsa_restorer;
1507         }

1509         return (0);
1510 }

1512 int
```



```
1513 lx_rt_sigaction(uintptr_t lx_sig, uintptr_t actp, uintptr_t oactp,
1514     uintptr_t setsize)
1515 {
1516         /*
1517          * The "new" rt_sigaction call checks the setsize
1518          * parameter.
1519          */
1520         if ((size_t)setsize != sizeof (lx_sigset_t))
1521                 return (-EINVAL);

1523         return (lx_sigaction_common(lx_sig, (struct lx_sigaction *)actp,
1524             (struct lx_sigaction *)oactp));
1525 }

1527 /*
1528  * Convert signal syscall to a call to the lx_sigaction() syscall
1529  */
1530 int
1531 lx_signal(uintptr_t lx_sig, uintptr_t handler)
1532 {
1533         struct sigaction act;
1534         struct sigaction oact;
1535         int rc;

1537         /*
1538          * Use sigaction to mimic SYSV signal() behavior; glibc will
1539          * actually call sigaction(2) itself, so we're really reaching
1540          * back for signal(2) semantics here.
1541          */
1542         bzero(&act, sizeof (act));
1543         act.sa_handler = (void (*)())handler;
1544         act.sa_flags = SA_RESETHAND | SA_NODEFER;

1546         rc = lx_sigaction(lx_sig, (uintptr_t)&act, (uintptr_t)&oact);
1547         return ((rc == 0) ? ((int)oact.sa_handler) : rc);
1548 }

1550 int
1551 lx_tgkill(uintptr_t tgid, uintptr_t pid, uintptr_t sig)
1552 {
1553         if (((pid_t)tgid <= 0) || ((pid_t)pid <= 0))
1554                 return (-EINVAL);

1556         if (tgid != pid) {
1557                 lx_unsupported(gettext(
1558                     "BrandZ tgkill(2) does not support gid != pid\n"));
1559                 return (-ENOTSUP);
1560         }

1562         /*
1563          * Pad the lx_tkill() call with NULLs to match the IN_KERNEL_SYSCALL
1564          * prototype generated for it by IN_KERNEL_SYSCALL in lx_brand.c.
1565          */
1566         return (lx_tkill(pid, sig, NULL, NULL, NULL, NULL));
1567 }

1569 /*
1570  * This C routine to save the passed %gs value into the thread-specific save
1571  * area is called by the assembly routine lx_sigacthandler.
1572  */
1573 void
1574 lx_sigsavegs(uintptr_t signalled_gs)
1575 {
1576         lx_tsd_t *lx_tsd;
1577         int err;
```

```
1579         signalled_gs &= 0xffff;         /* gs is only 16 bits */

1581         /*
1582          * While a %gs of 0 is technically legal (as long as the application
1583          * never dereferences memory using %gs), Solaris has its own ideas as
1584          * to how a zero %gs should be handled in _update_sregs(), such that
1585          * any 32-bit user process with a %gs of zero running on a system with
1586          * a 64-bit kernel will have its %gs hidden base register stomped on on
1587          * return from a system call, leaving an incorrect base address in
1588          * place until the next time %gs is actually reloaded (forcing a reload
1589          * of the base address from the appropriate descriptor table.)
1590          *
1591          * Of course the kernel will once again stomp on THAT base address when
1592          * returning from a system call, resulting in an application
1593          * segmentation fault.
1594          *
1595          * To avoid this situation, disallow a save of a zero %gs here in order
1596          * to try and capture any Linux process that takes a signal with a zero
1597          * %gs installed.
1598          */
1599         assert(signalled_gs != 0);

1601         if (signalled_gs != LWPGS_SEL) {
1602                 if ((err = thr_getspecific(lx_tsd_key,
1603                     (void **)&lx_tsd)) != 0)
1604                         lx_err_fatal(gettext(
1605                             "%s: unable to read thread-specific data: %s"),
1606                             "sigsavegs", strerror(err));

1608                 assert(lx_tsd != 0);

1610                 lx_tsd->lxtsd_gs = signalled_gs;

1612                 lx_debug("lx_sigsavegs(): gsp 0x%p, saved gs: 0x%x\n",
1613                     lx_tsd, signalled_gs);
1614         }
1615 }

1617 int
1618 lx_siginit(void)
1619 {
1620         extern void set_setcontext_enforcement(int);
1621         extern void lx_sigacthandler(int, siginfo_t *, void *);

1623         struct sigaction sa;
1624         sigset_t new_set, oset;
1625         int lx_sig, sig;

1627         /*
1628          * Block all signals possible while setting up the signal imposition
1629          * mechanism.
1630          */
1631         (void) sigfillset(&new_set);

1633         if (sigprocmask(SIG_BLOCK, &new_set, &oset) < 0)
1634                 lx_err_fatal(gettext("unable to block signals while setting up "
1635                     "imposition mechanism: %s"), strerror(errno));

1637         /*
1638          * Ignore any signals that have no Linux analog so that those
1639          * signals cannot be sent to Linux processes from the global zone
1640          */
1641         for (sig = 1; sig < NSIG; sig++)
1642                 if (stol_signo[sig] < 0)
1643                         (void) sigignore(sig);
```



```
1645         /*
1646          * As mentioned previously, when a user signal handler is installed
1647          * via sigaction(), libc interposes on the mechanism by actually
1648          * installing an internal routine sigacthandler() as the signal
1649          * handler.  On receipt of the signal, libc does some thread-related
1650          * processing via sigacthandler(), then calls the registered user
1651          * signal handler on behalf of the user.
1652          *
1653          * We need to interpose on that mechanism to make sure the correct
1654          * %gs segment register value is installed before the libc routine
1655          * is called, otherwise the libc code will die with a segmentation
1656          * fault.
1657          *
1658          * The private libc routine setsigacthandler() will set our
1659          * interposition routine, lx_sigacthandler(), as the default
1660          * "sigacthandler" routine for all new signal handlers for this
1661          * thread.
1662          */
1663         setsigacthandler(lx_sigacthandler, &libc_sigacthandler);
1664         lx_debug("lx_sigacthandler installed, libc_sigacthandler = 0x%p",
1665             libc_sigacthandler);

1667         /*
1668          * Mark any signals that are ignored as ignored in our interposition
1669          * handler array
1670          */
1671         for (lx_sig = 1; lx_sig < LX_NSIG; lx_sig++) {
1672                 if (((sig = ltos_signo[lx_sig]) != -1) &&
1673                     (sigaction(sig, NULL, &sa) < 0))
1674                         lx_err_fatal(gettext("unable to determine previous "
1675                             "disposition for signal %d: %s"),
1676                             sig, strerror(errno));

1678                 if (sa.sa_handler == SIG_IGN) {
1679                         lx_debug("marking signal %d (lx %d) as SIG_IGN",
1680                             sig, lx_sig);
1681                         lx_sighandlers.lx_sa[lx_sig].lxsa_handler = SIG_IGN;
1682                 }
1683         }

1685         /*
1686          * Have our interposition handler handle SIGPWR to start with,
1687          * as it has a default action of terminating the process in Linux
1688          * but its default is to be ignored in Solaris.
1689          */
1690         (void) sigemptyset(&sa.sa_mask);
1691         sa.sa_sigaction = lx_call_user_handler;
1692         sa.sa_flags = SA_SIGINFO;

1694         if (sigaction(SIGPWR, &sa, NULL) < 0)
1695                 lx_err_fatal(gettext("%s failed: %s"), "sigaction(SIGPWR)",
1696                     strerror(errno));

1698         /*
1699          * Solaris' libc forces certain register values in the ucontext_t
1700          * used to restore a post-signal user context to be those Solaris
1701          * expects; however that is not what we want to happen if the signal
1702          * was taken while branded code was executing, so we must disable
1703          * that behavior.
1704          */
1705         set_setcontext_enforcement(0);

1707         /*
1708          * Reset the signal mask to what we came in with
1709          */
1710         (void) sigprocmask(SIG_SETMASK, &oset, NULL);
```

```
1712         lx_debug("interposition handler setup for SIGPWR");
1713         return (0);
1714 }
1715 #endif /* ! codereview */
```

```
**********************************************************
   40838 Tue Jan 14 16:17:05 2014
new/usr/src/lib/brand/lx/lx_brand/common/socket.c
LX zone support should now build and packages of relevance produced.
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <libintl.h>
#include <strings.h>
#include <alloca.h>
#include <ucred.h>

#include <sys/param.h>
#include <sys/brand.h>
#include <sys/syscall.h>
#include <sys/socket.h>
#include <sys/socketvar.h>
#include <sys/un.h>
#include <netinet/tcp.h>
#include <netinet/igmp.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/lx_debug.h>
#include <sys/lx_syscall.h>
#include <sys/lx_socket.h>
#include <sys/lx_brand.h>
#include <sys/lx_misc.h>

/*
 * This string is used to prefix all abstract namespace unix sockets, ie all
 * abstract namespace sockets are converted to regular sockets in the /tmp
 * directory with .ABSK_ prefixed to their names.
 */
#define ABST_PRFX "/tmp/.ABSK_"
#define ABST_PRFX_LEN 11
```



```
static int lx_socket(ulong_t *);
static int lx_bind(ulong_t *);
static int lx_connect(ulong_t *);
static int lx_listen(ulong_t *);
static int lx_accept(ulong_t *);
static int lx_getsockname(ulong_t *);
static int lx_getpeername(ulong_t *);
static int lx_socketpair(ulong_t *);
static int lx_send(ulong_t *);
static int lx_recv(ulong_t *);
static int lx_sendto(ulong_t *);
static int lx_recvfrom(ulong_t *);
static int lx_shutdown(ulong_t *);
static int lx_setsockopt(ulong_t *);
static int lx_getsockopt(ulong_t *);
static int lx_sendmsg(ulong_t *);
static int lx_recvmsg(ulong_t *);

typedef int (*sockfn_t)(ulong_t *);

static struct {
        sockfn_t s_fn;  /* Function implementing the subcommand */
        int s_nargs;    /* Number of arguments the function takes */
} sockfns[] = {
        lx_socket, 3,
        lx_bind, 3,
        lx_connect, 3,
        lx_listen, 2,
        lx_accept, 3,
        lx_getsockname, 3,
        lx_getpeername, 3,
        lx_socketpair, 4,
        lx_send, 4,
        lx_recv, 4,
        lx_sendto, 6,
        lx_recvfrom, 6,
        lx_shutdown, 2,
        lx_setsockopt, 5,
        lx_getsockopt, 5,
        lx_sendmsg, 3,
        lx_recvmsg, 3
};

/*
 * What follows are a series of tables we use to translate Linux constants
 * into equivalent Solaris constants and back again.  I wish this were
 * cleaner, more programmatic, and generally nicer.  Sadly, life is messy,
 * and Unix networking even more so.
 */
static const int ltos_family[LX_AF_MAX + 1] =  {
        AF_UNSPEC, AF_UNIX, AF_INET, AF_CCITT, AF_IPX,
        AF_APPLETALK, AF_NOTSUPPORTED, AF_OSI, AF_NOTSUPPORTED,
        AF_X25, AF_INET6, AF_CCITT, AF_DECnet,
        AF_802, AF_POLICY, AF_KEY, AF_ROUTE,
        AF_NOTSUPPORTED, AF_NOTSUPPORTED, AF_NOTSUPPORTED, AF_NOTSUPPORTED,
        AF_NOTSUPPORTED, AF_SNA, AF_NOTSUPPORTED, AF_NOTSUPPORTED,
        AF_NOTSUPPORTED, AF_NOTSUPPORTED, AF_NOTSUPPORTED, AF_NOTSUPPORTED,
        AF_NOTSUPPORTED, AF_NOTSUPPORTED, AF_NOTSUPPORTED, AF_NOTSUPPORTED
};

#define LTOS_FAMILY(d) ((d) <= LX_AF_MAX ? ltos_family[(d)] : AF_INVAL)

static const int ltos_socktype[LX_SOCK_PACKET + 1] = {
        SOCK_NOTSUPPORTED, SOCK_STREAM, SOCK_DGRAM, SOCK_RAW,
        SOCK_RDM, SOCK_SEQPACKET, SOCK_NOTSUPPORTED, SOCK_NOTSUPPORTED,
```

```
127         SOCK_NOTSUPPORTED, SOCK_NOTSUPPORTED, SOCK_NOTSUPPORTED
128 };

130 #define LTOS_SOCKTYPE(t)        \
131         ((t) <= LX_SOCK_PACKET ? ltos_socktype[(t)] : SOCK_INVAL)

133 /*
134  * Linux socket option type definitions
135  *
136  * The protocol 'levels' are well defined (see in.h) The option values are
137  * not so well defined. Linux often uses different values to Solaris
138  * although they mean the same thing. For example, IP_TOS in Linux is
139  * defined as value 1 but in Solaris it is defined as value 3. This table
140  * maps all the Protocol levels to their options and maps them between
141  * Linux and Solaris and vice versa.  Hence the reason for the complexity.
142  */

144 typedef struct lx_proto_opts {
145         const int *proto;       /* Linux to Solaris mapping table */
146         int maxentries;         /* max entries in this table */
147 } lx_proto_opts_t;

149 #define OPTNOTSUP       -1      /* we don't support it */

151 static const int ltos_ip_sockopts[LX_IP_DROP_MEMBERSHIP + 1] = {
152         OPTNOTSUP, IP_TOS, IP_TTL, IP_HDRINCL,
153         IP_OPTIONS, OPTNOTSUP, IP_RECVOPTS, IP_RETOPTS,
154         OPTNOTSUP, OPTNOTSUP, OPTNOTSUP, OPTNOTSUP,
155         IP_RECVTTL, OPTNOTSUP, OPTNOTSUP, OPTNOTSUP,
156         OPTNOTSUP, OPTNOTSUP, OPTNOTSUP, OPTNOTSUP,
157         OPTNOTSUP, OPTNOTSUP, OPTNOTSUP, OPTNOTSUP,
158         OPTNOTSUP, OPTNOTSUP, OPTNOTSUP, OPTNOTSUP,
159         OPTNOTSUP, OPTNOTSUP, OPTNOTSUP, OPTNOTSUP,
160         IP_MULTICAST_IF, IP_MULTICAST_TTL, IP_MULTICAST_LOOP,
161         IP_ADD_MEMBERSHIP, IP_DROP_MEMBERSHIP
162 };

164 static const int ltos_tcp_sockopts[LX_TCP_QUICKACK + 1] = {
165         OPTNOTSUP, TCP_NODELAY, TCP_MAXSEG, OPTNOTSUP,
166         OPTNOTSUP, OPTNOTSUP, OPTNOTSUP, OPTNOTSUP,
167         TCP_KEEPALIVE, OPTNOTSUP, OPTNOTSUP, OPTNOTSUP,
168         OPTNOTSUP
169 };

171 static const int ltos_igmp_sockopts[IGMP_MTRACE + 1] = {
172         OPTNOTSUP, OPTNOTSUP, OPTNOTSUP, OPTNOTSUP,
173         OPTNOTSUP, OPTNOTSUP, OPTNOTSUP, OPTNOTSUP,
174         IGMP_MINLEN, OPTNOTSUP, OPTNOTSUP, /* XXX: was IGMP_TIMER_SCALE */
175         OPTNOTSUP, OPTNOTSUP, OPTNOTSUP, OPTNOTSUP,
176         OPTNOTSUP, OPTNOTSUP, IGMP_MEMBERSHIP_QUERY,
177         IGMP_V1_MEMBERSHIP_REPORT, IGMP_DVMRP,
178         IGMP_PIM, OPTNOTSUP, IGMP_V2_MEMBERSHIP_REPORT,
179         IGMP_V2_LEAVE_GROUP, OPTNOTSUP, OPTNOTSUP,
180         OPTNOTSUP, OPTNOTSUP, OPTNOTSUP, OPTNOTSUP,
181         IGMP_MTRACE_RESP, IGMP_MTRACE
182 };

184 static const int ltos_socket_sockopts[LX_SO_ACCEPTCONN + 1] = {
185         OPTNOTSUP,      SO_DEBUG,       SO_REUSEADDR,   SO_TYPE,
186         SO_ERROR,       SO_DONTROUTE,   SO_BROADCAST,   SO_SNDBUF,
187         SO_RCVBUF,      SO_KEEPALIVE,   SO_OOBINLINE,   OPTNOTSUP,
188         OPTNOTSUP,      SO_LINGER,      OPTNOTSUP,      OPTNOTSUP,
189         OPTNOTSUP,      OPTNOTSUP,      SO_RCVLOWAT,    SO_SNDLOWAT,
190         SO_RCVTIMEO,    SO_SNDTIMEO,    OPTNOTSUP,      OPTNOTSUP,
191         OPTNOTSUP,      OPTNOTSUP,      OPTNOTSUP,      OPTNOTSUP,
192         OPTNOTSUP,      OPTNOTSUP,      SO_ACCEPTCONN
```



```
193 };

195 #define PROTO_SOCKOPTS(opts)    \
196         { (opts), sizeof ((opts)) / sizeof ((opts)[0]) }

198 /*
199  * The main Linux to Solaris protocol to options mapping table
200  * IPPROTO_TAB_SIZE can be set up to IPPROTO_MAX. All entries above
201  * IPPROTO_TAB_SIZE are in effect not implemented,
202  */

204 #define IPPROTO_TAB_SIZE        8

206 static const lx_proto_opts_t ltos_proto_opts[IPPROTO_TAB_SIZE] = {
207         /* IPPROTO_IP           0 */
208         PROTO_SOCKOPTS(ltos_ip_sockopts),
209         /* SOL_SOCKET           1 */
210         PROTO_SOCKOPTS(ltos_socket_sockopts),
211         /* IPPROTO_IGMP         2 */
212         PROTO_SOCKOPTS(ltos_igmp_sockopts),
213         /* NOT IMPLEMENTED      3 */
214         { NULL, 0 },
215         /* NOT IMPLEMENTED      4 */
216         { NULL, 0 },
217         /* NOT IMPLEMENTED      5 */
218         { NULL, 0 },
219         /* IPPROTO_TCP          6 */
220         PROTO_SOCKOPTS(ltos_tcp_sockopts),
221         /* NOT IMPLEMENTED      7 */
222         { NULL, 0 }
223 };

225 /*
226  * Lifted from socket.h, since these definitions are contained within
227  * _KERNEL guards.
228  */
229 #define _CMSG_HDR_ALIGNMENT     4
230 #define _CMSG_HDR_ALIGN(x)      (((uintptr_t)(x) + _CMSG_HDR_ALIGNMENT - 1) & \
231                                     ~(_CMSG_HDR_ALIGNMENT - 1))
232 #define CMSG_FIRSTHDR(m)                                                \
233         (((m)->msg_controllen < sizeof (struct cmsghdr)) ?              \
234             (struct cmsghdr *)0 : (struct cmsghdr *)((m)->msg_control))

236 #define CMSG_NXTHDR(m, c)                                               \
237         (((c) == 0) ? CMSG_FIRSTHDR(m) :                                \
238         ((((uintptr_t)_CMSG_HDR_ALIGN((char *)(c) +                     \
239         ((struct cmsghdr *)(c))->cmsg_len) + sizeof (struct cmsghdr)) > \
240         (((uintptr_t)((struct lx_msghdr *)(m))->msg_control) +          \
241         ((uintptr_t)((struct lx_msghdr *)(m))->msg_controllen))) ?      \
242         ((struct cmsghdr *)0) :                                         \
243         ((struct cmsghdr *)_CMSG_HDR_ALIGN((char *)(c) +                \
244             ((struct cmsghdr *)(c))->cmsg_len))))

246 #define LX_TO_SOL       1
247 #define SOL_TO_LX       2

249 static int
250 convert_cmsgs(int direction, struct lx_msghdr *msg, char *caller)
251 {
252         struct cmsghdr *cmsg, *last;
253         int err = 0;

255         cmsg = CMSG_FIRSTHDR(msg);
256         while (cmsg != NULL && err == 0) {
257                 if (direction == LX_TO_SOL) {
258                         if (cmsg->cmsg_level == LX_SOL_SOCKET) {
```

```
259                                 cmsg->cmsg_level = SOL_SOCKET;
260                                 if (cmsg->cmsg_type == LX_SCM_RIGHTS)
261                                         cmsg->cmsg_type = SCM_RIGHTS;
262                                 else if (cmsg->cmsg_type == LX_SCM_CRED)
263                                         cmsg->cmsg_type = SCM_UCRED;
264                                 else
265                                         err = ENOTSUP;
266                         } else {
267                                 err = ENOTSUP;
268                         }
269                 } else {
270                         if (cmsg->cmsg_level == SOL_SOCKET) {
271                                 cmsg->cmsg_level = LX_SOL_SOCKET;
272                                 if (cmsg->cmsg_type == SCM_RIGHTS)
273                                         cmsg->cmsg_type = LX_SCM_RIGHTS;
274                                 else if (cmsg->cmsg_type == SCM_UCRED)
275                                         cmsg->cmsg_type = LX_SCM_CRED;
276                                 else
277                                         err = ENOTSUP;
278                         } else {
279                                 err = ENOTSUP;
280                         }
281                 }

283                 last = cmsg;
284                 cmsg = CMSG_NXTHDR(msg, last);
285         }
286         if (err)
287                 lx_unsupported("Unsupported socket control message in %s\n.",
288                     caller);

290         return (err);
291 }

293 /*
294  * If inaddr is an abstract namespace unix socket, this function expects addr
295  * to have enough memory to hold the expanded socket name, ie it must be of
296  * size *len + ABST_PRFX_LEN.
297  */
298 static int
299 convert_sockaddr(struct sockaddr *addr, socklen_t *len,
300         struct sockaddr *inaddr, socklen_t inlen)
301 {
302         sa_family_t family;
303         int lx_in6_len;
304         int size;
305         int i, orig_len;

307         /*
308          * Note that if the buffer at inaddr is ever smaller than inlen bytes,
309          * we may erroneously return EFAULT rather than a possible EINVAL
310          * as the copy comes before the various checks as to whether inlen
311          * is of the proper length for the socket type.
312          *
313          * This isn't an issue at present because all callers to this routine
314          * do meet that constraint.
315          */
316         if ((ssize_t)inlen < 0)
317                 return (-EINVAL);
318         if (uucopy(inaddr, addr, inlen) != 0)
319                 return (-errno);

321         family = LTOS_FAMILY(addr->sa_family);

323         switch (family) {
324                 case (sa_family_t)AF_NOTSUPPORTED:
```



```
325                         return (-EPROTONOSUPPORT);
326                 case (sa_family_t)AF_INVAL:
327                         return (-EAFNOSUPPORT);
328                 case AF_INET:
329                         size = sizeof (struct sockaddr);

331                         if (inlen < size)
332                                 return (-EINVAL);

334                         *len = size;
335                         break;

337                 case AF_INET6:
338                         /*
339                          * The Solaris sockaddr_in6 has one more 32-bit
340                          * field than the Linux version.
341                          */
342                         size = sizeof (struct sockaddr_in6);
343                         lx_in6_len = size - sizeof (uint32_t);

345                         if (inlen != lx_in6_len)
346                                 return (-EINVAL);

348                         *len = (sizeof (struct sockaddr_in6));
349                         bzero((char *)addr + lx_in6_len, sizeof (uint32_t));
350                         break;

352                 case AF_UNIX:
353                         if (inlen > sizeof (struct sockaddr_un))
354                                 return (-EINVAL);

356                         *len = inlen;

358                         /*
359                          * Linux supports abstract unix sockets, which are
360                          * simply sockets that do not exist on the file system.
361                          * These sockets are denoted by beginning the path with
362                          * a NULL character. To support these, we strip out the
363                          * leading NULL character and change the path to point
364                          * to a real place in /tmp directory, by prepending
365                          * ABST_PRFX and replacing all illegal characters with
366                          * '_'.
367                          */
368                         if (addr->sa_data[0] == '\0') {

370                                 /*
371                                  * inlen is the entire size of the sockaddr_un
372                                  * data structure, including the sun_family, so
373                                  * we need to subtract this out. We subtract
374                                  * 1 since we want to overwrite the leadin NULL
375                                  * character, and thus do not include it in the
376                                  * length.
377                                  */
378                                 orig_len = inlen - sizeof (addr->sa_family) - 1;

380                                 /*
381                                  * Since abstract paths can contain illegal
382                                  * filename characters, we simply replace these
383                                  * with '_'
384                                  */
385                                 for (i = 1; i < orig_len + 1; i++) {
386                                         if (addr->sa_data[i] == '\0' ||
387                                             addr->sa_data[i] == '/')
388                                                 addr->sa_data[i] = '_';
389                                 }
```

```
391                                 /*
392                                  * prepend ABST_PRFX to file name, minus the
393                                  * leading NULL character. This places the
394                                  * socket as a hidden file in the /tmp
395                                  * directory.
396                                  */
397                                 (void) memmove(addr->sa_data + ABST_PRFX_LEN,
398                                     addr->sa_data + 1, orig_len);
399                                 bcopy(ABST_PRFX, addr->sa_data, ABST_PRFX_LEN);

401                                 /*
402                                  * Since abstract socket paths may not be NULL
403                                  * terminated, we must explicitly NULL terminate
404                                  * our string.
405                                  */
406                                 addr->sa_data[orig_len + ABST_PRFX_LEN] = '\0';

408                                 /*
409                                  * Make len reflect the new len of our string.
410                                  * Although we removed the NULL character at the
411                                  * beginning of the string, we added a NULL
412                                  * character to the end, so the net gain in
413                                  * length is simply ABST_PRFX_LEN.
414                                  */
415                                 *len = inlen + ABST_PRFX_LEN;
416                         }
417                         break;

419                 default:
420                         *len = inlen;
421         }

423         addr->sa_family = family;
424         return (0);
425 }

427 static int
428 convert_sock_args(int in_dom, int in_type, int in_protocol, int *out_dom,
429     int *out_type)
430 {
431         int domain, type;

433         if (in_dom < 0 || in_type < 0 || in_protocol < 0)
434                 return (-EINVAL);

436         domain = LTOS_FAMILY(in_dom);
437         if (domain == AF_NOTSUPPORTED || domain == AF_UNSPEC)
438                 return (-EAFNOSUPPORT);
439         if (domain == AF_INVAL)
440                 return (-EINVAL);

442         type = LTOS_SOCKTYPE(in_type);
443         if (type == SOCK_NOTSUPPORTED)
444                 return (-ESOCKTNOSUPPORT);
445         if (type == SOCK_INVAL)
446                 return (-EINVAL);

448         /*
449          * Linux does not allow the app to specify IP Protocol for raw
450          * sockets.  Solaris does, so bail out here.
451          */
452         if (type == SOCK_RAW && in_protocol == IPPROTO_IP)
453                 return (-ESOCKTNOSUPPORT);

455         *out_dom = domain;
456         *out_type = type;
```



```
457         return (0);
458 }

460 static int
461 convert_sockflags(int lx_flags)
462 {
463         int solaris_flags = 0;

465         if (lx_flags & LX_MSG_OOB)
466                 solaris_flags |= MSG_OOB;

468         if (lx_flags & LX_MSG_PEEK)
469                 solaris_flags |= MSG_PEEK;

471         if (lx_flags & LX_MSG_DONTROUTE)
472                 solaris_flags |= MSG_DONTROUTE;

474         if (lx_flags & LX_MSG_CTRUNC)
475                 solaris_flags |= MSG_CTRUNC;

477         if (lx_flags & LX_MSG_TRUNC)
478                 solaris_flags |= MSG_TRUNC;

480         if (lx_flags & LX_MSG_WAITALL)
481                 solaris_flags |= MSG_WAITALL;

483         if (lx_flags & LX_MSG_DONTWAIT)
484                 solaris_flags |= MSG_DONTWAIT;

486         if (lx_flags & LX_MSG_EOR)
487                 solaris_flags |= MSG_EOR;

489         if (lx_flags & LX_MSG_PROXY)
490                 lx_unsupported("socket operation with MSG_PROXY flag set");

492         if (lx_flags & LX_MSG_FIN)
493                 lx_unsupported("socket operation with MSG_FIN flag set");

495         if (lx_flags & LX_MSG_SYN)
496                 lx_unsupported("socket operation with MSG_SYN flag set");

498         if (lx_flags & LX_MSG_CONFIRM)
499                 lx_unsupported("socket operation with MSG_CONFIRM set");

501         if (lx_flags & LX_MSG_RST)
502                 lx_unsupported("socket operation with MSG_RST flag set");

504         if (lx_flags & LX_MSG_MORE)
505                 lx_unsupported("socket operation with MSG_MORE flag set");

507         return (solaris_flags);
508 }

510 static int
511 lx_socket(ulong_t *args)
512 {
513         int domain;
514         int type;
515         int protocol = (int)args[2];
516         int fd;
517         int err;

519         err = convert_sock_args((int)args[0], (int)args[1], protocol,
520             &domain, &type);
521         if (err != 0)
522                 return (err);
```

```
524         lx_debug("\tsocket(%d, %d, %d)", domain, type, protocol);

526         /* Right now IPv6 sockets don't work */
527         if (domain == AF_INET6)
528                 return (-EAFNOSUPPORT);

530         /*
531          * Clients of the auditing subsystem used by CentOS 4 and 5 expect to
532          * be able to create AF_ROUTE SOCK_RAW sockets to communicate with the
533          * auditing daemons. Failure to create these sockets will cause login,
534          * ssh and useradd, amoung other programs to fail. To trick these
535          * programs into working, we convert the socket domain and type to
536          * something that we do support. Then when sendto is called on these
537          * sockets, we return an error code. See lx_sendto.
538          */
539         if (domain == AF_ROUTE && type == SOCK_RAW) {
540                 domain = AF_INET;
541                 type = SOCK_STREAM;
542                 protocol = 0;
543         }

545         fd = socket(domain, type, protocol);
546         if (fd >= 0)
547                 return (fd);

549         if (errno == EPROTONOSUPPORT)
550                 return (-ESOCKTNOSUPPORT);

552         return (-errno);
553 }

555 static int
556 lx_bind(ulong_t *args)
557 {
558         int sockfd = (int)args[0];
559         struct stat64 statbuf;
560         struct sockaddr *name, oldname;
561         socklen_t len;
562         int r, r2, ret, tmperrno;
563         int abst_sock;
564         struct stat sb;

566         if (uucopy((struct sockaddr *)args[1], &oldname,
567             sizeof (struct sockaddr)) != 0)
568                 return (-errno);

570         /*
571          * Handle Linux abstract sockets, which are UNIX sockets whose path
572          * begins with a NULL character.
573          */
574         abst_sock = (oldname.sa_family == AF_UNIX) &&
575             (oldname.sa_data[0] == '\0');

577         /*
578          * convert_sockaddr will expand the socket path if it is abstract, so
579          * we need to allocate extra memory for it now.
580          */
581         if ((name = SAFE_ALLOCA((socklen_t)args[2] +
582             abst_sock * ABST_PRFX_LEN)) == NULL)
583                 return (-EINVAL);

585         if ((r = convert_sockaddr(name, &len, (struct sockaddr *)args[1],
586             (socklen_t)args[2])) < 0)
587                 return (r);
```



```
589         /*
590          * Linux abstract namespace unix sockets are simply socket that do not
591          * exist on the filesystem. We emulate them by changing their paths
592          * in covert_sockaddr so that they point real files names on the
593          * filesystem. Because in Linux they do not exist on the filesystem
594          * applications do not have to worry about deleting files, however in
595          * our filesystem based emulation we do. To solve this problem, we first
596          * check to see if the socket already exists before we create one. If it
597          * does we attempt to connect to it to see if it is in use, or just
598          * left over from a previous lx_bind call. If we are unable to connect,
599          * we assume it is not in use and remove the file, then continue on
600          * as if the file never existed.
601          */
602         if (abst_sock && stat(name->sa_data, &sb) == 0 &&
603             S_ISSOCK(sb.st_mode)) {
604                 if ((r2 = socket(AF_UNIX, SOCK_STREAM, 0)) < 0)
605                         return (-ENOSR);
606                 ret = connect(r2, name, len);
607                 tmperrno = errno;
608                 if (close(r2) < 0)
609                         return (-EINVAL);

611                 /*
612                  * if we can't connect to the socket, assume no one is using it
613                  * and remove it, otherwise assume it is in use and return
614                  * EADDRINUSE.
615                  */
616                 if ((ret < 0) && (tmperrno == ECONNREFUSED)) {
617                         if (unlink(name->sa_data) < 0) {
618                                 return (-EADDRINUSE);
619                         }
620                 } else {
621                         return (-EADDRINUSE);
622                 }
623         }

625         lx_debug("\tbind(%d, 0x%p, %d)", sockfd, name, len);

627         if (name->sa_family == AF_UNIX)
628                 lx_debug("\t\tAF_UNIX, path = %s", name->sa_data);

630         r = bind(sockfd, name, len);

632         /*
633          * Linux returns EADDRINUSE for attempts to bind to UNIX domain
634          * sockets that aren't sockets.
635          */
636         if ((r < 0) && (errno == EINVAL) && (name->sa_family == AF_UNIX) &&
637             ((stat64(name->sa_data, &statbuf) == 0) &&
638             (!S_ISSOCK(statbuf.st_mode))))
639                 return (-EADDRINUSE);

641         return ((r < 0) ? -errno : r);
642 }

644 static int
645 lx_connect(ulong_t *args)
646 {
647         int sockfd = (int)args[0];
648         struct sockaddr *name, oldname;
649         socklen_t len;
650         int r;
651         int abst_sock;

653         if (uucopy((struct sockaddr *)args[1], &oldname,
654             sizeof (struct sockaddr)) != 0)
```

```
655                 return (-errno);


658         /* Handle Linux abstract sockets */
659         abst_sock = (oldname.sa_family == AF_UNIX) &&
660             (oldname.sa_data[0] == '\0');

662         /*
663          * convert_sockaddr will expand the socket path, if it is abstract, so
664          * we need to allocate extra memory for it now.
665          */
666         if ((name = SAFE_ALLOCA((socklen_t)args[2] +
667             abst_sock * ABST_PRFX_LEN)) == NULL)
668                 return (-EINVAL);

670         if ((r = convert_sockaddr(name, &len, (struct sockaddr *)args[1],
671             (socklen_t)args[2])) < 0)
672                 return (r);

674         lx_debug("\tconnect(%d, 0x%p, %d)", sockfd, name, len);

676         if (name->sa_family == AF_UNIX)
677                 lx_debug("\t\tAF_UNIX, path = %s", name->sa_data);

679         r = connect(sockfd, name, len);

681         return ((r < 0) ? -errno : r);
682 }

684 static int
685 lx_listen(ulong_t *args)
686 {
687         int sockfd = (int)args[0];
688         int backlog = (int)args[1];
689         int r;

691         lx_debug("\tlisten(%d, %d)", sockfd, backlog);
692         r = listen(sockfd, backlog);

694         return ((r < 0) ? -errno : r);
695 }

697 static int
698 lx_accept(ulong_t *args)
699 {
700         int sockfd = (int)args[0];
701         struct sockaddr *name = (struct sockaddr *)args[1];
702         socklen_t namelen = 0;
703         int r;

705         lx_debug("\taccept(%d, 0x%p, 0x%p", sockfd, args[1], args[2]);

707         /*
708          * The Linux man page says that -1 is returned and errno is set to
709          * EFAULT if the "name" address is bad, but it is silent on what to
710          * set errno to if the "namelen" address is bad.  Experimentation
711          * shows that Linux (at least the 2.4.21 kernel in CentOS) actually
712          * sets errno to EINVAL in both cases.
713          *
714          * Note that we must first check the name pointer, as the Linux
715          * docs state nothing is copied out if the "name" pointer is NULL.
716          * If it is NULL, we don't care about the namelen pointer's value
717          * or about dereferencing it.
718          *
719          * Happily, Solaris' accept(3SOCKET) treats NULL name pointers and
720          * zero namelens the same way.
```



```
721          */
722         if ((name != NULL) &&
723             (uucopy((void *)args[2], &namelen, sizeof (socklen_t)) != 0))
724                 return ((errno == EFAULT) ? -EINVAL : -errno);

726         lx_debug("\taccept namelen = %d", namelen);

728         if ((r = accept(sockfd, name, &namelen)) < 0)
729                 return ((errno == EFAULT) ? -EINVAL : -errno);

731         lx_debug("\taccept namelen returned %d bytes", namelen);

733         /*
734          * In Linux, accept()ed sockets do not inherit anything set by
735          * fcntl(), so filter those out.
736          */
737         if (fcntl(r, F_SETFL, 0) < 0)
738                 return (-errno);

740         /*
741          * Once again, a bad "namelen" address sets errno to EINVAL, not
742          * EFAULT.  If namelen was zero, there's no need to copy a zero back
743          * out.
744          *
745          * Logic might dictate that we should check if we can write to
746          * the namelen pointer earlier so we don't accept a pending connection
747          * only to fail the call because we can't write the namelen value back
748          * out. However, testing shows Linux does indeed fail the call after
749          * accepting the connection so we must behave in a compatible manner.
750          */
751         if ((name != NULL) && (namelen != 0) &&
752             (uucopy(&namelen, (void *)args[2], sizeof (socklen_t)) != 0))
753                 return ((errno == EFAULT) ? -EINVAL : -errno);

755         return (r);
756 }

758 static int
759 lx_getsockname(ulong_t *args)
760 {
761         int sockfd = (int)args[0];
762         struct sockaddr *name = NULL;
763         socklen_t namelen, namelen_orig;

765         if (uucopy((void *)args[2], &namelen, sizeof (socklen_t)) != 0)
766                 return (-errno);
767         namelen_orig = namelen;

769         lx_debug("\tgetsockname(%d, 0x%p, 0x%p (=%d))",
770             sockfd, args[1], args[2], namelen);

772         if (namelen > 0) {
773                 if ((name = SAFE_ALLOCA(namelen)) == NULL)
774                         return (-EINVAL);
775                 bzero(name, namelen);
776         }

778         if ((getsockname(sockfd, name, &namelen)) < 0)
779                 return (-errno);

781         /*
782          * If the name that getsockname() want's to return is larger
783          * than namelen, getsockname() will copy out the maximum amount
784          * of data possible and then update namelen to indicate the
785          * actually size of all the data that it wanted to copy out.
786          */
```

```
787         if (uucopy(name, (void *)args[1], namelen_orig) != 0)
788                 return (-errno);
789         if (uucopy(&namelen, (void *)args[2], sizeof (socklen_t)) != 0)
790                 return (-errno);
791
792         return (0);
793 }
794
795 static int
796 lx_getpeername(ulong_t *args)
797 {
798         int sockfd = (int)args[0];
799         struct sockaddr *name;
800         socklen_t namelen;
801
802         if (uucopy((void *)args[2], &namelen, sizeof (socklen_t)) != 0)
803                 return (-errno);
804
805         lx_debug("\tgetpeername(%d, 0x%p, 0x%p (=%d))",
806             sockfd, args[1], args[2], namelen);
807
808         /*
809          * Linux returns EFAULT in this case, even if the namelen parameter
810          * is 0.  This check will not catch other illegal addresses, but
811          * the benefit catching a non-null illegal address here is not
812          * worth the cost of another system call.
813          */
814         if ((void *)args[1] == NULL)
815                 return (-EFAULT);
816
817         if ((name = SAFE_ALLOCA(namelen)) == NULL)
818                 return (-EINVAL);
819         if ((getpeername(sockfd, name, &namelen)) < 0)
820                 return (-errno);
821
822         if (uucopy(name, (void *)args[1], namelen) != 0)
823                 return (-errno);
824
825         if (uucopy(&namelen, (void *)args[2], sizeof (socklen_t)) != 0)
826                 return (-errno);
827
828         return (0);
829 }
830
831 static int
832 lx_socketpair(ulong_t *args)
833 {
834         int domain;
835         int type;
836         int protocol = (int)args[2];
837         int *sv = (int *)args[3];
838         int fds[2];
839         int r;
840
841         r = convert_sock_args((int)args[0], (int)args[1], protocol,
842             &domain, &type);
843         if (r != 0)
844                 return (r);
845
846         lx_debug("\tsocketpair(%d, %d, %d, 0x%p)", domain, type, protocol, sv);
847
848         r = socketpair(domain, type, protocol, fds);
849
850         if (r == 0) {
851                 if (uucopy(fds, sv, sizeof (fds)) != 0) {
852                         r = errno;
```



```
853                         (void) close(fds[0]);
854                         (void) close(fds[1]);
855                         return (-r);
856                 }
857                 return (0);
858         }
859
860         if (errno == EPROTONOSUPPORT)
861                 return (-ESOCKTNOSUPPORT);
862
863         return (-errno);
864 }
865
866 static ssize_t
867 lx_send(ulong_t *args)
868 {
869         int sockfd = (int)args[0];
870         void *buf = (void *)args[1];
871         size_t len = (size_t)args[2];
872         int flags = (int)args[3];
873         ssize_t r;
874
875         int nosigpipe = flags & LX_MSG_NOSIGNAL;
876         struct sigaction newact, oact;
877
878         lx_debug("\tsend(%d, 0x%p, 0x%d, 0x%x)", sockfd, buf, len, flags);
879
880         flags = convert_sockflags(flags);
881
882         /*
883          * If nosigpipe is set, we want to emulate the Linux action of
884          * not sending a SIGPIPE to the caller if the remote socket has
885          * already been closed.
886          *
887          * As SIGPIPE is a directed signal sent only to the thread that
888          * performed the action, we can emulate this behavior by momentarily
889          * resetting the action for SIGPIPE to SIG_IGN, performing the socket
890          * call, and resetting the action back to its previous value.
891          */
892         if (nosigpipe) {
893                 newact.sa_handler = SIG_IGN;
894                 newact.sa_flags = 0;
895                 (void) sigemptyset(&newact.sa_mask);
896
897                 if (sigaction(SIGPIPE, &newact, &oact) < 0)
898                         lx_err_fatal(gettext(
899                             "%s: could not ignore SIGPIPE to emulate "
900                             "LX_MSG_NOSIGNAL"), "send()");
901         }
902
903         r = send(sockfd, buf, len, flags);
904
905         if ((nosigpipe) && (sigaction(SIGPIPE, &oact, NULL) < 0))
906                 lx_err_fatal(
907                     gettext("%s: could not reset SIGPIPE handler to "
908                     "emulate LX_MSG_NOSIGNAL"), "send()");
909
910         return ((r < 0) ? -errno : r);
911 }
912
913 static ssize_t
914 lx_recv(ulong_t *args)
915 {
916         int sockfd = (int)args[0];
917         void *buf = (void *)args[1];
918         size_t len = (size_t)args[2];
```

```
        int flags = (int)args[3];
        ssize_t r;

        int nosigpipe = flags & LX_MSG_NOSIGNAL;
        struct sigaction newact, oact;

        lx_debug("\trecv(%d, 0x%p, 0x%d, 0x%x)", sockfd, buf, len, flags);

        flags = convert_sockflags(flags);

        /*
         * If nosigpipe is set, we want to emulate the Linux action of
         * not sending a SIGPIPE to the caller if the remote socket has
         * already been closed.
         *
         * As SIGPIPE is a directed signal sent only to the thread that
         * performed the action, we can emulate this behavior by momentarily
         * resetting the action for SIGPIPE to SIG_IGN, performing the socket
         * call, and resetting the action back to its previous value.
         */
        if (nosigpipe) {
                newact.sa_handler = SIG_IGN;
                newact.sa_flags = 0;
                (void) sigemptyset(&newact.sa_mask);

                if (sigaction(SIGPIPE, &newact, &oact) < 0)
                        lx_err_fatal(gettext(
                            "%s: could not ignore SIGPIPE to emulate "
                            "LX_MSG_NOSIGNAL"), "recv()");
        }

        r = recv(sockfd, buf, len, flags);

        if ((nosigpipe) && (sigaction(SIGPIPE, &oact, NULL) < 0))
                lx_err_fatal(
                    gettext("%s: could not reset SIGPIPE handler to "
                    "emulate LX_MSG_NOSIGNAL"), "recv()");

        return ((r < 0) ? -errno : r);
}

static ssize_t
lx_sendto(ulong_t *args)
{
        int sockfd = (int)args[0];
        void *buf = (void *)args[1];
        size_t len = (size_t)args[2];
        int flags = (int)args[3];
        struct sockaddr *to = NULL, oldto;
        socklen_t tolen = 0;
        ssize_t r;
        int abst_sock;

        int nosigpipe = flags & LX_MSG_NOSIGNAL;
        struct sigaction newact, oact;

        if ((args[4] != NULL) && (args[5] > 0)) {
                if (uucopy((struct sockaddr *)args[4], &oldto,
                    sizeof (struct sockaddr)) != 0)
                        return (-errno);

                /* Handle Linux abstract sockets */
                abst_sock = (oldto.sa_family == AF_UNIX) &&
                    (oldto.sa_data[0] == '\0');

                /*
```



```
                 * convert_sockaddr will expand the socket path, if it is
                 * abstract, so we need to allocate extra memory for it now.
                 */
                if ((to = SAFE_ALLOCA(args[5] + abst_sock * ABST_PRFX_LEN))
                    == NULL)
                        return (-EINVAL);

                if ((r = convert_sockaddr(to, &tolen,
                    (struct sockaddr *)args[4], (socklen_t)args[5])) < 0)
                        return (r);
        }


        lx_debug("\tsendto(%d, 0x%p, 0x%d, 0x%x, 0x%x, %d)", sockfd, buf, len,
            flags, to, tolen);

        flags = convert_sockflags(flags);

        /* return this error to make auditing subsystem happy */
        if (to && to->sa_family == AF_ROUTE) {
                return (-ECONNREFUSED);
        }

        /*
         * If nosigpipe is set, we want to emulate the Linux action of
         * not sending a SIGPIPE to the caller if the remote socket has
         * already been closed.
         *
         * As SIGPIPE is a directed signal sent only to the thread that
         * performed the action, we can emulate this behavior by momentarily
         * resetting the action for SIGPIPE to SIG_IGN, performing the socket
         * call, and resetting the action back to its previous value.
         */
        if (nosigpipe) {
                newact.sa_handler = SIG_IGN;
                newact.sa_flags = 0;
                (void) sigemptyset(&newact.sa_mask);

                if (sigaction(SIGPIPE, &newact, &oact) < 0)
                        lx_err_fatal(gettext(
                            "%s: could not ignore SIGPIPE to emulate "
                            "LX_MSG_NOSIGNAL"), "sendto()");
        }

        r = sendto(sockfd, buf, len, flags, to, tolen);

        if ((nosigpipe) && (sigaction(SIGPIPE, &oact, NULL) < 0))
                lx_err_fatal(
                    gettext("%s: could not reset SIGPIPE handler to "
                    "emulate LX_MSG_NOSIGNAL"), "sendto()");

        if (r < 0) {
                /*
                 * according to the man page and LTP, the expected error in
                 * this case is EPIPE.
                 */
                if (errno == ENOTCONN)
                        return (-EPIPE);
                else
                        return (-errno);
        }
        return (r);
}

static ssize_t
lx_recvfrom(ulong_t *args)
```

```
1051 {
1052         int sockfd = (int)args[0];
1053         void *buf = (void *)args[1];
1054         size_t len = (size_t)args[2];
1055         int flags = (int)args[3];
1056         struct sockaddr *from = (struct sockaddr *)args[4];
1057         socklen_t *from_lenp = (socklen_t *)args[5];
1058         ssize_t r;

1060         int nosigpipe = flags & LX_MSG_NOSIGNAL;
1061         struct sigaction newact, oact;

1063         lx_debug("\trecvfrom(%d, 0x%p, 0x%d, 0x%x, 0x%x, 0x%p)", sockfd, buf,
1064             len, flags, from, from_lenp);

1066         flags = convert_sockflags(flags);

1068         /*
1069          * If nosigpipe is set, we want to emulate the Linux action of
1070          * not sending a SIGPIPE to the caller if the remote socket has
1071          * already been closed.
1072          *
1073          * As SIGPIPE is a directed signal sent only to the thread that
1074          * performed the action, we can emulate this behavior by momentarily
1075          * resetting the action for SIGPIPE to SIG_IGN, performing the socket
1076          * call, and resetting the action back to its previous value.
1077          */
1078         if (nosigpipe) {
1079                 newact.sa_handler = SIG_IGN;
1080                 newact.sa_flags = 0;
1081                 (void) sigemptyset(&newact.sa_mask);

1083                 if (sigaction(SIGPIPE, &newact, &oact) < 0)
1084                         lx_err_fatal(gettext(
1085                             "%s: could not ignore SIGPIPE to emulate "
1086                             "LX_MSG_NOSIGNAL"), "recvfrom()");
1087         }

1089         r = recvfrom(sockfd, buf, len, flags, from, from_lenp);

1091         if ((nosigpipe) && (sigaction(SIGPIPE, &oact, NULL) < 0))
1092                 lx_err_fatal(
1093                     gettext("%s: could not reset SIGPIPE handler to "
1094                     "emulate LX_MSG_NOSIGNAL"), "recvfrom()");

1096         return ((r < 0) ? -errno : r);
1097 }

1099 static int
1100 lx_shutdown(ulong_t *args)
1101 {
1102         int sockfd = (int)args[0];
1103         int how = (int)args[1];
1104         int r;

1106         lx_debug("\tshutdown(%d, %d)", sockfd, how);
1107         r = shutdown(sockfd, how);

1109         return ((r < 0) ? -errno : r);
1110 }

1112 static int
1113 lx_setsockopt(ulong_t *args)
1114 {
1115         int sockfd = (int)args[0];
1116         int level = (int)args[1];
```



```
1117         int optname = (int)args[2];
1118         void *optval = (void *)args[3];
1119         int optlen = (int)args[4];
1120         int internal_opt;
1121         int r;

1123         lx_debug("\tsetsockopt(%d, %d, %d, 0x%p, %d)", sockfd, level, optname,
1124             optval, optlen);

1126         /*
1127          * The kernel returns EFAULT for all invalid addresses except NULL,
1128          * for which it returns EINVAL.  Linux wants EFAULT for NULL too.
1129          */
1130         if (optval == NULL)
1131                 return (-EFAULT);

1133         /*
1134          * Do a table lookup of the Solaris equivalent of the given option
1135          */
1136         if (level < IPPROTO_IP || level >= IPPROTO_TAB_SIZE)
1137                 return (-ENOPROTOOPT);

1139         if (ltos_proto_opts[level].maxentries == 0 ||
1140             optname <= 0 || optname >= (ltos_proto_opts[level].maxentries))
1141                 return (-ENOPROTOOPT);

1143         /*
1144          * Linux sets this option when it wants to send credentials over a
1145          * socket. Currently we just ignore it to make Linux programs happy.
1146          */
1147         if ((level == LX_SOL_SOCKET) && (optname == LX_SO_PASSCRED))
1148                 return (0);


1151         if ((level == IPPROTO_TCP) && (optname == LX_TCP_CORK)) {
1152                 /*
1153                  * TCP_CORK is a Linux-only option that instructs the TCP
1154                  * stack not to send out partial frames.  Solaris doesn't
1155                  * include this option but some apps require it.  So, we do
1156                  * our best to emulate the option by disabling TCP_NODELAY.
1157                  * If the app requests that we disable TCP_CORK, we just
1158                  * ignore it since enabling TCP_NODELAY may be
1159                  * overcompensating.
1160                  */
1161                 optname = TCP_NODELAY;
1162                 if (optlen != sizeof (int))
1163                         return (-EINVAL);
1164                 if (uucopy(optval, &internal_opt, sizeof (int)) != 0)
1165                         return (-errno);
1166                 if (internal_opt == 0)
1167                         return (0);
1168                 internal_opt = 1;
1169                 optval = &internal_opt;
1170         } else {
1171                 optname = ltos_proto_opts[level].proto[optname];

1173                 if (optname == OPTNOTSUP)
1174                         return (-ENOPROTOOPT);
1175         }

1177         if (level == LX_SOL_SOCKET)
1178                 level = SOL_SOCKET;

1180         r = setsockopt(sockfd, level, optname, optval, optlen);

1182         return ((r < 0) ? -errno : r);
```

```
1183 }
1185 static int
1186 lx_getsockopt(ulong_t *args)
1187 {
1188         int sockfd = (int)args[0];
1189         int level = (int)args[1];
1190         int optname = (int)args[2];
1191         void *optval = (void *)args[3];
1192         int *optlenp = (int *)args[4];
1193         int r;
1195         lx_debug("\tgetsockopt(%d, %d, %d, 0x%p, 0x%p)", sockfd, level, optname,
1196             optval, optlenp);
1198         /*
1199          * According to the Linux man page, a NULL optval should indicate
1200          * (as in Solaris) that no return value is expected.  Instead, it
1201          * actually triggers an EFAULT error.
1202          */
1203         if (optval == NULL)
1204                 return (-EFAULT);
1206         /*
1207          * Do a table lookup of the Solaris equivalent of the given option
1208          */
1209         if (level < IPPROTO_IP || level >= IPPROTO_TAB_SIZE)
1210                 return (-EOPNOTSUPP);
1212         if (ltos_proto_opts[level].maxentries == 0 ||
1213             optname <= 0 || optname >= (ltos_proto_opts[level].maxentries))
1214                 return (-ENOPROTOOPT);
1216         if (((level == LX_SOL_SOCKET) && (optname == LX_SO_PASSCRED)) ||
1217             ((level == IPPROTO_TCP) && (optname == LX_TCP_CORK))) {
1218                 /*
1219                  * Linux sets LX_SO_PASSCRED when it wants to send credentials
1220                  * over a socket. Since we do not support it, it is never set
1221                  * and we return 0.
1222                  *
1223                  * We don't support TCP_CORK but some apps rely on it.  So,
1224                  * rather than return an error we just return 0.  This
1225                  * isn't exactly a lie, since this option really isn't set,
1226                  * but it's not the whole truth either.  Fortunately, we
1227                  * aren't under oath.
1228                  */
1229                 r = 0;
1230                 if (uucopy(&r, optval, sizeof (int)) != 0)
1231                         return (-errno);
1232                 r = sizeof (int);
1233                 if (uucopy(&r, optlenp, sizeof (int)) != 0)
1234                         return (-errno);
1235                 return (0);
1236         }
1237         if ((level == LX_SOL_SOCKET) && (optname == LX_SO_PEERCRED)) {
1238                 struct lx_ucred lx_ucred;
1239                 ucred_t         *ucp;
1241                 /*
1242                  * We don't support SO_PEERCRED, but we do have equivalent
1243                  * functionality in getpeerucred() so invoke that here.
1244                  */
1246                 /* Verify there's going to be enough room for the results. */
1247                 if (uucopy(optlenp, &r, sizeof (int)) != 0)
1248                         return (-errno);
```



```
1249                 if (r < sizeof (struct lx_ucred))
1250                         return (-EOVERFLOW);
1252                 /*
1253                  * We allocate a ucred_t ourselves rather than allow
1254                  * getpeerucred() to do it for us because getpeerucred()
1255                  * uses malloc(3C) and we'd rather use SAFE_ALLOCA().
1256                  */
1257                 if ((ucp = (ucred_t *)SAFE_ALLOCA(ucred_size())) == NULL)
1258                         return (-ENOMEM);
1260                 /* Get the credential for the remote end of this socket. */
1261                 if (getpeerucred(sockfd, &ucp) != 0)
1262                         return (-errno);
1263                 if (((lx_ucred.lxu_pid = ucred_getpid(ucp)) == -1) ||
1264                     ((lx_ucred.lxu_uid = ucred_geteuid(ucp)) == (uid_t)-1) ||
1265                     ((lx_ucred.lxu_gid = ucred_getegid(ucp)) == (gid_t)-1)) {
1266                         return (-errno);
1267                 }
1269                 /* Copy out the results. */
1270                 if ((uucopy(&lx_ucred, optval, sizeof (lx_ucred))) != 0)
1271                         return (-errno);
1272                 r = sizeof (lx_ucred);
1273                 if ((uucopy(&r, optlenp, sizeof (int))) != 0)
1274                         return (-errno);
1275                 return (0);
1276         }
1278         optname = ltos_proto_opts[level].proto[optname];
1280         if (optname == OPTNOTSUP)
1281                 return (-ENOPROTOOPT);
1283         if (level == LX_SOL_SOCKET)
1284                 level = SOL_SOCKET;
1286         r = getsockopt(sockfd, level, optname, optval, optlenp);
1288         return ((r < 0) ? -errno : r);
1289 }
1291 /*
1292  * libc routines that issue these system calls.  We bypass the libsocket
1293  * wrappers since they explicitly turn off the MSG_XPG_2 flag we need for
1294  * Linux compatibility.
1295  */
1296 extern int _so_sendmsg();
1297 extern int _so_recvmsg();
1299 static int
1300 lx_sendmsg(ulong_t *args)
1301 {
1302         int sockfd = (int)args[0];
1303         struct lx_msghdr msg;
1304         struct cmsghdr *cmsg;
1305         int flags = (int)args[2];
1306         int r;
1308         int nosigpipe = flags & LX_MSG_NOSIGNAL;
1309         struct sigaction newact, oact;
1311         lx_debug("\tsendmsg(%d, 0x%p, 0x%x)", sockfd, (void *)args[1], flags);
1313         flags = convert_sockflags(flags);
```

```
1315         if ((uucopy((void *)args[1], &msg, sizeof (msg))) != 0)
1316                 return (-errno);
1317
1318         /*
1319          * If there are control messages bundled in this message, we need
1320          * to convert them from Linux to Solaris.
1321          */
1322         if (msg.msg_control != NULL) {
1323                 if (msg.msg_controllen == 0) {
1324                         cmsg = NULL;
1325                 } else {
1326                         cmsg = SAFE_ALLOCA(msg.msg_controllen);
1327                         if (cmsg == NULL)
1328                                 return (-EINVAL);
1329                 }
1330                 if ((uucopy(msg.msg_control, cmsg, msg.msg_controllen)) != 0)
1331                         return (-errno);
1332                 msg.msg_control = cmsg;
1333                 if ((r = convert_cmsgs(LX_TO_SOL, &msg, "sendmsg()")) != 0)
1334                         return (-r);
1335         }
1336
1337         /*
1338          * If nosigpipe is set, we want to emulate the Linux action of
1339          * not sending a SIGPIPE to the caller if the remote socket has
1340          * already been closed.
1341          *
1342          * As SIGPIPE is a directed signal sent only to the thread that
1343          * performed the action, we can emulate this behavior by momentarily
1344          * resetting the action for SIGPIPE to SIG_IGN, performing the socket
1345          * call, and resetting the action back to its previous value.
1346          */
1347         if (nosigpipe) {
1348                 newact.sa_handler = SIG_IGN;
1349                 newact.sa_flags = 0;
1350                 (void) sigemptyset(&newact.sa_mask);
1351
1352                 if (sigaction(SIGPIPE, &newact, &oact) < 0)
1353                         lx_err_fatal(gettext(
1354                             "%s: could not ignore SIGPIPE to emulate "
1355                             "LX_MSG_NOSIGNAL"), "sendmsg()");
1356         }
1357
1358         r = _so_sendmsg(sockfd, (struct msghdr *)&msg, flags | MSG_XPG4_2);
1359
1360         if ((nosigpipe) && (sigaction(SIGPIPE, &oact, NULL) < 0))
1361                 lx_err_fatal(
1362                     gettext("%s: could not reset SIGPIPE handler to "
1363                     "emulate LX_MSG_NOSIGNAL"), "sendmsg()");
1364
1365         if (r < 0) {
1366                 /*
1367                  * according to the man page and LTP, the expected error in
1368                  * this case is EPIPE.
1369                  */
1370                 if (errno == ENOTCONN)
1371                         return (-EPIPE);
1372                 else
1373                         return (-errno);
1374         }
1375
1376         return (r);
1377 }
1378
1379 static int
1380 lx_recvmsg(ulong_t *args)
```



```
1381 {
1382         int sockfd = (int)args[0];
1383         struct lx_msghdr msg;
1384         struct lx_msghdr *msgp = (struct lx_msghdr *)args[1];
1385         struct cmsghdr *cmsg = NULL;
1386         int flags = (int)args[2];
1387         int r, err;
1388
1389         int nosigpipe = flags & LX_MSG_NOSIGNAL;
1390         struct sigaction newact, oact;
1391
1392         lx_debug("\trecvmsg(%d, 0x%p, 0x%x)", sockfd, (void *)args[1], flags);
1393
1394         flags = convert_sockflags(flags);
1395
1396         if ((uucopy(msgp, &msg, sizeof (msg))) != 0)
1397                 return (-errno);
1398
1399         /*
1400          * If we are expecting to have to convert any control messages,
1401          * then we should receive them into our address space instead of
1402          * the app's.
1403          */
1404         if (msg.msg_control != NULL) {
1405                 cmsg = msg.msg_control;
1406                 if (msg.msg_controllen == 0) {
1407                         msg.msg_control = NULL;
1408                 } else {
1409                         msg.msg_control = SAFE_ALLOCA(msg.msg_controllen);
1410                         if (msg.msg_control == NULL)
1411                                 return (-EINVAL);
1412                 }
1413         }
1414
1415         /*
1416          * If nosigpipe is set, we want to emulate the Linux action of
1417          * not sending a SIGPIPE to the caller if the remote socket has
1418          * already been closed.
1419          *
1420          * As SIGPIPE is a directed signal sent only to the thread that
1421          * performed the action, we can emulate this behavior by momentarily
1422          * resetting the action for SIGPIPE to SIG_IGN, performing the socket
1423          * call, and resetting the action back to its previous value.
1424          */
1425         if (nosigpipe) {
1426                 newact.sa_handler = SIG_IGN;
1427                 newact.sa_flags = 0;
1428                 (void) sigemptyset(&newact.sa_mask);
1429
1430                 if (sigaction(SIGPIPE, &newact, &oact) < 0)
1431                         lx_err_fatal(gettext(
1432                             "%s: could not ignore SIGPIPE to emulate "
1433                             "LX_MSG_NOSIGNAL"), "recvmsg()");
1434         }
1435
1436         r = _so_recvmsg(sockfd, (struct msghdr *)&msg, flags | MSG_XPG4_2);
1437
1438         if ((nosigpipe) && (sigaction(SIGPIPE, &oact, NULL) < 0))
1439                 lx_err_fatal(
1440                     gettext("%s: could not reset SIGPIPE handler to "
1441                     "emulate LX_MSG_NOSIGNAL"), "recvmsg()");
1442
1443         if (r >= 0 && msg.msg_control != NULL) {
1444                 /*
1445                  * If there are control messages bundled in this message,
1446                  * we need to convert them from Linux to Solaris.
```

```
1447                  */
1448                 if ((err = convert_cmsgs(SOL_TO_LX, &msg, "recvmsg()")) != 0)
1449                         return (-err);

1451                 if ((uucopy(msg.msg_control, cmsg, msg.msg_controllen)) != 0)
1452                         return (-errno);
1453         }

1455         /*
1456          * A handful of the values in the msghdr are set by the recvmsg()
1457          * call, so copy their values back to the caller.  Rather than iterate,
1458          * just copy the whole structure back.
1459          */
1460         if (uucopy(&msg, msgp, sizeof (msg)) != 0)
1461                 return (-errno);

1463         return ((r < 0) ? -errno : r);
1464 }

1466 int
1467 lx_socketcall(uintptr_t p1, uintptr_t p2)
1468 {
1469         int subcmd = (int)p1 - 1; /* subcommands start at 1 - not 0 */
1470         ulong_t args[6];
1471         int r;

1473         if (subcmd < 0 || subcmd >= LX_RECVMSG)
1474                 return (-EINVAL);

1476         /*
1477          * Copy the arguments to the subcommand in from the app's address
1478          * space, returning EFAULT if we get a bogus pointer.
1479          */
1480         if (uucopy((void *)p2, args,
1481             sockfns[subcmd].s_nargs * sizeof (ulong_t)))
1482                 return (-errno);

1484         r = (sockfns[subcmd].s_fn)(args);

1486         return (r);
1487 }
1488 #endif /* ! codereview */
```

```
**********************************************************
   13443 Tue Jan 14 16:17:05 2014
new/usr/src/lib/brand/lx/lx_brand/common/stat.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * when a stat() is done for a non-device file, the devt returned
 * via the stat is the devt of the device backing the filesystem which
 * contains the file the stat was performed on.  these devts are currently
 * untranslated.  if this turns out to cause problems in the future then
 * we might want to add more devt translators to convert sd and cmdk
 * devts into linux devts that normally represent disks.
 *
 * XXX this may not be the best place to have the devt translation code.
 * devt translation will also be needed for /proc fs support, which will
 * probably be done in the kernel.  we may need to move this code into
 * the kernel and add a brand syscall to do the translation for us.  this
 * will need to be worked out before putback.
 */

#include <assert.h>
#include <errno.h>
#include <stdio.h>
#include <strings.h>
#include <unistd.h>
#include <libintl.h>
#include <sys/fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/lx_types.h>
#include <sys/lx_stat.h>
#include <sys/lx_misc.h>
#include <sys/lx_debug.h>
#include <sys/lx_ptm.h>
#include <sys/lx_audio.h>
#include <sys/lx_fcntl.h>
#include <sys/modctl.h>

/* define _KERNEL to get the devt manipulation macros */
#define _KERNEL
#include <sys/sysmacros.h>
```



```
#undef  _KERNEL


#define LX_PTS_MAJOR_MIN        136
#define LX_PTS_MAJOR_MAX        143
#define LX_PTS_MAX              \
        ((LX_PTS_MAJOR_MAX - LX_PTS_MAJOR_MIN + 1) * LX_MINORMASK)

#define LX_PTM_MAJOR            5
#define LX_PTM_MINOR            2

/* values for dt_type */
#define DTT_INVALID     0
#define DTT_LIST        1
#define DTT_CUSTOM      2

/* convience macros for access the dt_minor union */
#define dt_list         dt_minor.dtm_list
#define dt_custom       dt_minor.dtm_custom

/*
 * structure used to define devt translators
 */
typedef struct minor_translator {
        char    *mt_path;       /* solaris minor node path */
        minor_t mt_minor;       /* solaris minor node number */
        int     mt_lx_major;    /* linux major node number */
        int     mt_lx_minor;    /* linux minor node number */
} minor_translator_t;

typedef struct devt_translator {
        char                            *dt_driver; /* solaris driver name */
        major_t                         dt_major; /* solaris driver number */

        /* dt_type dictates how we intrepret dt_minor */
        int                             dt_type;
        union {
                uintptr_t               dtm_foo; /* required to compile */
                minor_translator_t      *dtm_list;
                int                     (*dtm_custom)(dev_t, lx_dev_t *, int);
        } dt_minor;
} devt_translator_t;


/*
 * forward declerations
 */
static devt_translator_t devt_translators[];

/*
 * called to initialize the devt translation subsystem
 */
int
lx_stat_init()
{
        minor_translator_t      *mt;
        struct stat             st;
        major_t                 major;
        char                    *driver;
        int                     i, j, ret;

        for (i = 0; devt_translators[i].dt_driver != NULL; i++) {

                assert(devt_translators[i].dt_type != DTT_INVALID);

                /* figure out the major numbers for our devt translators */
```

```
128                 driver = devt_translators[i].dt_driver;
129                 ret = modctl(MODGETMAJBIND,
130                     driver, strlen(driver) + 1, &major);
131                 if (ret != 0) {
132                         lx_err(gettext("%s%s) failed: %s\n"),
133                             "lx_stat_init(): modctl(MODGETMAJBIND, ",
134                             driver, strerror(errno));
135                         lx_err(gettext("%s: %s translator disabled for: %s\n"),
136                             "lx_stat_init()", "devt", driver);
137                         devt_translators[i].dt_major = (major_t)-1;
138                         continue;
139                 }

141                 /* save the major node value */
142                 devt_translators[i].dt_major = major;

144                 /* if this translator doesn't use a list mapping we're done. */
145                 if (devt_translators[i].dt_type != DTT_LIST)
146                         continue;

148                 /* for each device listed, lookup the minor node number */
149                 mt = devt_translators[i].dt_list;
150                 for (j = 0; mt[j].mt_path != NULL; j++) {

152                         /* stat the device */
153                         ret = stat(mt[j].mt_path, &st);
154                         if (ret != 0) {
155                                 lx_err(gettext("%s%s) failed: %s\n"),
156                                     "lx_stat_init(): stat(",
157                                     mt[j].mt_path, strerror(errno));
158                                 lx_err(gettext(
159                                     "%s: %s translator disabled for: %s\n"),
160                                     "lx_stat_init()", "devt",
161                                     mt[j].mt_path);
162                                 st.st_rdev = NODEV;
163                         } else {
164                                 /* make sure the major node matches */
165                                 assert(getmajor(st.st_rdev) == major);
166                                 assert(mt[j].mt_minor < LX_MINORMASK);
167                         }

169                         /* save the minor node value */
170                         mt[j].mt_minor = getminor(st.st_rdev);
171                 }
172         }
173         return (0);
174 }

176 static int
177 /*ARGSUSED*/
178 pts_devt_translator(dev_t dev, lx_dev_t *jdev, int fd)
179 {
180         minor_t min = getminor(dev);
181         int     lx_maj;
182         int     lx_min;

184         /*
185          * linux has a really small minor number name space (8 bits).
186          * so if pts devices are limited to one major number you could
187          * only have 256 of them.  linux addresses this issue by using
188          * multiple major numbers for pts devices.
189          */
190         if (min >= LX_PTS_MAX)
191                 return (EOVERFLOW);

193         lx_maj = LX_PTS_MAJOR_MIN + (min / LX_MINORMASK);
```



```
194         lx_min = min % LX_MINORMASK;

196         *jdev = LX_MAKEDEVICE(lx_maj, lx_min);
197         return (0);
198 }


201 static int
202 /*ARGSUSED*/
203 ptm_devt_translator(dev_t dev, lx_dev_t *jdev, int fd)
204 {
205         *jdev = LX_MAKEDEVICE(LX_PTM_MAJOR, LX_PTM_MINOR);
206         return (0);
207 }

209 static int
210 audio_devt_translator(dev_t dev, lx_dev_t *jdev, int fd)
211 {
212         int s_minor, l_minor;

214         if (fd == -1) {
215                 s_minor = getminor(dev);
216         } else {
217                 /*
218                  * this is a cloning device so we have to ask the driver
219                  * what kind of minor node this is
220                  */
221                 if (ioctl(fd, LXA_IOC_GETMINORNUM, &s_minor) < 0)
222                         return (-EINVAL);
223         }

225         switch (s_minor) {
226         case LXA_MINORNUM_DSP:
227                 l_minor = 3;
228                 break;
229         case LXA_MINORNUM_MIXER:
230                 l_minor = 0;
231                 break;
232         default:
233                 return (-EINVAL);
234         }

236         *jdev = LX_MAKEDEVICE(14, l_minor);
237         return (0);
238 }

240 static void
241 s2l_dev_report(dev_t dev, lx_dev_t jdev)
242 {
243         major_t                 maj;
244         minor_t                 min;
245         int                     lx_maj, lx_min;

247         if (lx_debug_enabled == 0)
248                 return;

250         maj = getmajor(dev);
251         min = getminor(dev);

253         lx_maj = LX_GETMAJOR(jdev);
254         lx_min = LX_GETMINOR(jdev);

256         lx_debug("\ttranslated devt [%d, %d] -> [%d, %d]",
257             maj, min, lx_maj, lx_min);
258 }
```

```
260 static int
261 s2l_devt(dev_t dev, lx_dev_t *jdev, int fd)
262 {
263         minor_translator_t      *mt;
264         int                     i, j, err;
265         major_t                 maj = getmajor(dev);
266         minor_t                 min = getminor(dev);
267
268         /* look for a devt translator for this major number */
269         for (i = 0; devt_translators[i].dt_driver != NULL; i++) {
270                 if (devt_translators[i].dt_major == maj)
271                         break;
272         }
273         if (devt_translators[i].dt_driver != NULL) {
274
275                 /* try to translate the solaris devt to a linux devt */
276                 switch (devt_translators[i].dt_type) {
277                 case DTT_LIST:
278                         mt = devt_translators[i].dt_list;
279                         for (j = 0; mt[j].mt_path != NULL; j++) {
280                                 if (mt[j].mt_minor == min) {
281                                         assert(mt[j].mt_minor < LX_MINORMASK);
282
283                                         /* found a translation */
284                                         *jdev = LX_MAKEDEVICE(
285                                             mt[j].mt_lx_major,
286                                             mt[j].mt_lx_minor);
287                                         s2l_dev_report(dev, *jdev);
288                                         return (0);
289                                 }
290                         }
291                         break;
292
293                 case DTT_CUSTOM:
294                         err = devt_translators[i].dt_custom(dev, jdev, fd);
295                         if (err == 0)
296                                 s2l_dev_report(dev, *jdev);
297                         return (err);
298                         break;
299                 }
300         }
301
302         /* we don't have a translator for this device */
303         *jdev = LX_MAKEDEVICE(maj, min);
304         return (0);
305 }
306
307 static int
308 stat_convert(uintptr_t lx_statp, struct stat *s, int fd)
309 {
310         struct lx_stat  buf;
311         lx_dev_t        st_dev, st_rdev;
312         int             err;
313
314         if ((err = s2l_devt(s->st_dev, &st_dev, fd)) != 0)
315                 return (err);
316         if ((err = s2l_devt(s->st_rdev, &st_rdev, fd)) != 0)
317                 return (err);
318
319         if ((st_dev > USHRT_MAX) || (st_rdev > USHRT_MAX) ||
320             (s->st_nlink  > USHRT_MAX) || (s->st_size > ULONG_MAX))
321                 return (-EOVERFLOW);
322
323         /* Linux seems to report a 0 st_size for all block devices */
324         if ((s->st_mode & S_IFMT) == S_IFBLK)
325                 s->st_size = 0;
```



```
326
327         bzero(&buf, sizeof (buf));
328         buf.st_dev = st_dev;
329         buf.st_rdev = st_rdev;
330         buf.st_ino = s->st_ino;
331         buf.st_mode = s->st_mode;
332         buf.st_nlink = s->st_nlink;
333         buf.st_uid = LX_UID32_TO_UID16(s->st_uid);
334         buf.st_gid = LX_GID32_TO_GID16(s->st_gid);
335         buf.st_size = s->st_size;
336         buf.st_blksize = s->st_blksize;
337         buf.st_blocks = s->st_blocks;
338         buf.st_atime.ts_sec = s->st_atim.tv_sec;
339         buf.st_atime.ts_nsec = s->st_atim.tv_nsec;
340         buf.st_ctime.ts_sec = s->st_ctim.tv_sec;
341         buf.st_ctime.ts_nsec = s->st_ctim.tv_nsec;
342         buf.st_mtime.ts_sec = s->st_mtim.tv_sec;
343         buf.st_mtime.ts_nsec = s->st_mtim.tv_nsec;
344
345         if (uucopy(&buf, (void *)lx_statp, sizeof (buf)) != 0)
346                 return (-errno);
347
348         return (0);
349 }
350
351 static int
352 stat64_convert(uintptr_t lx_statp, struct stat64 *s, int fd)
353 {
354         struct lx_stat64        buf;
355         lx_dev_t                st_dev, st_rdev;
356         int                     err;
357
358         if ((err = s2l_devt(s->st_dev, &st_dev, fd)) != 0)
359                 return (err);
360         if ((err = s2l_devt(s->st_rdev, &st_rdev, fd)) != 0)
361                 return (err);
362
363         /* Linux seems to report a 0 st_size for all block devices */
364         if ((s->st_mode & S_IFMT) == S_IFBLK)
365                 s->st_size = 0;
366
367         bzero(&buf, sizeof (buf));
368         buf.st_dev = st_dev;
369         buf.st_rdev = st_rdev;
370         buf.st_small_ino = (lx_ino_t)(s->st_ino & UINT_MAX);
371         buf.st_ino = (lx_ino64_t)s->st_ino;
372         buf.st_mode = s->st_mode;
373         buf.st_nlink = s->st_nlink;
374         buf.st_uid = s->st_uid;
375         buf.st_gid = s->st_gid;
376         buf.st_size = s->st_size;
377         buf.st_blksize = s->st_blksize;
378         buf.st_blocks = s->st_blocks;
379         buf.st_atime.ts_sec = s->st_atim.tv_sec;
380         buf.st_atime.ts_nsec = s->st_atim.tv_nsec;
381         buf.st_ctime.ts_sec = s->st_ctim.tv_sec;
382         buf.st_ctime.ts_nsec = s->st_ctim.tv_nsec;
383         buf.st_mtime.ts_sec = s->st_mtim.tv_sec;
384         buf.st_mtime.ts_nsec = s->st_mtim.tv_nsec;
385
386         if (uucopy(&buf, (void *)lx_statp, sizeof (buf)) != 0)
387                 return (-errno);
388
389         return (0);
390 }
```

```
392 int
393 lx_stat(uintptr_t p1, uintptr_t p2)
394 {
395         char            *path = (char *)p1;
396         struct stat     sbuf;

398         lx_debug("\tstat(%s, ...)", path);
399         if (stat(path, &sbuf))
400                 return (-errno);

402         return (stat_convert(p2, &sbuf, -1));
403 }


406 int
407 lx_fstat(uintptr_t p1, uintptr_t p2)
408 {
409         int             fd = (int)p1;
410         struct stat     sbuf;
411         char            *path, path_buf[MAXPATHLEN];

413         if (lx_debug_enabled != 0) {
414                 path = lx_fd_to_path(fd, path_buf, sizeof (path_buf));
415                 if (path == NULL)
416                         path = "?";

418                 lx_debug("\tfstat(%d - %s, ...)", fd, path);
419         }
420         if (fstat(fd, &sbuf))
421                 return (-errno);

423         return (stat_convert(p2, &sbuf, fd));
424 }


427 int
428 lx_lstat(uintptr_t p1, uintptr_t p2)
429 {
430         char            *path = (char *)p1;
431         struct stat     sbuf;

433         lx_debug("\tlstat(%s, ...)", path);
434         if (lstat(path, &sbuf))
435                 return (-errno);

437         return (stat_convert(p2, &sbuf, -1));
438 }

440 int
441 lx_stat64(uintptr_t p1, uintptr_t p2)
442 {
443         char                    *path = (char *)p1;
444         struct stat64           sbuf;

446         lx_debug("\tstat64(%s, ...)", path);
447         if (stat64(path, &sbuf))
448                 return (-errno);

450         return (stat64_convert(p2, &sbuf, -1));
451 }


454 int
455 lx_fstat64(uintptr_t p1, uintptr_t p2)
456 {
457         int                     fd = (int)p1;
```



```
458         struct stat64           sbuf;
459         char                    *path, path_buf[MAXPATHLEN];

461         if (lx_debug_enabled != 0) {
462                 path = lx_fd_to_path(fd, path_buf, sizeof (path_buf));
463                 if (path == NULL)
464                         path = "?";

466                 lx_debug("\tfstat64(%d - %s, ...)", fd, path);
467         }
468         if (fstat64(fd, &sbuf))
469                 return (-errno);

471         return (stat64_convert(p2, &sbuf, fd));
472 }

474 int
475 lx_fstatat64(uintptr_t p1, uintptr_t p2, uintptr_t p3, uintptr_t p4)
476 {
477         int atfd = (int)p1;
478         const char *path = (const char *)p2;
479         int flag;
480         struct stat64 sbuf;

482         if (atfd == LX_AT_FDCWD)
483                 atfd = AT_FDCWD;

485         flag = ltos_at_flag(p4, AT_SYMLINK_NOFOLLOW);
486         if (flag < 0)
487                 return (-EINVAL);

489         if (fstatat64(atfd, path, &sbuf, flag))
490                 return (-errno);

492         return (stat64_convert(p3, &sbuf, -1));
493 }


496 int
497 lx_lstat64(uintptr_t p1, uintptr_t p2)
498 {
499         char                    *path = (char *)p1;
500         struct stat64           sbuf;

502         lx_debug("\tlstat64(%s, ...)", path);
503         if (lstat64(path, &sbuf))
504                 return (-errno);

506         return (stat64_convert(p2, &sbuf, -1));
507 }

509 /*
510  * devt translator definitions
511  */
512 #define MINOR_TRANSLATOR(path, lx_major, lx_minor)      \
513         { path, 0, lx_major, lx_minor }

515 #define MINOR_TRANSLATOR_END                            \
516         { NULL, 0, 0, 0 }

518 #define DEVT_TRANSLATOR(drv, flags, i)          \
519         { drv, 0, flags, (uintptr_t)i }

521 /*
522  * translators for devts
523  */
```

```
524 static minor_translator_t mtranslator_mm[] = {
525         MINOR_TRANSLATOR("/dev/null", 1, 3),
526         MINOR_TRANSLATOR("/dev/zero", 1, 5),
527         MINOR_TRANSLATOR_END
528 };
529 static minor_translator_t mtranslator_random[] = {
530         MINOR_TRANSLATOR("/dev/random", 1, 8),
531         MINOR_TRANSLATOR("/dev/urandom", 1, 9),
532         MINOR_TRANSLATOR_END
533 };
534 static minor_translator_t mtranslator_sy[] = {
535         MINOR_TRANSLATOR("/dev/tty", 5, 0),
536         MINOR_TRANSLATOR_END
537 };
538 static minor_translator_t mtranslator_zcons[] = {
539         MINOR_TRANSLATOR("/dev/console", 5, 1),
540         MINOR_TRANSLATOR_END
541 };
542 static devt_translator_t devt_translators[] = {
543         DEVT_TRANSLATOR("mm",           DTT_LIST,       &mtranslator_mm),
544         DEVT_TRANSLATOR("random",       DTT_LIST,       &mtranslator_random),
545         DEVT_TRANSLATOR("sy",           DTT_LIST,       &mtranslator_sy),
546         DEVT_TRANSLATOR("zcons",        DTT_LIST,       &mtranslator_zcons),
547         DEVT_TRANSLATOR(LX_AUDIO_DRV,   DTT_CUSTOM,     audio_devt_translator),
548         DEVT_TRANSLATOR(LX_PTM_DRV,     DTT_CUSTOM,     ptm_devt_translator),
549         DEVT_TRANSLATOR("pts",          DTT_CUSTOM,     pts_devt_translator),
550         DEVT_TRANSLATOR(NULL,           0,              0)
551 };
552 #endif /* ! codereview */
```

```
**********************************************************
    7811 Tue Jan 14 16:17:05 2014
new/usr/src/lib/brand/lx/lx_brand/common/statfs.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <assert.h>
#include <errno.h>
#include <libintl.h>
#include <string.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/statvfs.h>
#include <sys/param.h>

#include <sys/lx_debug.h>
#include <sys/lx_misc.h>
#include <sys/lx_statfs.h>

/*
 * these defines must exist before we include regexp.h, see regexp(5)
 */
#define RE_SIZE         1024
#define INIT            char *sp = instring;
#define GETC()          (*sp++)
#define PEEKC()         (*sp)
#define UNGETC(c)       (--sp)
#define RETURN(c)       return (NULL);
#define ERROR(c)        return ((char *)c);

/*
 * for regular expressions we're using regexp(5).
 *
 * we'd really prefer to use some other nicer regular expressions
 * interfaces (like regcmp(3c), regcomp(3c), or re_comp(3c)) but we
 * can't because all these other interfaces rely on the ability
 * to allocate memory via libc malloc()/calloc() calls, which
 * we can't really do here.
 *
 * we could optionally use regexpr(3gen) but we don't since the
```



```
 * interfaces there are incredibly similar to the regexp(5)
 * interfaces we're already using and we'd have the added
 * requirement of linking against libgen.
 *
 * another option that was considered is fnmatch(3c) but the
 * limited pattern expansion capability of this interface would
 * force us to include more patterns to check against.
 */
#include <regexp.h>

static struct lx_ftype_path {
        char            *lfp_path;
        char            lfp_re[RE_SIZE];
        int             lfp_magic;
        char            *lfp_magic_str;
} ftype_path_list[] = {
        { "^/dev/pts$",         "",
                LX_DEVPTS_SUPER_MAGIC,  "LX_DEVPTS_SUPER_MAGIC" },
        { "^/dev/pts/$",        "",
                LX_DEVPTS_SUPER_MAGIC,  "LX_DEVPTS_SUPER_MAGIC" },
        { "^/dev/pts/[0-9][0-9]*$",     "",
                LX_DEVPTS_SUPER_MAGIC,  "LX_DEVPTS_SUPER_MAGIC" },
        { NULL,                 "",
                0,                      NULL                    }
};

/*
 * For lack of linux equivalents, we present lofs and zfs as being ufs.
 */
static struct lx_ftype_name {
        const char      *lfn_name;
        int             lfn_magic;
        char            *lfn_magic_str;
} ftype_name_list[] = {
        { "hsfs",       LX_ISOFS_SUPER_MAGIC,   "LX_ISOFS_SUPER_MAGIC"  },
        { "nfs",        LX_NFS_SUPER_MAGIC,     "LX_NFS_SUPER_MAGIC"    },
        { "pcfs",       LX_MSDOS_SUPER_MAGIC,   "LX_MSDOS_SUPER_MAGIC"  },
        { "lx_proc",    LX_PROC_SUPER_MAGIC,    "LX_PROC_SUPER_MAGIC"   },
        { "ufs",        LX_UFS_MAGIC,           "LX_UFS_MAGIC"          },
        { "lofs",       LX_UFS_MAGIC,           "LX_UFS_MAGIC"          },
        { "zfs",        LX_UFS_MAGIC,           "LX_UFS_MAGIC"          },
        { NULL,         0,                      NULL    }
};

int
lx_statfs_init()
{
        int     i;
        char    *rv;

        for (i = 0; ftype_path_list[i].lfp_path != NULL; i++) {
                rv = compile(
                    ftype_path_list[i].lfp_path,
                    ftype_path_list[i].lfp_re,
                    ftype_path_list[i].lfp_re + RE_SIZE, '\0');
                if (rv == NULL)
                        continue;

                lx_debug("lx_statfs_init compile(\"%s\") failed",
                    ftype_path_list[i].lfp_path);
                return (1);
        }
        return (0);
}

static int
```

```
128 stol_type(const char *path, const char *name)
129 {
130         int     i;
131         lx_debug("\tstol_type(\"%s\", \"%s\")\n", path == NULL ? "NULL" : path,
132             name == NULL ? "NULL" : name);
133
134         if (path != NULL) {
135                 char userpath[MAXPATHLEN];
136
137                 if (uucopystr(path, userpath, MAXPATHLEN) == -1)
138                         return (-errno);
139
140                 for (i = 0; ftype_path_list[i].lfp_path != NULL; i++) {
141                         if (step(userpath, ftype_path_list[i].lfp_re) == 0)
142                                 continue;
143
144                         /* got a match on the fs path */
145                         lx_debug("\ttranslated f_type to 0x%x - %s",
146                             ftype_path_list[i].lfp_magic,
147                             ftype_path_list[i].lfp_magic_str);
148                         return (ftype_path_list[i].lfp_magic);
149                 }
150         }
151
152         assert(name != NULL);
153         for (i = 0; ftype_name_list[i].lfn_name != NULL; i++) {
154                 if (strcmp(name, ftype_name_list[i].lfn_name) == 0) {
155
156                         /* got a match on the fs name */
157                         lx_debug("\ttranslated f_type to 0x%x - %s",
158                             ftype_name_list[i].lfn_magic,
159                             ftype_name_list[i].lfn_magic_str);
160                         return (ftype_name_list[i].lfn_magic);
161                 }
162         }
163
164         /* we don't know what the fs type is so just set it to 0 */
165         return (0);
166 }
167
168 /*
169  * The Linux statfs() is similar to the Solaris statvfs() call, the main
170  * difference being the use of a numeric 'f_type' identifier instead of the
171  * 'f_basetype' string.
172  */
173 static int
174 stol_statfs(const char *path, struct lx_statfs *l, struct statvfs *s)
175 {
176         int type;
177
178         if ((type = stol_type(path, s->f_basetype)) < 0)
179                 return (type);
180
181         l->f_type = type;
182         l->f_bsize = s->f_bsize;
183         l->f_blocks = s->f_blocks;
184         l->f_bfree = s->f_bfree;
185         l->f_bavail = s->f_bavail;
186         l->f_files = s->f_files;
187         l->f_ffree = s->f_ffree;
188         l->f_fsid = s->f_fsid;
189         l->f_namelen = s->f_namemax;
190         l->f_frsize = s->f_frsize;
191         bzero(&(l->f_spare), sizeof (l->f_spare));
192
193         return (0);
```



```
194 }
195
196 static int
197 stol_statfs64(const char *path, struct lx_statfs64 *l, struct statvfs64 *s)
198 {
199         int type;
200
201         if ((type = stol_type(path, s->f_basetype)) < 0)
202                 return (type);
203
204         l->f_type = type;
205         l->f_bsize = s->f_bsize;
206         l->f_blocks = s->f_blocks;
207         l->f_bfree = s->f_bfree;
208         l->f_bavail = s->f_bavail;
209         l->f_files = s->f_files;
210         l->f_ffree = s->f_ffree;
211         l->f_fsid = s->f_fsid;
212         l->f_namelen = s->f_namemax;
213         l->f_frsize = s->f_frsize;
214         bzero(&(l->f_spare), sizeof (l->f_spare));
215
216         return (0);
217 }
218
219 int
220 lx_statfs(uintptr_t p1, uintptr_t p2)
221 {
222         const char *path = (const char *)p1;
223         struct lx_statfs lxfs, *fs = (struct lx_statfs *)p2;
224         struct statvfs vfs;
225         int err;
226
227         lx_debug("\tfstatvfs(%s, 0x%p)", path, fs);
228         if (statvfs(path, &vfs) != 0)
229                 return (-errno);
230
231         if ((err = stol_statfs(path, &lxfs, &vfs)) != 0)
232                 return (err);
233
234         if (uucopy(&lxfs, fs, sizeof (struct lx_statfs)) != 0)
235                 return (-errno);
236
237         return (0);
238 }
239
240 int
241 lx_fstatfs(uintptr_t p1, uintptr_t p2)
242 {
243         struct lx_statfs lxfs, *fs = (struct lx_statfs *)p2;
244         struct statvfs vfs;
245         char *path, path_buf[MAXPATHLEN];
246         int fd = (int)p1;
247         int err;
248
249         lx_debug("\tfstatvfs(%d, 0x%p)", fd, fs);
250         if (fstatvfs(fd, &vfs) != 0)
251                 return (-errno);
252
253         path = lx_fd_to_path(fd, path_buf, sizeof (path_buf));
254
255         if ((err = stol_statfs(path, &lxfs, &vfs)) != 0)
256                 return (err);
257
258         if (uucopy(&lxfs, fs, sizeof (struct lx_statfs)) != 0)
259                 return (-errno);
```

```
261         return (0);
262 }

264 /* ARGSUSED */
265 int
266 lx_statfs64(uintptr_t p1, uintptr_t p2, uintptr_t p3)
267 {
268         const char *path = (const char *)p1;
269         struct lx_statfs64 lxfs, *fs = (struct lx_statfs64 *)p3;
270         struct statvfs64 vfs;
271         int err;

273         lx_debug("\tstatvfs64(%s, %d, 0x%p)", path, p2, fs);
274         if (statvfs64(path, &vfs) != 0)
275                 return (-errno);

277         if ((err = stol_statfs64(path, &lxfs, &vfs)) != 0)
278                 return (err);

280         if (uucopy(&lxfs, fs, sizeof (struct lx_statfs64)) != 0)
281                 return (-errno);

283         return (0);
284 }

286 /* ARGSUSED */
287 int
288 lx_fstatfs64(uintptr_t p1, uintptr_t p2, uintptr_t p3)
289 {
290         struct lx_statfs64 lxfs, *fs = (struct lx_statfs64 *)p3;
291         struct statvfs64 vfs;
292         char *path, path_buf[MAXPATHLEN];
293         int fd = (int)p1;
294         int err;

296         lx_debug("\tfstatvfs64(%d, %d, 0x%p)", fd, p2, fs);
297         if (fstatvfs64(fd, &vfs) != 0)
298                 return (-errno);

300         path = lx_fd_to_path(fd, path_buf, sizeof (path_buf));

302         if ((err = stol_statfs64(path, &lxfs, &vfs)) != 0)
303                 return (err);

305         if (uucopy(&lxfs, fs, sizeof (struct lx_statfs64)) != 0)
306                 return (-errno);

308         return (0);
309 }
310 #endif /* ! codereview */
```

```
**********************************************************
    3557 Tue Jan 14 16:17:05 2014
new/usr/src/lib/brand/lx/lx_brand/common/sysctl.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <alloca.h>
#include <errno.h>
#include <stdio.h>
#include <string.h>
#include <sys/lx_syscall.h>
#include <sys/lx_misc.h>
#include <sys/lx_debug.h>

/*
 * sysctl() implementation.  The full set of possible values is incredibly
 * large; we only implement the bare minimum here, namely basic kernel
 * information.
 *
 * For the moment, we also print out debugging messages if the application
 * attempts to write or access any other values, so we can tell if we are not
 * supporting something we should be.
 */

struct lx_sysctl_args {
        int *name;
        int nlen;
        void *oldval;
        size_t *oldlenp;
        void *newval;
        size_t newlen;
};

#define LX_CTL_KERN             1

#define LX_KERN_OSTYPE          1
#define LX_KERN_OSRELEASE       2
#define LX_KERN_OSREV           3
#define LX_KERN_VERSION         4
```



```
int
lx_sysctl(uintptr_t raw)
{
        struct lx_sysctl_args args;
        int name[2];
        size_t oldlen;
        char *namebuf;

        if (uucopy((void *)raw, &args, sizeof (args)) < 0)
                return (-EFAULT);

        /*
         * We only allow [ CTL_KERN, KERN_* ] pairs, so reject anything that
         * doesn't have exactly two values starting with LX_CTL_KERN.
         */
        if (args.nlen != 2)
                return (-ENOTDIR);

        if (uucopy(args.name, name, sizeof (name)) < 0)
                return (-EFAULT);

        if (name[0] != LX_CTL_KERN) {
                lx_debug("sysctl: read of [%d, %d] unsupported",
                    name[0], name[1]);
                return (-ENOTDIR);
        }

        /* We don't support writing new sysctl values. */
        if ((args.newval != NULL) || (args.newlen != 0)) {
                lx_debug("sysctl: write of [%d, %d] unsupported",
                    name[0], name[1]);
                return (-EPERM);
        }

        /*
         * It may seem silly, but passing in a NULL oldval pointer and not
         * writing any new values is a perfectly legal thing to do and should
         * succeed.
         */
        if (args.oldval == NULL)
                return (0);

        /*
         * Likewise, Linux specifies that setting a non-NULL oldval but a
         * zero *oldlenp should result in an errno of EFAULT.
         */
        if ((uucopy(args.oldlenp, &oldlen, sizeof (oldlen)) < 0) ||
            (oldlen == 0))
                return (-EFAULT);

        namebuf = SAFE_ALLOCA(oldlen);
        if (namebuf == NULL)
                return (-ENOMEM);

        switch (name[1]) {
        case LX_KERN_OSTYPE:
                (void) strlcpy(namebuf, LX_UNAME_SYSNAME, oldlen);
                break;
        case LX_KERN_OSRELEASE:
                (void) strlcpy(namebuf, lx_release, oldlen);
                break;
        case LX_KERN_VERSION:
                (void) strlcpy(namebuf, LX_UNAME_VERSION, oldlen);
                break;
        default:
                lx_debug("sysctl: read of [CTL_KERN, %d] unsupported", name[1]);
```

```
128                 return (-ENOTDIR);
129         }

131         oldlen = strlen(namebuf);

133         if ((uucopy(namebuf, args.oldval, oldlen) < 0) ||
134             (uucopy(&oldlen, args.oldlenp, sizeof (oldlen)) < 0))
135                 return (-EFAULT);

137         return (0);
138 }
139 #endif /* ! codereview */
```

```
**********************************************************
   19815 Tue Jan 14 16:17:05 2014
new/usr/src/lib/brand/lx/lx_brand/common/sysv_ipc.c
LX zone support should now build and packages of relevance produced.
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <errno.h>
#include <unistd.h>
#include <strings.h>
#include <rctl.h>
#include <alloca.h>
#include <values.h>
#include <sys/syscall.h>
#include <sys/msg.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/lx_debug.h>
#include <sys/lx_types.h>
#include <sys/lx_sysv_ipc.h>
#include <sys/lx_misc.h>
#include <sys/lx_syscall.h>

#define SLOT_SEM        0
#define SLOT_SHM        1
#define SLOT_MSG        2

static int
get_rctlval(rctlblk_t *rblk, char *name)
{
        rctl_qty_t r;

        if (getrctl(name, NULL, rblk, RCTL_FIRST) == -1)
                return (-errno);

        r = rctlblk_get_value(rblk);
        if (r > MAXINT)
```



```
                return (-EOVERFLOW);
        return (r);
}

/*
 * Given a slot number and a maximum number of ids to extract from the
 * kernel, return the msgid in the provided slot.
 */
static int
slot_to_id(int type, int slot)
{
        uint_t nids, max;
        int *idbuf = NULL;
        int r = 0;

        nids = 0;
        for (;;) {
                switch (type) {
                case SLOT_SEM:
                        r = semids(idbuf, nids, &max);
                        break;
                case SLOT_SHM:
                        r = shmids(idbuf, nids, &max);
                        break;
                case SLOT_MSG:
                        r = msgids(idbuf, nids, &max);
                        break;
                }

                if (r < 0)
                        return (-errno);

                if (max == 0)
                        return (-EINVAL);

                if (max <= nids)
                        return (idbuf[slot]);

                nids = max;
                if ((idbuf = (int *)SAFE_ALLOCA(sizeof (int) * nids)) == NULL)
                        return (-ENOMEM);
        }
}

/*
 * Semaphore operations.
 */
static int
lx_semget(key_t key, int nsems, int semflg)
{
        int sol_flag;
        int r;

        lx_debug("\nsemget(%d, %d, %d)\n", key, nsems, semflg);
        sol_flag = semflg & S_IAMB;
        if (semflg & LX_IPC_CREAT)
                sol_flag |= IPC_CREAT;
        if (semflg & LX_IPC_EXCL)
                sol_flag |= IPC_EXCL;

        r = semget(key, nsems, sol_flag);
        return ((r < 0) ? -errno : r);
}

static int
lx_semop(int semid, struct sembuf *sops, size_t nsops)
```

```
127 {
128         int r;
129 
130         lx_debug("\nsemop(%d, 0x%p, %u)\n", semid, sops, nsops);
131         if (nsops == 0)
132                 return (-EINVAL);
133 
134         r = semop(semid, sops, nsops);
135         return ((r < 0) ? -errno : r);
136 }
137 
138 static int
139 lx_semctl_ipcset(int semid, void *buf)
140 {
141         struct lx_semid_ds semds;
142         struct semid_ds sol_semds;
143         int r;
144 
145         if (uucopy(buf, &semds, sizeof (semds)))
146                 return (-errno);
147 
148         bzero(&sol_semds, sizeof (sol_semds));
149         sol_semds.sem_perm.uid = semds.sem_perm.uid;
150         sol_semds.sem_perm.gid = semds.sem_perm.gid;
151         sol_semds.sem_perm.mode = semds.sem_perm.mode;
152 
153         r = semctl(semid, 0, IPC_SET, &sol_semds);
154         return ((r < 0) ? -errno : r);
155 }
156 
157 static int
158 lx_semctl_ipcstat(int semid, void *buf)
159 {
160         struct lx_semid_ds semds;
161         struct semid_ds sol_semds;
162 
163         if (semctl(semid, 0, IPC_STAT, &sol_semds) != 0)
164                 return (-errno);
165 
166         bzero(&semds, sizeof (semds));
167         semds.sem_perm.key = sol_semds.sem_perm.key;
168         semds.sem_perm.seq = sol_semds.sem_perm.seq;
169         semds.sem_perm.uid = sol_semds.sem_perm.uid;
170         semds.sem_perm.gid = sol_semds.sem_perm.gid;
171         semds.sem_perm.cuid = sol_semds.sem_perm.cuid;
172         semds.sem_perm.cgid = sol_semds.sem_perm.cgid;
173 
174         /* Linux only uses the bottom 9 bits */
175         semds.sem_perm.mode = sol_semds.sem_perm.mode & S_IAMB;
176         semds.sem_otime = sol_semds.sem_otime;
177         semds.sem_ctime = sol_semds.sem_ctime;
178         semds.sem_nsems = sol_semds.sem_nsems;
179 
180         if (uucopy(&semds, buf, sizeof (semds)))
181                 return (-errno);
182 
183         return (0);
184 }
185 
186 static int
187 lx_semctl_ipcinfo(void *buf)
188 {
189         struct lx_seminfo i;
190         rctlblk_t *rblk;
191         int rblksz;
192         uint_t nids;
```



```
193         int idbuf;
194 
195         rblksz = rctlblk_size();
196         if ((rblk = (rctlblk_t *)SAFE_ALLOCA(rblksz)) == NULL)
197                 return (-ENOMEM);
198 
199         bzero(&i, sizeof (i));
200         if ((i.semmni = get_rctlval(rblk, "project.max-sem-ids")) < 0)
201                 return (i.semmni);
202         if ((i.semmsl = get_rctlval(rblk, "process.max-sem-nsems")) < 0)
203                 return (i.semmsl);
204         if ((i.semopm = get_rctlval(rblk, "process.max-sem-ops")) < 0)
205                 return (i.semopm);
206 
207         /*
208          * We don't have corresponding rctls for these fields.  The values
209          * are taken from the formulas used to derive the defaults listed
210          * in the Linux header file.  We're lying, but trying to be
211          * coherent about it.
212          */
213         i.semmap = i.semmni;
214         i.semmns = i.semmni * i.semmsl;
215         i.semmnu = INT_MAX;
216         i.semume = INT_MAX;
217         i.semvmx = LX_SEMVMX;
218         if (semids(&idbuf, 0, &nids) < 0)
219                 return (-errno);
220         i.semusz = nids;
221         i.semaem = INT_MAX;
222 
223         if (uucopy(&i, buf, sizeof (i)) != 0)
224                 return (-errno);
225 
226         return (nids);
227 }
228 
229 static int
230 lx_semctl_semstat(int slot, void *buf)
231 {
232         int r, semid;
233 
234         semid = slot_to_id(SLOT_SEM, slot);
235         if (semid < 0)
236                 return (semid);
237 
238         r = lx_semctl_ipcstat(semid, buf);
239         return (r < 0 ? r : semid);
240 }
241 
242 /*
243  * For the SETALL operation, we have to examine each of the semaphore
244  * values to be sure it is legal.
245  */
246 static int
247 lx_semctl_setall(int semid, union lx_semun *arg)
248 {
249         struct semid_ds semds;
250         ushort_t *vals;
251         int i, sz, r;
252 
253         /*
254          * Find out how many semaphores are involved, reserve enough
255          * memory for an internal copy of the array, and then copy it in
256          * from the process.
257          */
258         if (semctl(semid, 0, IPC_STAT, &semds) != 0)
```

```
259                 return (-errno);
260         sz = semds.sem_nsems * sizeof (ushort_t);
261         if ((vals = SAFE_ALLOCA(sz)) == NULL)
262                 return (-ENOMEM);
263         if (uucopy(arg->sems, vals, sz))
264                 return (-errno);

266         /* Validate each of the values. */
267         for (i = 0; i < semds.sem_nsems; i++)
268                 if (vals[i] > LX_SEMVMX)
269                         return (-ERANGE);

271         r = semctl(semid, 0, SETALL, arg->sems);

273         return ((r < 0) ? -errno : r);
274 }

276 static int
277 lx_semctl(int semid, int semnum, int cmd, void *ptr)
278 {
279         union lx_semun arg;
280         int rval;
281         int opt = cmd & ~LX_IPC_64;
282         int use_errno = 0;

284         lx_debug("\nsemctl(%d, %d, %d, 0x%p)\n", semid, semnum, cmd, ptr);

286         /*
287          * The final arg to semctl() is a pointer to a union.  For some
288          * commands we can hand that pointer directly to the kernel.  For
289          * these commands, we need to extract an argument from the union
290          * before calling into the kernel.
291          */
292         if (opt == LX_SETVAL || opt == LX_SETALL || opt == LX_GETALL ||
293             opt == LX_IPC_SET || opt == LX_IPC_STAT || opt == LX_SEM_STAT ||
294             opt == LX_IPC_INFO || opt == LX_SEM_INFO)
295                 if (uucopy(ptr, &arg, sizeof (arg)))
296                         return (-errno);

298         switch (opt) {
299         case LX_GETVAL:
300                 use_errno = 1;
301                 rval = semctl(semid, semnum, GETVAL, NULL);
302                 break;
303         case LX_SETVAL:
304                 if (arg.val > LX_SEMVMX) {
305                         rval = -ERANGE;
306                         break;
307                 }
308                 use_errno = 1;
309                 rval = semctl(semid, semnum, SETVAL, arg.val);
310                 break;
311         case LX_GETPID:
312                 use_errno = 1;
313                 rval = semctl(semid, semnum, GETPID, NULL);
314                 break;
315         case LX_GETNCNT:
316                 use_errno = 1;
317                 rval = semctl(semid, semnum, GETNCNT, NULL);
318                 break;
319         case LX_GETZCNT:
320                 use_errno = 1;
321                 rval = semctl(semid, semnum, GETZCNT, NULL);
322                 break;
323         case LX_GETALL:
324                 use_errno = 1;
```



```
325                 rval = semctl(semid, semnum, GETALL, arg.sems);
326                 break;
327         case LX_SETALL:
328                 rval = lx_semctl_setall(semid, &arg);
329                 break;
330         case LX_IPC_RMID:
331                 use_errno = 1;
332                 rval = semctl(semid, semnum, IPC_RMID, NULL);
333                 break;
334         case LX_SEM_STAT:
335                 rval = lx_semctl_semstat(semid, arg.semds);
336                 break;
337         case LX_IPC_STAT:
338                 rval = lx_semctl_ipcstat(semid, arg.semds);
339                 break;

341         case LX_IPC_SET:
342                 rval = lx_semctl_ipcset(semid, arg.semds);
343                 break;

345         case LX_IPC_INFO:
346         case LX_SEM_INFO:
347                 rval = lx_semctl_ipcinfo(arg.semds);
348                 break;

350         default:
351                 rval = -EINVAL;
352         }

354         if (use_errno == 1 && rval < 0)
355                 return (-errno);
356         return (rval);
357 }

359 /*
360  * msg operations.
361  */
362 static int
363 lx_msgget(key_t key, int flag)
364 {
365         int sol_flag;
366         int r;

368         lx_debug("\tlx_msgget(%d, %d)\n", key, flag);

370         sol_flag = flag & S_IAMB;
371         if (flag & LX_IPC_CREAT)
372                 sol_flag |= IPC_CREAT;
373         if (flag & LX_IPC_EXCL)
374                 sol_flag |= IPC_EXCL;

376         r = msgget(key, sol_flag);
377         return (r < 0 ? -errno : r);
378 }

380 static int
381 lx_msgsnd(int id, struct msgbuf *buf, size_t sz, int flag)
382 {
383         int sol_flag = 0;
384         int r;

386         lx_debug("\tlx_msgsnd(%d, 0x%p, %d, %d)\n", id, buf, sz, flag);

388         if (flag & LX_IPC_NOWAIT)
389                 sol_flag |= IPC_NOWAIT;
```

```
391         if (((ssize_t)sz < 0) || (sz > LX_MSGMAX))
392                 return (-EINVAL);

394         r = msgsnd(id, buf, sz, sol_flag);
395         return (r < 0 ? -errno : r);
396 }

398 static int
399 lx_msgrcv(int id, struct msgbuf *buf, size_t sz, int flag)
400 {
401         int sol_flag = 0;
402         struct {
403                 void *msgp;
404                 long msgtype;
405         } args;
406         int r;

408         /*
409          * Rather than passing 5 args into ipc(2) directly, glibc passes 4
410          * args and uses the buf argument to point to a structure
411          * containing two args: a pointer to the message and the message
412          * type.
413          */
414         if (uucopy(buf, &args, sizeof (args)))
415                 return (-errno);

417         lx_debug("\tlx_msgrcv(%d, 0x%p, %d, %d, %ld, %d)\n",
418             id, args.msgp, sz, args.msgtype, flag);

420         /*
421          * Check for a negative sz parameter.
422          *
423          * Unlike msgsnd(2), the Linux man page does not specify that
424          * msgrcv(2) should return EINVAL if (sz > MSGMAX), only if (sz < 0).
425          */
426         if ((ssize_t)sz < 0)
427                 return (-EINVAL);

429         if (flag & LX_MSG_NOERROR)
430                 sol_flag |= MSG_NOERROR;
431         if (flag & LX_IPC_NOWAIT)
432                 sol_flag |= IPC_NOWAIT;

434         r = msgrcv(id, args.msgp, sz, args.msgtype, sol_flag);
435         return (r < 0 ? -errno : r);
436 }

438 static int
439 lx_msgctl_ipcstat(int msgid, void *buf)
440 {
441         struct lx_msqid_ds msgids;
442         struct msqid_ds sol_msgids;
443         int r;

445         r = msgctl(msgid, IPC_STAT, &sol_msgids);
446         if (r < 0)
447                 return (-errno);

449         bzero(&msgids, sizeof (msgids));
450         msgids.msg_perm.key = sol_msgids.msg_perm.key;
451         msgids.msg_perm.seq = sol_msgids.msg_perm.seq;
452         msgids.msg_perm.uid = sol_msgids.msg_perm.uid;
453         msgids.msg_perm.gid = sol_msgids.msg_perm.gid;
454         msgids.msg_perm.cuid = sol_msgids.msg_perm.cuid;
455         msgids.msg_perm.cgid = sol_msgids.msg_perm.cgid;
```



```
457         /* Linux only uses the bottom 9 bits */
458         msgids.msg_perm.mode = sol_msgids.msg_perm.mode & S_IAMB;

460         msgids.msg_stime = sol_msgids.msg_stime;
461         msgids.msg_rtime = sol_msgids.msg_rtime;
462         msgids.msg_ctime = sol_msgids.msg_ctime;
463         msgids.msg_qbytes = sol_msgids.msg_qbytes;
464         msgids.msg_cbytes = sol_msgids.msg_cbytes;
465         msgids.msg_qnum = sol_msgids.msg_qnum;
466         msgids.msg_lspid = sol_msgids.msg_lspid;
467         msgids.msg_lrpid = sol_msgids.msg_lrpid;

469         if (uucopy(&msgids, buf, sizeof (msgids)))
470                 return (-errno);

472         return (0);
473 }

475 static int
476 lx_msgctl_ipcinfo(int cmd, void *buf)
477 {
478         struct lx_msginfo m;
479         rctlblk_t *rblk;
480         int idbuf, rblksz, msgseg, maxmsgs;
481         uint_t nids;
482         int rval;

484         rblksz = rctlblk_size();
485         if ((rblk = (rctlblk_t *)SAFE_ALLOCA(rblksz)) == NULL)
486                 return (-ENOMEM);

488         bzero(&m, sizeof (m));
489         if ((m.msgmni = get_rctlval(rblk, "project.max-msg-ids")) < 0)
490                 return (m.msgmni);
491         if ((m.msgmnb = get_rctlval(rblk, "process.max-msg-qbytes")) < 0)
492                 return (m.msgmnb);

494         if (cmd == LX_IPC_INFO) {
495                 if ((maxmsgs = get_rctlval(rblk,
496                     "process.max-msg-messages")) < 0)
497                         return (maxmsgs);
498                 m.msgtql = maxmsgs * m.msgmni;
499                 m.msgmap = m.msgmnb;
500                 m.msgpool = m.msgmax * m.msgmnb;
501                 rval = 0;
502         } else {
503                 if (msgids(&idbuf, 0, &nids) < 0)
504                         return (-errno);
505                 m.msgpool = nids;

507                 /*
508                  * For these fields, we can't even come up with a good fake
509                  * approximation.  These are listed as 'obsolete' or
510                  * 'unused' in the header files, so hopefully nobody is
511                  * relying on them anyway.
512                  */
513                 m.msgtql = INT_MAX;
514                 m.msgmap = INT_MAX;
515                 rval = nids;
516         }

518         /*
519          * We don't have corresponding rctls for these fields.  The values
520          * are taken from the formulas used to derive the defaults listed
521          * in the Linux header file.  We're lying, but trying to be
522          * coherent about it.
```

```
523          */
524         m.msgmax = m.msgmnb;
525         m.msgssz = 16;
526         msgseg = (m.msgpool * 1024) / m.msgssz;
527         m.msgseg = (msgseg > 0xffff) ? 0xffff : msgseg;

529         if (uucopy(&m, buf, sizeof (m)))
530                 return (-errno);
531         return (rval);
532 }

534 static int
535 lx_msgctl_ipcset(int msgid, void *buf)
536 {
537         struct lx_msqid_ds msgids;
538         struct msqid_ds sol_msgids;
539         int r;

541         if (uucopy(buf, &msgids, sizeof (msgids)))
542                 return (-errno);

544         bzero(&sol_msgids, sizeof (sol_msgids));
545         sol_msgids.msg_perm.uid = LX_UID16_TO_UID32(msgids.msg_perm.uid);
546         sol_msgids.msg_perm.gid = LX_UID16_TO_UID32(msgids.msg_perm.gid);

548         /* Linux only uses the bottom 9 bits */
549         sol_msgids.msg_perm.mode = msgids.msg_perm.mode & S_IAMB;
550         sol_msgids.msg_qbytes = msgids.msg_qbytes;

552         r = msgctl(msgid, IPC_SET, &sol_msgids);
553         return (r < 0 ? -errno : r);
554 }

556 static int
557 lx_msgctl_msgstat(int slot, void *buf)
558 {
559         int r, msgid;

561         lx_debug("msgstat(%d, 0x%p)\n", slot, buf);

563         msgid = slot_to_id(SLOT_MSG, slot);

565         if (msgid < 0)
566                 return (msgid);

568         r = lx_msgctl_ipcstat(msgid, buf);
569         return (r < 0 ? r : msgid);
570 }

572 /*
573  * Split off the various msgctl's here
574  */
575 static int
576 lx_msgctl(int msgid, int cmd, void *buf)
577 {
578         int r;

580         lx_debug("\tlx_msgctl(%d, %d, 0x%p)\n", msgid, cmd, buf);
581         switch (cmd & ~LX_IPC_64) {
582         case LX_IPC_RMID:
583                 r = msgctl(msgid, IPC_RMID, NULL);
584                 if (r < 0)
585                         r = -errno;
586                 break;
587         case LX_IPC_SET:
588                 r = lx_msgctl_ipcset(msgid, buf);
```



```
589                 break;
590         case LX_IPC_STAT:
591                 r = lx_msgctl_ipcstat(msgid, buf);
592                 break;
593         case LX_MSG_STAT:
594                 r = lx_msgctl_msgstat(msgid, buf);
595                 break;

597         case LX_IPC_INFO:
598         case LX_MSG_INFO:
599                 r = lx_msgctl_ipcinfo(cmd, buf);
600                 break;

602         default:
603                 r = -EINVAL;
604                 break;
605         }

607         return (r);
608 }

610 /*
611  * shm-related operations.
612  */
613 static int
614 lx_shmget(key_t key, size_t size, int flag)
615 {
616         int sol_flag;
617         int r;

619         lx_debug("\tlx_shmget(%d, %d, %d)\n", key, size, flag);

621         sol_flag = flag & S_IAMB;
622         if (flag & LX_IPC_CREAT)
623                 sol_flag |= IPC_CREAT;
624         if (flag & LX_IPC_EXCL)
625                 sol_flag |= IPC_EXCL;

627         r = shmget(key, size, sol_flag);
628         return (r < 0 ? -errno : r);
629 }

631 static int
632 lx_shmat(int shmid, void *addr, int flags, void **rval)
633 {
634         int sol_flags;
635         void *ptr;

637         lx_debug("\tlx_shmat(%d, 0x%p, %d, 0%o)\n", shmid, addr, flags);

639         sol_flags = 0;
640         if (flags & LX_SHM_RDONLY)
641                 sol_flags |= SHM_RDONLY;
642         if (flags & LX_SHM_RND)
643                 sol_flags |= SHM_RND;
644         if ((flags & LX_SHM_REMAP) && (addr == NULL))
645                 return (-EINVAL);

647         ptr = shmat(shmid, addr, sol_flags);
648         if (ptr == (void *)-1)
649                 return (-errno);
650         if (uucopy(&ptr, rval, sizeof (ptr)) != 0)
651                 return (-errno);

653         return (0);
654 }
```

```
656 static int
657 lx_shmctl_ipcinfo(void *buf)
658 {
659         struct lx_shminfo s;
660         rctlblk_t *rblk;
661         int rblksz;
662
663         rblksz = rctlblk_size();
664         if ((rblk = (rctlblk_t *)SAFE_ALLOCA(rblksz)) == NULL)
665                 return (-ENOMEM);
666
667         bzero(&s, sizeof (s));
668         if ((s.shmmni = get_rctlval(rblk, "project.max-shm-ids")) < 0)
669                 return (s.shmmni);
670         if ((s.shmmax = get_rctlval(rblk, "project.max-shm-memory")) < 0)
671                 return (s.shmmax);
672
673         /*
674          * We don't have corresponding rctls for these fields.  The values
675          * are taken from the formulas used to derive the defaults listed
676          * in the Linux header file.  We're lying, but trying to be
677          * coherent about it.
678          */
679         s.shmmin = 1;
680         s.shmseg = INT_MAX;
681         s.shmall = s.shmmax / getpagesize();
682
683         if (uucopy(&s, buf, sizeof (s)))
684                 return (-errno);
685
686         return (0);
687 }
688
689 static int
690 lx_shmctl_ipcstat(int shmid, void *buf)
691 {
692         struct lx_shmid_ds shmds;
693         struct shmid_ds sol_shmds;
694
695         if (shmctl(shmid, IPC_STAT, &sol_shmds) != 0)
696                 return (-errno);
697
698         bzero(&shmds, sizeof (shmds));
699         shmds.shm_perm.key = sol_shmds.shm_perm.key;
700         shmds.shm_perm.seq = sol_shmds.shm_perm.seq;
701         shmds.shm_perm.uid = sol_shmds.shm_perm.uid;
702         shmds.shm_perm.gid = sol_shmds.shm_perm.gid;
703         shmds.shm_perm.cuid = sol_shmds.shm_perm.cuid;
704         shmds.shm_perm.cgid = sol_shmds.shm_perm.cgid;
705         shmds.shm_perm.mode = sol_shmds.shm_perm.mode & S_IAMB;
706         if (sol_shmds.shm_lkcnt > 0)
707                 shmds.shm_perm.mode |= LX_SHM_LOCKED;
708         shmds.shm_segsz = sol_shmds.shm_segsz;
709         shmds.shm_atime  = sol_shmds.shm_atime;
710         shmds.shm_dtime = sol_shmds.shm_dtime;
711         shmds.shm_ctime = sol_shmds.shm_ctime;
712         shmds.shm_cpid = sol_shmds.shm_cpid;
713         shmds.shm_lpid = sol_shmds.shm_lpid;
714         shmds.shm_nattch = (ushort_t)sol_shmds.shm_nattch;
715
716         if (uucopy(&shmds, buf, sizeof (shmds)))
717                 return (-errno);
718
719         return (0);
720 }
```



```
722 static int
723 lx_shmctl_ipcset(int shmid, void *buf)
724 {
725         struct lx_shmid_ds shmds;
726         struct shmid_ds sol_shmds;
727         int r;
728
729         if (uucopy(buf, &shmds, sizeof (shmds)))
730                 return (-errno);
731
732         bzero(&sol_shmds, sizeof (sol_shmds));
733         sol_shmds.shm_perm.uid = shmds.shm_perm.uid;
734         sol_shmds.shm_perm.gid = shmds.shm_perm.gid;
735         sol_shmds.shm_perm.mode = shmds.shm_perm.mode & S_IAMB;
736
737         r = shmctl(shmid, IPC_SET, &sol_shmds);
738         return (r < 0 ? -errno : r);
739 }
740
741 /*
742  * Build and return a shm_info structure. We only return the bare
743  * essentials required by ipcs. The rest of the info is not readily
744  * available.
745  */
746 static int
747 lx_shmctl_shminfo(void *buf)
748 {
749         struct lx_shm_info shminfo;
750         uint_t nids;
751         int idbuf;
752
753         bzero(&shminfo, sizeof (shminfo));
754
755         if (shmids(&idbuf, 0, &nids) < 0)
756                 return (-errno);
757
758         shminfo.used_ids = nids;
759         if (uucopy(&shminfo, buf, sizeof (shminfo)) != 0)
760                 return (-errno);
761
762         return (nids);
763 }
764
765 static int
766 lx_shmctl_shmstat(int slot, void *buf)
767 {
768         int r, shmid;
769
770         lx_debug("shmctl_shmstat(%d, 0x%p)\n", slot, buf);
771         shmid = slot_to_id(SLOT_SHM, slot);
772         if (shmid < 0)
773                 return (shmid);
774
775         r = lx_shmctl_ipcstat(shmid, buf);
776         return (r < 0 ? r : shmid);
777 }
778
779 static int
780 lx_shmctl(int shmid, int cmd, void *buf)
781 {
782         int r;
783         int use_errno = 0;
784
785         lx_debug("\tlx_shmctl(%d, %d, 0x%p)\n", shmid, cmd, buf);
786         switch (cmd & ~LX_IPC_64) {
```

```
787         case LX_IPC_RMID:
788                 use_errno = 1;
789                 r = shmctl(shmid, IPC_RMID, NULL);
790                 break;

792         case LX_IPC_SET:
793                 r = lx_shmctl_ipcset(shmid, buf);
794                 break;

796         case LX_IPC_STAT:
797                 r = lx_shmctl_ipcstat(shmid, buf);
798                 break;

800         case LX_IPC_INFO:
801                 r = lx_shmctl_ipcinfo(buf);
802                 break;

804         case LX_SHM_LOCK:
805                 use_errno = 1;
806                 r = shmctl(shmid, SHM_LOCK, NULL);
807                 break;

809         case LX_SHM_UNLOCK:
810                 use_errno = 1;
811                 r = shmctl(shmid, SHM_UNLOCK, NULL);
812                 break;

814         case LX_SHM_INFO:
815                 r = lx_shmctl_shminfo(buf);
816                 break;

818         case LX_SHM_STAT:
819                 r = lx_shmctl_shmstat(shmid, buf);
820                 break;
821         default:
822                 r = -EINVAL;
823                 break;
824         }

826         if (use_errno == 1 && r < 0)
827                 return (-errno);

829         return (r);
830 }

832 /*
833  * Under Linux, glibc funnels all of the sysv IPC operations into this
834  * single ipc(2) system call.  We need to blow that up and filter the
835  * remnants into the proper Solaris system calls.
836  */
837 int
838 lx_ipc(uintptr_t cmd, uintptr_t arg1, uintptr_t arg2, uintptr_t arg3,
839     uintptr_t arg4)
840 {
841         int r;
842         void *bufptr = (void *)arg4;

844         lx_debug("lx_ipc(%d, %d, %d, %d, 0x%p, %d)\n",
845                 cmd, arg1, arg2, arg3, bufptr, arg4);

847         switch (cmd) {
848         case LX_MSGGET:
849                 r = lx_msgget((key_t)arg1, (int)arg2);
850                 break;
851         case LX_MSGSND:
852                 r = lx_msgsnd((int)arg1, bufptr, (size_t)arg2, (int)arg3);
```



```
853                 break;
854         case LX_MSGRCV:
855                 r = lx_msgrcv((int)arg1, bufptr, (size_t)arg2, (int)arg3);
856                 break;
857         case LX_MSGCTL:
858                 r = lx_msgctl((int)arg1, (int)arg2, bufptr);
859                 break;
860         case LX_SEMCTL:
861                 r = lx_semctl((int)arg1, (size_t)arg2, (int)arg3, bufptr);
862                 break;
863         case LX_SEMOP:
864                 /*
865                  * 'struct sembuf' is the same on Linux and Solaris, so we
866                  * pass bufptr straight through.
867                  */
868                 r = lx_semop((int)arg1, bufptr, (size_t)arg2);
869                 break;
870         case LX_SEMGET:
871                 r = lx_semget((int)arg1, (size_t)arg2, (int)arg3);
872                 break;
873         case LX_SHMAT:
874                 r = lx_shmat((int)arg1, bufptr, (size_t)arg2, (void *)arg3);
875                 break;
876         case LX_SHMDT:
877                 r = shmdt(bufptr);
878                 if (r < 0)
879                         r = -errno;
880                 break;
881         case LX_SHMGET:
882                 r = lx_shmget((int)arg1, (size_t)arg2, (int)arg3);
883                 break;
884         case LX_SHMCTL:
885                 r = lx_shmctl((int)arg1, (int)arg2, bufptr);
886                 break;

888         default:
889                 r = -EINVAL;
890         }

892         return (r);
893 }
894 #endif /* ! codereview */
```

```
**********************************************************
    4903 Tue Jan 14 16:17:05 2014
new/usr/src/lib/brand/lx/lx_brand/common/time.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <errno.h>
#include <time.h>
#include <string.h>
#include <strings.h>
#include <sys/times.h>
#include <sys/lx_syscall.h>
#include <sys/lx_misc.h>

/*
 * time() - This cannot be passthrough because on Linux a bad buffer will
 *          set errno to EFAULT, and on Solaris the failure mode is documented
 *          as "undefined."
 *
 *          (At present, Solaris' time(2) will segmentation fault, as the call
 *          is simply a libc wrapper atop the time() syscall that will
 *          dereference the passed  pointer if it is non-zero.)
 */
int
lx_time(uintptr_t p1)
{
        time_t ret = time((time_t *)0);

        if ((ret == (time_t)-1) ||
            ((p1 != 0) && (uucopy(&ret, (time_t *)p1, sizeof (ret)) != 0)))
                return (-errno);

        return (ret);
}

/*
 * times() - The Linux implementation avoids writing to NULL, while Solaris
 *           returns EFAULT.
 */
```



```
int
lx_times(uintptr_t p1)
{
        clock_t ret;
        struct tms buf, *tp = (struct tms *)p1;

        ret = times(&buf);

        if ((ret == -1) ||
            ((tp != NULL) && uucopy((void *)&buf, tp, sizeof (buf)) != 0))
                return (-errno);

        return ((ret == -1) ? -errno : ret);
}

/*
 * setitimer() - the Linux implementation can handle tv_usec values greater
 *               than 1,000,000 where Solaris would return EINVAL.
 *
 *               There's still an issue here where Linux can handle a
 *               tv_sec value greater than 100,000,000 but Solaris cannot,
 *               but that would also mean setting an interval timer to fire
 *               over _three years_ in the future so it's unlikely anything
 *               other than Linux test suites will trip over it.
 */
int
lx_setitimer(uintptr_t p1, uintptr_t p2, uintptr_t p3)
{
        struct itimerval itv;
        struct itimerval *itp = (struct itimerval *)p2;

        if (itp != NULL) {
                if (uucopy(itp, &itv, sizeof (itv)) != 0)
                        return (-errno);

                /*
                 * Adjust any tv_usec fields >= 1,000,000 by adding any whole
                 * seconds so indicated to tv_sec and leaving tv_usec as the
                 * remainder.
                 */
                if (itv.it_interval.tv_usec >= MICROSEC) {
                        itv.it_interval.tv_sec +=
                            itv.it_interval.tv_usec / MICROSEC;

                        itv.it_interval.tv_usec %= MICROSEC;
                }
                if (itv.it_value.tv_usec >= MICROSEC) {
                        itv.it_value.tv_sec +=
                            itv.it_value.tv_usec / MICROSEC;

                        itv.it_value.tv_usec %= MICROSEC;
                }

                itp = &itv;
        }

        return ((setitimer((int)p1, itp, (struct itimerval *)p3) != 0) ?
                -errno : 0);
}

/*
 * NOTE: The Linux man pages state this structure is obsolete and is
 *       unsupported, so it is declared here for sizing purposes only.
 */
struct lx_timezone {
        int tz_minuteswest;     /* minutes W of Greenwich */
```

```
128         int tz_dsttime;         /* type of dst correction */
129 };

131 /*
132  * lx_gettimeofday() and lx_settimeofday() are implemented here rather than
133  * as pass-through calls to Solaris' libc due to the need to return EFAULT
134  * for a bad buffer rather than die with a segmentation fault.
135  */
136 int
137 lx_gettimeofday(uintptr_t p1, uintptr_t p2)
138 {
139         struct timeval tv;
140         struct lx_timezone tz;

142         bzero(&tz, sizeof (tz));
143         (void) gettimeofday(&tv, NULL);

145         if ((p1 != NULL) &&
146             (uucopy(&tv, (struct timeval *)p1, sizeof (tv)) < 0))
147                 return (-errno);

149         /*
150          * The Linux man page states use of the second parameter is obsolete,
151          * but gettimeofday(2) should still return EFAULT if it is set
152          * to a bad non-NULL pointer (sigh...)
153          */
154         if ((p2 != NULL) &&
155             (uucopy(&tz, (struct lx_timezone *)p2, sizeof (tz)) < 0))
156                 return (-errno);

158         return (0);
159 }

161 int
162 lx_settimeofday(uintptr_t p1, uintptr_t p2)
163 {
164         struct timeval tv;
165         struct lx_timezone tz;

167         if ((p1 != NULL) &&
168             (uucopy((struct timeval *)p1, &tv, sizeof (tv)) < 0))
169                 return (-errno);

171         /*
172          * The Linux man page states use of the second parameter is obsolete,
173          * but settimeofday(2) should still return EFAULT if it is set
174          * to a bad non-NULL pointer (sigh...)
175          */
176         if ((p2 != NULL) &&
177             (uucopy((struct lx_timezone *)p2, &tz, sizeof (tz)) < 0))
178                 return (-errno);

180         if ((p1 != NULL) && (settimeofday(&tv, NULL) < 0))
181                 return (-errno);

183         return (0);
184 }
185 #endif /* ! codereview */
```

```
**********************************************************
    1766 Tue Jan 14 16:17:06 2014
new/usr/src/lib/brand/lx/lx_brand/common/truncate.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <errno.h>
#include <unistd.h>
#include <sys/lx_types.h>
#include <sys/lx_misc.h>

/*
 * On Solaris, truncate() and ftruncate() are implemented in libc, so these are
 * layered on those interfaces.
 */

int
lx_truncate(uintptr_t path, uintptr_t length)
{
        return (truncate((const char *)path, (off_t)length) == 0 ? 0 : -errno);
}

int
lx_ftruncate(uintptr_t fd, uintptr_t length)
{
        return (ftruncate((int)fd, (off_t)length) == 0 ? 0 : -errno);
}

int
lx_truncate64(uintptr_t path, uintptr_t length_lo, uintptr_t length_hi)
{
        return (truncate64((const char *)path,
            LX_32TO64(length_lo, length_hi)) == 0 ? 0 : -errno);
}

int
lx_ftruncate64(uintptr_t fd, uintptr_t length_lo, uintptr_t length_hi)
{
        return (ftruncate64((int)fd,
```



```
            LX_32TO64(length_lo, length_hi)) == 0 ? 0 : -errno);
}
#endif /* ! codereview */
```

```
**********************************************************
    7834 Tue Jan 14 16:17:06 2014
new/usr/src/lib/brand/lx/lx_brand/common/wait.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

/*
 * wait() family of functions.
 *
 * The first minor difference between the Linux and Solaris family of wait()
 * calls is that the values for WNOHANG and WUNTRACED are different.  Solaris
 * also has additional options (WCONTINUED, WNOWAIT) which should be flagged as
 * invalid on Linux. Thankfully, the exit status values are identical between
 * the two implementations.
 *
 * Things get very different and very complicated when we introduce the Linux
 * threading model.  Under linux, both threads and child processes are
 * represented as processes.  However, the behavior of wait() with respect to
 * each child varies according to the flags given to clone()
 *
 *      SIGCHLD         The SIGCHLD signal should be sent on termination
 *      CLONE_THREAD    The child shares the same thread group as the parent
 *      CLONE_DETACHED  The parent receives no notification when the child exits
 *
 * The following flags control the Linux behavior w.r.t. the above attributes:
 *
 *      __WALL          Wait on all children, regardless of type
 *      __WCLONE        Wait only on non-SIGCHLD children
 *      __WNOTHREAD     Don't wait on children of other threads in this group
 *
 * The following chart shows whether wait() returns when the child exits:
 *
 *                           default    __WCLONE    __WALL
 *           no SIGCHLD         -           X         X
 *              SIGCHLD         X           -         X
 *
 * The following chart shows whether wait() returns when the grandchild exits:
 *
 *                           default   __WNOTHREAD
```



```
 *      no CLONE_THREAD         -           -
 *         CLONE_THREAD         X           -
 *
 * The CLONE_DETACHED flag is universal - when the child exits, no state is
 * stored and wait() has no effect.
 *
 * XXX Support the above combination of options, or some reasonable subset that
 *     covers at least fork() and pthread_create().
 */

#include <errno.h>
#include <sys/wait.h>
#include <sys/lx_types.h>
#include <sys/lx_signal.h>
#include <sys/lx_misc.h>
#include <sys/lx_syscall.h>
#include <sys/times.h>
#include <strings.h>
#include <unistd.h>
#include <assert.h>

/*
 * Convert between Linux options and Solaris options, returning -1 if any
 * invalid flags are found.
 */
#define LX_WNOHANG      0x1
#define LX_WUNTRACED    0x2

#define LX_WNOTHREAD    0x20000000
#define LX_WALL         0x40000000
#define LX_WCLONE       0x80000000

#define LX_P_ALL        0x0
#define LX_P_PID        0x1
#define LX_P_GID        0x2

static int
ltos_options(uintptr_t options)
{
        int newoptions = 0;

        if (((options) & ~(LX_WNOHANG | LX_WUNTRACED | LX_WNOTHREAD |
            LX_WALL | LX_WCLONE)) != 0) {
                return (-1);
        }
        /* XXX implement LX_WNOTHREAD, LX_WALL, LX_WCLONE */

        if (options & LX_WNOHANG)
                newoptions |= WNOHANG;
        if (options & LX_WUNTRACED)
                newoptions |= WUNTRACED;

        return (newoptions);
}

static int
lx_wstat(int code, int status)
{
        int stat = 0;

        switch (code) {
        case CLD_EXITED:
                stat = status << 8;
                break;
        case CLD_DUMPED:
                stat = stol_signo[status];
```

```
128                 assert(stat != -1);
129                 stat |= WCOREFLG;
130                 break;
131         case CLD_KILLED:
132                 stat = stol_signo[status];
133                 assert(stat != -1);
134                 break;
135         case CLD_TRAPPED:
136         case CLD_STOPPED:
137                 stat = stol_signo[status];
138                 assert(stat != -1);
139                 stat <<= 8;
140                 stat |= WSTOPFLG;
141                 break;
142         case CLD_CONTINUED:
143                 stat = WCONTFLG;
144                 break;
145         }

147         return (stat);
148 }

150 /* wrapper to make solaris waitid work properly with ptrace */
151 static int
152 lx_waitid_helper(idtype_t idtype, id_t id, siginfo_t *info, int options)
153 {
154         do {
155                 /*
156                  * It's possible that we return EINVAL here if the idtype is
157                  * P_PID or P_PGID and id is out of bounds for a valid pid or
158                  * pgid, but Linux expects to see ECHILD. No good way occurs to
159                  * handle this so we'll punt for now.
160                  */
161                 if (waitid(idtype, id, info, options) < 0)
162                         return (-errno);

164                 /*
165                  * If the WNOHANG flag was specified and no child was found
166                  * return 0.
167                  */
168                 if ((options & WNOHANG) && info->si_pid == 0)
169                         return (0);

171                 /*
172                  * It's possible that we may have a spurious return for one of
173                  * the child processes created by the ptrace subsystem. If
174                  * that's the case, we simply try again.
175                  */
176         } while (lx_ptrace_wait(info) == -1);
177         return (0);
178 }

180 int
181 lx_wait4(uintptr_t p1, uintptr_t p2, uintptr_t p3, uintptr_t p4)
182 {
183         siginfo_t info = { 0 };
184         struct rusage ru = { 0 };
185         idtype_t idtype;
186         id_t id;
187         int options, status = 0;
188         pid_t pid = (pid_t)p1;
189         int rval;

191         if ((options = ltos_options(p3)) == -1)
192                 return (-EINVAL);
```



```
194         /*
195          * While not listed as a valid return code, Linux's wait4(2) does,
196          * in fact, get an EFAULT if either the status pointer or rusage
197          * pointer is invalid. Since a failed waitpid should leave child
198          * process in a state where a future wait4(2) will succeed, we
199          * check them by copying out the values their buffers originally
200          * contained.  (We need to do this as a failed system call should
201          * never affect the contents of a passed buffer.)
202          *
203          * This will fail if the buffers in question are write-only.
204          */
205         if ((void *)p2 != NULL &&
206             ((uucopy((void *)p2, &status, sizeof (status)) != 0) ||
207             (uucopy(&status, (void *)p2, sizeof (status)) != 0)))
208                 return (-EFAULT);

210         if ((void *)p4 != NULL) {
211                 if ((uucopy((void *)p4, &ru, sizeof (ru)) != 0) ||
212                     (uucopy(&ru, (void *)p4, sizeof (ru)) != 0))
213                         return (-EFAULT);
214         }

216         if (pid < -1) {
217                 idtype = P_PGID;
218                 id = -pid;
219         } else if (pid == -1) {
220                 idtype = P_ALL;
221                 id = 0;
222         } else if (pid == 0) {
223                 idtype = P_PGID;
224                 id = getpgrp();
225         } else {
226                 idtype = P_PID;
227                 id = pid;
228         }

230         options |= WEXITED | WTRAPPED;

232         if ((rval = lx_waitid_helper(idtype, id, &info, options)) < 0)
233                 return (rval);
234         /*
235          * If the WNOHANG flag was specified and no child was found return 0.
236          */
237         if ((options & WNOHANG) && info.si_pid == 0)
238                 return (0);

240         status = lx_wstat(info.si_code, info.si_status);

242         /*
243          * Unfortunately if this attempt to copy out either the status or the
244          * rusage fails, the process will be in an inconsistent state as
245          * subsequent calls to wait for the same child will fail where they
246          * should succeed on a Linux system. This, however, is rather
247          * unlikely since we tested the validity of both above.
248          */
249         if (p2 != NULL && uucopy(&status, (void *)p2, sizeof (status)) != 0)
250                 return (-EFAULT);

252         if (p4 != NULL && (rval = lx_getrusage(LX_RUSAGE_CHILDREN, p4)) != 0)
253                 return (rval);

255         return (info.si_pid);
256 }

258 int
259 lx_waitpid(uintptr_t p1, uintptr_t p2, uintptr_t p3)
```

```
260 {
261         return (lx_wait4(p1, p2, p3, NULL));
262 }

264 int
265 lx_waitid(uintptr_t idtype, uintptr_t id, uintptr_t infop, uintptr_t opt)
266 {
267         int rval, options;
268         siginfo_t s_infop = {0};
269         if ((options = ltos_options(opt)) == -1)
270                 return (-1);
271         switch (idtype) {
272         case LX_P_ALL:
273                 idtype = P_ALL;
274                 break;
275         case LX_P_PID:
276                 idtype = P_PID;
277                 break;
278         case LX_P_GID:
279                 idtype = P_GID;
280                 break;
281         default:
282                 return (-EINVAL);
283         }
284         if ((rval = lx_waitid_helper(idtype, (id_t)id, &s_infop, options)) < 0)
285                 return (rval);

287         return (stol_siginfo(&s_infop, (lx_siginfo_t *)infop));
288 }
289 #endif /* ! codereview */
```

```
**********************************************************
    1387 Tue Jan 14 16:17:06 2014
new/usr/src/lib/brand/lx/lx_brand/i386/Makefile
Bring back LX zones.
**********************************************************
```

```
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #
  21 #
  22 # Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  23 # Use is subject to license terms.
  24 #
  25 # ident "%Z%%M% %I%     %E% SMI"
  26 #
  27 # lib/brand/lx/i386/Makefile

  29 ISASRCDIR=.

  31 ASFLAGS += -P -D_ASM

  33 include ../Makefile.com

  35 POFILE=         lx_brand.po
  36 MSGFILES=       $(CSRCS)

  38 ASSYMDEP_OBJS = lx_handler.o

  40 $(ASSYMDEP_OBJS:%=pics/%): assym.h

  42 OFFSETS = ../$(MACH)/offsets.in

  44 assym.h: $(OFFSETS)
  45         $(OFFSETS_CREATE) $(CTF_FLAGS) < $(OFFSETS) > $@

  47 CLOBBERFILES += assym.h

  49 install: all $(ROOTLIBS)

  51 $(POFILE): $(MSGFILES)
  52         $(BUILDPO.msgfiles)

  54 _msg: $(MSGDOMAINPOFILE)

  56 include $(SRC)/Makefile.msg.targ
  57 #endif /* ! codereview */
```

```
**********************************************************
    1681 Tue Jan 14 16:17:06 2014
new/usr/src/lib/brand/lx/lx_brand/i386/lx_crt.s
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ident  "%Z%%M% %I%     %E% SMI"

#include <sys/asm_linkage.h>

#if defined(lint)

void
_start(void)
{
}

#else   /* lint */

        /*
         * C language startup routine for the lx brand shared library.
         */
        ENTRY_NP(_start)
        pushl   $0                      / Build a stack frame. retpc = NULL
        pushl   $0                      / fp = NULL
        movl    %esp, %ebp              / first stack frame

        /*
         * Calculate the location of the envp array by adding the size of
         * the argv array to the start of the argv array.
         */
        movl    8(%ebp), %eax           / argc in %eax
        leal    16(%ebp,%eax,4), %edx   / envp in %edx
        andl    $-16, %esp
        pushl   %edx                    / push envp
        leal    12(%ebp),%edx           / compute &argv[0]
        pushl   %edx                    / push argv
        pushl   %eax                    / push argc
        call    lx_init
        /*
         * lx_init will never return.
```



```
         */
        SET_SIZE(_start)

#endif  /* lint */
#endif /* ! codereview */
```

```
**********************************************************
   10796 Tue Jan 14 16:17:06 2014
new/usr/src/lib/brand/lx/lx_brand/i386/lx_handler.s
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/asm_linkage.h>
#include <sys/regset.h>
#include <sys/segments.h>
#include <sys/syscall.h>
#include <sys/lx_brand.h>

#if defined(_ASM)
#include <sys/lx_signal.h>
#include <sys/lx_syscall.h>
#endif  /* _ASM */

#include "assym.h"

#define PIC_SETUP(r)                                    \
        call    9f;                                     \
9:      popl    r;                                      \
        addl    $_GLOBAL_OFFSET_TABLE_ + [. - 9b], r

/*
 * Each JMP must occupy 16 bytes
 */
#define JMP     \
        pushl   $_CONST(. - lx_handler_table); \
        jmp     lx_handler;     \
        .align  16;

#define JMP4    JMP; JMP; JMP; JMP
#define JMP16   JMP4; JMP4; JMP4; JMP4
#define JMP64   JMP16; JMP16; JMP16; JMP16
#define JMP256  JMP64; JMP64; JMP64; JMP64

/*
 * Alternate jump table that turns on lx_traceflag before proceeding with
 * the normal emulation routine.
```



```
 */
#define TJMP    \
        pushl   $_CONST(. - lx_handler_trace_table); \
        jmp     lx_handler_trace;       \
        .align  16;

#define TJMP4   TJMP; TJMP; TJMP; TJMP
#define TJMP16  TJMP4; TJMP4; TJMP4; TJMP4
#define TJMP64  TJMP16; TJMP16; TJMP16; TJMP16
#define TJMP256 TJMP64; TJMP64; TJMP64; TJMP64

#if defined(lint)

#include <sys/types.h>
#include <sys/regset.h>
#include <sys/signal.h>

void
lx_handler_table(void)
{}

void
lx_handler(void)
{}

/* ARGSUSED */
void
lx_setup_clone(uintptr_t gs, void *retaddr, void *stk)
{}

/* ARGSUSED */
void
lx_sigdeliver(int sig, siginfo_t *sip, void *p, size_t stacksz,
    void (*stack_frame_builder)(void), void (*lx_sighandler)(void),
    uintptr_t gs)
{}

/* ARGSUSED */
void
lx_sigacthandler(int sig, siginfo_t *s, void *p)
{}

void
lx_sigreturn_tramp(void)
{}

void
lx_rt_sigreturn_tramp(void)
{}

/* ARGSUSED */
void
lx_sigreturn_tolibc(uintptr_t sp)
{}

#else   /* lint */

        /*
         * On entry to this table, %eax will hold the return address. The
         * location where we enter the table is a function of the system
         * call number. The table needs the same alignment as the individual
         * entries.
         */
        .align  16
        ENTRY_NP(lx_handler_trace_table)
```

```
128         TJMP256
129         TJMP64
130         SET_SIZE(lx_handler_trace_table)
131
132         .align  16
133         ENTRY_NP(lx_handler_table)
134         JMP256
135         JMP64
136         SET_SIZE(lx_handler_table)
137
138         ENTRY_NP(lx_handler_trace)
139         pushl   %esi
140         PIC_SETUP(%esi)
141         movl    lx_traceflag@GOT(%esi), %esi
142         movl    $1, (%esi)
143         popl    %esi
144         /*
145          * While we could just fall through to lx_handler(), we "tail-call" it
146          * instead to make ourselves a little more comprehensible to trace
147          * tools.
148          */
149         jmp     lx_handler
150         SET_SIZE(lx_handler_trace)
151
152         ALTENTRY(lx_handler)
153         /*
154          * %ebp isn't always going to be a frame pointer on Linux, but when
155          * it is, saving it here lets us have a coherent stack backtrace.
156          */
157         pushl   %ebp
158
159         /*
160          * Fill in a lx_regs_t structure on the stack.
161          */
162         subl    $SIZEOF_LX_REGS_T, %esp
163
164         /*
165          * Save %ebp and then fill it with what would be its usual value as
166          * the frame pointer. The value we save for %esp needs to be the
167          * stack pointer at the time of the interrupt so we need to skip the
168          * saved %ebp and (what will be) the return address.
169          */
170         movl    %ebp, LXR_EBP(%esp)
171         movl    %esp, %ebp
172         addl    $_CONST(SIZEOF_LX_REGS_T), %ebp
173         movl    %ebp, LXR_ESP(%esp)
174         addl    $_CONST(_MUL(CPTRSIZE, 2)), LXR_ESP(%esp)
175
176         movl    $0, LXR_GS(%esp)
177         movw    %gs, LXR_GS(%esp)
178         movl    %edi, LXR_EDI(%esp)
179         movl    %esi, LXR_ESI(%esp)
180         movl    %ebx, LXR_EBX(%esp)
181         movl    %edx, LXR_EDX(%esp)
182         movl    %ecx, LXR_ECX(%esp)
183         movl    %eax, LXR_EIP(%esp)
184
185         /*
186          * The kernel drops us into the middle of one of the tables above
187          * that then pushes that table offset onto the stack, and calls into
188          * lx_handler. That offset indicates the system call number while
189          * %eax holds the return address for the system call. We replace the
190          * value on the stack with the return address, and use the value to
191          * compute the system call number by dividing by the table entry size.
192          */
193         xchgl   CPTRSIZE(%ebp), %eax
```



```
194         shrl    $4, %eax
195         movl    %eax, LXR_EAX(%esp)
196
197         /*
198          * Switch to the Solaris libc's %gs.
199          */
200         movl    $LWPGS_SEL, %ebx
201         movw    %bx, %gs
202
203         /*
204          * Call lx_emulate() whose only argument is a pointer to the
205          * lx_regs_t structure we've placed on the stack.
206          */
207         pushl   %esp
208         call    lx_emulate
209
210         /*
211          * We use this global symbol to identify this return site when
212          * walking the stack backtrace. It needs to remain immediately
213          * after the call to lx_emulate().
214          */
215         ALTENTRY(lx_emulate_done)
216
217         /*
218          * Clean up the argument to lx_emulate().
219          */
220         addl    $4, %esp
221
222         /*
223          * Restore the saved register state; we get %ebp, %esp and %esp from
224          * the ordinary locations rather than the saved state.
225          */
226         movl    LXR_EDI(%esp), %edi
227         movl    LXR_ESI(%esp), %esi
228         movl    LXR_EBX(%esp), %ebx
229         movl    LXR_EDX(%esp), %edx
230         movl    LXR_ECX(%esp), %ecx
231         movl    LXR_EAX(%esp), %eax
232         movw    LXR_GS(%esp), %gs
233
234         addl    $SIZEOF_LX_REGS_T, %esp
235
236         movl    %ebp, %esp
237         popl    %ebp
238         ret
239         SET_SIZE(lx_handler)
240
241         ENTRY_NP(lx_swap_gs)
242         push    %eax            /* save the current eax value */
243         movl    0xc(%esp),%eax  /* 2nd param is a pointer */
244         movw    %gs,(%eax)      /* use the pointer to save current gs */
245         movl    0x8(%esp),%eax  /* first parameter is the new gs value */
246         movw    %ax, %gs        /* switch to the new gs value */
247         pop     %eax            /* restore eax */
248         ret
249         SET_SIZE(lx_swap_gs)
250
251         ENTRY_NP(lx_setup_clone)
252         xorl    %ebp, %ebp      /* terminating stack */
253         popl    %edx            /* eat the start_clone() return address */
254         popl    %gs             /* Switch back to the Linux libc's %gs */
255         popl    %edx            /* Linux clone() return address */
256         popl    %esp            /* New stack pointer */
257         xorl    %eax, %eax      /* child returns 0 to SYS_clone() */
258         jmp     *%edx           /* return to Linux app. */
259         SET_SIZE(lx_setup_clone)
```

```
261         /*
262          * lx_sigdeliver(sig, siginfo_t *, ucontext_t *, stack_size,
263          *     stack_build_routine, signal_handler, glibc_gs)
264          *
265          * This routine allocates stack space for the Linux signal stack,
266          * calls a routine to build the signal stack and then calls the Linux
267          * signal handler.  This is written in assembly because of the way
268          * we need to directly manipulate the stack and pass the resulting
269          * stack to the signal handler with the Linux signal stack on top.
270          *
271          * When the Linux signal handler is called, the stack will look
272          * like this:
273          *
274          *      ================================================
275          *      | Linux signal frame built by lx_stackbuilder() |
276          *      ================================================
277          *      | LX_SIGRT_MAGIC                               |
278          *      ================================================
279          *      | %ebp                                         |
280          *      ================================================
281          */
282         ENTRY_NP(lx_sigdeliver)
283         pushl   %ebp
284         movl    %esp, %ebp
285         movl    16(%ebp), %edx          /* pointer to Solaris ucontext_t */
286         pushl   %edx                    /* save ucontext_t ptr for later */
287         pushl   $LX_SIGRT_MAGIC         /* marker value for lx_(rt)_sigreturn */

289         subl    20(%ebp), %esp          /* create stack buffer */
290         pushl   %esp                    /* push stack pointer */
291         pushl   %edx                    /* push pointer to ucontext_t */
292         pushl   12(%ebp)                /* push pointer to siginfo_t */
293         pushl   8(%ebp)                 /* push signal number */
294         call    *24(%ebp)               /* lx_stackbuilder(sig, sip, ucp, sp) */
295         add     $16, %esp               /* remove args from stack */
296         movw    32(%ebp), %gs           /* only low 16 bits are used */

298         mov     4(%ebp),%eax            /* fetch old %ebp from stack */
299         mov     28(%ebp), %edx          /* get address of Linux handler */
300         mov     %eax, %ebp              /* restore old %ebp */
301         jmp     *%edx                   /* jmp to the Linux signal handler */
302         SET_SIZE(lx_sigdeliver)

304         /*
305          * Due to the nature of signals, we need to be able to force the %gs
306          * value to that used by Solaris by running any Solaris code.
307          *
308          * This routine does that, then calls a C routine that will save the
309          * %gs value at the time of the signal off into a thread-specific data
310          * structure.  Finally, we trampoline to the libc code that would
311          * normally interpose itself before calling a signal handler.
312          *
313          * The libc routine that calls user signal handlers ends with a
314          * setcontext, so we would never return here even if we used a call
315          * rather than a jmp.
316          *
317          * %esi is used for the PIC as it is guaranteed by the 386 ABI to
318          * survive the call to lx_sigsavegs.  The downside is we must also
319          * preserve its value for our caller.
320          *
321          * Note that because lx_sigsavegs and libc_sigacthandler are externs,
322          * they need to be dereferenced via the GOT.
323          *
324          * IMPORTANT:  Because libc apparently gets upset if extra data is
325          *             left on its stack, this routine needs to be crafted
```



```
326          *             in assembly so that the jmp to the libc interposer
327          *             doesn't leave any cruft lying around.
328          */
329         ENTRY_NP(lx_sigacthandler)
330         pushl   %esi                            /* save %esi */
331         pushl   %gs                             /* push the Linux %gs */
332         pushl   $LWPGS_SEL
333         popl    %gs                             /* install the Solaris %gs */

335         PIC_SETUP(%esi)
336         movl    lx_sigsavegs@GOT(%esi), %eax
337         call    *%eax                           /* save the Linux %gs */
338         movl    libc_sigacthandler@GOT(%esi), %eax
339         add     $4, %esp                        /* clear Linux %gs from stack */
340         popl    %esi                            /* restore %esi */
341         jmp     *(%eax)                         /* jmp to libc's interposer */
342         SET_SIZE(lx_sigacthandler)

344         /*
345          * Trampoline code is called by the return at the end of a Linux
346          * signal handler to return control to the interrupted application
347          * via the lx_sigreturn() or lx_rt_sigreturn() syscalls.
348          *
349          * (lx_sigreturn() is called for legacy signal handling, and
350          * lx_rt_sigreturn() is called for "new"-style signals.)
351          *
352          * These two routines must consist of the EXACT code sequences below
353          * as gdb looks at the sequence of instructions a routine will return
354          * to determine whether it is in a signal handler or not.
355          */
356         ENTRY_NP(lx_sigreturn_tramp)
357         popl    %eax
358         movl    $LX_SYS_sigreturn, %eax
359         int     $0x80
360         SET_SIZE(lx_sigreturn_tramp)

362         ENTRY_NP(lx_rt_sigreturn_tramp)
363         movl    $LX_SYS_rt_sigreturn, %eax
364         int     $0x80
365         SET_SIZE(lx_rt_sigreturn_tramp)

367         /*
368          * Manipulate the stack in the way necessary for it to appear to libc
369          * that the signal handler it invoked via call_user_handler() is
370          * returning.
371          */
372         ENTRY_NP(lx_sigreturn_tolibc)
373         movl    4(%esp), %esp           /* set %esp to passed value */
374         popl    %ebp                    /* restore proper %ebp */
375         ret                             /* return to libc interposer */
376         SET_SIZE(lx_sigreturn_tolibc)
377 #endif  /* lint */
378 #endif /* ! codereview */
```

```
**********************************************************
    1586 Tue Jan 14 16:17:07 2014
new/usr/src/lib/brand/lx/lx_brand/i386/lx_runexe.s
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ident  "%Z%%M% %I%     %E% SMI"

#include <sys/asm_linkage.h>

#if defined(lint)

/*ARGSUSED*/
void
lx_runexe(void *argv, int32_t entry)
{
}

#else   /* lint */

        /*
         * Set our stack pointer, clear the general registers,
         * and jump to the brand linker's entry point.
         */
        ENTRY_NP(lx_runexe)
        movl    4(%esp), %eax           / %eax = &argv[0]
        movl    8(%esp), %ebx           / Brand linker's entry point in %ebx
        subl    $4, %eax                / Top of stack - must point at argc
        movl    %eax, %esp              / Set %esp to what linkers expect

        movl    $0, %eax
        movl    $0, %ecx
        movl    $0, %edx
        movl    $0, %esi
        movl    $0, %edi
        movl    $0, %ebp

        jmp     *%ebx                   / And away we go...
        SET_SIZE(lx_runexe)

#endif  /* lint */
```



```
#endif /* ! codereview */
```

```
**********************************************************
    1092 Tue Jan 14 16:17:07 2014
new/usr/src/lib/brand/lx/lx_brand/i386/offsets.in
Bring back LX zones.
**********************************************************
```

```
\
\ Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
\ Use is subject to license terms.
\
\ CDDL HEADER START
\
\ The contents of this file are subject to the terms of the
\ Common Development and Distribution License (the "License").
\ You may not use this file except in compliance with the License.
\
\ You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
\ or http://www.opensolaris.org/os/licensing.
\ See the License for the specific language governing permissions
\ and limitations under the License.
\
\ When distributing Covered Code, include this CDDL HEADER in each
\ file and include the License file at usr/src/OPENSOLARIS.LICENSE.
\ If applicable, add the following below this CDDL HEADER, with the
\ fields enclosed by brackets "[]" replaced with your own identifying
\ information: Portions Copyright [yyyy] [name of copyright owner]
\
\ CDDL HEADER END
\

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/lx_brand.h>

lx_regs_t       SIZEOF_LX_REGS_T
        lxr_gs
        lxr_edi
        lxr_esi
        lxr_ebp
        lxr_esp
        lxr_ebx
        lxr_edx
        lxr_ecx
        lxr_eax
        lxr_eip
        lxr_orig_eax
#endif /* ! codereview */
```

```
**********************************************************
    1351 Tue Jan 14 16:17:07 2014
new/usr/src/lib/brand/lx/lx_brand/sys/lx_debug.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _LX_DEBUG_H
#define _LX_DEBUG_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

#ifdef  __cplusplus
extern "C" {
#endif

/* initialize the debugging subsystem */
extern void lx_debug_init(void);

/* printf() style debug message functionality */
extern void lx_debug(const char *, ...);

/* set non-zero if the debugging subsystem is enabled */
extern int lx_debug_enabled;

#ifdef  __cplusplus
}
#endif

#endif  /* _LX_DEBUG_H */
#endif /* ! codereview */
```

```
**********************************************************
    2633 Tue Jan 14 16:17:07 2014
new/usr/src/lib/brand/lx/lx_brand/sys/lx_fcntl.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _SYS_LX_FCNTL_H
#define _SYS_LX_FCNTL_H

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * Lx open/fcntl flags
 */
#define LX_O_RDONLY             00
#define LX_O_WRONLY             01
#define LX_O_RDWR               02
#define LX_O_CREAT              0100
#define LX_O_EXCL               0200
#define LX_O_NOCTTY             0400
#define LX_O_TRUNC              01000
#define LX_O_APPEND             02000
#define LX_O_NONBLOCK           04000
#define LX_O_NDELAY             LX_O_NONBLOCK
#define LX_O_SYNC               010000
#define LX_O_FSYNC              LX_O_SYNC
#define LX_O_ASYNC              020000
#define LX_O_DIRECT             040000
#define LX_O_LARGEFILE          0100000
#define LX_O_DIRECTORY          0200000
#define LX_O_NOFOLLOW           0400000

#define LX_F_DUPFD              0
#define LX_F_GETFD              1
#define LX_F_SETFD              2
#define LX_F_GETFL              3
#define LX_F_SETFL              4
#define LX_F_GETLK              5
#define LX_F_SETLK              6
#define LX_F_SETLKW             7
```



```
#define LX_F_SETOWN             8
#define LX_F_GETOWN             9
#define LX_F_SETSIG             10
#define LX_F_GETSIG             11

#define LX_F_GETLK64            12
#define LX_F_SETLK64            13
#define LX_F_SETLKW64           14

#define LX_F_SETLEASE           1024
#define LX_F_GETLEASE           1025
#define LX_F_NOTIFY             1026

#define LX_F_RDLCK              0
#define LX_F_WRLCK              1
#define LX_F_UNLCK              2

/*
 * Lx flock codes.
 */
#define LX_NAME_MAX             255
#define LX_LOCK_SH              1       /* shared */
#define LX_LOCK_EX              2       /* exclusive */
#define LX_LOCK_NB              4       /* non-blocking */
#define LX_LOCK_UN              8       /* unlock */

#define LX_AT_FDCWD             -100
#define LX_AT_EACCESS           512
#define LX_AT_REMOVEDIR         512
#define LX_AT_SYMLINK_NOFOLLOW  256
#define LX_AT_SYMLINK_FOLLOW    1024

struct lx_flock {
        short           l_type;
        short           l_whence;
        long            l_start;
        long            l_len;
        int             l_pid;
};

struct lx_flock64 {
        short           l_type;
        short           l_whence;
        long long       l_start;
        long long       l_len;
        int             l_pid;
};

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_LX_FCNTL_H */
#endif /* ! codereview */
```

```
**********************************************************
   11182 Tue Jan 14 16:17:07 2014
new/usr/src/lib/brand/lx/lx_brand/sys/lx_ioctl.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _SYS_LX_IOCTL_H
#define _SYS_LX_IOCTL_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

#ifdef __cplusplus
extern "C" {
#endif

extern int lx_ioctl_init(void);

/*
 * LX_NCC must be different from LX_NCCS since while the termio and termios
 * structures may look similar they are fundamentally different sizes and
 * have different members.
 */
#define LX_NCC  8
#define LX_NCCS 19

struct lx_termio {
        unsigned short c_iflag;         /* input mode flags */
        unsigned short c_oflag;         /* output mode flags */
        unsigned short c_cflag;         /* control mode flags */
        unsigned short c_lflag;         /* local mode flags */
        unsigned char c_line;           /* line discipline */
        unsigned char c_cc[LX_NCC];     /* control characters */
};

struct lx_termios {
        uint32_t c_iflag;               /* input mode flags */
        uint32_t c_oflag;               /* output mode flags */
        uint32_t c_cflag;               /* control mode flags */
        uint32_t c_lflag;               /* local mode flags */
        unsigned char c_line;           /* line discipline */
        unsigned char c_cc[LX_NCCS];    /* control characters */
};
```



```
/*
 * c_cc characters which are valid for lx_termio and lx_termios
 */
#define LX_VINTR        0
#define LX_VQUIT        1
#define LX_VERASE       2
#define LX_VKILL        3
#define LX_VEOF         4
#define LX_VTIME        5
#define LX_VMIN         6
#define LX_VSWTC        7

/*
 * c_cc characters which are valid for lx_termios
 */
#define LX_VSTART       8
#define LX_VSTOP        9
#define LX_VSUSP        10
#define LX_VEOL         11
#define LX_VREPRINT     12
#define LX_VDISCARD     13
#define LX_VWERASE      14
#define LX_VLNEXT       15
#define LX_VEOL2        16

/*
 * Sound formats
 */
#define LX_AFMT_QUERY           0x00000000
#define LX_AFMT_MU_LAW          0x00000001
#define LX_AFMT_A_LAW           0x00000002
#define LX_AFMT_IMA_ADPCM       0x00000004
#define LX_AFMT_U8              0x00000008
#define LX_AFMT_S16_LE          0x00000010
#define LX_AFMT_S16_BE          0x00000020
#define LX_AFMT_S8              0x00000040
#define LX_AFMT_U16_LE          0x00000080
#define LX_AFMT_U16_BE          0x00000100
#define LX_AFMT_MPEG            0x00000200
#define LX_AFMT_AC3             0x00000400

/*
 * Supported ioctls
 */
#define LX_TCGETS               0x5401
#define LX_TCSETS               0x5402
#define LX_TCSETSW              0x5403
#define LX_TCSETSF              0x5404
#define LX_TCGETA               0x5405
#define LX_TCSETA               0x5406
#define LX_TCSETAW              0x5407
#define LX_TCSETAF              0x5408
#define LX_TCSBRK               0x5409
#define LX_TCXONC               0x540a
#define LX_TCFLSH               0x540b
#define LX_TIOCEXCL             0x540c
#define LX_TIOCNXCL             0x540d
#define LX_TIOCSCTTY            0x540e
#define LX_TIOCGPGRP            0x540f
#define LX_TIOCSPGRP            0x5410
#define LX_TIOCOUTQ             0x5411
#define LX_TIOCSTI              0x5412
#define LX_TIOCGWINSZ           0x5413
#define LX_TIOCSWINSZ           0x5414
#define LX_TIOCMGET             0x5415
```

```
128 #define LX_TIOCMBIS             0x5416
129 #define LX_TIOCMBIC             0x5417
130 #define LX_TIOCMSET             0x5418
131 #define LX_TIOCGSOFTCAR         0x5419
132 #define LX_TIOCSSOFTCAR         0x541a
133 #define LX_FIONREAD             0x541b
134 #define LX_TIOCPKT              0x5420
135 #define LX_FIONBIO              0x5421
136 #define LX_TIOCNOTTY            0x5422
137 #define LX_TIOCSETD             0x5423
138 #define LX_TIOCGETD             0x5424
139 #define LX_TCSBRKP              0x5425
140 #define LX_TIOCGSID             0x5429
141 #define LX_TIOCGPTN             0x80045430
142 #define LX_TIOCSPTLCK           0x40045431
143 #define LX_FIONCLEX             0x5450
144 #define LX_FIOCLEX              0x5451
145 #define LX_FIOASYNC             0x5452
146 #define LX_FIOSETOWN            0x8901
147 #define LX_SIOCSPGRP            0x8902
148 #define LX_FIOGETOWN            0x8903
149 #define LX_SIOCGPGRP            0x8904
150 #define LX_SIOCATMARK           0x8905
151 #define LX_SIOCGIFCONF          0x8912
152 #define LX_SIOCGIFFLAGS         0x8913
153 #define LX_SIOCSIFFLAGS         0x8914
154 #define LX_SIOCGIFADDR          0x8915
155 #define LX_SIOCSIFADDR          0x8916
156 #define LX_SIOCGIFDSTADDR       0x8917
157 #define LX_SIOCSIFDSTADDR       0x8918
158 #define LX_SIOCGIFBRDADDR       0x8919
159 #define LX_SIOCSIFBRDADDR       0x891a
160 #define LX_SIOCGIFNETMASK       0x891b
161 #define LX_SIOCSIFNETMASK       0x891c
162 #define LX_SIOCGIFMETRIC        0x891d
163 #define LX_SIOCSIFMETRIC        0x891e
164 #define LX_SIOCGIFMEM           0x891f
165 #define LX_SIOCSIFMEM           0x8920
166 #define LX_SIOCGIFMTU           0x8921
167 #define LX_SIOCSIFMTU           0x8922
168 #define LX_SIOCSIFHWADDR        0x8924
169 #define LX_SIOCGIFHWADDR        0x8927

171 /*
172  * /dev/dsp ioctls - supported
173  */
174 #define LX_OSS_SNDCTL_DSP_RESET         0x5000
175 #define LX_OSS_SNDCTL_DSP_SYNC          0x5001
176 #define LX_OSS_SNDCTL_DSP_SPEED         0xc0045002
177 #define LX_OSS_SNDCTL_DSP_STEREO        0xc0045003
178 #define LX_OSS_SNDCTL_DSP_GETBLKSIZE    0xc0045004
179 #define LX_OSS_SNDCTL_DSP_SETFMTS       0xc0045005
180 #define LX_OSS_SNDCTL_DSP_CHANNELS      0xc0045006
181 #define LX_OSS_SNDCTL_DSP_SETFRAGMENT   0xc004500a
182 #define LX_OSS_SNDCTL_DSP_GETFMTS       0x8004500b
183 #define LX_OSS_SNDCTL_DSP_GETOSPACE     0x8010500c
184 #define LX_OSS_SNDCTL_DSP_GETCAPS       0x8004500f
185 #define LX_OSS_SNDCTL_DSP_SETTRIGGER    0x40045010
186 #define LX_OSS_SNDCTL_DSP_GETOPTR       0x800c5012
187 #define LX_OSS_SNDCTL_DSP_GETISPACE     0x8010500d

189 /*
190  * support for /dev/dsp SNDCTL_DSP_GETFMTS and SNDCTL_DSP_SETFMTS
191  */
192 #define LX_OSS_AFMT_QUERY               0x0000
193 #define LX_OSS_AFMT_MU_LAW              0x0001
```



```
194 #define LX_OSS_AFMT_A_LAW               0x0002
195 #define LX_OSS_AFMT_IMA_ADPCM           0x0004
196 #define LX_OSS_AFMT_U8                  0x0008
197 #define LX_OSS_AFMT_S16_LE              0x0010
198 #define LX_OSS_AFMT_S16_BE              0x0020
199 #define LX_OSS_AFMT_S8                  0x0040
200 #define LX_OSS_AFMT_U16_LE              0x0080
201 #define LX_OSS_AFMT_U16_BE              0x0100
202 #define LX_OSS_AFMT_MPEG                0x0200

204 #ifdef _LITTLE_ENDIAN
205 #define LX_OSS_AFMT_S16_NE              LX_OSS_AFMT_S16_LE
206 #define LX_OSS_AFMT_U16_NE              LX_OSS_AFMT_U16_LE
207 #elif defined(_BIG_ENDIAN)
208 #define LX_OSS_AFMT_S16_NE              LX_OSS_AFMT_S16_BE
209 #define LX_OSS_AFMT_U16_NE              LX_OSS_AFMT_U16_BE
210 #else /* _LITTLE_ENDIAN */
211 #error  NO ENDIAN defined.
212 #endif /* _LITTLE_ENDIAN */

214 /*
215  * support for /dev/dsp SNDCTL_DSP_GETISPACE and SNDCTL_DSP_GETOSPACE
216  */
217 typedef struct lx_oss_audio_buf_info {
218         int fragments;  /* fragments that can be rd/wr without blocking */
219         int fragstotal; /* total number of fragments allocated for buffering */
220         int fragsize;   /* size of fragments, same as SNDCTL_DSP_GETBLKSIZE */
221         int bytes;      /* what can be rd/wr immediatly without blocking */
222 } lx_oss_audio_buf_info_t;

224 /*
225  * support for /dev/dsp SNDCTL_DSP_GETOPTR
226  */
227 typedef struct lx_oss_count_info {
228         /* # of bytes processed since opening the device */
229         int bytes;

231         /*
232          * # of fragment transitions since last call to this function.
233          * only valid for mmap acess mode.
234          */
235         int blocks;

237         /*
238          * byte offset of the current recording/playback position from
239          * the beginning of the audio buffer.  only valid for mmap access
240          * mode.
241          */
242         int ptr;
243 } lx_oss_count_info_t;

245 /*
246  * support for /dev/dsp SNDCTL_DSP_GETCAPS
247  */
248 #define LX_OSS_DSP_CAP_TRIGGER          0x1000
249 #define LX_OSS_DSP_CAP_MMAP             0x2000

251 /*
252  * support for /dev/dsp/ SNDCTL_DSP_SETTRIGGER
253  */
254 #define LX_OSS_PCM_DISABLE_OUTPUT       0
255 #define LX_OSS_PCM_ENABLE_OUTPUT        2

257 /*
258  * /dev/mixer ioctl macros
259  */
```

```
#define LX_OSS_SM_NRDEVICES     25
#define LX_OSS_SM_READ(x)       (0x80044d00 | (x))
#define LX_OSS_SM_WRITE(x)      (0xc0044d00 | (x))

/*
 * /dev/mixer ioctls - supported
 */
#define LX_OSS_SOUND_MIXER_READ_VOLUME  LX_OSS_SM_READ(LX_OSS_SM_VOLUME)
#define LX_OSS_SOUND_MIXER_READ_PCM     LX_OSS_SM_READ(LX_OSS_SM_PCM)
#define LX_OSS_SOUND_MIXER_READ_MIC     LX_OSS_SM_READ(LX_OSS_SM_MIC)
#define LX_OSS_SOUND_MIXER_READ_IGAIN   LX_OSS_SM_READ(LX_OSS_SM_IGAIN)
#define LX_OSS_SOUND_MIXER_WRITE_VOLUME LX_OSS_SM_WRITE(LX_OSS_SM_VOLUME)
#define LX_OSS_SOUND_MIXER_WRITE_PCM    LX_OSS_SM_WRITE(LX_OSS_SM_PCM)
#define LX_OSS_SOUND_MIXER_WRITE_MIC    LX_OSS_SM_WRITE(LX_OSS_SM_MIC)
#define LX_OSS_SOUND_MIXER_WRITE_IGAIN  LX_OSS_SM_WRITE(LX_OSS_SM_IGAIN)
#define LX_OSS_SOUND_MIXER_READ_STEREODEVS LX_OSS_SM_READ(LX_OSS_SM_STEREODEVS)
#define LX_OSS_SOUND_MIXER_READ_RECMASK LX_OSS_SM_READ(LX_OSS_SM_RECMASK)
#define LX_OSS_SOUND_MIXER_READ_DEVMASK LX_OSS_SM_READ(LX_OSS_SM_DEVMASK)
#define LX_OSS_SOUND_MIXER_READ_RECSRC  LX_OSS_SM_READ(LX_OSS_SM_RECSRC)

/*
 * /dev/mixer channels
 */
#define LX_OSS_SM_VOLUME        0
#define LX_OSS_SM_BASS          1
#define LX_OSS_SM_TREBLE        2
#define LX_OSS_SM_SYNTH         3
#define LX_OSS_SM_PCM           4
#define LX_OSS_SM_SPEAKER       5
#define LX_OSS_SM_LINE          6
#define LX_OSS_SM_MIC           7
#define LX_OSS_SM_CD            8
#define LX_OSS_SM_MIX           9
#define LX_OSS_SM_PCM2          10
#define LX_OSS_SM_REC           11
#define LX_OSS_SM_IGAIN         12
#define LX_OSS_SM_OGAIN         13
#define LX_OSS_SM_LINE1         14
#define LX_OSS_SM_LINE2         15
#define LX_OSS_SM_LINE3         16
#define LX_OSS_SM_DIGITAL1      17
#define LX_OSS_SM_DIGITAL2      18
#define LX_OSS_SM_DIGITAL3      19
#define LX_OSS_SM_PHONEIN       20
#define LX_OSS_SM_PHONEOUT      21
#define LX_OSS_SM_VIDEO         22
#define LX_OSS_SM_RADIO         23
#define LX_OSS_SM_MONITOR       24

/*
 * /dev/mixer operations
 */
#define LX_OSS_SM_STEREODEVS    251
#define LX_OSS_SM_CAPS          252
#define LX_OSS_SM_RECMASK       253
#define LX_OSS_SM_DEVMASK       254
#define LX_OSS_SM_RECSRC        255

/*
 * /dev/mixer value conversion macros
 *
 * solaris expects gain level on a scale of 0 - 255
 * oss expects gain level on a scale of 0 - 100
 *
 * oss also encodes multiple channels volume values in a single int,
 * one channel value per byte.
```



```
 */
#define LX_OSS_S2L_GAIN(v)              (((v) * 100) / 255)
#define LX_OSS_L2S_GAIN(v)              (((v) * 255) / 100)
#define LX_OSS_MIXER_DEC1(v)            ((v) & 0xff)
#define LX_OSS_MIXER_DEC2(v)            (((v) >> 8) & 0xff)
#define LX_OSS_MIXER_ENC2(v1, v2)       (((v2) << 8) | (v1))

/*
 * /dev/mixer value verification macros
 */
#define LX_OSS_MIXER_VCHECK(x)  (((int)(x) >= 0) && ((int)(x) <= 100))
#define LX_OSS_MIXER_1CH_OK(x)  ((((x) & ~0xff) == 0) &&        \
        LX_OSS_MIXER_VCHECK(LX_OSS_MIXER_DEC1(x)))
#define LX_OSS_MIXER_2CH_OK(x)  ((((x) & ~0xffff) == 0) &&      \
        LX_OSS_MIXER_VCHECK(LX_OSS_MIXER_DEC1(x)) &&            \
        LX_OSS_MIXER_VCHECK(LX_OSS_MIXER_DEC2(x)))

/*
 * Unsupported ioctls (NOT a comprehensive list)
 */
#define LX_TIOCLINUX            0x541c
#define LX_TIOCCONS             0x541d
#define LX_TIOCGSERIAL          0x541e
#define LX_TIOCSSERIAL          0x541f
#define LX_TIOCTTYGSTRUCT       0x5426
#define LX_TIOCSERCONFIG        0x5453
#define LX_TIOCSERGWILD         0x5454
#define LX_TIOCSERSWILD         0x5455
#define LX_TIOCGLCKTRMIOS       0x5456
#define LX_TIOCSLCKTRMIOS       0x5457
#define LX_TIOCSERGSTRUCT       0x5458
#define LX_TIOCSERGETLSR        0x5459
#define LX_TIOCSERGETMULTI      0x545a
#define LX_TIOCSERSETMULTI      0x545b
#define LX_OLD_SIOCGIFHWADDR    0x8923
#define LX_SIOCSIFENCAP         0x8926
#define LX_SIOCGIFSLAVE         0x8929
#define LX_SIOCSIFSLAVE         0x8930
#define LX_SIOCADDMULTI         0x8931
#define LX_SIOCDELMULTI         0x8932
#define LX_SIOCADDRTOLD         0x8940
#define LX_SIOCDELRTOLD         0x8941
#define LX_SIOCGIFTXQLEN        0x8942
#define LX_SIOCDARP             0x8950
#define LX_SIOCGARP             0x8951
#define LX_SIOCSARP             0x8952
#define LX_SIOCDRARP            0x8960
#define LX_SIOCGRARP            0x8961
#define LX_SIOCSRARP            0x8962
#define LX_SIOCGIFMAP           0x8970
#define LX_SIOCSIFMAP           0x8971

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_LX_IOCTL_H */
#endif /* ! codereview */
```

```
**********************************************************
    4248 Tue Jan 14 16:17:07 2014
new/usr/src/lib/brand/lx/lx_brand/sys/lx_misc.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _SYS_LX_H
#define _SYS_LX_H

#include <stdio.h>
#include <alloca.h>
#include <sys/types.h>
#include <sys/param.h>
#include <sys/lwp.h>

#include <sys/lx_brand.h>

#ifdef __cplusplus
extern "C" {
#endif

extern char lx_release[128];
extern pid_t zoneinit_pid;

/*
 * Support for the unfortunate RPM race condition workaround.
 */
extern int lx_rpm_delay;
extern boolean_t lx_is_rpm;

/*
 * Values Linux expects for init
 */
#define LX_INIT_PGID    0
#define LX_INIT_SID     0
#define LX_INIT_PID     1

/*
 * Codes to reboot(2).
 */
#define LINUX_REBOOT_MAGIC1             0xfee1dead
```



```
#define LINUX_REBOOT_MAGIC2             672274793
#define LINUX_REBOOT_MAGIC2A            85072278
#define LINUX_REBOOT_MAGIC2B            369367448
#define LINUX_REBOOT_MAGIC2C            537993216

/*
 * This was observed as coming from Red Hat's init process, but it's not in
 * their reboot(2) man page.
 */
#define LINUX_REBOOT_MAGIC2D            0x28121969

#define LINUX_REBOOT_CMD_RESTART        0x1234567
#define LINUX_REBOOT_CMD_HALT           0xcdef0123
#define LINUX_REBOOT_CMD_POWER_OFF      0x4321fedc
#define LINUX_REBOOT_CMD_RESTART2       0xa1b2c3d4
#define LINUX_REBOOT_CMD_CAD_ON         0x89abcdef
#define LINUX_REBOOT_CMD_CAD_OFF        0

/*
 * the maximum length of messages to be output with lx_msg(), lx_err(),
 * lx_debug(), or lx_unsupported().
 */
#define LX_MSG_MAXLEN                   (128 + MAXPATHLEN)

/*
 * Linux scheduler priority ranges.
 */
#define LX_SCHED_PRIORITY_MIN_OTHER     0
#define LX_SCHED_PRIORITY_MAX_OTHER     0
#define LX_SCHED_PRIORITY_MIN_RRFIFO    1
#define LX_SCHED_PRIORITY_MAX_RRFIFO    99

/*
 * Constants to indicate who getrusage() should return information about.
 */
#define LX_RUSAGE_SELF          0
#define LX_RUSAGE_CHILDREN      (-1)

/*
 * normally we never want to write to stderr or stdout because it's unsafe
 * to make assumptions about the underlying file descriptors.  to protect
 * against writes to these file descriptors we go ahead and close them
 * our brand process initalization code.  but there are still occasions
 * where we are willing to make assumptions about our file descriptors
 * and write to them.  at thes times we should use one lx_msg() or
 * lx_msg_error()
 */
extern void lx_msg(char *, ...);
extern void lx_err(char *, ...);
extern void lx_err_fatal(char *, ...);
extern void lx_unsupported(char *, ...);

struct ucontext;

extern void lx_handler_table(void);
extern void lx_handler_trace_table(void);
extern void lx_emulate_done(void);
extern lx_regs_t *lx_syscall_regs(void);

extern char *lx_fd_to_path(int fd, char *buf, int buf_size);
extern int lx_lpid_to_spair(pid_t, pid_t *, lwpid_t *);
extern int lx_lpid_to_spid(pid_t, pid_t *);

extern int lx_ptrace_wait(siginfo_t *);
extern void lx_ptrace_fork(void);
```

```
128 extern int lx_get_kern_version(void);

130 extern int lx_check_alloca(size_t);
131 #define SAFE_ALLOCA(sz) (lx_check_alloca(sz) ? alloca(sz) : NULL)

133 extern int ltos_at_flag(int lflag, int allow);

135 /*
136  * NO_UUCOPY disables calls to the uucopy* system calls to help with
137  * debugging brand library accesses to linux application memory.
138  */
139 #ifdef NO_UUCOPY

141 int uucopy_unsafe(const void *src, void *dst, size_t n);
142 int uucopystr_unsafe(const void *src, void *dst, size_t n);

144 #define uucopy(src, dst, n)     uucopy_unsafe((src), (dst), (n))
145 #define uucopystr(src, dst, n)  uucopystr_unsafe((src), (dst), (n))

147 #endif /* NO_UUCOPY */

149 #ifdef  __cplusplus
150 }
151 #endif

153 #endif  /* _SYS_LX_H */
154 #endif /* ! codereview */
```

```
**********************************************************
    4195 Tue Jan 14 16:17:08 2014
new/usr/src/lib/brand/lx/lx_brand/sys/lx_mount.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _LX_MOUNT_H
#define _LX_MOUNT_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

#ifdef __cplusplus
extern "C" {
#endif

#include <rpc/rpc.h>
#include <nfs/nfs.h>

/*
 * mount() is significantly different between Linux and Solaris.  The main
 * difference is between the set of flags.  Some flags on Linux can be
 * translated to a Solaris equivalent, some are converted to a
 * filesystem-specific option, while others have no equivalent whatsoever.
 */
#define LX_MS_MGC_VAL           0xC0ED0000
#define LX_MS_RDONLY            0x00000001
#define LX_MS_NOSUID            0x00000002
#define LX_MS_NODEV             0x00000004
#define LX_MS_NOEXEC            0x00000008
#define LX_MS_SYNCHRONOUS       0x00000010
#define LX_MS_REMOUNT           0x00000020
#define LX_MS_MANDLOCK          0x00000040
#define LX_MS_NOATIME           0x00000400
#define LX_MS_NODIRATIME        0x00000800
#define LX_MS_BIND              0x00001000
#define LX_MS_SUPPORTED         (LX_MS_MGC_VAL | \
                                 LX_MS_RDONLY | LX_MS_NOSUID | \
                                 LX_MS_NODEV | LX_MS_NOEXEC | \
                                 LX_MS_REMOUNT | LX_MS_NOATIME | \
                                 LX_MS_BIND)

/*
```



```
 * support for nfs mounts
 */
#define LX_NMD_MAXHOSTNAMELEN           256

#define LX_NFS_MOUNT_SOFT               0x00000001
#define LX_NFS_MOUNT_INTR               0x00000002
#define LX_NFS_MOUNT_SECURE             0x00000004
#define LX_NFS_MOUNT_POSIX              0x00000008
#define LX_NFS_MOUNT_NOCTO              0x00000010
#define LX_NFS_MOUNT_NOAC               0x00000020
#define LX_NFS_MOUNT_TCP                0x00000040
#define LX_NFS_MOUNT_VER3               0x00000080
#define LX_NFS_MOUNT_KERBEROS           0x00000100
#define LX_NFS_MOUNT_NONLM              0x00000200
#define LX_NFS_MOUNT_BROKEN_SUID        0x00000400
#define LX_NFS_MOUNT_SUPPORTED          (LX_NFS_MOUNT_SOFT | \
                                         LX_NFS_MOUNT_INTR | \
                                         LX_NFS_MOUNT_POSIX | \
                                         LX_NFS_MOUNT_NOCTO | \
                                         LX_NFS_MOUNT_NOAC | \
                                         LX_NFS_MOUNT_TCP | \
                                         LX_NFS_MOUNT_VER3 | \
                                         LX_NFS_MOUNT_NONLM)

#define LX_NMD_DEFAULT_RSIZE            0
#define LX_NMD_DEFAULT_WSIZE            0

/*
 * the nfs v3 file handle structure definitions are _almost_ the same
 * on linux and solaris.  the key difference are:
 *
 * 1) on linux fh3_length is an unsigned short where as on solaris it's
 *    an int.
 *
 * 2) on linux the file handle data doesn't 32 bit members, so the structure
 *    is not 32 bit aligned.  (where as on solaris it is.)
 *
 * so rather than defining a structure that would allow us to intrepret
 * all the contents of the nfs v3 file handle here, we decide to treate
 * the file handle as an array of chars.  this works just fine since it
 * avoids the alignment issues and the actual file handle handle contects
 * are defined by the nfs specification so they are common across solaris
 * and linux.  we do the same thing for nfs v2 file handles.
 */
struct lx_nfs_fh2 {
        unsigned char   lx_fh_data[NFS_FHSIZE];
} lx_nfs_fh2;

struct lx_nfs_fh3 {
        unsigned short  lx_fh3_length;
        unsigned char   lx_fh3_data[NFS3_FHSIZE];
} lx_nfs_fh3;

typedef struct lx_nfs_mount_data {
        int                     nmd_version;
        int                     nmd_fd;
        struct lx_nfs_fh2       nmd_old_root;
        int                     nmd_flags;
        int                     nmd_rsize;
        int                     nmd_wsize;
        int                     nmd_timeo;
        int                     nmd_retrans;
        int                     nmd_acregmin;
        int                     nmd_acregmax;
        int                     nmd_acdirmin;
        int                     nmd_acdirmax;
```

```
128         struct sockaddr_in      nmd_addr;
129         char                    nmd_hostname[LX_NMD_MAXHOSTNAMELEN];
130         int                     nmd_namlen;
131         uint_t                  nmd_bsize;
132         struct lx_nfs_fh3       nmd_root;
133 } lx_nfs_mount_data_t;

135 #ifdef  __cplusplus
136 }
137 #endif

139 #endif  /* _LX_MOUNT_H */
140 #endif /* ! codereview */
```

```
**********************************************************
    1726 Tue Jan 14 16:17:08 2014
new/usr/src/lib/brand/lx/lx_brand/sys/lx_poll.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.        All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _SYS_LX_POLL_H
#define _SYS_LX_POLL_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

#ifdef __cplusplus
extern "C" {
#endif

/*
 * These events are identical between Linux and Solaris
 */
#define LX_POLLIN       0x001
#define LX_POLLPRI      0x002
#define LX_POLLOUT      0x004
#define LX_POLLERR      0x008
#define LX_POLLHUP      0x010
#define LX_POLLNVAL     0x020
#define LX_POLLRDNORM   0x040
#define LX_POLLRDBAND   0x080

#define LX_POLL_COMMON_EVENTS (LX_POLLIN | LX_POLLPRI | LX_POLLOUT |    \
        LX_POLLERR | LX_POLLHUP | LX_POLLNVAL | LX_POLLRDNORM | LX_POLLRDBAND)

/*
 * These events differ between Linux and Solaris
 */
#define LX_POLLWRNORM   0x100
#define LX_POLLWRBAND   0x200

#define LX_POLL_SUPPORTED_EVENTS        \
        (LX_POLL_COMMON_EVENTS | LX_POLLWRNORM | LX_POLLWRBAND)

#ifdef  __cplusplus
}
#endif
```



```
#endif  /* _SYS_LX_POLL_H */
#endif /* ! codereview */
```

```
**********************************************************
    7213 Tue Jan 14 16:17:08 2014
new/usr/src/lib/brand/lx/lx_brand/sys/lx_signal.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _SYS_LX_SIGNAL_H
#define _SYS_LX_SIGNAL_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

#if !defined(_ASM)
#include <sys/lx_types.h>
#include <lx_signum.h>

#endif  /* !defined(_ASM) */

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * Linux sigaction flags
 */
#define LX_SA_NOCLDSTOP         0x00000001
#define LX_SA_NOCLDWAIT         0x00000002
#define LX_SA_SIGINFO           0x00000004
#define LX_SA_RESTORER          0x04000000
#define LX_SA_ONSTACK           0x08000000
#define LX_SA_RESTART           0x10000000
#define LX_SA_NODEFER           0x40000000
#define LX_SA_RESETHAND         0x80000000
#define LX_SA_NOMASK            LX_SA_NODEFER
#define LX_SA_ONESHOT           LX_SA_RESETHAND

#define LX_SIG_BLOCK            0
#define LX_SIG_UNBLOCK          1
#define LX_SIG_SETMASK          2

#define LX_MINSIGSTKSZ          2048
#define LX_SS_ONSTACK           1
#define LX_SS_DISABLE           2
```



```
#define LX_SIGRT_MAGIC          0xdeadf00d

#if !defined(_ASM)

/*
 * NOTE:  Linux uses different definitions for sigset_ts and sigaction_ts
 *        depending on whether the definition is for user space or the kernel.
 *
 *        The definitions below MUST correspond to the Linux kernel versions,
 *        as glibc will do the necessary translation from the Linux user
 *        versions.
 */
typedef struct {
        ulong_t __bits[LX_NSIG_WORDS];
} lx_sigset_t;

#define LX_NBITS                (sizeof (ulong_t) * NBBY)
#define lx_sigmask(n)           (1UL << (((n) - 1) % LX_NBITS))
#define lx_sigword(n)           (((ulong_t)((n) - 1))>>5)
#define lx_sigismember(s, n)    (lx_sigmask(n) & (s)->__bits[lx_sigword(n)])
#define lx_sigaddset(s, n)      ((s)->__bits[lx_sigword(n)] |= lx_sigmask(n))

typedef struct lx_sigaction {
        void (*lxsa_handler)();
        int lxsa_flags;
        void (*lxsa_restorer)(void);
        lx_sigset_t lxsa_mask;
} lx_sigaction_t;

typedef uint32_t lx_osigset_t;

#define OSIGSET_NBITS           (sizeof (lx_osigset_t) * NBBY)
#define OSIGSET_BITSET(sig)     (1U << (((sig) - 1) % OSIGSET_NBITS))

/*
 * Flag settings to determine whether common routines should operate on
 * lx_sigset_ts or lx_osigset_ts.
 */
#define USE_OSIGSET     0
#define USE_SIGSET      1

typedef struct lx_osigaction {
        void (*lxsa_handler)();
        lx_osigset_t lxsa_mask;
        int lxsa_flags;
        void (*lxsa_restorer)(void);
} lx_osigaction_t;

#define LX_SI_MAX_SIZE  128
#define LX_SI_PAD_SIZE  ((LX_SI_MAX_SIZE/sizeof (int)) - 3)

typedef struct lx_siginfo {
        int lsi_signo;
        int lsi_errno;
        int lsi_code;
        union {
                int _pad[LX_SI_PAD_SIZE];

                struct {
                        pid_t _pid;
                        lx_uid16_t _uid;
                } _kill;

                struct {
                        uint_t _timer1;
```

```
128                     uint_t _timer2;
129             } _timer;

131             struct {
132                     pid_t _pid;             /* sender's pid */
133                     lx_uid16_t _uid;        /* sender's uid */
134                     union sigval _sigval;
135             } _rt;

137             struct {
138                     pid_t _pid;             /* which child */
139                     lx_uid16_t _uid;        /* sender's uid */
140                     int _status;            /* exit code */
141                     clock_t _utime;
142                     clock_t _stime;
143             } _sigchld;

145             struct {
146                     void *_addr;            /* faulting insn/memory ref. */
147             } _sigfault;

149             struct {
150                     int _band;              /* POLL_IN,POLL_OUT,POLL_MSG */
151                     int _fd;
152             } _sigpoll;
153     } _sifields;
154 } lx_siginfo_t;

156 /*
157  * lx_siginfo_t lsi_code values
158  *
159  *      LX_SI_ASYNCNL:  Sent by asynch name lookup completion
160  *      LX_SI_TKILL:    Sent by tkill
161  *      LX_SI_SIGIO:    Sent by queued SIGIO
162  *      LX_SI_ASYNCIO:  Sent by asynchronous I/O completion
163  *      LX_SI_MESGQ:    Sent by real time message queue state change
164  *      LX_SI_TIMER:    Sent by timer expiration
165  *      LX_SI_QUEUE:    Sent by sigqueue
166  *      LX_SI_USER:     Sent by kill, sigsend, raise, etc.
167  *      LX_SI_KERNEL:   Sent by kernel
168  *
169  * At present, LX_SI_ASYNCNL and LX_SI_SIGIO are unused by BrandZ.
170  */
171 #define LX_SI_ASYNCNL   (-60)
172 #define LX_SI_TKILL     (-6)
173 #define LX_SI_SIGIO     (-5)
174 #define LX_SI_ASYNCIO   (-4)
175 #define LX_SI_MESGQ     (-3)
176 #define LX_SI_TIMER     (-2)
177 #define LX_SI_QUEUE     (-1)
178 #define LX_SI_USER      (0)
179 #define LX_SI_KERNEL    (0x80)

181 typedef struct lx_sighandlers {
182         struct lx_sigaction lx_sa[LX_NSIG];
183 } lx_sighandlers_t;

185 typedef struct lx_sigaltstack {
186         void *ss_sp;
187         int ss_flags;
188         size_t ss_size;
189 } lx_stack_t;

191 struct lx_fpreg {
192         ushort_t significand[4];
193         ushort_t exponent;
```



```
194 };

196 struct lx_fpxreg {
197         ushort_t significand[4];
198         ushort_t exponent;
199         ushort_t padding[3];
200 };

202 struct lx_xmmreg {
203         uint32_t element[4];
204 };

206 #define LX_X86_FXSR_MAGIC       0x0000
207 #define LX_X86_FXSR_NONE        0xffff

209 typedef struct lx_fpstate {
210         /* Regular FPU environment */
211         ulong_t cw;
212         ulong_t sw;
213         ulong_t tag;
214         ulong_t ipoff;
215         ulong_t cssel;
216         ulong_t dataoff;
217         ulong_t datasel;
218         struct lx_fpreg _st[8];
219         ushort_t status;
220         ushort_t magic;                 /* 0xffff = regular FPU data */

222         /* FXSR FPU environment */
223         ulong_t _fxsr_env[6];           /* env is ignored */
224         ulong_t mxcsr;
225         ulong_t reserved;
226         struct lx_fpxreg _fxsr_st[8];  /* reg data is ignored */
227         struct lx_xmmreg _xmm[8];
228         ulong_t padding[56];
229 } lx_fpstate_t;

231 typedef struct lx_sigcontext {
232         ulong_t sc_gs;
233         ulong_t sc_fs;
234         ulong_t sc_es;
235         ulong_t sc_ds;
236         ulong_t sc_edi;
237         ulong_t sc_esi;
238         ulong_t sc_ebp;
239         ulong_t sc_esp;
240         ulong_t sc_ebx;
241         ulong_t sc_edx;
242         ulong_t sc_ecx;
243         ulong_t sc_eax;
244         ulong_t sc_trapno;
245         ulong_t sc_err;
246         ulong_t sc_eip;
247         ulong_t sc_cs;
248         ulong_t sc_eflags;
249         ulong_t sc_esp_at_signal;
250         ulong_t  sc_ss;
251         lx_fpstate_t *sc_fpstate;
252         ulong_t sc_mask;
253         ulong_t sc_cr2;
254 } lx_sigcontext_t;

256 typedef struct lx_ucontext {
257         ulong_t uc_flags;
258         struct lx_ucontext *uc_link;
259         lx_stack_t uc_stack;
```

```
260         lx_sigcontext_t uc_sigcontext;
261         lx_sigset_t uc_sigmask;
262 } lx_ucontext_t;

264 #define LX_SI_MAX_SIZE  128
265 #define LX_SI_PAD_SIZE  ((LX_SI_MAX_SIZE/sizeof (int)) - 3)

267 #define lsi_pid         _sifields._kill._pid
268 #define lsi_uid         _sifields._kill._uid
269 #define lsi_status      _sifields._sigchld._status
270 #define lsi_utime       _sifields._sigchld._utime
271 #define lsi_stime       _sifields._sigchld._stime
272 #define lsi_value       _sifields._rt._sigval
273 #define lsi_int         _sifields._rt._sigval.sivalx_int
274 #define lsi_ptr         _sifields._rt._sigval.sivalx_ptr
275 #define lsi_addr        _sifields._sigfault._addr
276 #define lsi_band        _sifields._sigpoll._band
277 #define lsi_fd          _sifields._sigpoll._fd

279 extern const int ltos_signo[];
280 extern const int stol_signo[];

282 extern void setsigacthandler(void (*)(int, siginfo_t *, void *),
283     void (**)(int, siginfo_t *, void *));

285 extern int lx_siginit(void);

287 extern void lx_sigreturn_tolibc(uintptr_t);
288 extern void lx_sigdeliver(int, siginfo_t *, void *, size_t, void (*)(),
289     void (*)(), uintptr_t);

291 extern int stol_siginfo(siginfo_t *siginfop, lx_siginfo_t *lx_siginfop);

293 #endif  /* !defined(_ASM) */

295 #ifdef  __cplusplus
296 }
297 #endif

299 #endif  /* _SYS_LX_SIGNAL_H */
300 #endif /* ! codereview */
```

```
**********************************************************
    7857 Tue Jan 14 16:17:08 2014
new/usr/src/lib/brand/lx/lx_brand/sys/lx_socket.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.        All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _SYS_LX_SOCKET_H
#define _SYS_LX_SOCKET_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

#ifdef __cplusplus
extern "C" {
#endif

#include <sys/lx_types.h>

/*
 * Linux address family definitions
 * Some of these are not supported
 */
#define LX_AF_UNSPEC            0  /* Unspecified */
#define LX_AF_UNIX              1  /* local file/pipe name */
#define LX_AF_INET              2  /* IP protocol family */
#define LX_AF_AX25              3  /* Amateur Radio AX.25 */
#define LX_AF_IPX               4  /* Novell Internet Protocol */
#define LX_AF_APPLETALK         5  /* Appletalk */
#define LX_AF_NETROM            6  /* Amateur radio */
#define LX_AF_BRIDGE            7  /* Multiprotocol bridge */
#define LX_AF_ATMPVC            8  /* ATM PVCs */
#define LX_AF_X25               9  /* X.25 */
#define LX_AF_INET6             10 /* IPV 6 */
#define LX_AF_ROSE              11 /* Amateur Radio X.25 */
#define LX_AF_DECnet            12 /* DECnet */
#define LX_AF_NETBEUI           13 /* 802.2LLC */
#define LX_AF_SECURITY          14 /* Security callback */
#define LX_AF_KEY               15 /* key management */
#define LX_AF_ROUTE             16 /* Alias to emulate 4.4BSD */
#define LX_AF_PACKET            17 /* Packet family */
#define LX_AF_ASH               18 /* Ash ? */
#define LX_AF_ECONET            19 /* Acorn Econet */
#define LX_AF_ATMSVC            20 /* ATM SVCs */
```



```
#define LX_AF_SNA               22 /* Linux SNA */
#define LX_AF_IRDA              23 /* IRDA sockets */
#define LX_AF_PPPOX             24 /* PPPoX sockets */
#define LX_AF_WANPIPE           25 /* Wanpipe API sockets */
#define LX_AF_BLUETOOTH         31 /* Bluetooth sockets */
#define LX_AF_MAX               32 /* MAX socket type  */

#define AF_NOTSUPPORTED         -1
#define AF_INVAL                -2

/*
 * Linux ARP protocol hardware identifiers
 */
#define LX_ARPHRD_ETHER         1       /* Ethernet */
#define LX_ARPHRD_LOOPBACK      772     /* Loopback */
#define LX_ARPHRD_VOID          0xffff  /* Unknown */

/*
 * Linux socket type definitions
 */
#define LX_SOCK_STREAM          1  /* Connection-based byte streams */
#define LX_SOCK_DGRAM           2  /* Connectionless, datagram */
#define LX_SOCK_RAW             3  /* Raw protocol interface */
#define LX_SOCK_RDM             4  /* Reliably-delivered message */
#define LX_SOCK_SEQPACKET       5  /* Sequenced packet stream */
#define LX_SOCK_PACKET          10 /* Linux specific */
#define LX_SOCK_MAX             11

#define SOCK_NOTSUPPORTED       -1
#define SOCK_INVAL              -2

/*
 * Options for use with [gs]etsockopt at the IP level.
 * IPPROTO_IP
 */
#define LX_IP_TOS               1
#define LX_IP_TTL               2
#define LX_IP_HDRINCL           3
#define LX_IP_OPTIONS           4
#define LX_IP_ROUTER_ALERT      5
#define LX_IP_RECVOPTS          6
#define LX_IP_RETOPTS           7
#define LX_IP_PKTINFO           8
#define LX_IP_PKTOPTIONS        9
#define LX_IP_MTU_DISCOVER      10
#define LX_IP_RECVERR           11
#define LX_IP_RECVTTL           12
#define LX_IP_RECVTOS           13
#define LX_IP_MTU               14
#define LX_IP_FREEBIND          15
#define LX_IP_MULTICAST_IF      32
#define LX_IP_MULTICAST_TTL     33
#define LX_IP_MULTICAST_LOOP    34
#define LX_IP_ADD_MEMBERSHIP    35
#define LX_IP_DROP_MEMBERSHIP   36

/*
 * Options for use with [gs]etsockopt at the TCP level.
 * IPPROTO_TCP
 */
#define LX_TCP_NODELAY          1  /* Don't delay send to coalesce packets  */
#define LX_TCP_MAXSEG           2  /* Set maximum segment size  */
#define LX_TCP_CORK             3  /* Control sending of partial frames  */
#define LX_TCP_KEEPIDLE         4  /* Start keeplives after this period */
#define LX_TCP_KEEPINTVL        5  /* Interval between keepalives */
#define LX_TCP_KEEPCNT          6  /* Number of keepalives before death */
```

```
#define LX_TCP_SYNCNT           7  /* Number of SYN retransmits */
#define LX_TCP_LINGER2          8  /* Life time of orphaned FIN-WAIT-2 state */
#define LX_TCP_DEFER_ACCEPT     9  /* Wake up listener only when data arrive */
#define LX_TCP_WINDOW_CLAMP     10 /* Bound advertised window */
#define LX_TCP_INFO             11 /* Information about this connection. */
#define LX_TCP_QUICKACK         12 /* Bock/reenable quick ACKs.  */

/*
 * Options for use with [gs]etsockopt at the IGMP level.
 * IPPROTO_IGMP
 */
#define LX_IGMP_MINLEN                          8
#define LX_IGMP_MAX_HOST_REPORT_DELAY           10
#define LX_IGMP_HOST_MEMBERSHIP_QUERY           0x11
#define LX_IGMP_HOST_MEMBERSHIP_REPORT          0x12
#define LX_IGMP_DVMRP                           0x13
#define LX_IGMP_PIM                             0x14
#define LX_IGMP_TRACE                           0x15
#define LX_IGMP_HOST_NEW_MEMBERSHIP_REPORT      0x16
#define LX_IGMP_HOST_LEAVE_MESSAGE              0x17
#define LX_IGMP_MTRACE_RESP                     0x1e
#define LX_IGMP_MTRACE                          0x1f

/*
 * Options for use with [gs]etsockopt at the SOL_SOCKET level.
 */
#define LX_SOL_SOCKET                           1

#define LX_SCM_RIGHTS                           1
#define LX_SCM_CRED                             2

#define LX_SO_DEBUG                             1
#define LX_SO_REUSEADDR                         2
#define LX_SO_TYPE                              3
#define LX_SO_ERROR                             4
#define LX_SO_DONTROUTE                         5
#define LX_SO_BROADCAST                         6
#define LX_SO_SNDBUF                            7
#define LX_SO_RCVBUF                            8
#define LX_SO_KEEPALIVE                         9
#define LX_SO_OOBINLINE                         10
#define LX_SO_NO_CHECK                          11
#define LX_SO_PRIORITY                          12
#define LX_SO_LINGER                            13
#define LX_SO_BSDCOMPAT                         14
/* To add :#define      LX_SO_REUSEPORT 15 */
#define LX_SO_PASSCRED                          16
#define LX_SO_PEERCRED                          17
#define LX_SO_RCVLOWAT                          18
#define LX_SO_SNDLOWAT                          19
#define LX_SO_RCVTIMEO                          20
#define LX_SO_SNDTIMEO                          21
/* Security levels - as per NRL IPv6 - don't actually do anything */
#define LX_SO_SECURITY_AUTHENTICATION           22
#define LX_SO_SECURITY_ENCRYPTION_TRANSPORT     23
#define LX_SO_SECURITY_ENCRYPTION_NETWORK       24
#define LX_SO_BINDTODEVICE                      25
/* Socket filtering */
#define LX_SO_ATTACH_FILTER                     26
#define LX_SO_DETACH_FILTER                     27
#define LX_SO_PEERNAME                          28
#define LX_SO_TIMESTAMP                         29
#define LX_SCM_TIMESTAMP                        LX_SO_TIMESTAMP
#define LX_SO_ACCEPTCONN                        30

/*
```



```
 * Linux socketcall indices.
 * These constitute all 17 socket related system calls
 *
 * These system calls are called via a single system call socketcall().
 * The first arg being the endex of the system call type
 */
#define LX_SOCKET               1
#define LX_BIND                 2
#define LX_CONNECT              3
#define LX_LISTEN               4
#define LX_ACCEPT               5
#define LX_GETSOCKNAME          6
#define LX_GETPEERNAME          7
#define LX_SOCKETPAIR           8
#define LX_SEND                 9
#define LX_RECV                 10
#define LX_SENDTO               11
#define LX_RECVFROM             12
#define LX_SHUTDOWN             13
#define LX_SETSOCKOPT           14
#define LX_GETSOCKOPT           15
#define LX_SENDMSG              16
#define LX_RECVMSG              17

/*
 * Linux socket flags for use with recv(2)/send(2)/recvmsg(2)/sendmsg(2)
 */
#define LX_MSG_OOB              1
#define LX_MSG_PEEK             2
#define LX_MSG_DONTROUTE        4
#define LX_MSG_CTRUNC           8
#define LX_MSG_PROXY            0x10
#define LX_MSG_TRUNC            0x20
#define LX_MSG_DONTWAIT         0x40
#define LX_MSG_EOR              0x80
#define LX_MSG_WAITALL          0x100
#define LX_MSG_FIN              0x200
#define LX_MSG_SYN              0x400
#define LX_MSG_CONFIRM          0x800
#define LX_MSG_RST              0x1000
#define LX_MSG_ERRQUEUE         0x2000
#define LX_MSG_NOSIGNAL         0x4000
#define LX_MSG_MORE             0x8000

struct lx_msghdr {
        void            *msg_name;              /* optional address */
        socklen_t       msg_namelen;            /* size of address */
        struct iovec    *msg_iov;               /* scatter/gather array */
        int             msg_iovlen;             /* # elements in msg_iov */
        void            *msg_control;           /* ancillary data */
        socklen_t       msg_controllen;         /* ancillary data buffer len */
        int             msg_flags;              /* flags on received message */
};

struct lx_ucred {
        pid_t           lxu_pid;
        lx_uid_t        lxu_uid;
        lx_gid_t        lxu_gid;
};

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_LX_SOCKET_H */
#endif /* ! codereview */
```

```
**********************************************************
    2337 Tue Jan 14 16:17:08 2014
new/usr/src/lib/brand/lx/lx_brand/sys/lx_stat.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _SYS_LX_STAT_H
#define _SYS_LX_STAT_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

#ifdef  __cplusplus
extern "C" {
#endif

#include <sys/lx_types.h>
#include <sys/stat.h>

#define LX_MAJORSHIFT           8
#define LX_MINORMASK            ((1 << LX_MAJORSHIFT) - 1)
#define LX_MAKEDEVICE(lx_maj, lx_min)   \
        ((lx_dev_t)((lx_maj) << LX_MAJORSHIFT | ((lx_min) & LX_MINORMASK)))

#define LX_GETMAJOR(lx_dev)     ((lx_dev) >> LX_MAJORSHIFT)
#define LX_GETMINOR(lx_dev)     ((lx_dev) & LX_MINORMASK)

#undef st_atime
#undef st_mtime
#undef st_ctime

struct lx_stat {
        lx_dev16_t              st_dev;
        uint16_t                st_pad1;
        lx_ino_t                st_ino;
        lx_mode16_t             st_mode;
        uint16_t                st_nlink;
        lx_uid16_t              st_uid;
        lx_gid16_t              st_gid;
        lx_dev16_t              st_rdev;
        uint16_t                st_pad2;
        lx_off_t                st_size;
        lx_blksize_t            st_blksize;
```



```
        lx_blkcnt_t             st_blocks;
        struct lx_timespec      st_atime;
        struct lx_timespec      st_mtime;
        struct lx_timespec      st_ctime;
        uint32_t                st_pad3;
        uint32_t                st_pad4;
};

struct lx_stat64 {
        lx_dev_t                st_dev;
        uint32_t                st_pad1;
        lx_ino_t                st_small_ino;
        lx_mode_t               st_mode;
        uint_t                  st_nlink;
        lx_uid_t                st_uid;
        lx_gid_t                st_gid;
        lx_dev_t                st_rdev;
        uint32_t                st_pad2;
        lx_off64_t              st_size;
        lx_blksize_t            st_blksize;
        lx_blkcnt64_t           st_blocks;
        struct lx_timespec      st_atime;
        struct lx_timespec      st_mtime;
        struct lx_timespec      st_ctime;
        lx_ino64_t              st_ino;
};

extern int lx_stat_init(void);

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_LX_STAT_H */
#endif /* ! codereview */
```

```
**********************************************************
    1881 Tue Jan 14 16:17:09 2014
new/usr/src/lib/brand/lx/lx_brand/sys/lx_statfs.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _LX_STATFS_H
#define _LX_STATFS_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

#ifdef  __cplusplus
extern "C" {
#endif

extern int lx_statfs_init(void);

struct lx_statfs {
        int             f_type;
        int             f_bsize;
        ulong_t         f_blocks;
        ulong_t         f_bfree;
        ulong_t         f_bavail;
        ulong_t         f_files;
        ulong_t         f_ffree;
        u_longlong_t    f_fsid;
        int             f_namelen;
        int             f_frsize;
        int             f_spare[5];
};

struct lx_statfs64 {
        int             f_type;
        int             f_bsize;
        u_longlong_t    f_blocks;
        u_longlong_t    f_bfree;
        u_longlong_t    f_bavail;
        u_longlong_t    f_files;
        u_longlong_t    f_ffree;
        u_longlong_t    f_fsid;
        int             f_namelen;
        int             f_frsize;
```



```
        int             f_spare[5];
};

/*
 * These magic values are taken mostly from statfs(2).
 */
#define LX_ISOFS_SUPER_MAGIC            0x9660
#define LX_NFS_SUPER_MAGIC              0x6969
#define LX_MSDOS_SUPER_MAGIC            0x4d44
#define LX_PROC_SUPER_MAGIC             0x9fa0
#define LX_UFS_MAGIC                    0x00011954
#define LX_DEVPTS_SUPER_MAGIC           0x1cd1

#ifdef  __cplusplus
}
#endif

#endif  /* _LX_STATFS_H */
#endif /* ! codereview */
```

```
**********************************************************
   18562 Tue Jan 14 16:17:09 2014
new/usr/src/lib/brand/lx/lx_brand/sys/lx_syscall.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _SYS_LX_SYSCALL_H
#define _SYS_LX_SYSCALL_H

#if !defined(_ASM)

#include <sys/types.h>
#include <sys/procset.h>

#ifdef __cplusplus
extern "C" {
#endif

extern int lx_install;

extern int lx_openat(uintptr_t, uintptr_t, uintptr_t, uintptr_t);
extern int lx_mkdirat(uintptr_t, uintptr_t, uintptr_t);
extern int lx_mknodat(uintptr_t, uintptr_t, uintptr_t, uintptr_t);
extern int lx_fchownat(uintptr_t, uintptr_t, uintptr_t, uintptr_t, uintptr_t);
extern int lx_futimesat(uintptr_t, uintptr_t, uintptr_t);
extern int lx_fstatat64(uintptr_t, uintptr_t, uintptr_t, uintptr_t);
extern int lx_unlinkat(uintptr_t, uintptr_t, uintptr_t);
extern int lx_renameat(uintptr_t, uintptr_t, uintptr_t, uintptr_t);
extern int lx_linkat(uintptr_t, uintptr_t, uintptr_t, uintptr_t, uintptr_t);
extern int lx_symlinkat(uintptr_t, uintptr_t, uintptr_t);
extern int lx_readlinkat(uintptr_t, uintptr_t, uintptr_t, uintptr_t);
extern int lx_fchmodat(uintptr_t, uintptr_t, uintptr_t, uintptr_t);
extern int lx_faccessat(uintptr_t, uintptr_t, uintptr_t, uintptr_t);

extern int lx_stat(uintptr_t, uintptr_t);
extern int lx_fstat(uintptr_t, uintptr_t);
extern int lx_lstat(uintptr_t, uintptr_t);
extern int lx_stat64(uintptr_t, uintptr_t);
extern int lx_fstat64(uintptr_t, uintptr_t);
extern int lx_lstat64(uintptr_t, uintptr_t);
extern int lx_fcntl(uintptr_t, uintptr_t, uintptr_t);
```



```
extern int lx_fcntl64(uintptr_t, uintptr_t, uintptr_t);
extern int lx_flock(uintptr_t, uintptr_t);
extern int lx_open(uintptr_t, uintptr_t, uintptr_t);
extern int lx_readdir(uintptr_t, uintptr_t, uintptr_t);
extern int lx_getdents64(uintptr_t, uintptr_t, uintptr_t);
extern int lx_getpid(void);
extern int lx_execve(uintptr_t, uintptr_t, uintptr_t);
extern int lx_dup2(uintptr_t, uintptr_t);
extern int lx_ioctl(uintptr_t, uintptr_t, uintptr_t);
extern int lx_vhangup(void);

extern int lx_read(uintptr_t, uintptr_t, uintptr_t);
extern int lx_readv(uintptr_t, uintptr_t, uintptr_t);
extern int lx_writev(uintptr_t, uintptr_t, uintptr_t);
extern int lx_pread64(uintptr_t, uintptr_t, uintptr_t, uintptr_t, uintptr_t);
extern int lx_pwrite64(uintptr_t, uintptr_t, uintptr_t, uintptr_t, uintptr_t);

extern int lx_socketcall(uintptr_t, uintptr_t);
extern int lx_select(uintptr_t, uintptr_t, uintptr_t, uintptr_t, uintptr_t);
extern int lx_poll(uintptr_t, uintptr_t, uintptr_t);
extern int lx_oldgetrlimit(uintptr_t, uintptr_t);
extern int lx_getrlimit(uintptr_t, uintptr_t);
extern int lx_setrlimit(uintptr_t, uintptr_t);
extern int lx_gettimeofday(uintptr_t, uintptr_t);
extern int lx_settimeofday(uintptr_t, uintptr_t);
extern int lx_getrusage(uintptr_t, uintptr_t);
extern int lx_mknod(uintptr_t, uintptr_t, uintptr_t);

extern int lx_getpgrp(void);
extern int lx_getpgid(uintptr_t);
extern int lx_setpgid(uintptr_t, uintptr_t);
extern int lx_getsid(uintptr_t);
extern int lx_setsid(void);
extern int lx_setgroups(uintptr_t, uintptr_t);


extern int lx_waitpid(uintptr_t, uintptr_t, uintptr_t);
extern int lx_waitid(uintptr_t, uintptr_t, uintptr_t, uintptr_t);
extern int lx_wait4(uintptr_t, uintptr_t, uintptr_t, uintptr_t);

extern int lx_getuid16(void);
extern int lx_getgid16(void);
extern int lx_geteuid16(void);
extern int lx_getegid16(void);
extern int lx_geteuid(void);
extern int lx_getegid(void);
extern int lx_getresuid16(uintptr_t, uintptr_t, uintptr_t);
extern int lx_getresgid16(uintptr_t, uintptr_t, uintptr_t);
extern int lx_getresuid(uintptr_t, uintptr_t, uintptr_t);
extern int lx_getresgid(uintptr_t, uintptr_t, uintptr_t);

extern int lx_setuid16(uintptr_t);
extern int lx_setreuid16(uintptr_t, uintptr_t);
extern int lx_setregid16(uintptr_t, uintptr_t);
extern int lx_setgid16(uintptr_t);
extern int lx_setfsuid16(uintptr_t);
extern int lx_setfsgid16(uintptr_t);

extern int lx_setfsuid(uintptr_t);
extern int lx_setfsgid(uintptr_t);

extern int lx_clock_settime(int, struct timespec *);
extern int lx_clock_gettime(int, struct timespec *);
extern int lx_clock_getres(int, struct timespec *);
extern int lx_clock_nanosleep(int, int flags, struct timespec *,
    struct timespec *);
```

```
129 extern int lx_truncate(uintptr_t, uintptr_t);
130 extern int lx_ftruncate(uintptr_t, uintptr_t);
131 extern int lx_truncate64(uintptr_t, uintptr_t, uintptr_t);
132 extern int lx_ftruncate64(uintptr_t, uintptr_t, uintptr_t);

134 extern int lx_sysctl(uintptr_t);
135 extern int lx_fsync(uintptr_t);
136 extern int lx_fdatasync(uintptr_t);
137 extern int lx_pipe(uintptr_t);
138 extern int lx_link(uintptr_t, uintptr_t);
139 extern int lx_unlink(uintptr_t);
140 extern int lx_rmdir(uintptr_t);
141 extern int lx_chown16(uintptr_t, uintptr_t, uintptr_t);
142 extern int lx_fchown16(uintptr_t, uintptr_t, uintptr_t);
143 extern int lx_lchown16(uintptr_t, uintptr_t, uintptr_t);
144 extern int lx_chown(uintptr_t, uintptr_t, uintptr_t);
145 extern int lx_fchown(uintptr_t, uintptr_t, uintptr_t);
146 extern int lx_chmod(uintptr_t, uintptr_t);
147 extern int lx_rename(uintptr_t, uintptr_t);
148 extern int lx_utime(uintptr_t, uintptr_t);
149 extern int lx_llseek(uintptr_t, uintptr_t, uintptr_t, uintptr_t, uintptr_t);
150 extern int lx_lseek(uintptr_t, uintptr_t, uintptr_t);
151 extern int lx_sysfs(uintptr_t, uintptr_t, uintptr_t);

153 extern int lx_getcwd(uintptr_t, uintptr_t);
154 extern int lx_uname(uintptr_t);
155 extern int lx_reboot(uintptr_t, uintptr_t, uintptr_t, uintptr_t);
156 extern int lx_getgroups16(uintptr_t, uintptr_t);
157 extern int lx_setgroups16(uintptr_t, uintptr_t);
158 extern int lx_personality(uintptr_t);

160 extern int lx_query_module(uintptr_t, uintptr_t, uintptr_t, uintptr_t,
161     uintptr_t);

163 extern int lx_time(uintptr_t);
164 extern int lx_times(uintptr_t);
165 extern int lx_setitimer(uintptr_t, uintptr_t, uintptr_t);

167 extern int lx_clone(uintptr_t, uintptr_t, uintptr_t, uintptr_t, uintptr_t);
168 extern int lx_exit(uintptr_t);
169 extern int lx_group_exit(uintptr_t);

171 extern int lx_mlock(uintptr_t, uintptr_t);
172 extern int lx_mlockall(uintptr_t);
173 extern int lx_munlock(uintptr_t, uintptr_t);
174 extern int lx_munlockall(void);
175 extern int lx_msync(uintptr_t, uintptr_t, uintptr_t);
176 extern int lx_madvise(uintptr_t, uintptr_t, uintptr_t);
177 extern int lx_mprotect(uintptr_t, uintptr_t, uintptr_t);
178 extern int lx_mmap(uintptr_t, uintptr_t, uintptr_t, uintptr_t, uintptr_t,
179     uintptr_t);
180 extern int lx_mmap2(uintptr_t, uintptr_t, uintptr_t, uintptr_t, uintptr_t,
181     uintptr_t);

183 extern int lx_mount(uintptr_t, uintptr_t, uintptr_t, uintptr_t, uintptr_t);
184 extern int lx_umount(uintptr_t);
185 extern int lx_umount2(uintptr_t, uintptr_t);

187 extern int lx_statfs(uintptr_t, uintptr_t);
188 extern int lx_fstatfs(uintptr_t, uintptr_t);
189 extern int lx_statfs64(uintptr_t, uintptr_t, uintptr_t);
190 extern int lx_fstatfs64(uintptr_t, uintptr_t, uintptr_t);

192 extern int lx_sigreturn(void);
193 extern int lx_rt_sigreturn(void);
```



```
194 extern int lx_signal(uintptr_t, uintptr_t);
195 extern int lx_sigaction(uintptr_t, uintptr_t, uintptr_t);
196 extern int lx_rt_sigaction(uintptr_t, uintptr_t, uintptr_t, uintptr_t);
197 extern int lx_sigaltstack(uintptr_t, uintptr_t);
198 extern int lx_sigpending(uintptr_t);
199 extern int lx_rt_sigpending(uintptr_t, uintptr_t);
200 extern int lx_sigprocmask(uintptr_t, uintptr_t, uintptr_t);
201 extern int lx_rt_sigprocmask(uintptr_t, uintptr_t, uintptr_t, uintptr_t);
202 extern int lx_sigsuspend(uintptr_t);
203 extern int lx_rt_sigsuspend(uintptr_t, uintptr_t);
204 extern int lx_sigwaitinfo(uintptr_t, uintptr_t);
205 extern int lx_rt_sigwaitinfo(uintptr_t, uintptr_t, uintptr_t);
206 extern int lx_sigtimedwait(uintptr_t, uintptr_t, uintptr_t);
207 extern int lx_rt_sigtimedwait(uintptr_t, uintptr_t, uintptr_t, uintptr_t);

209 extern int lx_sync(void);

211 extern int lx_futex(uintptr_t, uintptr_t, uintptr_t, uintptr_t, uintptr_t,
212     uintptr_t);

214 extern int lx_tkill(uintptr_t, uintptr_t, uintptr_t, uintptr_t, uintptr_t,
215     uintptr_t);
216 extern int lx_tgkill(uintptr_t, uintptr_t, uintptr_t);

218 extern int lx_sethostname(uintptr_t, uintptr_t);
219 extern int lx_setdomainname(uintptr_t, uintptr_t);

221 extern int lx_sendfile(uintptr_t, uintptr_t, uintptr_t, uintptr_t);
222 extern int lx_sendfile64(uintptr_t, uintptr_t, uintptr_t, uintptr_t);

224 extern int lx_fork(void);
225 extern int lx_vfork(void);
226 extern int lx_exec(uintptr_t, uintptr_t, uintptr_t);

228 extern int lx_getpriority(uintptr_t, uintptr_t);
229 extern int lx_setpriority(uintptr_t, uintptr_t, uintptr_t);

231 extern int lx_ptrace(uintptr_t, uintptr_t, uintptr_t, uintptr_t);

233 extern int lx_sched_getaffinity(uintptr_t, uintptr_t, uintptr_t);
234 extern int lx_sched_setaffinity(uintptr_t, uintptr_t, uintptr_t);
235 extern int lx_sched_getparam(uintptr_t, uintptr_t);
236 extern int lx_sched_setparam(uintptr_t, uintptr_t);
237 extern int lx_sched_rr_get_interval(uintptr_t pid, uintptr_t);
238 extern int lx_sched_getscheduler(uintptr_t);
239 extern int lx_sched_setscheduler(uintptr_t, uintptr_t, uintptr_t);
240 extern int lx_sched_get_priority_min(uintptr_t);
241 extern int lx_sched_get_priority_max(uintptr_t);

243 extern int lx_keyctl(void);

245 extern int lx_ipc(uintptr_t, uintptr_t, uintptr_t, uintptr_t, uintptr_t);

247 #endif  /* !defined(_ASM) */

249 #define EBP_HAS_ARG6            0x01

251 /*
252  * Linux syscall numbers
253  */
254 #define LX_SYS_exit               1
255 #define LX_SYS_fork               2
256 #define LX_SYS_read               3
257 #define LX_SYS_write              4
258 #define LX_SYS_open               5
259 #define LX_SYS_close              6
```

```
260 #define LX_SYS_waitpid            7
261 #define LX_SYS_creat              8
262 #define LX_SYS_link               9
263 #define LX_SYS_unlink            10
264 #define LX_SYS_execve            11
265 #define LX_SYS_chdir             12
266 #define LX_SYS_time              13
267 #define LX_SYS_mknod             14
268 #define LX_SYS_chmod             15
269 #define LX_SYS_lchown            16
270 #define LX_SYS_break             17
271 #define LX_SYS_oldstat           18
272 #define LX_SYS_lseek             19
273 #define LX_SYS_getpid            20
274 #define LX_SYS_mount             21
275 #define LX_SYS_umount            22
276 #define LX_SYS_setuid            23
277 #define LX_SYS_getuid            24
278 #define LX_SYS_stime             25
279 #define LX_SYS_ptrace            26
280 #define LX_SYS_alarm             27
281 #define LX_SYS_oldfstat          28
282 #define LX_SYS_pause             29
283 #define LX_SYS_utime             30
284 #define LX_SYS_stty              31
285 #define LX_SYS_gtty              32
286 #define LX_SYS_access            33
287 #define LX_SYS_nice              34
288 #define LX_SYS_ftime             35
289 #define LX_SYS_sync              36
290 #define LX_SYS_kill              37
291 #define LX_SYS_rename            38
292 #define LX_SYS_mkdir             39
293 #define LX_SYS_rmdir             40
294 #define LX_SYS_dup               41
295 #define LX_SYS_pipe              42
296 #define LX_SYS_times             43
297 #define LX_SYS_prof              44
298 #define LX_SYS_brk               45
299 #define LX_SYS_setgid            46
300 #define LX_SYS_getgid            47
301 #define LX_SYS_signal            48
302 #define LX_SYS_geteuid           49
303 #define LX_SYS_getegid           50
304 #define LX_SYS_acct              51
305 #define LX_SYS_umount2           52
306 #define LX_SYS_lock              53
307 #define LX_SYS_ioctl             54
308 #define LX_SYS_fcntl             55
309 #define LX_SYS_mpx               56
310 #define LX_SYS_setpgid           57
311 #define LX_SYS_ulimit            58
312 #define LX_SYS_oldolduname       59
313 #define LX_SYS_umask             60
314 #define LX_SYS_chroot            61
315 #define LX_SYS_ustat             62
316 #define LX_SYS_dup2              63
317 #define LX_SYS_getppid           64
318 #define LX_SYS_getpgrp           65
319 #define LX_SYS_setsid            66
320 #define LX_SYS_sigaction         67
321 #define LX_SYS_sgetmask          68
322 #define LX_SYS_ssetmask          69
323 #define LX_SYS_setreuid          70
324 #define LX_SYS_setregid          71
325 #define LX_SYS_sigsuspend        72
```



```
326 #define LX_SYS_sigpending        73
327 #define LX_SYS_sethostname       74
328 #define LX_SYS_setrlimit         75
329 #define LX_SYS_getrlimit         76
330 #define LX_SYS_getrusage         77
331 #define LX_SYS_gettimeofday      78
332 #define LX_SYS_settimeofday      79
333 #define LX_SYS_getgroups         80
334 #define LX_SYS_setgroups         81
335 #define LX_SYS_select            82
336 #define LX_SYS_symlink           83
337 #define LX_SYS_oldlstat          84
338 #define LX_SYS_readlink          85
339 #define LX_SYS_uselib            86
340 #define LX_SYS_swapon            87
341 #define LX_SYS_reboot            88
342 #define LX_SYS_readdir           89
343 #define LX_SYS_mmap              90
344 #define LX_SYS_munmap            91
345 #define LX_SYS_truncate          92
346 #define LX_SYS_ftruncate         93
347 #define LX_SYS_fchmod            94
348 #define LX_SYS_fchown            95
349 #define LX_SYS_getpriority       96
350 #define LX_SYS_setpriority       97
351 #define LX_SYS_profil            98
352 #define LX_SYS_statfs            99
353 #define LX_SYS_fstatfs          100
354 #define LX_SYS_ioperm           101
355 #define LX_SYS_socketcall       102
356 #define LX_SYS_syslog           103
357 #define LX_SYS_setitimer        104
358 #define LX_SYS_getitimer        105
359 #define LX_SYS_stat             106
360 #define LX_SYS_lstat            107
361 #define LX_SYS_fstat            108
362 #define LX_SYS_olduname         109
363 #define LX_SYS_iopl             110
364 #define LX_SYS_vhangup          111
365 #define LX_SYS_idle             112
366 #define LX_SYS_vm86old          113
367 #define LX_SYS_wait4            114
368 #define LX_SYS_swapoff          115
369 #define LX_SYS_sysinfo          116
370 #define LX_SYS_ipc              117
371 #define LX_SYS_fsync            118
372 #define LX_SYS_sigreturn        119
373 #define LX_SYS_clone            120
374 #define LX_SYS_setdomainname    121
375 #define LX_SYS_uname            122
376 #define LX_SYS_modify_ldt       123
377 #define LX_SYS_adjtimex         124
378 #define LX_SYS_mprotect         125
379 #define LX_SYS_sigprocmask      126
380 #define LX_SYS_create_module    127
381 #define LX_SYS_init_module      128
382 #define LX_SYS_delete_module    129
383 #define LX_SYS_get_kernel_syms  130
384 #define LX_SYS_quotactl         131
385 #define LX_SYS_getpgid          132
386 #define LX_SYS_fchdir           133
387 #define LX_SYS_sysfs            135
388 #define LX_SYS_setfsuid         138
389 #define LX_SYS_setfsgid         139
390 #define LX_SYS_llseek           140
391 #define LX_SYS_getdents         141
```

```
392 #define LX_SYS_newselect        142
393 #define LX_SYS_flock            143
394 #define LX_SYS_msync            144
395 #define LX_SYS_readv            145
396 #define LX_SYS_writev           146
397 #define LX_SYS_getsid           147
398 #define LX_SYS_fdatasync        148
399 #define LX_SYS_sysctl           149
400 #define LX_SYS_mlock            150
401 #define LX_SYS_munlock          151
402 #define LX_SYS_mlockall         152
403 #define LX_SYS_munlockall               153
404 #define LX_SYS_sched_setparam           154
405 #define LX_SYS_sched_getparam           155
406 #define LX_SYS_sched_setscheduler       156
407 #define LX_SYS_sched_getscheduler       157
408 #define LX_SYS_sched_yield              158
409 #define LX_SYS_sched_get_priority_max   159
410 #define LX_SYS_sched_get_priority_min   160
411 #define LX_SYS_sched_rr_get_interval    161
412 #define LX_SYS_nanosleep        162
413 #define LX_SYS_mremap           163
414 #define LX_SYS_setresuid        164
415 #define LX_SYS_getresuid        165
416 #define LX_SYS_poll             168
417 #define LX_SYS_setresgid        170
418 #define LX_SYS_getresgid        171
419 #define LX_SYS_prctl            172
420 #define LX_SYS_rt_sigreturn     173
421 #define LX_SYS_rt_sigaction     174
422 #define LX_SYS_rt_sigprocmask   175
423 #define LX_SYS_rt_sigpending    176
424 #define LX_SYS_rt_sigtimedwait  177
425 #define LX_SYS_rt_sigqueueinfo  178
426 #define LX_SYS_rt_sigsuspend    179
427 #define LX_SYS_pread            180
428 #define LX_SYS_pwrite           181
429 #define LX_SYS_chown            182
430 #define LX_SYS_getcwd           183
431 #define LX_SYS_capget           184
432 #define LX_SYS_capset           185
433 #define LX_SYS_sigaltstack      186
434 #define LX_SYS_sendfile         187
435 #define LX_SYS_getpmsg          188
436 #define LX_SYS_putpmsg          189
437 #define LX_SYS_vfork            190
438 #define LX_SYS_ugetrlimit       191
439 #define LX_SYS_mmap2            192
440 #define LX_SYS_truncate64       193
441 #define LX_SYS_ftruncate64      194
442 #define LX_SYS_stat64           195
443 #define LX_SYS_lstat64          196
444 #define LX_SYS_fstat64          197
445 #define LX_SYS_lchown32         198
446 #define LX_SYS_getuid32         199
447 #define LX_SYS_getgid32         200
448 #define LX_SYS_geteuid32        201
449 #define LX_SYS_getegid32        202
450 #define LX_SYS_setreuid32       203
451 #define LX_SYS_setregid32       204
452 #define LX_SYS_getgroups32      205
453 #define LX_SYS_setgroups32      206
454 #define LX_SYS_fchown32         207
455 #define LX_SYS_setresuid32      208
456 #define LX_SYS_getresuid32      209
457 #define LX_SYS_setresgid32      210
```



```
458 #define LX_SYS_getresgid32      211
459 #define LX_SYS_chown32          212
460 #define LX_SYS_setuid32         213
461 #define LX_SYS_setgid32         214
462 #define LX_SYS_setfsuid32       215
463 #define LX_SYS_setfsgid32       216
464 #define LX_SYS_mincore          218
465 #define LX_SYS_madvise          219
466 #define LX_SYS_getdents64       220
467 #define LX_SYS_fcntl64          221
468 #define LX_SYS_gettid           224
469 #define LX_SYS_readahead        225
470 #define LX_SYS_setxattr         226
471 #define LX_SYS_lsetxattr        227
472 #define LX_SYS_fsetxattr        228
473 #define LX_SYS_getxattr         229
474 #define LX_SYS_lgetxattr        230
475 #define LX_SYS_fgetxattr        231
476 #define LX_SYS_listxattr        232
477 #define LX_SYS_llistxattr       233
478 #define LX_SYS_flistxattr       234
479 #define LX_SYS_removexattr      235
480 #define LX_SYS_lremovexattr     236
481 #define LX_SYS_fremovexattr     237
482 #define LX_SYS_tkill            238
483 #define LX_SYS_sendfile64       239
484 #define LX_SYS_futex            240
485 #define LX_SYS_sched_setaffinity        241
486 #define LX_SYS_sched_getaffinity        242
487 #define LX_SYS_set_thread_area  243
488 #define LX_SYS_get_thread_area  244
489 #define LX_SYS_fadvise64        250
490 #define LX_SYS_exit_group       252
491 #define LX_SYS_remap_file_pages 257
492 #define LX_SYS_set_tid_address  258
493 #define LX_SYS_timer_create     259
494 #define LX_SYS_timer_settime    260
495 #define LX_SYS_timer_gettime    261
496 #define LX_SYS_timer_getoverrun 262
497 #define LX_SYS_timer_delete     263
498 #define LX_SYS_clock_settime    264
499 #define LX_SYS_clock_gettime    265
500 #define LX_SYS_clock_getres     266
501 #define LX_SYS_clock_nanosleep  267
502 #define LX_SYS_tgkill           270
503 /* the following syscalls are for 2.6 and later kernels */
504 #define LX_SYS_utimes           271
505 #define LX_SYS_fadvise64_64     272
506 #define LX_SYS_vserver          273
507 #define LX_SYS_mbind            274
508 #define LX_SYS_get_mempolicyd   275
509 #define LX_SYS_set_mempolicy    276
510 #define LX_SYS_mq_open          277
511 #define LX_SYS_mq_unlink        278
512 #define LX_SYS_mq_timedsend     279
513 #define LX_SYS_mq_timedreceive  280
514 #define LX_SYS_mq_notify        281
515 #define LX_SYS_mq_getsetattr    282
516 #define LX_SYS_kexec_load       283
517 #define LX_SYS_waitid           284
518 #define LX_SYS_setaltroot       285
519 #define LX_SYS_add_key          286
520 #define LX_SYS_request_key      287
521 #define LX_SYS_keyctl           288
522 #define LX_SYS_ioprio_set       289
523 #define LX_SYS_ioprio_get       290
```

```
524 #define LX_SYS_inotify_init     291
525 #define LX_SYS_inotify_add_watch        292
526 #define LX_SYS_inotify_rm_watch 293
527 #define LX_SYS_migrate_pages    294
528 #define LX_SYS_openat           295
529 #define LX_SYS_mkdirat          296
530 #define LX_SYS_mknodat          297
531 #define LX_SYS_fchownat         298
532 #define LX_SYS_futimesat        299
533 #define LX_SYS_fstatat64        300
534 #define LX_SYS_unlinkat         301
535 #define LX_SYS_renameat         302
536 #define LX_SYS_linkat           303
537 #define LX_SYS_symlinkat        304
538 #define LX_SYS_readlinkat       305
539 #define LX_SYS_fchmodat         306
540 #define LX_SYS_faccessat        307
541 #define LX_SYS_pselect6         308
542 #define LX_SYS_ppoll            309
543 #define LX_SYS_unshare          310
544 #define LX_SYS_set_robust_list  311
545 #define LX_SYS_get_robust_list  312
546 #define LX_SYS_splice           313
547 #define LX_SYS_sync_file_range  314
548 #define LX_SYS_tee              315
549 #define LX_SYS_vmsplice         316
550 #define LX_SYS_move_pages       317

552 #ifdef  __cplusplus
553 }
554 #endif

556 #endif  /* _SYS_LX_SYSCALL_H */
557 #endif /* ! codereview */
```

```
**********************************************************
    3967 Tue Jan 14 16:17:09 2014
new/usr/src/lib/brand/lx/lx_brand/sys/lx_sysv_ipc.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _LX_SYSV_IPC_H
#define _LX_SYSV_IPC_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

#ifdef __cplusplus
extern "C" {
#endif

/*
 * msg-related definitions.
 */
#define LX_IPC_CREAT    00001000
#define LX_IPC_EXCL     00002000
#define LX_IPC_NOWAIT   00004000

#define LX_IPC_RMID     0
#define LX_IPC_SET      1
#define LX_IPC_STAT     2
#define LX_IPC_INFO     3

#define LX_IPC_64       0x0100

#define LX_SEMOP        1
#define LX_SEMGET       2
#define LX_SEMCTL       3
#define LX_MSGSND       11
#define LX_MSGRCV       12
#define LX_MSGGET       13
#define LX_MSGCTL       14
#define LX_SHMAT        21
#define LX_SHMDT        22
#define LX_SHMGET       23
#define LX_SHMCTL       24
```



```
#define LX_MSG_STAT     11
#define LX_MSG_INFO     12

#define LX_MSG_NOERROR  010000

/*
 * Linux hard codes the maximum msgbuf length to be 8192 bytes.  Really.
 */
#define LX_MSGMAX       8192

struct lx_ipc_perm {
        key_t           key;
        uid_t           uid;
        uid_t           gid;
        uid_t           cuid;
        uid_t           cgid;
        ushort_t        mode;
        ushort_t        _pad1;
        ushort_t        seq;
        ushort_t        _pad2;
        ulong_t         _unused1;
        ulong_t         _unused2;
};

struct lx_msqid_ds {
        struct lx_ipc_perm      msg_perm;
        time_t                  msg_stime;
        ulong_t                 _unused1;
        time_t                  msg_rtime;
        ulong_t                 _unused2;
        time_t                  msg_ctime;
        ulong_t                 _unused3;
        ulong_t                 msg_cbytes;
        ulong_t                 msg_qnum;
        ulong_t                 msg_qbytes;
        pid_t                   msg_lspid;
        pid_t                   msg_lrpid;
        ulong_t                 _unused4;
        ulong_t                 _unused5;
};

struct lx_msginfo {
        int             msgpool;
        int             msgmap;
        int             msgmax;
        int             msgmnb;
        int             msgmni;
        int             msgssz;
        int             msgtql;
        ushort_t        msgseg;
};

/*
 * semaphore-related definitions.
 */
#define LX_GETPID       11
#define LX_GETVAL       12
#define LX_GETALL       13
#define LX_GETNCNT      14
#define LX_GETZCNT      15
#define LX_SETVAL       16
#define LX_SETALL       17
#define LX_SEM_STAT     18
#define LX_SEM_INFO     19
#define LX_SEM_UNDO     0x1000
#define LX_SEMVMX       32767
```

```
129 struct lx_semid_ds {
130         struct lx_ipc_perm      sem_perm;
131         time_t                  sem_otime;
132         ulong_t                 _unused1;
133         time_t                  sem_ctime;
134         ulong_t                 _unused2;
135         ulong_t                 sem_nsems;
136         ulong_t                 _unused3;
137         ulong_t                 _unused4;
138 };

140 struct lx_seminfo {
141         int semmap;
142         int semmni;
143         int semmns;
144         int semmnu;
145         int semmsl;
146         int semopm;
147         int semume;
148         int semusz;
149         int semvmx;
150         int semaem;
151 };

153 union lx_semun {
154         int val;
155         struct lx_semid_ds *semds;
156         ushort_t *sems;
157         struct lx_seminfo *info;
158         uintptr_t dummy;
159 };

161 /*
162  * shm-related definitions
163  */
164 #define LX_SHM_LOCKED   02000
165 #define LX_SHM_RDONLY   010000
166 #define LX_SHM_RND      020000
167 #define LX_SHM_REMAP    040000

169 #define LX_SHM_LOCK     11
170 #define LX_SHM_UNLOCK   12
171 #define LX_SHM_STAT     13
172 #define LX_SHM_INFO     14

174 struct lx_shmid_ds {
175         struct lx_ipc_perm      shm_perm;
176         size_t                  shm_segsz;
177         time_t                  shm_atime;
178         ulong_t                 _unused1;
179         time_t                  shm_dtime;
180         ulong_t                 _unused2;
181         time_t                  shm_ctime;
182         ulong_t                 _unused3;
183         pid_t                   shm_cpid;
184         pid_t                   shm_lpid;
185         ushort_t                shm_nattch;
186         ulong_t                 _unused4;
187         ulong_t                 _unused5;
188 };

190 struct lx_shm_info {
191         int     used_ids;
192         ulong_t shm_tot;
193         ulong_t shm_rss;
```



```
194         ulong_t shm_swp;
195         ulong_t swap_attempts;
196         ulong_t swap_successes;
197 };

199 struct lx_shminfo {
200         int     shmmax;
201         int     shmmin;
202         int     shmmni;
203         int     shmseg;
204         int     shmall;
205 };

207 #ifdef  __cplusplus
208 }
209 #endif

211 #endif  /* _LX_SYSV_IPC_H */
212 #endif /* ! codereview */
```

```
**********************************************************
    1378 Tue Jan 14 16:17:09 2014
new/usr/src/lib/brand/lx/lx_brand/sys/lx_thread.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2006 Sun Microsystems, Inc.        All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _SYS_LX_THREAD_H
#define _SYS_LX_THREAD_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

#ifdef __cplusplus
extern "C" {
#endif

#include <thread.h>

typedef struct lx_tsd {
        uintptr_t       lxtsd_gs;
        int             lxtsd_exit;
        int             lxtsd_exit_status;
        ucontext_t      lxtsd_exit_context;
} lx_tsd_t;

extern thread_key_t     lx_tsd_key;     /* thread-specific Linux %gs value */

extern void             lx_swap_gs(long, long *);

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_LX_THREAD_H */
#endif /* ! codereview */
```

```
**********************************************************
    3867 Tue Jan 14 16:17:09 2014
new/usr/src/lib/brand/lx/lx_brand/sys/lx_thunk_server.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _LX_THUNK_SERVER_H
#define _LX_THUNK_SERVER_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

#ifdef  __cplusplus
extern "C" {
#endif

#include <netdb.h>
#include <procfs.h>

/*
 * Binary that should be exec'd to start up the thunking server
 */
#define LXT_SERVER_BINARY       "/native/usr/lib/brand/lx/lx_thunk"

/*
 * When the thunking server is started it will need to communicate
 * to the client via two fifos.  These fifos will be passed to the
 * thunking server via the following file descriptors:
 */
#define LXT_SERVER_FIFO_RD_FD   3
#define LXT_SERVER_FIFO_WR_FD   4

/*
 * Operations supported by the thunking server
 */
#define LXT_SERVER_OP_MIN               0
#define LXT_SERVER_OP_PING              0
#define LXT_SERVER_OP_NAME2HOST         1
#define LXT_SERVER_OP_ADDR2HOST         2
#define LXT_SERVER_OP_NAME2SERV         3
#define LXT_SERVER_OP_PORT2SERV         4
#define LXT_SERVER_OP_OPENLOG           5
```



```
#define LXT_SERVER_OP_SYSLOG            6
#define LXT_SERVER_OP_CLOSELOG          7
#define LXT_SERVER_OP_MAX               8

/*
 * Macros used to translate pointer into offsets for when they are
 * being transmitted between the client and server processes.
 *
 * NOTE: We're going to add 1 to every offset value.  The reason
 * for this is that some of the pointers we're converting to offsets are
 * stored in NULL terminated arrays, and if one of the members of
 * one of these arrays happened to be at the beginning of the storage
 * buffer it would have an offset of 0 and when the client tries to
 * translate the offsets back into pointers it wouldn't be able
 * to differentiate between the 0 offset from the end of the array.
 */
#define LXT_PTR_TO_OFFSET(ptr, base) \
                ((void *)((uintptr_t)(ptr) - (uintptr_t)(base) + 1))
#define LXT_OFFSET_TO_PTR(offset, base) \
                ((void *)((uintptr_t)(offset) + (uintptr_t)(base) - 1))

/*
 * Structures passed to the thunking server via door calls
 */
typedef struct lxt_server_arg {
        int             lxt_sa_op;
        int             lxt_sa_success;
        int             lxt_sa_errno;
        char            lxt_sa_data[1];
} lxt_server_arg_t;

typedef struct lxt_gethost_arg {
        struct hostent  lxt_gh_result;

        int             lxt_gh_h_errno;

        int             lxt_gh_type;
        int             lxt_gh_token_len;
        int             lxt_gh_buf_len;

        int             lxt_gh_storage_len;
        char            lxt_gh_storage[1];
} lxt_gethost_arg_t;

typedef struct lxt_getserv_arg {
        struct servent  lxt_gs_result;

        int             lxt_gs_token_len;
        int             lxt_gs_buf_len;
        char            lxt_gs_proto[5];

        int             lxt_gs_storage_len;
        char            lxt_gs_storage[1];
} lxt_getserv_arg_t;

typedef struct lxt_openlog_arg {
        int             lxt_ol_logopt;
        int             lxt_ol_facility;
        char            lxt_ol_ident[128];
} lxt_openlog_arg_t;

typedef struct lxt_syslog_arg {
        int             lxt_sl_priority;
        pid_t           lxt_sl_pid;
        char            lxt_sl_progname[PRFNSZ];
        char            lxt_sl_message[1024];
```

```
128 } lxt_syslog_arg_t;


131 /*
132  * Functions called by the brand library to manage startup of the
133  * thunk server process.
134  */
135 void lxt_server_init(int, char *[]);
136 int lxt_server_pid(int *pid);
137 void lxt_server_exec_check(void);

139 #ifdef  __cplusplus
140 }
141 #endif

143 #endif  /* _LX_THUNK_SERVER_H */
144 #endif /* ! codereview */
```

```
**********************************************************
    3293 Tue Jan 14 16:17:09 2014
new/usr/src/lib/brand/lx/lx_brand/sys/lx_types.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _SYS_LX_TYPES_H
#define _SYS_LX_TYPES_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

#ifdef  __cplusplus
extern "C" {
#endif

#define SHRT_MIN        (-32768)        /* min value of a "short int" */
#define SHRT_MAX        32767           /* max value of a "short int" */
#define USHRT_MAX       65535           /* max of "unsigned short int" */
#define INT_MIN         (-2147483647-1) /* min value of an "int" */
#define INT_MAX         2147483647      /* max value of an "int" */
#define UINT_MAX        4294967295U     /* max value of an "unsigned int" */
#define LONG_MIN        (-2147483647L-1L)
                                        /* min value of a "long int" */
#define LONG_MAX        2147483647L     /* max value of a "long int" */
#define ULONG_MAX       4294967295UL    /* max of "unsigned long int" */

#define LX_SYS_UTS_LN   65

struct lx_utsname {
        char    sysname[LX_SYS_UTS_LN];
        char    nodename[LX_SYS_UTS_LN];
        char    release[LX_SYS_UTS_LN];
        char    version[LX_SYS_UTS_LN];
        char    machine[LX_SYS_UTS_LN];
        char    domainname[LX_SYS_UTS_LN];
};

typedef uint64_t        lx_dev_t;
typedef uint16_t        lx_dev16_t;
typedef uint32_t        lx_ino_t;
typedef uint64_t        lx_ino64_t;
typedef uint32_t        lx_uid_t;
```



```
typedef uint16_t        lx_uid16_t;
typedef uint32_t        lx_gid_t;
typedef uint16_t        lx_gid16_t;
typedef uint32_t        lx_off_t;
typedef uint64_t        lx_off64_t;
typedef uint32_t        lx_blksize_t;
typedef uint32_t        lx_blkcnt_t;
typedef uint64_t        lx_blkcnt64_t;
typedef ulong_t         lx_mode_t;
typedef uint16_t        lx_mode16_t;

#define LX_UID16_TO_UID32(uid16)        \
        (((uid16) == (lx_uid16_t)-1) ? ((lx_uid_t)-1) : (lx_uid_t)(uid16))

#define LX_GID16_TO_GID32(gid16)     \
        (((gid16) == (lx_gid16_t)-1) ? ((lx_gid_t)-1) : (lx_gid_t)(gid16))

/* Overflow values default to NFS nobody. */

#define UID16_OVERFLOW  ((lx_uid16_t)65534)
#define GID16_OVERFLOW  ((lx_gid16_t)65534)

/*
 * All IDs with high word non-zero are converted to default overflow values to
 * avoid inadvertent truncation to zero (root) (!).
 */
#define LX_UID32_TO_UID16(uid32)        \
        ((((uid32) & 0xffff0000) == 0)  ? ((lx_uid16_t)(uid32)) : \
            (((uid32) == ((lx_uid_t)-1)) ? ((lx_uid16_t)-1) : UID16_OVERFLOW))

#define LX_GID32_TO_GID16(gid32)        \
        ((((gid32) & 0xffff0000) == 0)  ? ((lx_gid16_t)(gid32)) : \
            (((gid32) == ((lx_gid_t)-1)) ? ((lx_gid16_t)-1) : GID16_OVERFLOW))

struct lx_timespec {
        time_t  ts_sec;
        long    ts_nsec;
};

#define LX_32TO64(lo, hi)       \
        ((uint64_t)((uint64_t)(lo) | ((uint64_t)(hi) << 32)))

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_LX_TYPES_H */
#endif /* ! codereview */
```

```
**********************************************************
    1356 Tue Jan 14 16:17:10 2014
new/usr/src/lib/brand/lx/lx_nametoaddr/Makefile
Bring back LX zones.
**********************************************************
```

```
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #

  22 #
  23 # Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  24 # Use is subject to license terms.
  25 #
  26 # ident "%Z%%M% %I%     %E% SMI"
  27 #

  29 include ../../../Makefile.lib

  31 SUBDIRS = $(MACH)
  32 $(BUILD64)SUBDIRS += $(MACH64)

  34 LINT_SUBDIRS = $(MACH)
  35 $(BUILD64)LINT_SUBDIRS += $(MACH64)

  37 all :=          TARGET= all
  38 clean :=        TARGET= clean
  39 clobber :=      TARGET= clobber
  40 install :=      TARGET= install
  41 lint :=         TARGET= lint

  43 .KEEP_STATE:

  45 all install clean clobber: $(SUBDIRS)

  47 lint: $(LINT_SUBDIRS)

  49 $(SUBDIRS): FRC
  50         @cd $@; pwd; $(MAKE) $(TARGET)

  52 FRC:
  53 #endif /* ! codereview */
```

```
**********************************************************
    1702 Tue Jan 14 16:17:10 2014
new/usr/src/lib/brand/lx/lx_nametoaddr/Makefile.com
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "%Z%%M% %I%     %E% SMI"
#

LIBRARY =       lx_nametoaddr.a
VERS =          .1

COBJS =         lx_nametoaddr.o
OBJECTS =       $(COBJS)

include ../../../../Makefile.lib
include ../../Makefile.lx

MAPFILES =      ../common/mapfile-vers
MAPOPTS =       $(MAPFILES:%=-M%)

CSRCS =         $(COBJS:%o=../common/%c)
SRCS =          $(CSRCS)

SRCDIR =        ../common
LX_THUNK =      ../../lx_thunk

ASFLAGS +=      -P -D_ASM
LDLIBS +=       -lc -lnsl
CFLAGS +=       $(CCVERBOSE)
CPPFLAGS +=     -D_REENTRANT -I../ -I$(LX_THUNK)
DYNFLAGS +=     $(MAPOPTS) '-R$$ORIGIN'

LIBS =          $(DYNLIB)

LINTFLAGS +=    $(LX_THUNK)/$(MACH)/llib-llx_thunk.ln
LINTFLAGS64 +=  $(LX_THUNK)/$(MACH64)/llib-llx_thunk.ln

CLEANFILES =    $(DYNLIB)
ROOTLIBDIR =    $(ROOT)/usr/lib/brand/lx
ROOTLIBDIR64 =  $(ROOT)/usr/lib/brand/lx/$(MACH64)
```



```
.KEEP_STATE:

all: $(DYNLIB)

lint: lintcheck

include ../../../../Makefile.targ
#endif /* ! codereview */
```

```
**********************************************************
    1137 Tue Jan 14 16:17:10 2014
new/usr/src/lib/brand/lx/lx_nametoaddr/amd64/Makefile
Bring back LX zones.
**********************************************************
```

```
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #

  22 #
  23 # Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  24 # Use is subject to license terms.
  25 #
  26 # ident "%Z%%M% %I%     %E% SMI"
  27 #

  29 include ../Makefile.com
  30 include $(SRC)/lib/Makefile.lib.64

  32 DYNFLAGS +=     $(LX_THUNK)/$(MACH64)/lx_thunk.so.1
  33 CLOBBERFILES =  $(ROOTLIBDIR64)/$(DYNLIB) $(ROOTLIBDIR64)/$(LINTLIB)

  35 install: all $(ROOTLIBS64)
  36 #endif /* ! codereview */
```

```
**********************************************************
   13251 Tue Jan 14 16:17:10 2014
new/usr/src/lib/brand/lx/lx_nametoaddr/common/lx_nametoaddr.c
LX zone support should now build and packages of relevance produced.
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

/*
 * BrandZ lx name services translation library.
 *
 * This library is specified as the default name services translation
 * library in a custom netconfig(4) file that is only used when running
 * native solaris processes in a Linux branded zone.
 *
 * What this means it that when a native solaris process runs in a
 * Linux branded zone and issues a name service request to libnsl.so
 * (either directly or indirectly via any libraries the program may
 * be linked against) libnsl.so will dlopen(3c) this library and call
 * into it to service these requests.
 *
 * This library is in turn linked against lx_thunk.so and will attempt
 * to call interfaces in lx_thunk.so to resolve these requests.  The
 * functions that are called in lx_thunk.so are designed to have the
 * same signature and behavior as the existing solaris name service
 * interfaces.  The name services interfaces we call are:
 *
 *      Native Interface        -> lx_thunk.so Interface
 *      ----------------        -> --------------------
 *      gethostbyname_r         -> lxt_gethostbyname_r
 *      gethostbyaddr_r         -> lxt_gethostbyaddr_r
 *      getservbyname_r         -> lxt_getservbyname_r
 *      getservbyport_r         -> lxt_getservbyport_r
 *
 * This library also uses one additional interface from lx_thunk.so:
 *      lxt_debug
 * Information debugging messages are sent to lx_thunk.so via this
 * interface and that library can decided if it wants to drop the
 * messages or output them somewhere.
 */
```



```
#include <assert.h>
#include <dlfcn.h>
#include <errno.h>
#include <fcntl.h>
#include <netdb.h>
#include <netdir.h>
#include <nss_dbdefs.h>
#include <rpc/clnt.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/varargs.h>
#include <sys/wait.h>
#include <thread.h>
#include <tiuser.h>
#include <unistd.h>
#include <sys/lx_thunk.h>


/*
 * Private nametoaddr library interfaces.
 */
static int
netconfig_is_ipv4(struct netconfig *config)
{
        int i;
        /*
         * If we look at the rpc services registered on a Linux system
         * (this can be done via rpcinfo(1M)) for both on the loopback
         * interface and on any remote interfaces we only see services
         * registered for tcp and udp.  So here we'll limit our support
         * to these transports.
         */
        char *ipv4_netids[] = {
                "tcp",
                "udp",
                NULL
        };

        for (i = 0; ipv4_netids[i] != NULL; i++) {
                if (strcmp(ipv4_netids[i], config->nc_netid) == 0)
                        return (1);
        }
        return (0);
}

/*
 * Public nametoaddr library interfaces.
 *
 * These are the functional entry points that libnsl will lookup (via
 * the symbol names) when it loads this nametoaddr translation library.
 */

/*
 * _netdir_getbyname() returns all of the addresses for
 * a specified host and service.
 */
struct nd_addrlist *
_netdir_getbyname(struct netconfig *netconfigp,
    struct nd_hostserv *nd_hostservp)
```

```
{
        struct nd_addrlist      *rp = NULL;
        struct netbuf           *nbp = NULL;
        struct sockaddr_in      *sap = NULL;
        struct hostent          n2h_result;
        struct servent          n2s_result;
        char                    *n2h_buf = NULL, *n2s_buf = NULL;
        int                     h_errno, i, host_self = 0, r_count;
        int                     n2h_count = 0, n2s_count = 0;

        lxt_debug("_netdir_getbyname: request recieved\n");

        /* Make sure this is an ipv4 request. */
        if (!netconfig_is_ipv4(netconfigp)) {
                _nderror = ND_BADARG;
                goto fail;
        }

        /* Allocate memory for the queries. */
        if (((n2h_buf = malloc(NSS_BUFLEN_HOSTS)) == NULL) ||
            ((n2s_buf = malloc(NSS_BUFLEN_SERVICES)) == NULL))
                goto malloc_fail;

        /* Check if the host name specified is HOST_SELF. */
        if (strcmp(nd_hostservp->h_host, HOST_SELF) == 0)
                host_self = 1;

        /*
         * If the hostname specified is HOST_SELF, the we're just
         * just doing a service lookup so don't bother with trying
         * to lookup the host name.
         */
        if (!host_self) {
                /* Resolve the hostname. */
                lxt_debug("_netdir_getbyname: "
                    "resolving host name: %s\n", nd_hostservp->h_host);
                if (lxt_gethostbyname_r(nd_hostservp->h_host, &n2h_result,
                    n2h_buf, NSS_BUFLEN_HOSTS, &h_errno) == NULL) {
                        if (errno == ERANGE) {
                                _nderror = ND_SYSTEM;
                        } else if (h_errno == HOST_NOT_FOUND) {
                                _nderror = ND_NOHOST;
                        } else if (h_errno == TRY_AGAIN) {
                                _nderror = ND_TRY_AGAIN;
                        } else if (h_errno == NO_RECOVERY) {
                                _nderror = ND_NO_RECOVERY;
                        } else if (h_errno == NO_DATA) {
                                _nderror = ND_NO_DATA;
                        } else {
                                _nderror = ND_SYSTEM;
                        }
                        goto fail;
                }
                while (n2h_result.h_addr_list[n2h_count++] != NULL);
                n2h_count--;
        }

        if (nd_hostservp->h_serv != NULL) {
                /* Resolve the service name */
                lxt_debug("_netdir_getbyname: "
                    "resolving service name: %s\n", nd_hostservp->h_serv);
                if (lxt_getservbyname_r(nd_hostservp->h_serv,
                    netconfigp->nc_proto, &n2s_result,
                    n2s_buf, NSS_BUFLEN_SERVICES) == NULL) {
                        _nderror = ND_SYSTEM;
                        goto fail;
```



```
                }
                n2s_count = 1;
        }

        /* Make sure we got some results. */
        if ((n2h_count + n2s_count) == 0) {
                lxt_debug("_netdir_getbyname: no results!\n");
                goto exit;
        }
        r_count = (n2h_count != 0) ? n2h_count : 1;

        /*
         * Allocate the return buffers.  These buffers will be free'd
         * by libnsl'netdir_free(), so we need to allocate them in the
         * way that libnsl'netdir_free() expects.
         */
        if (((rp = calloc(1, sizeof (struct nd_addrlist))) == NULL) ||
            ((nbp = calloc(1, sizeof (struct netbuf) * r_count)) == NULL) ||
            ((sap = calloc(1, sizeof (struct sockaddr_in) * r_count)) == NULL))
                goto malloc_fail;

        /* Initialize the structures we're going to return. */
        rp->n_cnt = r_count;
        rp->n_addrs = nbp;
        for (i = 0; i < r_count; i++) {

                /* Initialize the netbuf. */
                nbp[i].maxlen = nbp[i].len = sizeof (struct sockaddr_in);
                nbp[i].buf = (char *)&sap[i];

                /* Initialize the sockaddr_in. */
                sap[i].sin_family = AF_INET;

                /* If we looked up any host address copy them out. */
                if (!host_self)
                        bcopy(n2h_result.h_addr_list[i], &sap[i].sin_addr,
                            sizeof (sap[i].sin_addr));

                /* If we looked up any service ports copy them out. */
                if (nd_hostservp->h_serv != NULL)
                        sap[i].sin_port = n2s_result.s_port;
        }

        /* We're finally done. */
        lxt_debug("_netdir_getbyname: success\n");
        return (rp);

malloc_fail:
        _nderror = ND_NOMEM;

fail:
        lxt_debug("_netdir_getbyname: failed!\n");

exit:
        if (n2h_buf == NULL)
                free(n2h_buf);
        if (n2s_buf == NULL)
                free(n2s_buf);
        if (rp == NULL)
                free(rp);
        if (nbp == NULL)
                free(nbp);
        if (sap == NULL)
                free(sap);
        return (NULL);
}
```

```
/*
 * _netdir_getbyaddr() takes an address (hopefully obtained from
 * someone doing a _netdir_getbyname()) and returns all hosts with
 * that address.
 */
struct nd_hostservlist *
/*ARGSUSED*/
_netdir_getbyaddr(struct netconfig *netconfigp, struct netbuf *nbp)
{
        struct nd_hostservlist  *rp = NULL;
        struct nd_hostserv      *hsp = NULL;
        struct sockaddr_in      *sap;
        struct servent          p2s_result;
        struct hostent          a2h_result;
        char                    *a2h_buf = NULL, *p2s_buf = NULL;
        int                     h_errno, i;
        int                     r_count = 0;
        int                     a2h_count = 0, p2s_count = 0;

        lxt_debug("_netdir_getbyaddr: request recieved\n");

        /* Make sure this is an ipv4 request. */
        if (!netconfig_is_ipv4(netconfigp)) {
                _nderror = ND_BADARG;
                goto fail;
        }

        /*
         * Make sure the netbuf contains one struct sockaddr_in of
         * type AF_INET.
         */
        if ((nbp->len != sizeof (struct sockaddr_in)) ||
            (nbp->len < nbp->maxlen)) {
                _nderror = ND_BADARG;
                goto fail;
        }
        /*LINTED*/
        sap = (struct sockaddr_in *)nbp->buf;
        if (sap->sin_family != AF_INET) {
                _nderror = ND_BADARG;
                goto fail;
        }

        /* Allocate memory for the queries. */
        if (((a2h_buf = malloc(NSS_BUFLEN_HOSTS)) == NULL) ||
            ((p2s_buf = malloc(NSS_BUFLEN_SERVICES)) == NULL))
                goto malloc_fail;

        if (sap->sin_addr.s_addr != INADDR_ANY) {
                lxt_debug("_netdir_getbyaddr: "
                    "resolving host address: 0x%x\n", sap->sin_addr.s_addr);
                if (lxt_gethostbyaddr_r((char *)&sap->sin_addr.s_addr,
                    sizeof (sap->sin_addr.s_addr), AF_INET,
                    &a2h_result, a2h_buf, NSS_BUFLEN_HOSTS,
                    &h_errno) == NULL) {
                        if (errno == ERANGE) {
                                _nderror = ND_SYSTEM;
                        } else if (h_errno == HOST_NOT_FOUND) {
                                _nderror = ND_NOHOST;
                        } else if (h_errno == TRY_AGAIN) {
                                _nderror = ND_TRY_AGAIN;
                        } else if (h_errno == NO_RECOVERY) {
                                _nderror = ND_NO_RECOVERY;
                        } else if (h_errno == NO_DATA) {
                                _nderror = ND_NO_DATA;
```



```
                        } else {
                                _nderror = ND_SYSTEM;
                        }
                        goto fail;
                }
                while (a2h_result.h_aliases[a2h_count++] != NULL);
                /*
                 * We need to count a2h_result.h_name as a valid name for
                 * for the address we just looked up.  Of course a2h_count
                 * is actually over estimated by one, so instead of
                 * decrementing it here we'll just leave it as it to
                 * account for a2h_result.h_name.
                 */
        }

        if (sap->sin_port != 0) {
                lxt_debug("_netdir_getbyaddr: "
                    "resolving service port: 0x%x\n", sap->sin_port);
                if (lxt_getservbyport_r(sap->sin_port,
                    netconfigp->nc_proto, &p2s_result,
                    p2s_buf, NSS_BUFLEN_SERVICES) == NULL) {
                        _nderror = ND_SYSTEM;
                        goto fail;
                }
                p2s_count = 1;
        }

        /* Make sure we got some results. */
        if ((a2h_count + p2s_count) == 0) {
                lxt_debug("_netdir_getbyaddr: no results!\n");
                goto exit;
        }
        r_count = (a2h_count != 0) ? a2h_count : 1;

        /*
         * Allocate the return buffers.  These buffers will be free'd
         * by libnsl'netdir_free(), so we need to allocate them in the
         * way that libnsl'netdir_free() expects.
         */
        if (((rp = calloc(1, sizeof (struct nd_hostservlist))) == NULL) ||
            ((hsp = calloc(1, sizeof (struct nd_hostserv) * r_count)) == NULL))
                goto malloc_fail;

        lxt_debug("_netdir_getbyaddr: hahaha0 - %d\n", r_count);
        rp->h_cnt = r_count;
        rp->h_hostservs = hsp;
        for (i = 0; i < r_count; i++) {
                /* If we looked up any host names copy them out. */
        lxt_debug("_netdir_getbyaddr: hahaha1 - %d\n", r_count);
                if ((a2h_count > 0) && (i == 0) &&
                    ((hsp[i].h_host = strdup(a2h_result.h_name)) == NULL))
                        goto malloc_fail;

                if ((a2h_count > 0) && (i > 0) &&
                    ((hsp[i].h_host =
                        strdup(a2h_result.h_aliases[i - 1])) == NULL))
                        goto malloc_fail;

        lxt_debug("_netdir_getbyaddr: hahaha2 - %d\n", r_count);
                /* If we looked up any service names copy them out. */
                if ((p2s_count > 0) &&
                    ((hsp[i].h_serv = strdup(p2s_result.s_name)) == NULL))
                        goto malloc_fail;
        lxt_debug("_netdir_getbyaddr: hahaha3 - %d\n", r_count);
        }
```

```
391         /* We're finally done. */
392         lxt_debug("_netdir_getbyaddr: success\n");
393         return (rp);

395 malloc_fail:
396         _nderror = ND_NOMEM;

398 fail:
399         lxt_debug("_netdir_getbyaddr: failed!\n");

401 exit:
402         if (a2h_buf == NULL)
403                 free(a2h_buf);
404         if (p2s_buf == NULL)
405                 free(p2s_buf);
406         if (rp == NULL)
407                 free(rp);
408         if (hsp != NULL) {
409                 for (i = 0; i < r_count; i++) {
410                         if (hsp[i].h_host != NULL)
411                                 free(hsp[i].h_host);
412                         if (hsp[i].h_serv != NULL)
413                                 free(hsp[i].h_serv);
414                 }
415                 free(hsp);
416         }
417         return (NULL);
418 }

420 char *
421 /* ARGSUSED */
422 _taddr2uaddr(struct netconfig *netconfigp, struct netbuf *nbp)
423 {
424         extern char             *inet_ntoa_r();

426         struct sockaddr_in      *sa;
427         char                    tmp[RPC_INET6_MAXUADDRSIZE];
428         unsigned short          myport;

430         if (netconfigp == NULL || nbp == NULL || nbp->buf == NULL) {
431                 _nderror = ND_BADARG;
432                 return (NULL);
433         }

435         if (strcmp(netconfigp->nc_protofmly, NC_INET) != 0) {
436                 /* we only support inet address translation */
437                 assert(0);
438                 _nderror = ND_SYSTEM;
439                 return (NULL);
440         }

442         /* LINTED pointer cast */
443         sa = (struct sockaddr_in *)(nbp->buf);
444         myport = ntohs(sa->sin_port);
445         (void) inet_ntoa_r(sa->sin_addr, tmp);

447         (void) sprintf(tmp + strlen(tmp), ".%d.%d",
448             myport >> 8, myport & 255);
449         return (strdup(tmp));   /* Doesn't return static data ! */
450 }

452 /*
453  * _uaddr2taddr() translates a universal address back into a
454  * netaddr structure.  Since the universal address is a string,
455  * put that into the TLI buffer (making sure to change all \ddd
456  * characters back and strip off the trailing \0 character).
```



```
457  */
458 struct netbuf *
459 /* ARGSUSED */
460 _uaddr2taddr(struct netconfig *netconfigp, char *uaddr)
461 {
462         assert(0);
463         _nderror = ND_SYSTEM;
464         return (NULL);
465 }

467 /*
468  * _netdir_options() is a "catch-all" routine that does
469  * transport specific things.  The only thing that these
470  * routines have to worry about is ND_MERGEADDR.
471  */
472 int
473 /* ARGSUSED */
474 _netdir_options(struct netconfig *netconfigp, int option, int fd, void *par)
475 {
476         assert(0);
477         _nderror = ND_SYSTEM;
478         return (0);
479 }
480 #endif /* ! codereview */
```

```
**********************************************************
    1388 Tue Jan 14 16:17:10 2014
new/usr/src/lib/brand/lx/lx_nametoaddr/common/mapfile-vers
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

#
# MAPFILE HEADER START
#
# WARNING:  STOP NOW.  DO NOT MODIFY THIS FILE.
# Object versioning must comply with the rules detailed in
#
#       usr/src/lib/README.mapfiles
#
# You should not be making modifications here until you've read the most current
# copy of that file. If you need help, contact a gatekeeper for guidance.
#
# MAPFILE HEADER END
#

SUNWprivate_1.1 {
        global:
                _netdir_getbyname;
                _netdir_getbyaddr;
                _taddr2uaddr;
                _uaddr2taddr;
                _netdir_options;

        local:
                *;
};
#endif /* ! codereview */
```

```
**********************************************************
    1094 Tue Jan 14 16:17:11 2014
new/usr/src/lib/brand/lx/lx_nametoaddr/i386/Makefile
Bring back LX zones.
**********************************************************
```

```
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #

  22 #
  23 # Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  24 # Use is subject to license terms.
  25 #
  26 # ident "%Z%%M% %I%     %E% SMI"
  27 #

  29 include ../Makefile.com

  31 DYNFLAGS +=     $(LX_THUNK)/$(MACH)/lx_thunk.so.1
  32 CLOBBERFILES =  $(ROOTLIBDIR)/$(DYNLIB) $(ROOTLIBDIR)/$(LINTLIB)

  34 install: all $(ROOTLIBS)
  35 #endif /* ! codereview */
```

****
1368 Tue Jan 14 16:17:11 2014
new/usr/src/lib/brand/lx/lx_support/Makefile
Bring back LX zones.
****

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

PROG =          lx_support
PROGS =         $(PROG)
OBJS =          lx_support

all:            $(PROG)

include ../Makefile.lx
include $(SRC)/cmd/Makefile.cmd

# override the install directory
ROOTBIN =       $(ROOTBRANDDIR)
CLOBBERFILES =  $(OBJS) $(ROOTPROGS)

UTSBASE =       $(SRC)/uts

CFLAGS +=       $(CCVERBOSE)
CPPFLAGS +=     -D_REENTRANT -I$(UTSBASE)/common/brand/lx
LDLIBS +=       -lzonecfg

.KEEP_STATE:

install:        all $(ROOTPROGS)

clean:
                $(RM) $(PROG) $(OBJS)

lint:           lint_PROG

include $(SRC)/cmd/Makefile.targ
#endif /* ! codereview */
```

```
**********************************************************
   16511 Tue Jan 14 16:17:11 2014
new/usr/src/lib/brand/lx/lx_support/lx_support.c
LX zone support should now build and packages of relevance produced.
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * lx_support is a small cli utility used to perform some brand-specific
 * tasks when booting, halting, or verifying a zone.  This utility is not
 * intended to be called by users - it is intended to be invoked by the
 * zones utilities.
 */

#include <ctype.h>
#include <errno.h>
#include <fcntl.h>
#include <libgen.h>
#include <limits.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <stropts.h>
#include <sys/ioccom.h>
#include <sys/stat.h>
#include <sys/systeminfo.h>
#include <sys/types.h>
#include <sys/varargs.h>
#include <unistd.h>
#include <libintl.h>
#include <locale.h>

#include <libzonecfg.h>
#include <sys/lx_audio.h>
#include <sys/lx_brand.h>

static void lxs_err(char *msg, ...) __NORETURN;
static void usage(void) __NORETURN;

#define CP_CMD          "/usr/bin/cp"
```



```
#define MOUNT_CMD       "/sbin/mount"

#define LXA_AUDIO_DEV           "/dev/brand/lx/audio_devctl"
#define INTSTRLEN               32
#define KVSTRLEN                10

static char *bname = NULL;
static char *zonename = NULL;
static char *zoneroot = NULL;

#if !defined(TEXT_DOMAIN)               /* should be defined by cc -D */
#define TEXT_DOMAIN     "SYS_TEST"      /* Use this only if it wasn't */
#endif

static void
lxs_err(char *msg, ...)
{
        char    buf[1024];
        va_list ap;

        va_start(ap, msg);
        /*LINTED*/
        (void) vsnprintf(buf, sizeof (buf), msg, ap);
        va_end(ap);

        (void) printf("%s error: %s\n", bname, buf);

        exit(1);
        /*NOTREACHED*/
}

/*
 * The Linux init(1M) command requires communication over the /dev/initctl
 * FIFO.  Since any attempt to create a file in /dev will fail, we must
 * create it here.
 */
static void
lxs_make_initctl()
{
        char            cmdbuf[ARG_MAX];
        char            path[MAXPATHLEN];
        char            special[MAXPATHLEN];
        struct stat     buf;
        int             err;

        if (snprintf(special, sizeof (special), "%s/dev/initctl", zoneroot) >=
            sizeof (special))
                lxs_err("%s: %s", gettext("Failed to create /dev/initctl"),
                    gettext("zoneroot is too long"));

        if (snprintf(path, sizeof (path), "%s/root/dev/initctl", zoneroot) >=
            sizeof (path))
                lxs_err("%s: %s", gettext("Failed to create /dev/initctl"),
                    gettext("zoneroot is too long"));

        /* create the actual fifo as <zoneroot>/dev/initctl */
        if (stat(special, &buf) != 0) {
                err = errno;
                if (err != ENOENT)
                        lxs_err("%s: %s",
                            gettext("Failed to create /dev/initctl"),
                            strerror(err));
                if (mkfifo(special, 0644) < 0) {
                        err = errno;
                        lxs_err("%s: %s",
                            gettext("Failed to create /dev/initctl"),
```

```
127                             strerror(err));
128                 }
129         } else {
130                 if ((buf.st_mode & S_IFIFO) == 0)
131                         lxs_err("%s: %s",
132                             gettext("Failed to create /dev/initctl"),
133                             gettext("It already exists, and is not a FIFO."));
134         }

136         /*
137          * now lofs mount the <zoneroot>/dev/initctl fifo onto
138          * <zoneroot>/root/dev/initctl
139          */
140         if (snprintf(cmdbuf, sizeof (cmdbuf), "%s -F lofs %s %s", MOUNT_CMD,
141             special, path) >= sizeof (cmdbuf))
142                 lxs_err("%s: %s", gettext("Failed to lofs mount /dev/initctl"),
143                     gettext("zoneroot is too long"));

145         if (system(cmdbuf) < 0) {
146                 err = errno;
147                 lxs_err("%s: %s", gettext("Failed to lofs mount /dev/initctl"),
148                     strerror(err));
149         }
150 }

152 /*
153  * fsck gets really confused when run inside a zone.  Removing this file
154  * prevents it from running
155  */
156 static void
157 lxs_remove_autofsck()
158 {
159         char    path[MAXPATHLEN];
160         int     err;

162         if (snprintf(path, MAXPATHLEN, "%s/root/.autofsck", zoneroot) >=
163             MAXPATHLEN)
164                 lxs_err("%s: %s", gettext("Failed to remove /.autofsck"),
165                     gettext("zoneroot is too long"));

167         if (unlink(path) < 0) {
168                 err = errno;
169                 if (err != ENOENT)
170                         lxs_err("%s: %s",
171                             gettext("Failed to remove /.autofsck"),
172                             strerror(err));
173         }
174 }

176 /*
177  * Extract any lx-supported attributes from the zone configuration file.
178  */
179 static void
180 lxs_getattrs(zone_dochandle_t zdh, boolean_t *restart, boolean_t *audio,
181     char **idev, char **odev, char **kvers)
182 {
183         struct zone_attrtab     attrtab;
184         int                     err;

186         /* initialize the attribute iterator */
187         if (zonecfg_setattrent(zdh) != Z_OK) {
188                 zonecfg_fini_handle(zdh);
189                 lxs_err(gettext("error accessing zone configuration"));
190         }

192         *idev = (char *)malloc(INTSTRLEN);
```



```
193         *odev = (char *)malloc(INTSTRLEN);
194         *kvers = (char *)malloc(KVSTRLEN);
195         if (*idev == NULL || *odev == NULL || *kvers == NULL)
196                 lxs_err(gettext("out of memory"));

198         *audio = B_FALSE;
199         *restart = B_FALSE;
200         bzero(*idev, INTSTRLEN);
201         bzero(*odev, INTSTRLEN);
202         bzero(*kvers, KVSTRLEN);
203         while ((err = zonecfg_getattrent(zdh, &attrtab)) == Z_OK) {
204                 if ((strcmp(attrtab.zone_attr_name, "init-restart") == 0) &&
205                     (zonecfg_get_attr_boolean(&attrtab, restart) != Z_OK))
206                         lxs_err(gettext("invalid type for zone attribute: %s"),
207                             attrtab.zone_attr_name);
208                 if ((strcmp(attrtab.zone_attr_name, "audio") == 0) &&
209                     (zonecfg_get_attr_boolean(&attrtab, audio) != Z_OK))
210                         lxs_err(gettext("invalid type for zone attribute: %s"),
211                             attrtab.zone_attr_name);
212                 if ((strcmp(attrtab.zone_attr_name, "audio-inputdev") == 0) &&
213                     (zonecfg_get_attr_string(&attrtab, *idev,
214                     INTSTRLEN) != Z_OK))
215                         lxs_err(gettext("invalid type for zone attribute: %s"),
216                             attrtab.zone_attr_name);
217                 if ((strcmp(attrtab.zone_attr_name, "audio-outputdev") == 0) &&
218                     (zonecfg_get_attr_string(&attrtab, *odev,
219                     INTSTRLEN) != Z_OK))
220                         lxs_err(gettext("invalid type for zone attribute: %s"),
221                             attrtab.zone_attr_name);
222                 if ((strcmp(attrtab.zone_attr_name, "kernel-version") == 0) &&
223                     (zonecfg_get_attr_string(&attrtab, *kvers,
224                     KVSTRLEN) != Z_OK))
225                         lxs_err(gettext("invalid type for zone attribute: %s"),
226                             attrtab.zone_attr_name);
227         }

229         if (strlen(*kvers) == 0) {
230                 free(*kvers);
231                 *kvers = NULL;
232         }

234         /* some kind of error while looking up attributes */
235         if (err != Z_NO_ENTRY)
236                 lxs_err(gettext("error accessing zone configuration"));
237 }

239 static int
240 lxs_iodev_ok(char *dev)
241 {
242         int i, j;

244         if ((j = strlen(dev)) == 0)
245                 return (1);
246         if (strcmp(dev, "default") == 0)
247                 return (1);
248         if (strcmp(dev, "none") == 0)
249                 return (1);
250         for (i = 0; i < j; i++) {
251                 if (!isdigit(dev[i]))
252                         return (0);
253         }
254         return (1);
255 }

257 /*
258  * The audio configuration settings are read from the zone configuration
```

```
 * file.  Audio configuration is specified via the following attributes
 * (settable via zonecfg):
 *      attr name: audio
 *      attr type: boolean
 *
 *      attr name: audio-inputdev
 *      attr type: string
 *      attr values: "none" | [0-9]+
 *
 *      attr name: audio-outputdev
 *      attr type: string
 *      attr values: "none" | [0-9]+
 *
 * The user can enable linux brand audio device (ie /dev/dsp and /dev/mixer)
 * for a zone by setting the "audio" attribute to true.  (The absence of
 * this attribute leads to an assumed value of false.)
 *
 * If the "audio" attribute is set to true and "audio-inputdev" and
 * "audio-outputdev" are not set, then when a linux applications access
 * audio devices these access will be mapped to the system default audio
 * device, ie /dev/audio and/dev/audioctl.
 *
 * If "audio-inputdev" is set to none, then audio input will be disabled.
 * If "audio-inputdev" is set to an integer, then when a Linux application
 * attempts to access audio devices these access will be mapped to
 * /dev/sound/<audio-inputdev attribute value>.  The same behavior will
 * apply to the "audio-outputdev" attribute for linux audio output
 * device accesses.
 *
 * If "audio-inputdev" or "audio-outputdev" exist but the audio attribute
 * is missing (or set to false) audio will not be enabled for the zone.
 */
static void
lxs_init_audio(char *idev, char *odev)
{
        int                     err, fd;
        lxa_zone_reg_t          lxa_zr;

        /* sanity check the input and output device properties */
        if (!lxs_iodev_ok(idev))
                lxs_err(gettext("invalid value for zone attribute: %s"),
                    "audio-inputdev");

        if (!lxs_iodev_ok(odev))
                lxs_err(gettext("invalid value for zone attribute: %s"),
                    "audio-outputdev");

        /* initialize the zone name in the ioctl request */
        bzero(&lxa_zr, sizeof (lxa_zr));
        (void) strlcpy(lxa_zr.lxa_zr_zone_name, zonename,
            sizeof (lxa_zr.lxa_zr_zone_name));

        /* initialize the input device property in the ioctl request */
        (void) strlcpy(lxa_zr.lxa_zr_inputdev, idev,
            sizeof (lxa_zr.lxa_zr_inputdev));
        if (lxa_zr.lxa_zr_inputdev[0] == '\0') {
                /*
                 * if no input device was specified, set the input device
                 * to "default"
                 */
                (void) strlcpy(lxa_zr.lxa_zr_inputdev, "default",
                    sizeof (lxa_zr.lxa_zr_inputdev));
        }

        /* initialize the output device property in the ioctl request */
        (void) strlcpy(lxa_zr.lxa_zr_outputdev, odev,
```



```
            sizeof (lxa_zr.lxa_zr_outputdev));
        if (lxa_zr.lxa_zr_outputdev[0] == '\0') {
                /*
                 * if no output device was specified, set the output device
                 * to "default"
                 */
                (void) strlcpy(lxa_zr.lxa_zr_outputdev, "default",
                    sizeof (lxa_zr.lxa_zr_outputdev));
        }

        /* open the audio device control node */
        if ((fd = open(LXA_AUDIO_DEV, O_RDWR)) < 0)
                lxs_err(gettext("error accessing lx_audio device"));

        /* enable audio for this zone */
        err = ioctl(fd, LXA_IOC_ZONE_REG, &lxa_zr);
        (void) close(fd);
        if (err != 0)
                lxs_err(gettext("error configuring lx_audio device"));
}

static int
lxs_boot()
{
        zoneid_t        zoneid;
        zone_dochandle_t zdh;
        boolean_t       audio, restart;
        char            *idev, *odev, *kvers;
        int             kversnum;

        lxs_make_initctl();
        lxs_remove_autofsck();

        if ((zdh = zonecfg_init_handle()) == NULL)
                lxs_err(gettext("unable to initialize zone handle"));

        if (zonecfg_get_handle((char *)zonename, zdh) != Z_OK) {
                zonecfg_fini_handle(zdh);
                lxs_err(gettext("unable to load zone configuration"));
        }

        /* Extract any relevant attributes from the config file. */
        lxs_getattrs(zdh, &restart, &audio, &idev, &odev, &kvers);
        zonecfg_fini_handle(zdh);

        /* Configure the zone's audio support (if any). */
        if (audio == B_TRUE)
                lxs_init_audio(idev, odev);

        /*
         * Let the kernel know whether or not this zone's init process
         * should be automatically restarted on its death.
         */
        if ((zoneid = getzoneidbyname(zonename)) < 0)
                lxs_err(gettext("unable to get zoneid"));
        if (zone_setattr(zoneid, LX_ATTR_RESTART_INIT, &restart,
            sizeof (boolean_t)) == -1)
                lxs_err(gettext("error setting zone's restart_init property"));

        if ((kvers != NULL) && (strcmp(kvers, "2.6") == 0))
                kversnum = LX_KERN_2_6;
        else
                kversnum = LX_KERN_2_4;

        if (zone_setattr(zoneid, LX_KERN_VERSION_NUM, &kversnum,
            sizeof (int)) < 0)
```

```
391                 lxs_err(gettext("unable to set kernel version"));

393         return (0);
394 }

396 static int
397 lxs_halt()
398 {
399         lxa_zone_reg_t  lxa_zr;
400         int             fd, rv;

402         /*
403          * We don't bother to check if audio is configured for this zone
404          * before issuing a request to unconfigure it.  There's no real
405          * reason to do this, it would require looking up the xml zone and
406          * brand configuration information (which could have been changed
407          * since the zone was booted), and it would involve more library
408          * calls there by increasing chances for failure.
409          */

411         /* initialize the zone name in the ioctl request */
412         bzero(&lxa_zr, sizeof (lxa_zr));
413         (void) strlcpy(lxa_zr.lxa_zr_zone_name, zonename,
414             sizeof (lxa_zr.lxa_zr_zone_name));

416         /* open the audio device control node */
417         if ((fd = open(LXA_AUDIO_DEV, O_RDWR)) < 0)
418                 lxs_err(gettext("error accessing lx_audio device"));

420         /*
421          * disable audio for this zone
422          *
423          * we ignore ENOENT errors here because it's possible that
424          * audio is not configured for this zone.  (either it was
425          * already unconfigured or someone could have added the
426          * audio resource to this zone after it was booted.)
427          */
428         rv = ioctl(fd, LXA_IOC_ZONE_UNREG, &lxa_zr);
429         (void) close(fd);
430         if ((rv == 0) || (errno == ENOENT))
431                 return (0);
432         lxs_err(gettext("error unconfiguring lx_audio device: %s"),
433             strerror(errno));
434         /*NOTREACHED*/
435 }

437 static int
438 lxs_verify(char *xmlfile)
439 {
440         zone_dochandle_t        handle;
441 /*      struct zone_fstab       fstab; */
442         struct zone_dstab       dstab;
443         struct zone_devtab      devtab;
444         boolean_t               audio, restart;
445         char                    *idev, *odev, *kvers;
446         zone_iptype_t           iptype;
447         char                    hostidp[HW_HOSTID_LEN];

449         if ((handle = zonecfg_init_handle()) == NULL)
450                 lxs_err(gettext("internal libzonecfg.so.1 error"), 0);

452         if (zonecfg_get_xml_handle(xmlfile, handle) != Z_OK) {
453                 zonecfg_fini_handle(handle);
454                 lxs_err(gettext("zonecfg provided an invalid XML file"));
455         }
```



```
457         /*
458          * Check to see whether the zone has any inherit-pkg-dirs
459          * configured.
460          */
461 /*      if (zonecfg_setipdent(handle) != Z_OK) {
462                 zonecfg_fini_handle(handle);
463                 lxs_err(gettext("zonecfg provided an invalid XML file"));
464         }

466         if (zonecfg_getipdent(handle, &fstab) == Z_OK) {
467                 zonecfg_fini_handle(handle);
468                 lxs_err(gettext("lx zones do not support inherit-pkg-dirs"));
469         }
470 */
471         /*
472          * Check to see whether the zone has any ZFS datasets configured.
473          */
474         if (zonecfg_setdsent(handle) != Z_OK) {
475                 zonecfg_fini_handle(handle);
476                 lxs_err(gettext("zonecfg provided an invalid XML file"));
477         }

479         if (zonecfg_getdsent(handle, &dstab) == Z_OK) {
480                 zonecfg_fini_handle(handle);
481                 lxs_err(gettext("lx zones do not support ZFS datasets"));
482         }

484         /*
485          * Check to see whether the zone has any devices configured.
486          */
487         if (zonecfg_setdevent(handle) != Z_OK) {
488                 zonecfg_fini_handle(handle);
489                 lxs_err(gettext("zonecfg provided an invalid XML file"));
490         }

492         if (zonecfg_getdevent(handle, &devtab) == Z_OK) {
493                 zonecfg_fini_handle(handle);
494                 lxs_err(gettext("lx zones do not support added devices"));
495         }

497         /*
498          * Check to see whether the zone has ip-type configured as exclusive
499          */
500         if (zonecfg_get_iptype(handle, &iptype) != Z_OK) {
501                 zonecfg_fini_handle(handle);
502                 lxs_err(gettext("zonecfg provided an invalid XML file"));
503         }

505         if (iptype == ZS_EXCLUSIVE) {
506                 zonecfg_fini_handle(handle);
507                 lxs_err(gettext("lx zones do not support an 'exclusive' "
508                     "ip-type"));
509         }

511         /*
512          * Check to see whether the zone has hostid emulation enabled.
513          */
514         if (zonecfg_get_hostid(handle, hostidp, sizeof (hostidp)) == Z_OK) {
515                 zonecfg_fini_handle(handle);
516                 lxs_err(gettext("lx zones do not support hostid emulation"));
517         }

519         /* Extract any relevant attributes from the config file. */
520         lxs_getattrs(handle, &restart, &audio, &idev, &odev, &kvers);
521         zonecfg_fini_handle(handle);
```

```
523         if (audio) {
524                 /* sanity check the input and output device properties */
525                 if (!lxs_iodev_ok(idev))
526                         lxs_err(gettext("invalid value for zone attribute: %s"),
527                             "audio-inputdev");

529                 if (!lxs_iodev_ok(odev))
530                         lxs_err(gettext("invalid value for zone attribute: %s"),
531                             "audio-outputdev");
532         }
533         if (kvers) {
534                 if ((strcmp(kvers, "2.4")) != 0 && (strcmp(kvers, "2.6") != 0))
535                         lxs_err(gettext("invalid value for zone attribute: %s"),
536                             "kernel-version");
537         }
538         return (0);
539 }

541 static void
542 usage()
543 {

545         (void) fprintf(stderr,
546             gettext("usage:\t%s boot <zoneroot> <zonename>\n"), bname);
547         (void) fprintf(stderr,
548             gettext("      \t%s halt <zoneroot> <zonename>\n"), bname);
549         (void) fprintf(stderr,
550             gettext("      \t%s verify <xml file>\n\n"), bname);
551         exit(1);
552 }

554 int
555 main(int argc, char *argv[])
556 {
557         (void) setlocale(LC_ALL, "");
558         (void) textdomain(TEXT_DOMAIN);

560         bname = basename(argv[0]);

562         if (argc < 3)
563                 usage();

565         if (strcmp(argv[1], "boot") == 0) {
566                 if (argc != 4)
567                         lxs_err(gettext("usage: %s %s <zoneroot> <zonename>"),
568                             bname, argv[1]);
569                 zoneroot = argv[2];
570                 zonename = argv[3];
571                 return (lxs_boot());
572         }

574         if (strcmp(argv[1], "halt") == 0) {
575                 if (argc != 4)
576                         lxs_err(gettext("usage: %s %s <zoneroot> <zonename>"),
577                             bname, argv[1]);
578                 zoneroot = argv[2];
579                 zonename = argv[3];
580                 return (lxs_halt());
581         }

583         if (strcmp(argv[1], "verify") == 0) {
584                 if (argc != 3)
585                         lxs_err(gettext("usage: %s verify <xml file>"),
586                             bname);
587                 return (lxs_verify(argv[2]));
588         }
```



```
590         usage();
591         /*NOTREACHED*/
592 }
593 #endif /* ! codereview */
```

```
**********************************************************
    1356 Tue Jan 14 16:17:11 2014
new/usr/src/lib/brand/lx/lx_thunk/Makefile
Bring back LX zones.
**********************************************************
```

```
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #

  22 #
  23 # Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  24 # Use is subject to license terms.
  25 #
  26 # ident "%Z%%M% %I%     %E% SMI"
  27 #

  29 include ../../../Makefile.lib

  31 SUBDIRS = $(MACH)
  32 $(BUILD64)SUBDIRS += $(MACH64)

  34 LINT_SUBDIRS = $(MACH)
  35 $(BUILD64)LINT_SUBDIRS += $(MACH64)

  37 all :=          TARGET= all
  38 clean :=        TARGET= clean
  39 clobber :=      TARGET= clobber
  40 install :=      TARGET= install
  41 lint :=         TARGET= lint

  43 .KEEP_STATE:

  45 all install clean clobber: $(SUBDIRS)

  47 lint: $(LINT_SUBDIRS)

  49 $(SUBDIRS): FRC
  50         @cd $@; pwd; $(MAKE) $(TARGET)

  52 FRC:
  53 #endif /* ! codereview */
```

```
**********************************************************
    1836 Tue Jan 14 16:17:11 2014
new/usr/src/lib/brand/lx/lx_thunk/Makefile.com
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "%Z%%M% %I%     %E% SMI"
#

LIBRARY =       lx_thunk.a
VERS =          .1

COBJS =         lx_thunk.o
OBJECTS =       $(COBJS)

include ../../../../Makefile.lib
include ../../Makefile.lx

#
# Since our name doesn't start with "lib", Makefile.lib incorrectly
# calculates LIBNAME. Therefore, we set it here.
#
LIBNAME =       lx_thunk

MAPFILES =      ../common/mapfile-vers
MAPOPTS =       $(MAPFILES:%=-M%)

CSRCS =         $(COBJS:%o=../common/%c)
SRCS =          $(CSRCS)

SRCDIR =        ../common
UTSBASE =       ../../../../../uts

ASFLAGS +=      -P -D_ASM
LDLIBS +=       -lc
CFLAGS +=       $(CCVERBOSE)
CPPFLAGS +=     -D_REENTRANT -I../ -I ../../lx_brand \
                        -I$(UTSBASE)/common/brand/lx

# lx_think.so.1 interposes on a number of libc.so.1 routines.
DYNFLAGS +=     $(MAPOPTS) $(ZINTERPOSE)
```



```
LIBS =          $(DYNLIB)

CLEANFILES =    $(DYNLIB)
ROOTLIBDIR =    $(ROOT)/usr/lib/brand/lx
ROOTLIBDIR64 =  $(ROOT)/usr/lib/brand/lx/$(MACH64)

.KEEP_STATE:

all: $(DYNLIB)

lint: $(LINTLIB) lintcheck

include ../../../../Makefile.targ
#endif /* ! codereview */
```

```
**********************************************************
    1089 Tue Jan 14 16:17:11 2014
new/usr/src/lib/brand/lx/lx_thunk/amd64/Makefile
Bring back LX zones.
**********************************************************
```

```
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #

  22 #
  23 # Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  24 # Use is subject to license terms.
  25 #
  26 # ident "%Z%%M% %I%     %E% SMI"
  27 #

  29 include ../Makefile.com
  30 include $(SRC)/lib/Makefile.lib.64

  32 CLOBBERFILES    = $(ROOTLIBDIR64)/$(DYNLIB) $(ROOTLIBDIR64)/$(LINTLIB)

  34 install: all $(ROOTLIBS64)
  35 #endif /* ! codereview */
```

```
**********************************************************
   31455 Tue Jan 14 16:17:12 2014
new/usr/src/lib/brand/lx/lx_thunk/common/lx_thunk.c
LX zone support should now build and packages of relevance produced.
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

/*
 * The BrandZ Linux thunking library.
 *
 * The interfaces defined in this file form the client side of a bridge
 * to allow native Solaris process to access Linux services.  Currently
 * the Linux services that is made accessible by these interfaces here
 * are:
 *      - Linux host <-> address naming services
 *      - Linux service <-> port naming services
 *      - Linux syslog
 *
 * Currently, to use this library it must be LD_PRELOADed into the
 * application that needs to access Linux services.  Once loaded
 * Linux services are accessed by the client application in two
 * different ways:
 *
 * - Direct library calls:
 *      lxt_gethostbyname_r
 *      lxt_gethostbyaddr_r
 *      lxt_getservbyname_r
 *      lxt_getservbyport_r
 *      lxt_debug
 *
 *   These library functions are used by the BrandZ lx name services
 *   translation library (lx_nametoaddr.so) to handle libnsl.so name
 *   service requests.
 *
 * - Intercepted library calls:
 *      openlog(3c)
 *      syslog(3c)
 *      vsyslog(3c)
 *      closelog(3c)
```



```
 *
 *   Via the LD_PRELOAD mechanism this library interposes itself on
 *   these interfaces and when the application calls these interfaces
 *   (either directly or indirectly via any libraries the program may
 *   be linked against) this library intercepts the request and passes
 *   it onto a Linux process to handle the request.
 *
 * Once this library receives a request that needs to be serviced by a
 * Linux process, it packs up that request and attempts to send it
 * to a doors server.  The door server interfaces are defined in
 * lx_thunk_server.h.  If the doors server is not running or not
 * responding, this library will attempt to spawn a new doors server
 * by forking and executing the following shell script (which runs as
 * a native /bin/sh Linux process):
 *      /native/usr/lib/brand/lx/lx_thunk
 *
 * Notes:
 * - This library also intercepts the following system calls:
 *      close(2) - We intercept close(2) to prevent the caller from
 *              accidentally closing any of the file descriptors we
 *              need to do our work.
 *
 *      setppriv(2) - We intercept setppriv(2) to prevent a process
 *              from dropping any of the privileges we'll need to create
 *              a new lx_thunk server process and to deal with service
 *              requests.
 *
 * - To facilitate the running of native Solaris programs and libraries
 *   when this library is preloaded into an application it will chroot()
 *   into /native.  This way the Solaris application and libraries can
 *   access files via their expected paths and we can avoid having to
 *   either do path mapping or modifying all libraries to make them
 *   aware of "/native" so that they can pre-pend it to all their
 *   filesystem operations.
 *
 * - This library can only be used with processes that are initially
 *   run by root in a zone.  The reason is that we use the chroot()
 *   system call and this requires the PRIV_PROC_CHROOT privilege,
 *   which non-root users don't have.
 */

#include <alloca.h>
#include <assert.h>
#include <dlfcn.h>
#include <door.h>
#include <errno.h>
#include <fcntl.h>
#include <netdb.h>
#include <netdir.h>
#include <priv.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <synch.h>
#include <sys/brand.h>
#include <sys/fcntl.h>
#include <sys/lx_thunk_server.h>
#include <sys/lx_thunk.h>
#include <sys/mman.h>
#include <sys/priv_impl.h>
#include <sys/stat.h>
#include <sys/syscall.h>
#include <sys/types.h>
#include <sys/wait.h>
```

```
127 #include <thread.h>
128 #include <unistd.h>
129 #include <sys/varargs.h>

131 #define LXT_DOOR_DIR            "/tmp"
132 #define LXT_DOOR_PREFIX         "lxt"
133 #define LXT_MSG_MAXLEN          (128 + MAXPATHLEN)

135 #pragma init(init)

137 typedef uintptr_t (*fp1_t)(uintptr_t);
138 typedef uintptr_t (*fp3_t)(uintptr_t, uintptr_t, uintptr_t);

140 static char     *lxt_debug_path = NULL;         /* debug output file path */
141 static char     lxt_debug_path_buf[MAXPATHLEN];
142 static int      root_fd;
143 static int      debug_fd = -1;

145 void lxt_debug(const char *msg, ...);

147 void
148 init(void)
149 {
150         if (getenv("LX_DEBUG") != NULL) {

152                 /* check if there's a debug log file specified */
153                 lxt_debug_path = getenv("LX_DEBUG_FILE");
154                 if (lxt_debug_path == NULL) {
155                         /* send all debugging output to /dev/tty */
156                         lxt_debug_path = "/dev/tty";
157                 }

159                 (void) strlcpy(lxt_debug_path_buf, lxt_debug_path,
160                     sizeof (lxt_debug_path_buf));
161                 lxt_debug_path = lxt_debug_path_buf;

163                 /*
164                  * Open the debugging output file.  We need to open it
165                  * and hold it open because we're going to call chroot()
166                  * in just a second, so we won't be able to open it later.
167                  */
168                 if ((debug_fd = open(lxt_debug_path,
169                     O_WRONLY|O_APPEND|O_CREAT|O_NDELAY|O_NOCTTY,
170                     0666)) != -1) {
171                         (void) fchmod(debug_fd, 0666);
172                 }
173         }
174         lxt_debug("lxt_init: executing native process");

176         /* Get a fd that points to the root directory */
177         if ((root_fd = open("/", O_RDONLY)) < 0) {
178                 lxt_debug("lxt_init(): "
179                     "failed to open root directory: %s", strerror(errno));
180                 exit(-1);
181         }

183         /*
184          * Now, so that we can avoid having to do path mapping,
185          * just chdir() and chroot() into /native.
186          */
187         if (chdir("/native") != 0) {
188                 lxt_debug("lxt_init(): "
189                     "failed to chdir to /native: %s", strerror(errno));
190                 exit(-1);
191         }
192         if (chroot("/native") != 0) {
```



```
193                 lxt_debug("lxt_init(): "
194                     "failed to chroot to /native: %s", strerror(errno));
195                 exit(-1);
196         }
197 }

199 /*
200  * Linux Thunking Interfaces - Client Side
201  */
202 static mutex_t  lxt_door_lock = DEFAULTMUTEX;
203 static int      lxt_door_fd = -1;

205 static void
206 lxt_server_exec(int fifo_wr, int fifo_rd)
207 {
208         extern const char       **environ;
209         char                    *nullist[] = { NULL };

211         lxt_debug("lxt_server_exec: server starting");

213         /*
214          * First we need to dup our fifos to the file descriptors
215          * the brand library is expecting them to be at.
216          */

218         /* Check if the write fifo needs to be moved aside */
219         if ((fifo_wr == LXT_SERVER_FIFO_RD_FD) &&
220             ((fifo_wr = dup(fifo_wr)) < 0))
221                 return;

223         /* Check if the read fifo needs to be moved aside */
224         if ((fifo_rd == LXT_SERVER_FIFO_WR_FD) &&
225             ((fifo_rd = dup(fifo_rd)) < 0))
226                 return;

228         if ((fifo_wr != LXT_SERVER_FIFO_WR_FD) &&
229             (dup2(fifo_wr, LXT_SERVER_FIFO_WR_FD) < 0))
230                 return;
231         if ((fifo_rd != LXT_SERVER_FIFO_RD_FD) &&
232             (dup2(fifo_rd, LXT_SERVER_FIFO_RD_FD) < 0))
233                 return;

235         /*
236          * We're about to execute a native Linux process.
237          * Since we've been loaded into a Solaris process with
238          * LD_PRELOAD and LD_LIBRARY_PATH we should clear these
239          * variables from the environment before calling exec.
240          */
241         (void) unsetenv("LD_PRELOAD");
242         (void) unsetenv("LD_LIBRARY_PATH");

244         /*
245          * Now we need to exec the thunk server process.  This is a
246          * branded Linux process that will act as a doors server and
247          * service our requests to perform native Linux operations.
248          * Since we're currently running as a native Solaris process
249          * to start up the server we'll use the brand system call to
250          * the kernel that the target of the exec will be a branded
251          * process.
252          */
253         lxt_debug("lxt_server_exec: execing as Linux process");
254         (void) syscall(SYS_brand, B_EXEC_BRAND,
255             LXT_SERVER_BINARY, nullist, environ);
256 }
```

```
static void *
lxt_door_waitpid(void *arg)
{
        pid_t   child_pid = (pid_t)(uintptr_t)arg;
        int     stat;

        (void) waitpid(child_pid, &stat, 0);
        return (NULL);
}

static char *
lxt_door_mkfifo()
{
        char    *path;

        for (;;) {
                path = tempnam(LXT_DOOR_DIR, LXT_DOOR_PREFIX);
                if (path == NULL)
                        return (NULL);
                if (mkfifo(path, S_IWUSR | S_IRUSR) != 0) {
                        if (errno != EEXIST) {
                                free(path);
                                return (NULL);
                        }
                        /* This file path exists, pick a new name. */
                        free(path);
                        continue;
                }
                /* We successfully created the fifo */
                break;
        }
        return (path);
}

static void
lxt_door_init()
{
        char            *fifo1_path = NULL, *fifo2_path = NULL;
        char            fifo1_path_native[MAXPATHLEN];
        int             fifo1_rd = -1, fifo1_wr = -1;
        int             fifo2_rd = -1, fifo2_wr = -1;
        int             junk;
        pid_t           child_pid;
        thread_t        tid;

        lxt_debug("lxt_door_init: preparint to start server");

        /* Create two new fifos. */
        if (((fifo1_path = lxt_door_mkfifo()) == NULL) ||
            ((fifo2_path = lxt_door_mkfifo()) == NULL))
                goto fail;

        (void) snprintf(fifo1_path_native, sizeof (fifo1_path_native),
            "/native%s", fifo1_path);

        /*
         * Open both fifos for reading and writing.  We have to open
         * the read side of the fifo first (because the write side will
         * fail to open if there is no reader) and we have to use the
         * O_NONBLOCK flag (because the read open with hang without it).
         */
        if (((fifo1_rd = open(fifo1_path, O_RDONLY | O_NONBLOCK)) < 0) ||
            ((fifo1_wr = open(fifo1_path, O_WRONLY)) < 0) ||
            ((fifo2_rd = open(fifo2_path, O_RDONLY | O_NONBLOCK)) < 0) ||
            ((fifo2_wr = open(fifo2_path, O_WRONLY)) < 0))
                goto fail;
```



```
        /*
         * Now we have to close the read side of fifo1 and fifo2 and re-open
         * them without the O_NONBLOCK flag.  This is because we're using
         * the fifos for synchronization and when we actually try to read
         * from them we want to block.
         */
        (void) close(fifo1_rd);
        if ((fifo1_rd = open(fifo1_path, O_RDONLY)) < 0)
                goto fail;
        (void) close(fifo2_rd);
        if ((fifo2_rd = open(fifo2_path, O_RDONLY)) < 0)
                goto fail;

        /*
         * Once fifo2 is opened no one will ever need to open it again
         * so delete it now.
         */
        (void) unlink(fifo2_path);
        free(fifo2_path);
        fifo2_path = NULL;

        /* Attempt to fork and start the door server */
        lxt_debug("lxt_door_init: starting server");
        switch (child_pid = fork1()) {
        case -1:
                /* fork1() failed. */
                goto fail;
        case 0:
                /* Child process - new door server. */
                (void) close(fifo1_rd);
                (void) close(fifo2_wr);

                /* Need to chroot back to the real root directory */
                if (fchroot(root_fd) != 0) {
                        lxt_debug("lxt_server_exec: "
                            "failed fchroot(\"/\"): %s", strerror(errno));
                        exit(-1);
                }
                (void) close(root_fd);

                /* Start the server */
                lxt_server_exec(fifo1_wr, fifo2_rd);
                lxt_debug("lxt_server_exec: server init failed");
                exit(-1);
                /*NOTREACHED*/
        }
        /* Parent process - door client. */

        /*
         * fifo2 is used to send the door path to the child.
         * (We can't simply pass it via the address space since the
         * child will need to exec.)  We'll write the name of the door
         * file to fifo2 before we close the read end of the fifo2 so
         * that if the child has exited for some reason we won't get
         * a SIGPIPE.  Note that we're reusing the name of fifo1 as
         * the door path.  Also note that we've pre-pended /native
         * to the fifo/door path.  The reason is that we're chroot'ed
         * to /native, but when the thunking server executes it will
         * be chroot'ed back to the real root directory.
         */
        (void) write(fifo2_wr,
            fifo1_path_native, strlen(fifo1_path_native) + 1);
        (void) close(fifo2_wr);
        (void) close(fifo2_rd);
```

```
391         /*
392          * Start up a thread that will perfom a waitpid() on the child
393          * door server process.  We do this because if the calling
394          * application that is using our interfaces is forking it's own
395          * children and using wait(), then it won't expect to see our
396          * children.  We take advantage of the fact that if there are
397          * wait() and a waitpid() calls in progress at the same time
398          * when a child exists,  preference will be given to any
399          * waitpid() calls that are explicity waiting for that child.
400          * There is of course a window of time where the child could
401          * exit after we've forked it but before we've called waitpid()
402          * where another wait() in this process could collect the result.
403          * There's nothing we can really do to prevent this short of
404          * stopping all the other threads in this process.
405          */
406         (void) thr_create(NULL, 0,
407             lxt_door_waitpid, (void *)(uintptr_t)child_pid, THR_DAEMON, &tid);

409         /*
410          * fifo1 is used for the child process to signal us that the
411          * door server is ready to take requests.
412          */
413         (void) close(fifo1_wr);
414         (void) read(fifo1_rd, &junk, 1);
415         (void) close(fifo1_rd);

417         /* If there was a door that was open, close it now. */

419         if (lxt_door_fd >= 0)
420                 (void) close(lxt_door_fd);
421         /*
422          * The server should be started up by now and fattach()ed the door
423          * server to the fifo/door path.  so if we re-open that path now we
424          * should get a fd to the door server.
425          */
426         lxt_door_fd = open(fifo1_path, O_RDWR);

428         lxt_debug("lxt_door_init: new server door = %d", lxt_door_fd);

430         /* We don't need the fifo/door anymore so delete it. */
431         (void) unlink(fifo1_path);
432         free(fifo1_path);
433         return;

435 fail:
436         if (fifo1_path != NULL)
437                 (void) unlink(fifo1_path);
438         if (fifo2_path != NULL)
439                 (void) unlink(fifo2_path);
440         if (fifo1_rd != -1)
441                 (void) close(fifo1_rd);
442         if (fifo1_wr != -1)
443                 (void) close(fifo1_wr);
444         if (fifo2_rd != -1)
445                 (void) close(fifo2_rd);
446         if (fifo2_wr != -1)
447                 (void) close(fifo2_wr);
448 }

450 static int
451 lxt_door_call(door_arg_t *door_arg, int lock_held)
452 {
453         int fd;

455         if (!lock_held)
456                 (void) mutex_lock(&lxt_door_lock);
```



```
458         /* Get a copy of lxt_door_fd */
459         fd = lxt_door_fd;

461         if (!lock_held)
462                 (void) mutex_unlock(&lxt_door_lock);

464         if (fd == -1) {
465                 lxt_debug("lxt_door_call: no door available");
466                 return (-1);
467         }

469         if (door_call(fd, door_arg) != 0) {
470                 lxt_debug("lxt_door_call: call failed");
471                 return (-1);
472         }
473         if (door_arg->rbuf == NULL) {
474                 lxt_debug("lxt_door_call: call returned NULL");
475                 return (-1);
476         }
477         return (0);
478 }

480 static int
481 lxt_door_request(door_arg_t *door_arg)
482 {
483         door_arg_t              door_ping;
484         lxt_server_arg_t        ping_request, *ping_result;
485         int                     rv, ping_success = 0;

487         /* First just try the door call. */
488         lxt_debug("lxt_door_request: calling server");
489         if (lxt_door_call(door_arg, 0) == 0)
490                 return (0);

492         /* Prepare a door server ping request. */
493         bzero(&door_ping, sizeof (door_ping));
494         bzero(&ping_request, sizeof (ping_request));
495         door_ping.data_ptr      = (char *)&ping_request;
496         door_ping.data_size     = sizeof (ping_request);
497         ping_request.lxt_sa_op = LXT_SERVER_OP_PING;

499         (void) mutex_lock(&lxt_door_lock);

501         /* Ping the doors server. */
502         lxt_debug("lxt_door_request: pinging server");
503         if (lxt_door_call(&door_ping, 1) == 0) {
504                 /*LINTED*/
505                 ping_result = (lxt_server_arg_t *)door_ping.rbuf;
506                 ping_success = ping_result->lxt_sa_success;
507                 (void) munmap(door_ping.rbuf, door_ping.rsize);
508         }

510         if (!ping_success) {
511                 /* The server is not responding so start up a new one. */
512                 lxt_door_init();
513         }
514         (void) mutex_unlock(&lxt_door_lock);

516         /* Retry the original request */
517         lxt_debug("lxt_door_request: calling server, retry");
518         if ((rv = lxt_door_call(door_arg, 0)) == 0)
519                 return (0);
520         return (rv);
521 }
```

```
523 static struct hostent *
524 lxt_gethost(int op, const char *token, int token_len, int type,
525     struct hostent *result, char *buf, int buf_len, int *h_errnop)
526 {
527         door_arg_t              door_arg;
528         lxt_gethost_arg_t       *data;
529         lxt_server_arg_t        *request;
530         int                     request_size, errno_tmp, i;

532         lxt_debug("lxt_gethost: request caught");

534         request_size = sizeof (*request) + sizeof (*data) +
535             token_len + buf_len - 1;
536         if ((request = calloc(1, request_size)) == NULL) {
537                 lxt_debug("lxt_gethost: calloc() failed");
538                 *h_errnop = TRY_AGAIN;
539                 return (NULL);
540         }
541         /*LINTED*/
542         data = (lxt_gethost_arg_t *)&request->lxt_sa_data[0];

544         /* Initialize the server request. */
545         request->lxt_sa_op = op;
546         data->lxt_gh_type = type;
547         data->lxt_gh_token_len = token_len;
548         data->lxt_gh_buf_len = buf_len;
549         data->lxt_gh_storage_len = token_len + token_len;
550         bcopy(token, &data->lxt_gh_storage[0], token_len);

552         /* Initialize door_call() arguments. */
553         bzero(&door_arg, sizeof (door_arg));
554         door_arg.data_ptr       = (char *)request;
555         door_arg.data_size      = request_size;

557         if (lxt_door_request(&door_arg) != 0) {
558                 lxt_debug("lxt_gethost: door_call() failed");
559                 /* Don't know what caused the error so clear errno. */
560                 errno = 0;
561                 *h_errnop = ND_SYSTEM;
562                 free(request);
563                 return (NULL);
564         }

566         free(request);

568         if (door_arg.rbuf == NULL) {
569                 lxt_debug("lxt_gethost: door_call() returned NULL");
570                 /* Don't know what caused the error so clear errno. */
571                 errno = 0;
572                 *h_errnop = ND_SYSTEM;
573                 return (NULL);
574         }

576         /*LINTED*/
577         request = (lxt_server_arg_t *)door_arg.rbuf;
578         /*LINTED*/
579         data = (lxt_gethost_arg_t *)&request->lxt_sa_data[0];

581         /* Check if the remote procedure call failed */
582         if (!request->lxt_sa_success) {
583                 lxt_debug("lxt_gethost: remote function call failed");
584                 errno_tmp = request->lxt_sa_errno;
585                 *h_errnop = data->lxt_gh_h_errno;
586                 (void) munmap(door_arg.rbuf, door_arg.rsize);
587                 errno = errno_tmp;
588                 return (NULL);
```



```
589         }

591         /* Copy out the results and output buffer. */
592         bcopy(&data->lxt_gh_result, result, sizeof (*result));
593         bcopy(&data->lxt_gh_storage[token_len], buf, buf_len);
594         (void) munmap(door_arg.rbuf, door_arg.rsize);

596         /* Now go through the results and convert all offsets to pointers */
597         result->h_name = LXT_OFFSET_TO_PTR(result->h_name, buf);
598         result->h_aliases = LXT_OFFSET_TO_PTR(result->h_aliases, buf);
599         result->h_addr_list = LXT_OFFSET_TO_PTR(result->h_addr_list, buf);
600         for (i = 0; result->h_aliases[i] != NULL; i++) {
601                 result->h_aliases[i] =
602                     LXT_OFFSET_TO_PTR(result->h_aliases[i], buf);
603         }
604         for (i = 0; result->h_addr_list[i] != NULL; i++) {
605                 result->h_addr_list[i] =
606                     LXT_OFFSET_TO_PTR(result->h_addr_list[i], buf);
607         }

609         return (result);
610 }

612 static struct servent *
613 lxt_getserv(int op, const char *token, const int token_len, const char *proto,
614     struct servent *result, char *buf, int buf_len)
615 {
616         door_arg_t              door_arg;
617         lxt_getserv_arg_t       *data;
618         lxt_server_arg_t        *request;
619         int                     request_size, errno_tmp, i;

621         lxt_debug("lxt_getserv: request caught");

623         request_size = sizeof (*request) + sizeof (*data) +
624             token_len + buf_len - 1;
625         if ((request = calloc(1, request_size)) == NULL) {
626                 lxt_debug("lxt_getserv: calloc() failed");
627                 return (NULL);
628         }
629         /*LINTED*/
630         data = (lxt_getserv_arg_t *)&request->lxt_sa_data[0];

632         /* Initialize the server request. */
633         request->lxt_sa_op = op;
634         data->lxt_gs_token_len = token_len;
635         data->lxt_gs_buf_len = buf_len;
636         data->lxt_gs_storage_len = token_len + token_len;
637         bcopy(token, &data->lxt_gs_storage[0], token_len);

639         bzero(data->lxt_gs_proto, sizeof (data->lxt_gs_proto));
640         if (proto != NULL)
641                 (void) strncpy(data->lxt_gs_proto, proto,
642                     sizeof (data->lxt_gs_proto));

644         /* Initialize door_call() arguments. */
645         bzero(&door_arg, sizeof (door_arg));
646         door_arg.data_ptr       = (char *)request;
647         door_arg.data_size      = request_size;

649         /* Call the doors server */
650         if (lxt_door_request(&door_arg) != 0) {
651                 lxt_debug("lxt_getserv: door_call() failed");
652                 /* Don't know what caused the error so clear errno */
653                 errno = 0;
654                 free(request);
```

```
655                 return (NULL);
656         }
657         free(request);

659         if (door_arg.rbuf == NULL) {
660                 lxt_debug("lxt_getserv: door_call() returned NULL");
661                 /* Don't know what caused the error so clear errno */
662                 errno = 0;
663                 return (NULL);
664         }
665         /*LINTED*/
666         request = (lxt_server_arg_t *)door_arg.rbuf;
667         /*LINTED*/
668         data = (lxt_getserv_arg_t *)&request->lxt_sa_data[0];

670         /* Check if the remote procedure call failed */
671         if (!request->lxt_sa_success) {
672                 lxt_debug("lxt_getserv: remote function call failed");
673                 errno_tmp = request->lxt_sa_errno;
674                 (void) munmap(door_arg.rbuf, door_arg.rsize);
675                 errno = errno_tmp;
676                 return (NULL);
677         }

679         /* Copy out the results and output buffer. */
680         bcopy(&data->lxt_gs_result, result, sizeof (*result));
681         bcopy(&data->lxt_gs_storage[token_len], buf, buf_len);
682         (void) munmap(door_arg.rbuf, door_arg.rsize);

684         /*
685          * Now go through the results and convert all offsets to pointers.
686          * See the comments in lxt_server_getserv() for why we need
687          * to subtract 1 from each offset.
688          */
689         result->s_name = LXT_OFFSET_TO_PTR(result->s_name, buf);
690         result->s_proto = LXT_OFFSET_TO_PTR(result->s_proto, buf);
691         result->s_aliases = LXT_OFFSET_TO_PTR(result->s_aliases, buf);
692         for (i = 0; result->s_aliases[i] != NULL; i++) {
693                 result->s_aliases[i] =
694                     LXT_OFFSET_TO_PTR(result->s_aliases[i], buf);
695         }

697         return (result);
698 }

700 static void
701 lxt_openlog(const char *ident, int logopt, int facility)
702 {
703         door_arg_t              door_arg;
704         lxt_openlog_arg_t       *data;
705         lxt_server_arg_t        *request;
706         int                     request_size;

708         request_size = sizeof (*request) + sizeof (*data);
709         if ((request = calloc(1, request_size)) == NULL) {
710                 lxt_debug("lxt_openlog: calloc() failed");
711                 return;
712         }
713         /*LINTED*/
714         data = (lxt_openlog_arg_t *)&request->lxt_sa_data[0];

716         /* Initialize the server request. */
717         request->lxt_sa_op = LXT_SERVER_OP_OPENLOG;
718         data->lxt_ol_facility = facility;
719         data->lxt_ol_logopt = logopt;
720         (void) strlcpy(data->lxt_ol_ident, ident, sizeof (data->lxt_ol_ident));
```



```
722         /* Initialize door_call() arguments. */
723         bzero(&door_arg, sizeof (door_arg));
724         door_arg.data_ptr       = (char *)request;
725         door_arg.data_size      = request_size;

727         /* Call the doors server */
728         if (lxt_door_request(&door_arg) != 0) {
729                 lxt_debug("lxt_openlog: door_call() failed");
730                 free(request);
731                 return;
732         }
733         free(request);

735         if (door_arg.rbuf == NULL) {
736                 lxt_debug("lxt_openlog: door_call() returned NULL");
737                 return;
738         }

740         /*LINTED*/
741         request = (lxt_server_arg_t *)door_arg.rbuf;

743         /* Check if the remote procedure call failed */
744         if (!request->lxt_sa_success) {
745                 lxt_debug("lxt_openlog: remote function call failed");
746         }
747         (void) munmap(door_arg.rbuf, door_arg.rsize);
748 }

750 static void
751 lxt_vsyslog(int priority, const char *message, va_list va)
752 {
753         door_arg_t              door_arg;
754         lxt_syslog_arg_t        *data;
755         lxt_server_arg_t        *request;
756         psinfo_t                p;
757         char                    procfile[PRFNSZ], *buf = NULL, *estr;
758         int                     buf_len, buf_i, estr_len, request_size, procfd;
759         int                     i, key, err_count = 0, tok_count = 0;
760         int                     errno_backup = errno;

762         /*
763          * Here we're going to use vsnprintf() to expand the message
764          * string passed in before we hand it off to a Linux process.
765          * Before we can call vsnprintf() we'll need to do modify the
766          * string to deal with certain special tokens.
767          *
768          * syslog() supports a special '%m' format token that expands to
769          * the error message string associated with the current value
770          * of errno.  Unfortunatly if we pass this token to vsnprintf()
771          * it will choke so we need to expand that token manually here.
772          *
773          * We also need to expand any "%%" characters into "%%%%".
774          * The reason is that we'll be calling vsnprintf() which will
775          * translate "%%%%" back to "%%", which is safe to pass to the
776          * Linux version if syslog.  If we didn't do this then vsnprintf()
777          * would translate "%%" to "%" and then the Linux syslog would
778          * attempt to intrepret "%" and whatever character follows it
779          * as a printf format style token.
780          */
781         for (key = i = 0; message[i] != '\0'; i++) {
782                 if (!key && message[i] == '%') {
783                         key = 1;
784                         continue;
785                 }
786                 if (key && message[i] == '%')
```

```
787                     tok_count++;
788             if (key && message[i] == 'm')
789                     err_count++;
790             key = 0;
791     }

793     /* We found some tokens that we need to expand. */
794     if (err_count || tok_count) {
795             estr = strerror(errno_backup);
796             estr_len = strlen(estr);
797             assert(estr_len >= 2);

799             /* Allocate a buffer to hold the expanded string. */
800             buf_len = i + 1 +
801                 (tok_count * 2) + (err_count * (estr_len - 2));
802             if ((buf = calloc(1, buf_len)) == NULL) {
803                     lxt_debug("lxt_vsyslog: calloc() failed");
804                     return;
805             }

807             /* Finally, expand %% and %m. */
808             for (key = buf_i = i = 0; message[i] != '\0'; i++) {
809                     assert(buf_i < buf_len);
810                     if (!key && message[i] == '%') {
811                             buf[buf_i++] = '%';
812                             key = 1;
813                             continue;
814                     }
815                     if (key && message[i] == 'm') {
816                             (void) bcopy(estr, &buf[buf_i - 1], estr_len);
817                             buf_i += estr_len - 1;
818                     } else if (key && message[i] == '%') {
819                             (void) bcopy("%%%%", &buf[buf_i - 1], 4);
820                             buf_i += 4 - 1;
821                     } else {
822                             buf[buf_i++] = message[i];
823                     }
824                     key = 0;
825             }
826             assert(buf[buf_i] == '\0');
827             assert(buf_i == (buf_len - 1));

829             /* Use the expanded buffer as our format string. */
830             message = buf;
831     }

833     /* Allocate the request we're going to send to the server */
834     request_size = sizeof (*request) + sizeof (*data);
835     if ((request = calloc(1, request_size)) == NULL) {
836             lxt_debug("lxt_vsyslog: calloc() failed");
837             return;
838     }

840     /*LINTED*/
841     data = (lxt_syslog_arg_t *)&request->lxt_sa_data[0];

843     /* Initialize the server request. */
844     request->lxt_sa_op = LXT_SERVER_OP_SYSLOG;
845     data->lxt_sl_priority = priority;
846     data->lxt_sl_pid = getpid();
847     (void) vsnprintf(data->lxt_sl_message, sizeof (data->lxt_sl_message),
848         message, va);

850     /* If we did token expansion then free the intermediate buffer. */
851     if (err_count || tok_count)
852             free(buf);
```



```
854     /* Add the current program name into the request */
855     (void) sprintf(procfile, "/proc/%u/psinfo", (int)getpid());
856     /* (void) sprintf(procfile, "/native/proc/%u/psinfo", (int)getpid()); */
857     if ((procfd = open(procfile, O_RDONLY)) >= 0) {
858             if (read(procfd, &p, sizeof (psinfo_t)) >= 0) {
859                     (void) strncpy(data->lxt_sl_progname, p.pr_fname,
860                         sizeof (data->lxt_sl_progname));
861             }
862             (void) close(procfd);
863     }

865     /* Initialize door_call() arguments. */
866     bzero(&door_arg, sizeof (door_arg));
867     door_arg.data_ptr       = (char *)request;
868     door_arg.data_size      = request_size;

870     /* Call the doors server */
871     if (lxt_door_request(&door_arg) != 0) {
872             lxt_debug("lxt_vsyslog: door_call() failed");
873             free(request);
874             return;
875     }
876     free(request);

878     if (door_arg.rbuf == NULL) {
879             lxt_debug("lxt_vsyslog: door_call() returned NULL");
880             return;
881     }

883     /*LINTED*/
884     request = (lxt_server_arg_t *)door_arg.rbuf;

886     /* Check if the remote procedure call failed */
887     if (!request->lxt_sa_success) {
888             lxt_debug("lxt_vsyslog: remote function call failed");
889     }
890     (void) munmap(door_arg.rbuf, door_arg.rsize);
891 }

893 static void
894 lxt_closelog(void)
895 {
896     door_arg_t              door_arg;
897     lxt_server_arg_t        *request;
898     int                     request_size;

900     request_size = sizeof (*request);
901     if ((request = calloc(1, request_size)) == NULL) {
902             lxt_debug("lxt_closelog: calloc() failed");
903             return;
904     }

906     /* Initialize the server request. */
907     request->lxt_sa_op = LXT_SERVER_OP_CLOSELOG;

909     /* Initialize door_call() arguments. */
910     bzero(&door_arg, sizeof (door_arg));
911     door_arg.data_ptr       = (char *)request;
912     door_arg.data_size      = request_size;

914     /* Call the doors server */
915     if (lxt_door_request(&door_arg) != 0) {
916             lxt_debug("lxt_closelog: door_call() failed");
917             free(request);
918             return;
```

```
919         }
920         free(request);

922         if (door_arg.rbuf == NULL) {
923                 lxt_debug("lxt_closelog: door_call() returned NULL");
924                 return;
925         }

927         /*LINTED*/
928         request = (lxt_server_arg_t *)door_arg.rbuf;

930         /* Check if the remote procedure call failed */
931         if (!request->lxt_sa_success) {
932                 lxt_debug("lxt_closelog: remote function call failed");
933         }
934         (void) munmap(door_arg.rbuf, door_arg.rsize);
935 }

937 static void
938 lxt_pset_keep(priv_op_t op, priv_ptype_t type, priv_set_t *pset,
939     const char *priv)
940 {
941         if (priv_ismember(pset, priv) == B_TRUE) {
942                 if (op == PRIV_OFF) {
943                         (void) priv_delset(pset, priv);
944                         lxt_debug("lxt_pset_keep: "
945                             "preventing drop of \"%s\" from \"%s\" set",
946                             priv, type);
947                 }
948         } else {
949                 if (op == PRIV_SET) {
950                         (void) priv_addset(pset, priv);
951                         lxt_debug("lxt_pset_keep: "
952                             "preventing drop of \"%s\" from \"%s\" set",
953                             priv, type);
954                 }
955         }
956 }

958 /*
959  * Public interfaces - used by lx_nametoaddr
960  */
961 void
962 lxt_vdebug(const char *msg, va_list va)
963 {
964         char            buf[LXT_MSG_MAXLEN + 1];
965         int             rv, n;

967         if (debug_fd == -1)
968                 return;

970         /* Prefix the message with pid/tid. */
971         if ((n = snprintf(buf, sizeof (buf), "%u/%u: ",
972             getpid(), thr_self())) == -1)
973                 return;

975         /* Format the message. */
976         if (vsnprintf(&buf[n], sizeof (buf) - n, msg, va) == -1)
977                 return;

979         /* Add a carrige return if there isn't one already. */
980         if ((buf[strlen(buf) - 1] != '\n') &&
981             (strlcat(buf, "\n", sizeof (buf)) >= sizeof (buf)))
982                 return;

984         /* We retry in case of EINTR */
```



```
985         do {
986                 rv = write(debug_fd, buf, strlen(buf));
987         } while ((rv == -1) && (errno == EINTR));
988 }

990 void
991 lxt_debug(const char *msg, ...)
992 {
993         va_list         va;
994         int             errno_backup;

996         if (debug_fd == -1)
997                 return;

999         errno_backup = errno;
1000        va_start(va, msg);
1001        lxt_vdebug(msg, va);
1002        va_end(va);
1003        errno = errno_backup;
1004 }

1006 struct hostent *
1007 lxt_gethostbyaddr_r(const char *addr, int addr_len, int type,
1008     struct hostent *result, char *buf, int buf_len, int *h_errnop)
1009 {
1010        lxt_debug("lxt_gethostbyaddr_r: request recieved");
1011        return (lxt_gethost(LXT_SERVER_OP_ADDR2HOST,
1012            addr, addr_len, type, result, buf, buf_len, h_errnop));
1013 }

1015 struct hostent *
1016 lxt_gethostbyname_r(const char *name,
1017     struct hostent *result, char *buf, int buf_len, int *h_errnop)
1018 {
1019        lxt_debug("lxt_gethostbyname_r: request recieved");
1020        return (lxt_gethost(LXT_SERVER_OP_NAME2HOST,
1021            name, strlen(name) + 1, 0, result, buf, buf_len, h_errnop));
1022 }

1024 struct servent *
1025 lxt_getservbyport_r(int port, const char *proto,
1026     struct servent *result, char *buf, int buf_len)
1027 {
1028        lxt_debug("lxt_getservbyport_r: request recieved");
1029        return (lxt_getserv(LXT_SERVER_OP_PORT2SERV,
1030            (const char *)&port, sizeof (int), proto, result, buf, buf_len));
1031 }

1033 struct servent *
1034 lxt_getservbyname_r(const char *name, const char *proto,
1035     struct servent *result, char *buf, int buf_len)
1036 {
1037        lxt_debug("lxt_getservbyname_r: request recieved");
1038        return (lxt_getserv(LXT_SERVER_OP_NAME2SERV,
1039            name, strlen(name) + 1, proto, result, buf, buf_len));
1040 }

1042 /*
1043  * "Public" interfaces - used to override public existing interfaces
1044  */
1045 #pragma weak _close = close
1046 int
1047 close(int fd)
1048 {
1049        static fp1_t    fp = NULL;
```

```
1051         /*
1052          * Don't let the process close our file descriptor that points
1053          * back to the root directory.
1054          */
1055         if (fd == root_fd)
1056                 return (0);
1057         if (fd == debug_fd)
1058                 return (0);

1060         if (fp == NULL)
1061                 fp = (fp1_t)dlsym(RTLD_NEXT, "close");
1062         return (fp((uintptr_t)fd));
1063 }

1065 int
1066 _setppriv(priv_op_t op, priv_ptype_t type, const priv_set_t *pset)
1067 {
1068         static fp3_t    fp = NULL;
1069         priv_set_t      *pset_new;
1070         int             rv;

1072         lxt_debug("_setppriv: request caught");

1074         if (fp == NULL)
1075                 fp = (fp3_t)dlsym(RTLD_NEXT, "_setppriv");

1077         while ((pset_new = priv_allocset()) == NULL)
1078                 (void) sleep(1);

1080         priv_copyset(pset, pset_new);
1081         lxt_pset_keep(op, type, pset_new, PRIV_PROC_EXEC);
1082         lxt_pset_keep(op, type, pset_new, PRIV_PROC_FORK);
1083         lxt_pset_keep(op, type, pset_new, PRIV_PROC_CHROOT);
1084         lxt_pset_keep(op, type, pset_new, PRIV_FILE_DAC_READ);
1085         lxt_pset_keep(op, type, pset_new, PRIV_FILE_DAC_WRITE);
1086         lxt_pset_keep(op, type, pset_new, PRIV_FILE_DAC_SEARCH);

1088         rv = fp(op, (uintptr_t)type, (uintptr_t)pset_new);
1089         priv_freeset(pset_new);
1090         return (rv);
1091 }

1093 void
1094 openlog(const char *ident, int logopt, int facility)
1095 {
1096         lxt_debug("openlog: request caught");
1097         lxt_openlog(ident, logopt, facility);
1098 }

1100 void
1101 syslog(int priority, const char *message, ...)
1102 {
1103         va_list va;

1105         lxt_debug("syslog: request caught");
1106         va_start(va, message);
1107         lxt_vsyslog(priority, message, va);
1108         va_end(va);
1109 }

1111 void
1112 vsyslog(int priority, const char *message, va_list va)
1113 {
1114         lxt_debug("vsyslog: request caught");
1115         lxt_vsyslog(priority, message, va);
1116 }
```



```
1118 void
1119 closelog(void)
1120 {
1121         lxt_debug("closelog: request caught");
1122         lxt_closelog();
1123 }
1124 #endif /* ! codereview */
```

```
**********************************************************
    1481 Tue Jan 14 16:17:12 2014
new/usr/src/lib/brand/lx/lx_thunk/common/mapfile-vers
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

#
# MAPFILE HEADER START
#
# WARNING:  STOP NOW.  DO NOT MODIFY THIS FILE.
# Object versioning must comply with the rules detailed in
#
#       usr/src/lib/README.mapfiles
#
# You should not be making modifications here until you've read the most current
# copy of that file. If you need help, contact a gatekeeper for guidance.
#
# MAPFILE HEADER END
#

SUNWprivate_1.1 {
        global:
                lxt_vdebug;
                lxt_debug;
                lxt_gethostbyaddr_r;
                lxt_gethostbyname_r;
                lxt_getservbyport_r;
                lxt_getservbyname_r;
                _close;
                _setppriv;
                openlog;
                syslog;
                vsyslog;
                closelog;

        local:
                *;
};
#endif /* ! codereview */
```

```
**********************************************************
    1048 Tue Jan 14 16:17:12 2014
new/usr/src/lib/brand/lx/lx_thunk/i386/Makefile
Bring back LX zones.
**********************************************************
```

```
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #

  22 #
  23 # Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  24 # Use is subject to license terms.
  25 #
  26 # ident "%Z%%M% %I%     %E% SMI"
  27 #

  29 include ../Makefile.com

  31 CLOBBERFILES    = $(ROOTLIBDIR)/$(DYNLIB) $(ROOTLIBDIR)/$(LINTLIB)

  33 install: all $(ROOTLIBS)
  34 #endif /* ! codereview */
```

```
**********************************************************
    1831 Tue Jan 14 16:17:12 2014
new/usr/src/lib/brand/lx/lx_thunk/sys/lx_thunk.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _LX_THUNK_H
#define _LX_THUNK_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

#ifdef  __cplusplus
extern "C" {
#endif

struct hostent *lxt_gethostbyaddr_r(const char *addr, int addr_len, int type,
    struct hostent *result, char *buf, int buf_len, int *h_errnop);
struct hostent *lxt_gethostbyname_r(const char *name,
    struct hostent *result, char *buf, int buf_len, int *h_errnop);
struct servent *lxt_getservbyport_r(int port, const char *proto,
    struct servent *result, char *buf, int buf_len);
struct servent *lxt_getservbyname_r(const char *name, const char *proto,
    struct servent *result, char *buf, int buf_len);

void openlog(const char *ident, int logopt, int facility);
void syslog(int priority, const char *message, ...);
void closelog(void);

void lxt_debug(const char *msg, ...);
void lxt_vdebug(const char *msg, va_list va);

#ifdef  __cplusplus
}
#endif

#endif  /* _LX_THUNK_H */
#endif /* ! codereview */
```

```
**********************************************************
    1227 Tue Jan 14 16:17:12 2014
new/usr/src/lib/brand/lx/netfiles/Makefile
Bring back LX zones.
**********************************************************
```

```
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #

  22 #
  23 # Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  24 # Use is subject to license terms.
  25 #
  26 # ident "%Z%%M% %I%     %E% SMI"
  27 #

  29 TXTS = etc_netconfig etc_default_nfs
  30 NFS_DFL = ../../../../cmd/fs.d/nfs/etc/nfs.dfl

  32 all: $(TXTS)

  34 include ../Makefile.lx

  36 lint:

  38 install: $(ROOTTXTS)

  40 clean:
  41         -$(RM) etc_default_nfs

  43 clobber: clean
  44         -$(RM) $(ROOTXMLDOCS) $(ROOTTXTS)

  46 etc_default_nfs: $(NFS_DFL)
  47         $(RM) $@
  48         $(CP) $(NFS_DFL) $@
  49 #endif /* ! codereview */
```

```
**********************************************************
    1555 Tue Jan 14 16:17:13 2014
new/usr/src/lib/brand/lx/netfiles/etc_netconfig
Bring back LX zones.
**********************************************************
```

```
  1 # CDDL HEADER START
  2 #
  3 # The contents of this file are subject to the terms of the
  4 # Common Development and Distribution License (the "License").
  5 # You may not use this file except in compliance with the License.
  6 #
  7 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
  8 # or http://www.opensolaris.org/os/licensing.
  9 # See the License for the specific language governing permissions
 10 # and limitations under the License.
 11 #
 12 # When distributing Covered Code, include this CDDL HEADER in each
 13 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 14 # If applicable, add the following below this CDDL HEADER, with the
 15 # fields enclosed by brackets "[]" replaced with your own identifying
 16 # information: Portions Copyright [yyyy] [name of copyright owner]
 17 #
 18 # CDDL HEADER END
 19 #
 20 # Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 21 # Use is subject to license terms.
 22 #
 23 # ident "%Z%%M% %I%     %E% SMI"
 24 #
 25 # The "Network Configuration" File.
 26 #
 27 # Each entry is of the form:
 28 #
 29 #       <network_id> <semantics> <flags> <protofamily> <protoname> \
 30 #               <device> <nametoaddr_libs>
 31 #
 32 # For running solaris daemons in a linux zone we use this non-default
 33 # /etc/netconfig.  The reason is that all name resolution has to be
 34 # done linux name service interfaces.  To do this we specify a custom
 35 # nametoaddr library that libnsl will invoke to do name service lookups.
 36 #
 37 udp        tpi_clts      v     inet    udp    /dev/udp        lx_nametoaddr.so.1
 38 tcp        tpi_cots_ord  v     inet    tcp    /dev/tcp        lx_nametoaddr.so.1
 39 #endif /* ! codereview */
```

```
**********************************************************
    1618 Tue Jan 14 16:17:13 2014
new/usr/src/lib/brand/lx/zone/Makefile
Final fixups and bugfixes
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "%Z%%M% %I%     %E% SMI"
#

PROGS = lx_install lx_distro_install lx_init_zone
SUBDIRS = distros
XMLDOCS =       config.xml platform.xml
TEMPLATES =     SUNWlx.xml SUNWlx26.xml

all:    $(PROGS)

include $(SRC)/cmd/Makefile.cmd
include ../Makefile.lx

all :=          TARGET= all
install :=      TARGET= install
clobber :=      TARGET= clobber

POFILES=        $(PROGS:%=%.po)
POFILE=         lx_zone.po

$(POFILE): $(POFILES)
        $(RM) $@
        $(BUILDPO.pofiles)

_msg: $(MSGDOMAINPOFILE)

install: $(PROGS) $(ROOTXMLDOCS) $(ROOTTEMPLATES) $(ROOTPROGS) $(SUBDIRS)

lint:

clean:
        -$(RM) $(PROGS)

clobber: clean $(SUBDIRS)
        -$(RM) $(ROOTXMLDOCS) $(ROOTPROGS) $(ROOTTEMPLATES)
```



```
$(SUBDIRS): FRC
        @cd $@; pwd; $(MAKE) $(TARGET)

FRC:

include $(SRC)/Makefile.msg.targ
#endif /* ! codereview */
```

****
1173 Tue Jan 14 16:17:13 2014
new/usr/src/lib/brand/lx/zone/SUNWlx.xml
Bring back LX zones.
****

```
<?xml version="1.0"?>

<!--
 Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 Use is subject to license terms.

 CDDL HEADER START

 The contents of this file are subject to the terms of the
 Common Development and Distribution License (the "License").
 You may not use this file except in compliance with the License.

 You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 or http://www.opensolaris.org/os/licensing.
 See the License for the specific language governing permissions
 and limitations under the License.

 When distributing Covered Code, include this CDDL HEADER in each
 file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 If applicable, add the following below this CDDL HEADER, with the
 fields enclosed by brackets "[]" replaced with your own identifying
 information: Portions Copyright [yyyy] [name of copyright owner]

 CDDL HEADER END

    ident       "%Z%%M% %I%     %E% SMI"

    DO NOT EDIT THIS FILE.  Use zonecfg(1M) instead.
-->

<!DOCTYPE zone PUBLIC "-//Sun Microsystems Inc//DTD Zones//EN" "file:///usr/shar

<zone name="default" zonepath="" autoboot="false" brand="lx">
</zone>
#endif /* ! codereview */
```

```
**********************************************************
    1231 Tue Jan 14 16:17:13 2014
new/usr/src/lib/brand/lx/zone/SUNWlx26.xml
Final fixups and bugfixes
**********************************************************
```

```
<?xml version="1.0"?>

<!--
 Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 Use is subject to license terms.

 CDDL HEADER START

 The contents of this file are subject to the terms of the
 Common Development and Distribution License (the "License").
 You may not use this file except in compliance with the License.

 You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 or http://www.opensolaris.org/os/licensing.
 See the License for the specific language governing permissions
 and limitations under the License.

 When distributing Covered Code, include this CDDL HEADER in each
 file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 If applicable, add the following below this CDDL HEADER, with the
 fields enclosed by brackets "[]" replaced with your own identifying
 information: Portions Copyright [yyyy] [name of copyright owner]

 CDDL HEADER END

    ident       "%Z%%M% %I%     %E% SMI"

    DO NOT EDIT THIS FILE.  Use zonecfg(1M) instead.
-->

<!DOCTYPE zone PUBLIC "-//Sun Microsystems Inc//DTD Zones//EN" "file:///usr/shar

<zone name="default" zonepath="" autoboot="false" brand="lx">
  <attr name="kernel-version" type="string" value="2.6"/>
</zone>
#endif /* ! codereview */
```

```
**********************************************************
    3806 Tue Jan 14 16:17:13 2014
new/usr/src/lib/brand/lx/zone/config.xml
Bring back LX zones.
**********************************************************
```

```
<?xml version="1.0"?>

<!--
 CDDL HEADER START

 The contents of this file are subject to the terms of the
 Common Development and Distribution License (the "License").
 You may not use this file except in compliance with the License.

 You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 or http://www.opensolaris.org/os/licensing.
 See the License for the specific language governing permissions
 and limitations under the License.

 When distributing Covered Code, include this CDDL HEADER in each
 file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 If applicable, add the following below this CDDL HEADER, with the
 fields enclosed by brackets "[]" replaced with your own identifying
 information: Portions Copyright [yyyy] [name of copyright owner]

 CDDL HEADER END

 Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.

 DO NOT EDIT THIS FILE.
-->

<!DOCTYPE brand PUBLIC "-//Sun Microsystems Inc//DTD Brands//EN"
    "file:///usr/share/lib/xml/dtd/brand.dtd.1">

<brand name="lx">
        <modname>lx_brand</modname>

        <initname>/sbin/init</initname>
        <login_cmd>/bin/login -h zone:%Z %u</login_cmd>
        <forcedlogin_cmd>/bin/login -h zone:%Z -f %u</forcedlogin_cmd>
        <user_cmd>/usr/bin/getent passwd %u</user_cmd>

        <install>/usr/lib/brand/lx/lx_install %z %R</install>
        <installopts>d:hsvX</installopts>
        <boot>/usr/lib/brand/lx/lx_support boot %R %z</boot>
        <halt>/usr/lib/brand/lx/lx_support halt %R %z</halt>
        <verify_cfg>/usr/lib/brand/lx/lx_support verify</verify_cfg>
        <verify_adm></verify_adm>
        <postclone></postclone>
        <postinstall></postinstall>

        <privilege set="default" name="contract_event" />
        <privilege set="default" name="contract_identity" />
        <privilege set="default" name="contract_observer" />
        <privilege set="default" name="file_chown" />
        <privilege set="default" name="file_chown_self" />
        <privilege set="default" name="file_dac_execute" />
        <privilege set="default" name="file_dac_read" />
        <privilege set="default" name="file_dac_search" />
        <privilege set="default" name="file_dac_write" />
        <privilege set="default" name="file_owner" />
        <privilege set="default" name="file_setid" />
        <privilege set="default" name="ipc_dac_read" />
        <privilege set="default" name="ipc_dac_write" />
        <privilege set="default" name="ipc_owner" />
```



```
        <privilege set="default" name="net_bindmlp" />
        <privilege set="default" name="net_icmpaccess" />
        <privilege set="default" name="net_mac_aware" />
        <privilege set="default" name="net_privaddr" />
        <privilege set="default" name="proc_chroot" />
        <privilege set="default" name="sys_audit" />
        <privilege set="default" name="proc_audit" />
        <privilege set="default" name="proc_lock_memory" />
        <privilege set="default" name="proc_owner" />
        <privilege set="default" name="proc_setid" />
        <privilege set="default" name="proc_taskid" />
        <privilege set="default" name="sys_acct" />
        <privilege set="default" name="sys_admin" />
        <privilege set="default" name="sys_mount" />
        <privilege set="default" name="sys_nfs" />
        <privilege set="default" name="sys_resource" />

        <privilege set="prohibited" name="dtrace_kernel" />
        <privilege set="prohibited" name="proc_zone" />
        <privilege set="prohibited" name="sys_config" />
        <privilege set="prohibited" name="sys_devices" />
        <privilege set="prohibited" name="sys_ip_config" />
        <privilege set="prohibited" name="sys_linkdir" />
        <privilege set="prohibited" name="sys_net_config" />
        <privilege set="prohibited" name="sys_res_config" />
        <privilege set="prohibited" name="sys_suser_compat" />
        <privilege set="prohibited" name="xvm_control" />
        <privilege set="prohibited" name="virt_manage" />

        <privilege set="required" name="proc_exec" />
        <privilege set="required" name="proc_fork" />
        <privilege set="required" name="sys_mount" />
</brand>
#endif /* ! codereview */
```

```
**********************************************************
    1348 Tue Jan 14 16:17:13 2014
new/usr/src/lib/brand/lx/zone/distros/Makefile
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

include ../../Makefile.lx

DISTROS =       centos35.distro centos36.distro centos37.distro \
                centos38.distro rhel35.distro rhel36.distro rhel37.distro \
                rhel38.distro rhel_centos_common

ROOTDISTRODIR=  $(ROOTBRANDDIR)/distros
ROOTDISTROS=    $(DISTROS:%=$(ROOTDISTRODIR)/%)

$(ROOTDISTROS) :=       FILEMODE = 444

$(ROOTDISTRODIR):
        $(INS.dir)

$(ROOTDISTRODIR)/%: %
        $(INS.file)

install: $(ROOTDISTROS)

lint clean all:

clobber:
        -$(RM) $(ROOTDISTROS)

#endif /* ! codereview */
```

```
**********************************************************
    2270 Tue Jan 14 16:17:14 2014
new/usr/src/lib/brand/lx/zone/distros/centos35.distro
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
# Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "%Z%%M% %I%     %E% SMI"
#

#
# Installation information for the CentOS 3.5 distribution disc set:
#
#       + Serial number (as found in the disc set's .discinfo file)
#       + Version Name
#       + Order CDs holding the distribution must be installed in
#       + MB of disk space required to hold a full install of the distribution
#
distro_serial=1118161135.08
distro_version="3.5"
set -A distro_cdorder 1 2 3

distro_mb_required=500

# Include the common_<cluster>_* definitions.
. ${distro_dir}/rhel_centos_common

# Define the CentOS 3.5 deltas from the common cluster lists
delta_miniroot_rpms=centos-release
delta_core_rpms="centos-yumconf centos-yumcache yum"
delta_server_rpms=$delta_core_rpms
delta_desktop_rpms="$delta_server_rpms \
        mozilla \
        mozilla-chat \
        mozilla-dom-inspector \
        mozilla-js-debugger \
        mozilla-mail \
        mozilla-nspr \
        mozilla-nss \
        openoffice.org-style-gnome"
delta_developer_rpms=$delta_desktop_rpms
delta_all_rpms=$delta_developer_rpms

# Define the final cluster lists for the installer
distro_miniroot_rpms="$common_miniroot_rpms $delta_miniroot_rpms"
```



```
distro_core_rpms="$common_core_rpms $delta_core_rpms"
distro_server_rpms="$common_server_rpms $delta_server_rpms"
distro_desktop_rpms="$common_desktop_rpms $delta_desktop_rpms"
distro_developer_rpms="$common_developer_rpms $delta_developer_rpms"
distro_all_rpms="$common_all_rpms $delta_all_rpms"
#endif /* ! codereview */
```

```
**********************************************************
    2292 Tue Jan 14 16:17:14 2014
new/usr/src/lib/brand/lx/zone/distros/centos36.distro
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
# Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "%Z%%M% %I%     %E% SMI"
#

#
# Installation information for the CentOS 3.6 distribution disc set:
#
#       + Serial number (as found in the disc set's .discinfo file)
#       + Version Name
#       + Order CDs holding the distribution must be installed in
#       + MB of disk space required to hold a full install of the distribution
#
distro_serial=1130453594.8
distro_version="3.6"
set -A distro_cdorder 1 2 3

distro_mb_required=500

# Include the common_<cluster>_* definitions.
. ${distro_dir}/rhel_centos_common

# Define the CentOS 3.6 deltas from the common cluster lists
delta_miniroot_rpms=centos-release
delta_core_rpms="centos-yumconf centos-yumcache yum"
delta_server_rpms=$delta_core_rpms
delta_desktop_rpms="$delta_server_rpms \
        mozilla \
        mozilla-chat \
        mozilla-dom-inspector \
        mozilla-js-debugger \
        mozilla-mail \
        mozilla-nspr \
        mozilla-nss \
        openoffice.org-style-gnome"
delta_developer_rpms="$delta_desktop_rpms gd-progs"
delta_all_rpms="$delta_developer_rpms emacs-nox"

# Define the final cluster lists for the installer
distro_miniroot_rpms="$common_miniroot_rpms $delta_miniroot_rpms"
```



```
distro_core_rpms="$common_core_rpms $delta_core_rpms"
distro_server_rpms="$common_server_rpms $delta_server_rpms"
distro_desktop_rpms="$common_desktop_rpms $delta_desktop_rpms"
distro_developer_rpms="$common_developer_rpms $delta_developer_rpms"
distro_all_rpms="$common_all_rpms $delta_all_rpms"
#endif /* ! codereview */
```

```
**********************************************************
    2327 Tue Jan 14 16:17:14 2014
new/usr/src/lib/brand/lx/zone/distros/centos37.distro
Bring back LX zones.
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
# Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "%Z%%M% %I%     %E% SMI"
#

#
# Installation information for the CentOS 3.7 distribution disc set:
#
#       + Serial number (as found in the disc set's .discinfo file)
#       + Version Name
#       + Order CDs holding the distribution must be installed in
#       + MB of disk space required to hold a full install of the distribution
#
distro_serial=1144177644.47
distro_version="3.7"
set -A distro_cdorder 1 2 3

distro_mb_required=500

# Include the common_<cluster>_* definitions.
. ${distro_dir}/rhel_centos_common

# Define the CentOS 3.7 deltas from the common cluster lists
delta_miniroot_rpms=centos-release
delta_core_rpms="centos-yumconf centos-yumcache yum"
delta_server_rpms="$delta_core_rpms nss_db-compat sendmail-doc qt-config"
delta_desktop_rpms="$delta_server_rpms \
        mozilla \
        mozilla-chat \
        mozilla-dom-inspector \
        mozilla-js-debugger \
        mozilla-mail \
        mozilla-nspr \
        mozilla-nss"
delta_developer_rpms="$delta_desktop_rpms gd-progs ruby-docs irb ruby-tcltk"
delta_all_rpms="$delta_developer_rpms emacs-nox"

# Define the final cluster lists for the installer
distro_miniroot_rpms="$common_miniroot_rpms $delta_miniroot_rpms"
distro_core_rpms="$common_core_rpms $delta_core_rpms"
```



```
distro_server_rpms="$common_server_rpms $delta_server_rpms"
distro_desktop_rpms="$common_desktop_rpms $delta_desktop_rpms"
distro_developer_rpms="$common_developer_rpms $delta_developer_rpms"
distro_all_rpms="$common_all_rpms $delta_all_rpms"
#endif /* ! codereview */
```

```
**********************************************************
    2518 Tue Jan 14 16:17:14 2014
new/usr/src/lib/brand/lx/zone/distros/centos38.distro
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
# Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "%Z%%M% %I%     %E% SMI"
#

#
# Installation information for the CentOS 3.8 distribution disc set:
#
#       + Serial number (as found in the disc set's .discinfo file)
#       + Version Name
#       + Order CDs holding the distribution must be installed in
#       + MB of disk space required to hold a full install of the distribution
#
distro_serial=1155307611.42
distro_version="3.8"
set -A distro_cdorder 1 2 3

distro_mb_required=500

# Include the common_<cluster>_* definitions.
. ${distro_dir}/rhel_centos_common

# Define the CentOS 3.8 deltas from the common cluster lists
delta_miniroot_rpms=centos-release
delta_core_rpms="centos-yumconf centos-yumcache yum"
delta_server_rpms="$delta_core_rpms nss_db-compat sendmail-doc qt-config"
delta_desktop_rpms="$delta_server_rpms \
        expectk \
        seamonkey \
        seamonkey-chat \
        seamonkey-mail \
        seamonkey-nspr \
        seamonkey-nss \
        tcl-html \
        tcllib"
delta_developer_rpms="$delta_desktop_rpms \
        gd-progs \
        freetype-demos \
        freetype-utils \
        glibc-debug \
```



```
        irb \
        python-docs \
        ruby-docs \
        ruby-tcltk \
        seamonkey-dom-inspector \
        seamonkey-js-debugger \
        seamonkey-devel \
        seamonkey-nspr-devel \
        seamonkey-nss-devel"
delta_all_rpms="$delta_developer_rpms emacs-nox"

# Define the final cluster lists for the installer
distro_miniroot_rpms="$common_miniroot_rpms $delta_miniroot_rpms"
distro_core_rpms="$common_core_rpms $delta_core_rpms"
distro_server_rpms="$common_server_rpms $delta_server_rpms"
distro_desktop_rpms="$common_desktop_rpms $delta_desktop_rpms"
distro_developer_rpms="$common_developer_rpms $delta_developer_rpms"
distro_all_rpms="$common_all_rpms $delta_all_rpms"
#endif /* ! codereview */
```

```
**********************************************************
    2804 Tue Jan 14 16:17:14 2014
new/usr/src/lib/brand/lx/zone/distros/rhel35.distro
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
# Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "%Z%%M% %I%     %E% SMI"
#

#
# Installation information for the RHEL 3 Update 5 distribution disc set:
#
#       + Serial number (as found in the disc set's .discinfo file)
#       + Version Name
#       + Order CDs holding the distribution must be installed in
#       + MB of disk space required to hold a full install of the distribution
#
distro_serial=1115874580.003298
distro_version="Update 5"
set -A distro_cdorder 2 3 4 1

distro_mb_required=500

# Include the common_<cluster>_* definitions.
. ${distro_dir}/rhel_centos_common

# Define the RHEL 3.5 deltas from the common cluster lists
delta_miniroot_rpms=redhat-release
delta_core_rpms=""
delta_server_rpms=$delta_core_rpms
delta_desktop_rpms="$delta_server_rpms \
        mozilla \
        mozilla-chat \
        mozilla-dom-inspector \
        mozilla-js-debugger \
        mozilla-mail \
        mozilla-nspr \
        mozilla-nss \
        openoffice.org-style-gnome"
delta_developer_rpms=$delta_desktop_rpms
delta_all_rpms="$delta_developer_rpms comps"

# Define the final cluster lists for the installer
distro_miniroot_rpms="$common_miniroot_rpms $delta_miniroot_rpms"
```



```
distro_core_rpms="$common_core_rpms $delta_core_rpms"
distro_server_rpms="$common_server_rpms $delta_server_rpms"
distro_desktop_rpms="$common_desktop_rpms $delta_desktop_rpms"
distro_developer_rpms="$common_developer_rpms $delta_developer_rpms"
distro_all_rpms="$common_all_rpms $delta_all_rpms"

#
# List of packages missing from the "WS" personality of this distribution
# as compared to the "AS" personality.
#
distro_WS_missing="amanda-server \
        caching-nameserver \
        finger-server \
        freeradius \
        inews \
        inn \
        krb5-server \
        netdump-server \
        openldap-servers \
        pxe \
        quagga \
        radvd \
        redhat-config-bind \
        samba-swat \
        tftp-server \
        tux \
        vsftpd \
        ypserv \
        arptables_jf \
        mtx \
        redhat-config-netboot"

#
# No packages are missing from the "ES" personality as compared to the "AS"
# personality.
#
unset distro_ES_missing
#endif /* ! codereview */
```

```
**********************************************************
    2810 Tue Jan 14 16:17:15 2014
new/usr/src/lib/brand/lx/zone/distros/rhel36.distro
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
# Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "%Z%%M% %I%     %E% SMI"
#

#
# Installation information for the RHEL 3 Update 6 distribution disc set:
#
#       + Serial number (as found in the disc set's .discinfo file)
#       + Version Name
#       + Order CDs holding the distribution must be installed in
#       + MB of disk space required to hold a full install of the distribution
#
distro_serial=1127323691.616555
distro_version="Update 6"
set -A distro_cdorder 2 3 4 1

distro_mb_required=500

# Include the common_<cluster>_* definitions.
. ${distro_dir}/rhel_centos_common

# Define the RHEL 3.6 deltas from the common cluster lists
delta_miniroot_rpms=redhat-release
delta_core_rpms=""
delta_server_rpms=$delta_core_rpms
delta_desktop_rpms="$delta_server_rpms \
        mozilla \
        mozilla-chat \
        mozilla-dom-inspector \
        mozilla-js-debugger \
        mozilla-mail \
        mozilla-nspr \
        mozilla-nss \
        openoffice.org-style-gnome"
delta_developer_rpms="$delta_desktop_rpms gd-progs"
delta_all_rpms="$delta_developer_rpms emacs-nox comps"

# Define the final cluster lists for the installer
distro_miniroot_rpms="$common_miniroot_rpms $delta_miniroot_rpms"
```



```
distro_core_rpms="$common_core_rpms $delta_core_rpms"
distro_server_rpms="$common_server_rpms $delta_server_rpms"
distro_desktop_rpms="$common_desktop_rpms $delta_desktop_rpms"
distro_developer_rpms="$common_developer_rpms $delta_developer_rpms"
distro_all_rpms="$common_all_rpms $delta_all_rpms"

#
# List of packages missing from the "WS" personality of this distribution
# as compared to the "AS" personality.
#
distro_WS_missing="amanda-server \
        caching-nameserver \
        finger-server \
        freeradius \
        inews \
        inn \
        netdump-server \
        openldap-servers \
        pxe \
        quagga \
        radvd \
        redhat-config-bind \
        samba-swat \
        tftp-server \
        tux \
        vsftpd \
        ypserv \
        arptables_jf \
        mtx \
        redhat-config-netboot"

#
# No packages are missing from the "ES" personality as compared to the "AS"
# personality.
#
unset distro_ES_missing
#endif /* ! codereview */
```

```
**********************************************************
    2844 Tue Jan 14 16:17:15 2014
new/usr/src/lib/brand/lx/zone/distros/rhel37.distro
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
# Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "%Z%%M% %I%     %E% SMI"
#

#
# Installation information for the RHEL 3 Update 7 distribution disc set:
#
#       + Serial number (as found in the disc set's .discinfo file)
#       + Version Name
#       + Order CDs holding the distribution must be installed in
#       + MB of disk space required to hold a full install of the distribution
#
distro_serial=1141679045.364586
distro_version="Update 7"
set -A distro_cdorder 2 3 4 1

distro_mb_required=500

# Include the common_<cluster>_* definitions.
. ${distro_dir}/rhel_centos_common

# Define the RHEL 3.7 deltas from the common cluster lists
delta_miniroot_rpms=redhat-release
delta_core_rpms=""
delta_server_rpms="$delta_core_rpms nss_db-compat sendmail-doc qt-config"
delta_desktop_rpms="$delta_server_rpms \
        mozilla \
        mozilla-chat \
        mozilla-dom-inspector \
        mozilla-js-debugger \
        mozilla-mail \
        mozilla-nspr \
        mozilla-nss"
delta_developer_rpms="$delta_desktop_rpms gd-progs ruby-docs irb ruby-tcltk"
delta_all_rpms="$delta_developer_rpms emacs-nox comps"

# Define the final cluster lists for the installer
distro_miniroot_rpms="$common_miniroot_rpms $delta_miniroot_rpms"
distro_core_rpms="$common_core_rpms $delta_core_rpms"
```



```
distro_server_rpms="$common_server_rpms $delta_server_rpms"
distro_desktop_rpms="$common_desktop_rpms $delta_desktop_rpms"
distro_developer_rpms="$common_developer_rpms $delta_developer_rpms"
distro_all_rpms="$common_all_rpms $delta_all_rpms"

#
# List of packages missing from the "WS" personality of this distribution
# as compared to the "AS" personality.
#
distro_WS_missing="amanda-server \
        caching-nameserver \
        finger-server \
        freeradius \
        inews \
        inn \
        netdump-server \
        openldap-servers \
        pxe \
        quagga \
        radvd \
        redhat-config-bind \
        samba-swat \
        tftp-server \
        tux \
        vsftpd \
        ypserv \
        arptables_jf \
        mtx \
        redhat-config-netboot"

#
# No packages are missing from the "ES" personality as compared to the "AS"
# personality.
#
unset distro_ES_missing
#endif /* ! codereview */
```

```
**********************************************************
    3125 Tue Jan 14 16:17:15 2014
new/usr/src/lib/brand/lx/zone/distros/rhel38.distro
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
# Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "%Z%%M% %I%     %E% SMI"
#

#
# Installation information for the RHEL 3 Update 8 distribution disc set:
#
#       + Serial number (as found in the disc set's .discinfo file)
#       + Version Name
#       + Order CDs holding the distribution must be installed in
#       + MB of disk space required to hold a full install of the distribution
#
distro_serial=1152738297.776178
distro_version="Update 8"
set -A distro_cdorder 2 3 4 1

distro_mb_required=500

# Include the common_<cluster>_* definitions.
. ${distro_dir}/rhel_centos_common

# Define the RHEL 3.8 deltas from the common cluster lists
delta_miniroot_rpms=redhat-release
delta_core_rpms=""
delta_server_rpms="$delta_core_rpms nss_db-compat sendmail-doc qt-config"
delta_desktop_rpms="$delta_server_rpms \
        seamonkey \
        seamonkey-chat \
        seamonkey-mail \
        seamonkey-nspr \
        seamonkey-nss"
delta_developer_rpms="$delta_desktop_rpms \
        gd-progs \
        irb \
        ruby-docs \
        ruby-tcltk \
        seamonkey-dom-inspector \
        seamonkey-js-debugger \
        seamonkey-devel \
```



```
        seamonkey-nspr-devel \
        seamonkey-nss-devel"
delta_all_rpms="$delta_developer_rpms emacs-nox comps"

# Define the final cluster lists for the installer
distro_miniroot_rpms="$common_miniroot_rpms $delta_miniroot_rpms"
distro_core_rpms="$common_core_rpms $delta_core_rpms"
distro_server_rpms="$common_server_rpms $delta_server_rpms"
distro_desktop_rpms="$common_desktop_rpms $delta_desktop_rpms"
distro_developer_rpms="$common_developer_rpms $delta_developer_rpms"
distro_all_rpms="$common_all_rpms $delta_all_rpms"

#
# List of packages missing from the "WS" personality of this distribution
# as compared to the "AS" personality.
#
distro_WS_missing="amanda-server \
        caching-nameserver \
        finger-server \
        freeradius \
        inews \
        inn \
        netdump-server \
        openldap-servers \
        pxe \
        quagga \
        radvd \
        redhat-config-bind \
        samba-swat \
        tftp-server \
        tux \
        vsftpd \
        ypserv \
        arptables_jf \
        mtx \
        redhat-config-netboot"

#
# No packages are missing from the "ES" personality as compared to the "AS"
# personality.
#
unset distro_ES_missing

#
# Identify the packages that need to be set aside for installation after
# all the other packages are installed.
#
deferred_rpms="openoffice.org openoffice.org-i18n openoffice.org-libs"
#endif /* ! codereview */
```

```
**********************************************************
   15489 Tue Jan 14 16:17:15 2014
new/usr/src/lib/brand/lx/zone/distros/rhel_centos_common
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
# Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "%Z%%M% %I%     %E% SMI"
#


#
# This file contains the basic cluster contents shared by all of the
# Linux distros we support.  Each distro has its own .distro file that
# expands on the basic cluster lists provided here.
#


#
# Required packages for the install miniroot, these are the minimum packages a
# system must have installed in order to run rpm (which is then used from
# within the zone to perform the balance of the installation.)
#
common_miniroot_rpms="SysVinit \
        basesystem \
        bash \
        beecrypt \
        bzip2-libs \
        coreutils \
        elfutils \
        elfutils-libelf \
        filesystem \
        glibc \
        glibc-common \
        gpm \
        initscripts \
        iptables \
        iptables-ipv6 \
        kernel-utils \
        laus-libs \
        libacl \
        libattr \
        libgcc \
        libtermcap \
        ncurses \
        pam \
        popt \
```



```
        rpm \
        rpm-libs \
        setup \
        termcap \
        zlib"

#
# This starts a listing of RPMs comprising a variety of install package options
# for a distribution.
#
# The supported package clusters are:
#
#       + core
#       + server
#       + desktop
#       + developer
#       + system
#
# The RPMs needed to install each cluster are listed in the shell variable
#
#       distro_<level>_rpms
#
# This file provides "common_<level>_rpms", which are lists of the packages
# in each cluster that are common to all distros.
#
# The package names are listed alphabetically for readability.  rpm will
# reorder the list to ensure that each package's dependencies are installed
# before it is.
#
# Note: Since the distro_install script uses a regular expression to expand
# RPM package names to filenames, there may be some tweaking required to
# guarantee a unique match between a package name and a corresponding RPM
# file on the install media.
#
# One such example below is the package "XFree86-4."  The official name of
# the package is "XFree86," but the regular expression in the script
# matches that package name to the XFree86-100dpi-fonts and
# XFree86-75dpi-fonts package RPMs in addition to the proper XFree86 RPM.
# Therefore the "XFree86" package name was modified to be "XFree86-4",
# which does result in a unique package name to RPM file match.
#
common_core_rpms="GConf2 \
        Glide3 \
        ORBit \
        ORBit2 \
        XFree86-Mesa-libGL \
        XFree86-Mesa-libGLU \
        XFree86-libs \
        XFree86-libs-data \
        Xaw3d \
        ash \
        at \
        atk \
        audiofile \
        autofs \
        bc \
        binutils \
        bonobo-activation \
        bzip2 \
        chkconfig \
        compat-pwdb \
        cpio \
        cpp \
        cracklib \
        cracklib-dicts \
        crontabs \
```

```
128         cups-libs \
129         cyrus-sasl \
130         cyrus-sasl-md5 \
131         db4 \
132         desktop-file-utils \
133         dev \
134         diffutils \
135         diskdumputils \
136         e2fsprogs \
137         ed \
138         ethtool \
139         expat \
140         file \
141         findutils \
142         finger \
143         fontconfig \
144         freetype \
145         ftp \
146         gail \
147         gawk \
148         gdbm \
149         gdk-pixbuf \
150         gettext \
151         glib \
152         glib2 \
153         glibc-headers \
154         glibc-kernheaders \
155         gmp \
156         gnupg \
157         grep \
158         groff \
159         gtk+ \
160         gtk2 \
161         gzip \
162         hesiod \
163         hwdata \
164         indexhtml \
165         info \
166         iproute \
167         iputils \
168         kernel \
169         kernel-BOOT \
170         krb5-libs \
171         krb5-workstation \
172         kudzu \
173         laus \
174         less \
175         libaio \
176         libart_lgpl \
177         libbonobo \
178         libcap \
179         libgcj \
180         libgcj-ssa \
181         libglade2 \
182         libgnomecanvas \
183         libjpeg \
184         libmng \
185         libogg \
186         libpng \
187         libpng10 \
188         libstdc++ \
189         libtiff \
190         libtool-libs \
191         libungif \
192         libusb \
193         libuser \
```



```
194         libvorbis \
195         libwnck \
196         libxml \
197         libxml2 \
198         libxml2-python \
199         libxslt \
200         linc \
201         lockdev \
202         logrotate \
203         losetup \
204         lsof \
205         lvm \
206         lynx \
207         m4 \
208         mailcap \
209         make \
210         man \
211         man-pages \
212         mingetty \
213         mkinitrd \
214         mkisofs \
215         mktemp \
216         modutils \
217         mount \
218         mtools \
219         nc \
220         net-snmp \
221         net-snmp-libs \
222         net-tools \
223         netdump \
224         newt \
225         nfs-utils \
226         nscd \
227         nss_db \
228         nss_ldap \
229         ntp \
230         ntsysv \
231         openldap \
232         openssh \
233         openssh-clients \
234         openssh-server \
235         openssl \
236         pango \
237         passwd \
238         patch \
239         pax \
240         pcre \
241         pdksh \
242         perl \
243         perl-CGI \
244         perl-DateManip \
245         perl-Filter \
246         perl-HTML-Parser \
247         perl-HTML-Tagset \
248         perl-Parse-Yapp \
249         perl-URI \
250         perl-XML-Dumper \
251         perl-XML-Encoding \
252         perl-XML-Grove \
253         perl-XML-Parser \
254         perl-XML-Twig \
255         perl-libwww-perl \
256         perl-libxml-enno \
257         perl-libxml-perl \
258         portmap \
259         procmail \
```

```
260         procps \
261         psacct \
262         psmisc \
263         pspell \
264         pygtk2 \
265         pygtk2-libglade \
266         python \
267         pyxf86config \
268         readline \
269         redhat-logos \
270         redhat-menus \
271         rhpl \
272         rpm-python \
273         rpmdb-redhat \
274         rsh \
275         rsync \
276         rusers \
277         rwho \
278         sed \
279         setarch \
280         sgml-common \
281         shadow-utils \
282         slang \
283         startup-notification \
284         sudo \
285         sysklogd \
286         syslinux \
287         tar \
288         tcl \
289         tcp_wrappers \
290         tcsh \
291         telnet \
292         time \
293         traceroute \
294         ttmkfdir \
295         tzdata \
296         units \
297         unix2dos \
298         unzip \
299         usermode \
300         utempter \
301         util-linux \
302         vim-common \
303         vim-minimal \
304         vixie-cron \
305         wget \
306         which \
307         words \
308         xinetd \
309         xml-common \
310         yp-tools \
311         ypbind \
312         zip"

314 common_server_rpms="$common_core_rpms \
315         4Suite \
316         MyODBC \
317         MySQL-python \
318         Omni \
319         Omni-foomatic \
320         PyXML \
321         VFlib2 \
322         XFree86-4 \
323         XFree86-base-fonts \
324         XFree86-font-utils \
325         XFree86-truetype-fonts \
```



```
326         XFree86-xauth \
327         XFree86-xdm \
328         XFree86-xfs \
329         acl \
330         alchemist \
331         amanda \
332         amanda-server \
333         arts \
334         aspell \
335         aspell-config \
336         at-spi \
337         authd \
338         bcel \
339         bind \
340         bind-chroot \
341         bind-libs \
342         bind-utils \
343         bitmap-fonts \
344         caching-nameserver \
345         chkfontpath \
346         commons-beanutils \
347         commons-collections \
348         commons-digester \
349         commons-logging \
350         commons-modeler \
351         compat-db \
352         compat-libstdc++ \
353         crypto-utils \
354         cup-v10k \
355         cups \
356         curl \
357         cyrus-sasl-gssapi \
358         cyrus-sasl-plain \
359         dhcp \
360         distcache \
361         distcache-devel \
362         esound \
363         expect \
364         fam \
365         finger-server \
366         foomatic \
367         freeradius \
368         gd \
369         ghostscript \
370         ghostscript-fonts \
371         gimp-print \
372         gnome-libs \
373         gnome-mime-data \
374         gnome-python2 \
375         gnome-python2-bonobo \
376         gnome-python2-canvas \
377         gnome-python2-gtkhtml2 \
378         gnome-vfs2 \
379         gnuplot \
380         gtkhtml2 \
381         htmlview \
382         httpd \
383         hwcrypto \
384         imap \
385         imap-utils \
386         imlib \
387         inews \
388         inn \
389         jakarta-regexp \
390         krb5-server \
391         krbafs \
```

```
392         libIDL \
393         libbonoboui \
394         libdbi \
395         libdbi-dbd-mysql \
396         libgnome \
397         libgnomeprint22 \
398         libgnomeprintui22 \
399         libgnomeui \
400         libgsf \
401         libole2 \
402         logwatch \
403         mailman \
404         mailx \
405         mod_auth_mysql \
406         mod_auth_pgsql \
407         mod_authz_ldap \
408         mod_perl \
409         mod_python \
410         mod_ssl \
411         mpage \
412         mtr \
413         mx \
414         mx4j \
415         mysql \
416         mysql-bench \
417         mysql-devel \
418         net-snmp-utils \
419         netdump-server \
420         newt-perl \
421         openldap-servers \
422         openssl-perl \
423         pam_krb5 \
424         perl-DBD-MySQL \
425         perl-DBD-Pg \
426         perl-DBI \
427         perl-DB_File \
428         perl-Digest-HMAC \
429         perl-Digest-SHA1 \
430         perl-Net-DNS \
431         perl-Time-HiRes \
432         php \
433         php-imap \
434         php-ldap \
435         php-mysql \
436         php-odbc \
437         php-pgsql \
438         pnm2ppa \
439         postfix \
440         postgresql-odbc \
441         pxe \
442         pyorbit \
443         qt \
444         qt-MySQL \
445         qt-ODBC \
446         quagga \
447         radvd \
448         rdist \
449         redhat-config-bind \
450         redhat-config-httpd \
451         redhat-config-printer \
452         redhat-config-printer-gui \
453         redhat-config-samba \
454         redhat-config-securitylevel \
455         redhat-config-securitylevel-tui \
456         redhat-config-services \
457         redhat-java-rpm-scripts \
```



```
458         redhat-switch-mail \
459         redhat-switch-mail-gnome \
460         rh-postgresql \
461         rh-postgresql-contrib \
462         rh-postgresql-docs \
463         rh-postgresql-jdbc \
464         rh-postgresql-libs \
465         rh-postgresql-python \
466         rh-postgresql-server \
467         rh-postgresql-tcl \
468         rh-postgresql-test \
469         rhdb-utils \
470         rsh-server \
471         rusers-server \
472         samba \
473         samba-client \
474         samba-common \
475         samba-swat \
476         sendmail \
477         sendmail-cf \
478         slocate \
479         spamassassin \
480         squid \
481         squirrelmail \
482         switchdesk \
483         sysreport \
484         telnet-server \
485         tftp-server \
486         tmpwatch \
487         tux \
488         unixODBC \
489         unixODBC-kde \
490         urw-fonts \
491         usermode-gtk \
492         vsftpd \
493         webalizer \
494         xalan-j \
495         xerces-j \
496         xinitrc \
497         ypserv"

499 common_desktop_rpms="$common_server_rpms \
500         Canna-libs \
501         FreeWnn-libs \
502         Gtk-Perl \
503         ImageMagick \
504         ImageMagick-perl \
505         SDL \
506         XFree86-100dpi-fonts \
507         XFree86-75dpi-fonts \
508         XFree86-Xnest \
509         XFree86-Xvfb \
510         XFree86-doc \
511         XFree86-tools \
512         XFree86-twm \
513         a2ps \
514         am-utils \
515         amanda-client \
516         anacron \
517         apel-xemacs \
518         aumix \
519         authconfig \
520         authconfig-gtk \
521         autorun \
522         cdparanoia-alpha9.8 \
523         cdparanoia-libs-alpha9.8 \
```

```
524         cdrecord \
525         cipe \
526         ckermit \
527         comps-extras \
528         control-center \
529         ctags \
530         desktop-backgrounds-basic \
531         desktop-printing \
532         dialog \
533         docbook-dtds \
534         docbook-style-dsssl \
535         docbook-style-xsl \
536         docbook-utils \
537         docbook-utils-pdf \
538         dtach \
539         dvd+rw-tools \
540         dvdrecord \
541         eel2 \
542         elinks \
543         enscript \
544         eog \
545         evolution \
546         evolution-connector \
547         fetchmail \
548         file-roller \
549         firstboot \
550         fontilus \
551         gaim \
552         gconf-editor \
553         gdm \
554         gedit \
555         gftp \
556         ggv \
557         gimp \
558         gimp-data-extras \
559         gimp-perl \
560         gimp-print-cups \
561         gimp-print-plugin \
562         gimp-print-utils \
563         gnome-applets \
564         gnome-audio \
565         gnome-desktop \
566         gnome-games \
567         gnome-icon-theme \
568         gnome-media \
569         gnome-panel \
570         gnome-pilot \
571         gnome-python2-applet \
572         gnome-session \
573         gnome-spell \
574         gnome-system-monitor \
575         gnome-terminal \
576         gnome-themes \
577         gnome-user-docs \
578         gnome-utils \
579         gnome-vfs2-extras \
580         gnomemeeting \
581         gphoto2 \
582         gsl \
583         gstreamer \
584         gstreamer-plugins \
585         gstreamer-tools \
586         gtk-engines \
587         gtk2-engines \
588         gtkam \
589         gtkam-gimp \
```



```
590         gtkglarea \
591         gtkhtml3 \
592         guile \
593         hotplug \
594         hpijs \
595         hpoj \
596         htdig \
597         hwbrowser \
598         intltool \
599         itcl \
600         jadetex \
601         kdeaddons \
602         kdeartwork \
603         kdebase \
604         kdegames \
605         kdegraphics \
606         kdelibs \
607         kdemultimedia \
608         kdenetwork \
609         kdepim \
610         kdeutils \
611         lftp \
612         libao \
613         libf2c \
614         libgail-gnome \
615         libgal2 \
616         libghttp \
617         libglade \
618         libgtop2 \
619         libmrproject \
620         libpcap \
621         libraw1394 \
622         librsvg2 \
623         libsoup \
624         linuxdoc-tools \
625         lm_sensors \
626         magicdev \
627         metacity \
628         mikmod \
629         mrproject \
630         mrtg \
631         mutt \
632         nautilus \
633         nautilus-cd-burner \
634         nautilus-media \
635         netpbm \
636         netpbm-progs \
637         open \
638         openh323 \
639         openjade \
640         openldap-clients \
641         openmotif \
642         openmotif21 \
643         openoffice.org \
644         openoffice.org-i18n \
645         openoffice.org-libs \
646         openssh-askpass \
647         openssh-askpass-gnome \
648         parted \
649         passivetex \
650         perl-PDL \
651         perl-SGMLSpm \
652         perl-suidperl \
653         pilot-link \
654         printman \
655         psutils \
```

```
656         pwlib    \
657         pyOpenSSL \
658         python-optik \
659         redhat-artwork \
660         redhat-config-date \
661         redhat-config-keyboard \
662         redhat-config-kickstart \
663         redhat-config-language \
664         redhat-config-mouse \
665         redhat-config-network \
666         redhat-config-network-tui \
667         redhat-config-nfs \
668         redhat-config-packages \
669         redhat-config-proc \
670         redhat-config-rootpassword \
671         redhat-config-soundcard \
672         redhat-config-users \
673         redhat-config-xfree86 \
674         redhat-logviewer \
675         rhn-applet \
676         rhnlib \
677         sane-backends \
678         sane-frontends \
679         screen \
680         scrollkeeper \
681         shapecfg \
682         sharutils \
683         sox \
684         star \
685         switchdesk-gnome \
686         switchdesk-kde \
687         sysstat \
688         talk \
689         tclx \
690         tetex \
691         tetex-afm \
692         tetex-dvips \
693         tetex-fonts \
694         tetex-latex \
695         tetex-xdvi \
696         tix \
697         tk \
698         tkinter \
699         transfig \
700         ttfprint \
701         umb-scheme \
702         up2date \
703         up2date-gnome \
704         usbutils \
705         uucp \
706         vim-enhanced \
707         vlock \
708         vnc \
709         vnc-server \
710         vorbis-tools \
711         vte \
712         w3c-libwww \
713         xchat \
714         xdelta \
715         xemacs \
716         xemacs-el \
717         xemacs-info \
718         xfig \
719         xhtml1-dtds \
720         xloadimage \
721         xmltex \
```



```
722         xmlto \
723         xmms \
724         xpdf \
725         xsane \
726         xsane-gimp \
727         xscreensaver \
728         xsri \
729         xterm \
730         yelp \
731         zsh"

733 common_developer_rpms="$common_desktop_rpms \
734         ElectricFence \
735         GConf2-devel \
736         ORBit-devel \
737         ORBit2-devel \
738         SDL-devel \
739         XFree86-devel \
740         ant \
741         ant-libs \
742         arts-devel \
743         at-spi-devel \
744         atk-devel \
745         audiofile-devel \
746         autoconf \
747         autoconf213 \
748         automake \
749         automake14 \
750         automake15 \
751         bison \
752         blas \
753         bonobo-activation-devel \
754         bug-buddy \
755         byacc \
756         cdecl \
757         cproto \
758         crash \
759         cscope \
760         cups-devel \
761         cvs \
762         ddd \
763         dejagnu \
764         dev86 \
765         diffstat \
766         doxygen \
767         eel2-devel \
768         emacs \
769         emacs-el \
770         emacs-leim \
771         esound-devel \
772         flex \
773         fontconfig-devel \
774         freetype-devel \
775         gail-devel \
776         gcc \
777         gcc-c++ \
778         gcc-c++-ssa \
779         gcc-g77 \
780         gcc-g77-ssa \
781         gcc-gnat \
782         gcc-java \
783         gcc-java-ssa \
784         gcc-objc \
785         gcc-objc-ssa \
786         gcc-ssa \
787         gd-devel \
```

```
788         gdb \
789         gdk-pixbuf-devel \
790         gdk-pixbuf-gnome \
791         glade2 \
792         glib-devel \
793         glib2-devel \
794         glibc-devel \
795         glibc-profile \
796         glibc-utils \
797         gnome-desktop-devel \
798         gnome-libs-devel \
799         gnome-vfs2-devel \
800         gperf \
801         gtk+-devel \
802         gtk-doc \
803         gtk2-devel \
804         gtkhtml2-devel \
805         httpd-devel \
806         im-sdk \
807         imlib-devel \
808         indent \
809         jaf \
810         javamail \
811         joe \
812         jpackage-utils \
813         junit \
814         kdebase-devel \
815         kdegraphics-devel \
816         kdelibs-devel \
817         kdenetwork-devel \
818         kdepim-devel \
819         kdesdk \
820         kdesdk-devel \
821         kdeutils-devel \
822         kdevelop \
823         kdoc \
824         kernel-doc \
825         kernel-source \
826         lam \
827         lapack \
828         lha \
829         libIDL-devel \
830         libacl-devel \
831         libart_lgpl-devel \
832         libattr-devel \
833         libbonobo-devel \
834         libbonoboui-devel \
835         libgcc-ssa \
836         libgcj-devel \
837         libgcj-ssa-devel \
838         libglade2-devel \
839         libgnat \
840         libgnome-devel \
841         libgnomecanvas-devel \
842         libgnomeprint22-devel \
843         libgnomeprintui22-devel \
844         libgnomeui-devel \
845         libjpeg-devel \
846         libmng-devel \
847         libmudflap \
848         libmudflap-devel \
849         libobjc \
850         libole2-devel \
851         libpng-devel \
852         librsvg2-devel \
853         libstdc++-devel \
```



```
854         libstdc++-ssa \
855         libstdc++-ssa-devel \
856         libtiff-devel \
857         libtool \
858         libungif-devel \
859         libxml2-devel \
860         libxslt-devel \
861         linc-devel \
862         ltrace \
863         memprof \
864         nasm \
865         ncurses-devel \
866         nedit \
867         netpbm-devel \
868         openmotif-devel \
869         oprofile \
870         pango-devel \
871         patchutils \
872         pcre-devel \
873         perl-CPAN \
874         perl-Crypt-SSLeay \
875         pilot-link-devel \
876         pkgconfig \
877         pstack \
878         pygtk2-devel \
879         python-devel \
880         python-tools \
881         qt-designer \
882         qt-devel \
883         rcs \
884         redhat-rpm-config \
885         rpm-build \
886         ruby \
887         ruby-libs \
888         ruby-mode \
889         sane-backends-devel \
890         sip \
891         sip-devel \
892         splint \
893         startup-notification-devel \
894         strace \
895         swig \
896         texinfo \
897         tora \
898         vim-X11 \
899         vte-devel \
900         zlib-devel"

902 common_all_rpms="$common_developer_rpms \
903         Canna
904         FreeWnn \
905         ImageMagick-c++-5.5.6 \
906         Wnn6-SDK \
907         ami \
908         amtu \
909         anaconda \
910         anaconda-help \
911         anaconda-images \
912         anaconda-product \
913         anaconda-runtime \
914         apmd \
915         arptables_jf \
916         attr \
917         bg5ps \
918         bitmap-fonts-cjk \
919         bogl \
```

```
920         bogl-bterm \
921         bootparamd \
922         booty \
923         bridge-utils \
924         busybox \
925         busybox-anaconda \
926         compat-gcc \
927         compat-gcc-c++ \
928         compat-glibc-7.x \
929         compat-libstdc++-devel \
930         compat-slang \
931         db4-java \
932         db4-utils \
933         dbskkd-cdb \
934         desktop-backgrounds-extra \
935         devlabel \
936         dhclient \
937         dietlibc \
938         dos2unix \
939         dosfstools \
940         dump \
941         eject \
942         emacspeak \
943         ethereal \
944         ethereal-gnome \
945         fbset \
946         festival \
947         grub \
948         h2ps \
949         hdparm \
950         ipsec-tools \
951         irda-utils \
952         iscsi-initiator-utils \
953         isdn4k-utils \
954         jfsutils \
955         jisksp14 \
956         jisksp16 \
957         jwhois \
958         kappa20 \
959         kbd \
960         kernel-pcmcia-cs \
961         knm_new \
962         kon2 \
963         kon2-fonts \
964         libtabe \
965         libwvstreams \
966         lilo \
967         linuxwacom \
968         lslk \
969         mdadm \
970         mgetty \
971         minicom \
972         mkbootdisk \
973         mt-st \
974         mtx \
975         nano \
976         ncompress \
977         net-snmp-perl \
978         netconfig \
979         nhpf \
980         nmap \
981         octave \
982         openssl096b \
983         pam_passwdqc \
984         pam_smb \
985         pinfo \
```



```
986         ppp \
987         prelink \
988         psgml \
989         pvm \
990         quota \
991         rdate \
992         rdesktop \
993         redhat-config-netboot \
994         rhgb \
995         rmt \
996         rootfiles \
997         rp-pppoe \
998         schedutils \
999         setserial \
1000        setuptool \
1001        sg3_utils \
1002        skkdic \
1003        skkinput
1004        specspo \
1005        stunnel \
1006        tcpdump \
1007        tftp \
1008        tn5250 \
1009        tsclient \
1010        vconfig \
1011        wireless-tools \
1012        wvdial \
1013        x3270 \
1014        x3270-text \
1015        x3270-x11 \
1016        xcin"
1017 #endif /* ! codereview */
```

```
**********************************************************
   67442 Tue Jan 14 16:17:15 2014
new/usr/src/lib/brand/lx/zone/lx_distro_install.ksh
Bring back LX zones.
**********************************************************
```

```
#!/bin/ksh -p
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

#
# This script is called from /usr/lib/brand/lx/lx_install.
#
# options passed down from lx_install:
#       -z $ZONENAME
#       -r $LINUX_ROOT
#
# options passed down from zoneadm -z <zone-name> install
#       -d <Linux-archives-dir>
#       [core | server | desktop | development | all]
#
# The desktop cluster will be installed by default.
#

# Restrict executables to /bin, /usr/bin and /usr/sbin
PATH=/bin:/usr/bin:/usr/sbin
export PATH


# Setup i18n output
TEXTDOMAIN="SUNW_OST_OSCMD"
export TEXTDOMAIN

# Log passed arguments to file descriptor 2
log()
{
        [[ -n $logfile ]] && echo "$@" >&2
}

#
# Send the provided printf()-style arguments to the screen and to the
# logfile.
#
screenlog()
{
        typeset fmt="$1"
```



```
        shift

        printf "$fmt\n" "$@"
        [[ -n $logfile ]] && printf "$fmt\n" "$@" >&2
}

# Print and log provided text if the shell variable "verbose_mode" is set
verbose()
{
        [[ -n $verbose_mode ]] && echo "$@"
        [[ -n $logfile ]] && [[ -n $verbose_mode ]] && echo "$@" >&2
}

#
# Print to the screen if the shell variable "verbose_mode" is set, but always
# send the output to the log.
#
verboselog()
{
        [[ -n $verbose_mode ]] && echo "$@"
        [[ -n $logfile ]] && echo "$@" >&2
}

bad_rpmdir=$(gettext "'%s' is not a valid RPM directory!")

mb_req=$(gettext "(%s MB required, %s MB available)")
no_space=$(gettext "Not enough free space available in '%s'")

inst_clust=$(gettext "Installing cluster '%s'")
unknown_clust=$(gettext "ERROR: Unknown cluster name: '%s'")

unknown_media=$(gettext "Unknown or unreadable media loaded in %s")

eject_fail=$(gettext "Attempt to eject '%s' failed.")

lofi_failed=$(gettext "Attempt to add '%s' as lofi device FAILED.")
lofs_failed=$(gettext "Attempt to lofs mount '%s' on '%s' FAILED.")

media_spec=$(gettext "the provided media (%s)")

distro_mediafail=\
$(gettext "Attempt to determine Linux distribution from\n  %s FAILED.")

mini_bootfail=$(gettext "Attempt to boot miniroot for zone '%s' FAILED.")
mini_copyfail=$(gettext "Attempt to copy miniroot for zone '%s' FAILED.")
mini_initfail=$(gettext "Attempt to initialize miniroot for zone '%s' FAILED.")
mini_instfail=$(gettext "Attempt to install RPM '%s' to miniroot FAILED.")
mini_mediafail=$(gettext "Install of zone '%s' miniroot from\n  %s FAILED.")
mini_setfail=$(gettext "Attempt to setup miniroot for zone '%s' FAILED.")

mini_mntfsfail=\
$(gettext "Attempt to mount miniroot filesystems for zone '%s' FAILED.")

rpm_initfail=\
$(gettext "Attempt to initialize RPM database for zone '%s' FAILED.")

symlink_failed=$(gettext "Attempt to symbolically link '%s' to '%s' FAILED.")

discinfo_nofile=$(gettext "ERROR: Discinfo file '%s' not found!")
discinfo_notreadable=$(gettext "ERROR: Discinfo file '%s': not readable!")
discinfo_wrongarch=\
$(gettext "ERROR: '%s': disc architecture is '%s'; install requires 'i386'!")

wrong_serial=$(gettext "Incorrect serial number found on provided %s.")
wrong_ser_expect=$(gettext "  (found #%s, expected #%s)")
```

```
128 wrong_cd=$(gettext "Incorrect CD inserted (found %s, wanted %s)")

130 zone_initrootfail=\
131 $(gettext "Attempt to initialize root filesystem for zone '%s' FAILED.")

133 zone_haltfail=$(gettext "Unable to halt zone '%s'!")
134 zone_instfail=$(gettext "Install of zone '%s' from '%s' FAILED '%s'.")
135 zone_mediafail=$(gettext "Install of zone '%s' from\n  %s FAILED.")

137 zone_rootfail=\
138 $(gettext "ERROR: The specified zone root directory '%s' could not be created.")
139 zone_rootsub=\
140 $(gettext "ERROR: The specified zone root subdirectory '%s' does not exist.")

142 mk_mntfail=$(gettext "Could not create the mount directory '%s'")
143 mountfail=$(gettext "Mount of '%s' on '%s' FAILED.")

145 insert_discmsg=\
146 $(gettext "Please insert %s, or a\n  %s DVD in the removable media")

148 mount_proper_iso1=$(gettext "Please mount the ISO for %s or a")
149 mount_proper_iso2=$(gettext "%s DVD on device '%s'")

151 silent_nodisc=$(gettext "ERROR: Cannot install from CDs in silent mode.")
152 silent_nolofi=\
153 $(gettext "ERROR: Cannot install from lofi-based CD ISOs in silent mode.")

155 install_msg=$(gettext "Installing zone '%s' from\n  %s.")
156 install_ndiscs=\
157 $(gettext "You will need CDs 1 - %s (or the equivalent DVD) to")
158 install_nisos=\
159 $(gettext "You will need ISO images representing CDs 1 - %s (or the equivalent")

161 locate_npkgs=$(gettext "Attempting to locate %s packages...")

163 install_one_rpm=$(gettext "Installing 1 %spackage.")
164 install_nrpms_few=\
165 $(gettext "Installing %s %spackages; this may take a few minutes...")
166 install_nrpms_several=\
167 $(gettext "Installing %s %spackages; this may take several minutes...")

169 install_longwait=\
170 $(gettext "NOTE: There may be a long delay before you see further output.")

172 install_defmkfail=$(gettext "Could not create the temporary directory '%s'")
173 install_defcpfail=$(gettext "Could not make a local copy of deferred RPM '%s'")
174 install_dist=$(gettext "Installing distribution '%s'...")
175 install_zonefail=$(gettext "Attempt to install zone '%s' FAILED.")

177 no_distropath=$(gettext "ERROR: Distribution path '%s' doesn't exist.")

179 install_done=$(gettext "Installation of %s to zone\n  '%s' completed %s.")
180 install_failed=$(gettext "Installation of %s to zone\n  '%s' FAILED %s.")

182 eject_final_msg=\
183 $(gettext "Would you like the system to eject the %sinstall %s when")
184 eject_final_prompt=$(gettext "installation of '%s' is complete? (%s)")
185 eject_final_status=$(gettext "The %sinstall %s %s be ejected.")

187 #
188 # Get the device underlying a specified mounted file system and return it in
189 # the shell variable "mount_dev"
190 #
191 # Returns 0 on success, 1 on failure.
192 #
193 get_mountdev()
```



```
194 {
195         typeset mount_dir="$1"
196         typeset device
197         unset mount_dev

199         #
200         # Obtain information on the specified mounted device.
201         #
202         device=`{ df -k "$mount_dir" | egrep "^/" ; } 2>/dev/null` || return 1
203         mount_dev=$(echo $device | awk -e '{print $1}' 2>/dev/null)

205         [[ "`echo $mount_dev | cut -c 1`" = "/" ]] && return 0

207         unset mount_dev
208         return 1
209 }

211 #
212 # Get the directory name a specified device is mounted as and return it in
213 # the shell variable "mount_dir"
214 #
215 # Returns 0 on success, 1 on failre.
216 #
217 get_mountdir()
218 {
219         typeset mount_dev="$1"
220         typeset dir
221         unset mount_dir

223         [[ -b "$mount_dev" ]] || return 1

225         #
226         # Obtain information on the specified mounted device.
227         #
228         dir=`{ df -k "$mount_dev" | egrep "^/" ; } 2>/dev/null` || return 1
229         mount_dir=$(echo $dir | awk -e '{print $6}' 2>/dev/null)

231         [[ "`echo $mount_dir | cut -c 1`" = "/" ]] && return 0

233         unset mount_dir
234         return 1
235 }

237 #
238 # Check the free disk space of the passed filesystem against the passed
239 # argument.
240 #
241 # Returns 0 on success, 1 on failure.
242 #
243 check_mbfree()
244 {
245         typeset dir="$1"
246         typeset mb_required=$2

248         #
249         # Return free space in partition containing passed argument in MB
250         #
251         typeset mbfree=`{ LC_ALL=C df -k "$dir" | \
252             egrep -v Filesystem ; } 2>/dev/null` || return 1
253         mbfree=$(echo $mbfree | awk -e '{print $4}' 2>/dev/null)

255         ((mbfree /= 1024))
256         if ((mbfree < mb_required)); then
257                 screenlog "$no_space" "$zoneroot"
258                 screenlog "$mb_req" "$mb_required" "$mb_free"
259                 return 1
```

```
260         fi
261         return 0
262 }

264 #
265 # Find packages by attempting to expand passed RPM names to their full filenames
266 # in the passed RPM directory.
267 #
268 # Arguments:
269 #
270 #       Argument 1:  Path to mounted install media
271 #       Arguments [2 - n]:  RPM names to process
272 #
273 # The expanded filenames are returned in the shell array "rpm_names."
274 #
275 # For example:
276 #
277 #       find_packages /mnt/iso dev kernel tetex redhat-menus
278 #
279 # would return something like:
280 #
281 #       rpms_found[0]:  dev-3.3.12.3-1.centos.0.i386.rpm
282 #       rpms_found[1]:  kernel-2.4.21-32.EL.i586.rpm
283 #       rpms_found[2]:  tetex-1.0.7-67.7.i386.rpm
284 #       rpms_found[3]:  redhat-menus-0.39-1.noarch.rpm
285 #
286 # The routine returns 0 on success, 1 on an error.
287 #
288 find_packages()
289 {
290         typeset found=0
291         typeset left=0

293         typeset rpmdir="$1/$rd_rpmdir"
294         typeset curdir=${PWD:=$(pwd)}

296         typeset arch
297         typeset procinfo
298         typeset rpmglob
299         typeset rpmfile

301         unset rpms_found
302         unset rpms_left

304         shift
305         cd "$rpmdir"

307         typeset rpmcheck="$(echo *.rpm)"

309         if [[ "$rpmcheck" = "*.rpm" ]]; then
310                 screenlog "$bad_rpmdir" "$rpmdir"
311                 cd "$curdir"
312                 return 1
313         fi

315         #
316         # If the miniroot is booted, and the archs list isn't already set,
317         # ask the zone's rpm command for the list of compatible architectures.
318         #
319         if [[ -n $miniroot_booted && -z $archs ]]; then
320                 procinfo=$(zlogin "$zonename" /bin/rpm --showrc | \
321                     grep "^compatible archs")

323                 [[ $? -eq 0 ]] &&
324                     archs=$(echo $procinfo | sed 's/^compatible archs : //')
```



```
326                 [[ -n $archs ]] &&
327                     log "RPM-reported compatible architectures: $archs"
328         fi

330         #
331         # Either the miniroot isn't booted or asking rpm for the information
332         # failed for some reason, so make some reasonable assumptions.
333         #
334         if [[ -z $archs ]]; then
335                 procinfo=$(LC_ALL=C psrinfo -vp | grep family)

337                 #
338                 # Check for additional processor capabilities
339                 #
340                 if [[ "$procinfo" = *" family 6 "* ||
341                     "$procinfo" = *" family 15 "* ||
342                     "$procinfo" = *" family 16 "* ||
343                     "$procinfo" = *" family 17 "* ]]; then
344                         if [[ "$procinfo" = *AuthenticAMD* ]]; then
345                                 #
346                                 # Linux gives "athlon" packages precedence
347                                 # over "i686" packages, so duplicate that
348                                 # here.
349                                 #
350                                 archs="athlon i686"
351                         else
352                                 archs="i686"
353                         fi
354                 fi

356                 archs="$archs i586 i486 i386 noarch"

358                 log "Derived compatible architectures: $archs"
359         fi

361         verboselog "RPM source directory:\n  \"$rpmdir\"\n"

363         if [[ $# -eq 1 ]]; then
364                 msg=$(gettext "Attempting to locate 1 package...")
365                 screenlog "$msg"
366         else
367                 screenlog "$locate_npkgs" "$#"
368         fi

370         for rpm in "$@"; do
371                 #
372                 # Search for the appropriate RPM, using the compatible
373                 # architecture list contained in "archs" to look for the best
374                 # match.
375                 #
376                 # For example, if the processor is an i686, and the rpm is
377                 # "glibc", the script will look for the files (in order):
378                 #
379                 #    glibc[.-][0-9]*.i686.rpm
380                 #    glibc[.-][0-9]*.i586.rpm
381                 #    glibc[.-][0-9]*.i486.rpm
382                 #    glibc[.-][0-9]*.i386.rpm
383                 #    glibc[.-][0-9]*.noarch.rpm
384                 #    glibc[.-][0-9]*.fat.rpm
385                 #
386                 # and will stop when it finds the first match.
387                 #
388                 # TODO: Once the miniroot is booted, we should verify that
389                 #       the rpm name has been expanded to "$rpmfile" properly
390                 #       by comparing "$rpm" and the output of:
391                 #
```

```
392                 #       zlogin -z <zone> /bin/rpm --qf '%{NAME}' -qp $rpmfile
393                 #
394                 for arch in $archs; do
395                         #
396                         # Use the filename globbing functionality of ksh's
397                         # echo command to search for the file we want.
398                         #
399                         # If no matching file is found, echo will simply
400                         # return the passed string.
401                         #
402                         rpmglob="$rpm[.-][0-9]*.$arch.rpm"
403                         rpmfile="$(echo $rpmglob)"

405                         [[ "$rpmfile" != "$rpmglob" ]] && break

407                         unset rpmfile
408                 done

410                 if [[ -z $rpmfile ]]; then
411                         rpms_left[$left]="$rpm"
412                         ((left += 1))
413                 else
414                         rpms_found[$found]="$rpmfile"
415                         ((found += 1))
416                 fi
417         done

419         cd "$curdir"
420         log "\"$rpmdir\": matched $found of $# packages."
421         log "\"$rpmdir\": $left RPMs remaining."
422         return 0
423 }

425 #
426 # Build the rpm lists used to install a machine.
427 #
428 # The first argument is the number of discs in the distribution.  The
429 # second, optional, argument is the metacluster to install.
430 #
431 # The array "distro_rpm[]" is built from the individual package RPM arrays
432 # read in from an individual distribution definition file.
433 #
434 build_rpm_list()
435 {
436         # Default to a desktop installation
437         typeset cluster=desktop
438         typeset cnt=0
439         typeset pkgs

441         for clust in "$@"; do
442                 ((cnt += 1))
443                 case $clust in
444                         core)   cluster=core ;;
445                         desk*)  cluster=desktop ;;
446                         serv*)  cluster=server ;;
447                         dev*)   cluster=developer ;;
448                         all)    cluster=all
449                                 break;;
450                         *)      screenlog "$unknown_clust" "$clust"
451                                 exit $ZONE_SUBPROC_USAGE ;;
452                 esac
453         done

455         if [ $cnt -gt 1 ]; then
456                 msg=$(gettext "Too many install clusters specified")
457                 screenlog "$msg"
```



```
458                 exit $ZONE_SUBPROC_USAGE
459         fi

461         screenlog "$inst_clust" $cluster

463         case $cluster in
464                 core)           distro_rpms=$distro_core_rpms ;;
465                 desktop)        distro_rpms=$distro_desktop_rpms ;;
466                 server)         distro_rpms=$distro_server_rpms ;;
467                 developer)      distro_rpms=$distro_developer_rpms ;;
468                 all)            distro_rpms=$distro_all_rpms ;;
469         esac

471         # The RPMs in the miniroot must all be installed properly as well
472         distro_rpms="$distro_miniroot_rpms $distro_rpms"
473 }

475 #
476 # Install the "miniroot" minimal Linux environment that is booted single-user
477 # to complete the install.
478 #
479 # This works by doing feeding the RPM list needed for the installation one
480 # by one to rpm2cpio(1).
481 #
482 # Usage:
483 #    install_miniroot <mounted media dir> <names of RPMS to install>
484 #
485 #
486 install_miniroot()
487 {
488         typeset mediadir="$1"
489         typeset rpm

491         shift

493         #
494         # There's a quirk in our version of ksh that sometimes resets the
495         # trap handler for the shell.  Since RPM operations will be the
496         # longest part of any given install, make sure that an interrupt while
497         # the command is running will bring the miniroot down and clean up
498         # the interrupted install.
499         #
500         trap trap_cleanup INT

502         if [[ $# -eq 1 ]]; then
503                 msg=$(gettext "Installing %s miniroot package...")
504         else
505                 msg=$(gettext "Installing %s miniroot packages...")
506         fi

508         screenlog "\n$msg" "$#"

510         for rpm in "$@"; do
511                 verboselog "\nInstalling \"$rpm\" to miniroot at\n" \
512                         "  \"$zoneroot\"..."

514                 rpm2cpio "$mediadir/$rd_rpmdir/$rpm" | \
515                     ( cd "$rootdir" && cpio -idu ) 1>&2

517                 if [[ $? -ne 0 ]]; then
518                         screenlog "$mini_instfail" "$rpm"
519                         return 1
520                 fi
521         done

523         screenlog ""
```

```
524         return 0
525 }

527 #
528 # Install the zone from the mounted disc image by feeding a list of RPMs to
529 # install from this image to RPM running on the zone via zlogin(1).
530 #
531 # Usage:
532 #    install_zone <path to mounted install media> [<names of RPMS to install>]
533 #
534 # If the caller doesn't supply a list of RPMs to install, we install any
535 # we previously stashed away in the deferred RPMs directory.
536 #
537 install_zone()
538 {
539         #
540         # Convert the passed install media pathname to a zone-relative path
541         # by stripping $rootpath from the head of the path.
542         #
543         typeset zonerpmdir="${1##$rootdir}/$rd_rpmdir"

545         typeset defdir="$rootdir/var/lx_install/deferred_rpms"
546         typeset mounted_root="$1"
547         typeset rpmopts="-i"

549         typeset defer
550         typeset deferred_found
551         typeset install_rpms
552         typeset nrpms
553         typeset rpm
554         typeset rpmerr

556         shift

558         #
559         # If the caller provided a list of RPMs, determine which of them
560         # should be installed now, and which should be deferred until
561         # later.
562         #
563         if [[ $# -gt 0 ]]; then
564                 if [[ -n $deferred_rpms ]]; then
565                         [[ -d $defdir ]] || if ! mkdir -p $defdir; then
566                                 screenlog "$install_defmkfail" "$mntdir"
567                                 return 1
568                         fi

570                         msg=$(gettext "Checking for deferred packages...")
571                         screenlog "$msg"

573                         find_packages "$mounted_root" $deferred_rpms
574                         deferred_found="${rpms_found[@]}"
575                         numdeferred=${#rpms_found[@]}
576                 else
577                         deferred_found=""
578                 fi

580                 install_rpms="$@"
581                 nrpms=$#

583                 #
584                 # If this distro has any deferred RPMs, we want to simply
585                 # copy them into the zone instead of installing them.  We
586                 # then remove them from the list of RPMs to be installed on
587                 # this pass.
588                 #
589                 for rpm in $deferred_found; do
```



```
590                         if echo "$install_rpms" | egrep -s "$rpm"; then
591                                 verboselog "Deferring installation of \"$rpm\""

593                                 #
594                                 # Remove the RPM from the install_rpms list
595                                 # and append it to the deferred_saved array
596                                 #
597                                 install_rpms=$(echo "$install_rpms " |
598                                     sed "s/ $rpm / /g")

600                                 # remove trailing spaces, if any
601                                 install_rpms=${install_rpms%%+( )}

603                                 deferred_saved[${#deferred_saved[@]}]="$rpm"

605                                 if ! cp "$mounted_root/$rd_rpmdir/$rpm" \
606                                     "$defdir"; then
607                                         screenlog "$install_defcpfail" "$rpm"
608                                         return 1
609                                 fi
610                         fi

612                         #
613                         # If we've deferred the installation of EVERYTHING,
614                         # simply return success
615                         #
616                         [[ -z $install_rpms ]] && return 0
617                 done

619                 [[ -n $deferred_found ]] & verbose ""
620         elif [[ -z $deferred_saved ]]; then
621                 # There are no deferred RPMs to install, so we're done.
622                 return 0
623         else
624                 # Install the RPMs listed in the deferred_saved array
625                 install_rpms=${deferred_saved[@]}
626                 nrpms=${#deferred_saved[@]}
627                 zonerpmdir=/var/lx_install/deferred_rpms
628                 defer="deferred "
629         fi

631         #
632         # There's a quirk in our version of ksh that sometimes resets the
633         # trap handler for the shell.  Since RPM operations will be the
634         # longest part of any given install, make sure that an interrupt while
635         # the command is running will bring the miniroot down and clean up
636         # the interrupted install.
637         #
638         trap trap_cleanup INT

640         #
641         # Print a message depending on how many RPMS we have to install.
642         #
643         # 25 RPMS seems like a reasonable boundary between when an install may
644         # take a "few" or "several" minutes; this may be tuned if needed.
645         #
646         screenlog ""

648         if [[ $nrpms -eq 1 ]]; then
649                 screenlog "$install_one_rpm" "$defer"
650         elif [[ $nrpms -lt 25 ]]; then
651                 screenlog "$install_nrpms_few" "$nrpms" "$defer"
652         else
653                 screenlog "$install_nrpms_several" "$nrpms" "$defer"

655                 #
```

```
656                 # For installs of over 600 packages or so, it can take rpm a
657                 # really, REALLY long time to output anything, even when
658                 # running in verbose mode.
659                 #
660                 # For example, when doing an "all" install from a DVD or DVD
661                 # ISO, depending on the speed of the optical drive and the
662                 # speed of the machine's CPU(s), it may be up to TEN MINUTES or
663                 # MORE before rpm prints out its "Processing..." message even
664                 # though it is, in fact, processing the entire package list,
665                 # checking for dependencies (something it is unfortunately
666                 # entirely silent about.)
667                 #
668                 # Since the user might otherwise think the install was hung
669                 # when running in verbose mode, warn them that it could be
670                 # quite a while before they see any further output from the
671                 # installer.
672                 #
673                 #
674                 [[ $nrpms -gt 600 ]] && verbose "$install_longwait"
675         fi

677         log ""
678         log "Installing: $install_rpms"
679         log ""
680         log "NOTE:  Any messages appearing below prefixed with \"warning:\""
681         log "       and/or that do not cause the installer to abort the"
682         log "       installation process may safely be ignored."
683         log ""

685         echo

687         # If verbose mode is selected, run rpm in verbose mode as well.
688         [[ -n $verbose_mode ]] && rpmopts="-ivh"

690         #
691         # LX_INSTALL must be defined when running this command in order to
692         # enable switches built into various emulated system calls to allow
693         # the dev package (which may not actually write to /dev) to function.
694         #
695         zlogin "$zonename" "( cd "$zonerpmdir" ; LX_INSTALL=1 \
696             /bin/rpm $rpmopts --force --aid --nosignature --root /a \
697             $install_rpms )"

699         rpmerr=$?

701         if [[ $rpmerr -ne 0 ]]; then
702                 log ""
703                 log "Zone rpm install command exited abnormally, code $rpmerr"
704                 log ""

706                 screenlog "$zone_instfail" "$zonename" "$zonerpmdir" "$rpmerr"
707                 return 1
708         fi

710         log ""
711         log "$nrpms package(s) installed."

713         return 0
714 }

716 #
717 # Attempt to unmount all file systems passed on the command line
718 #
719 # Returns 0 if all umounts succeeded, otherwise the number of umount failures
720 #
721 umount_list()
```



```
722 {
723         typeset failures=0
724         typeset mounted

726         unset umount_failures

728         for mounted in "$@"; do
729                 if ! umount "$mounted"; then
730                         umount_failures="$umount_failures $mounted"
731                         ((failures += 1))
732                 fi
733         done

735         return $failures
736 }

738 #
739 #
740 # Set up lofi mounts required for chroot(1M) to work on a new root directory
741 # located in /a within a zone.
742 #
743 newroot_lofimnt()
744 {
745         typeset dev
746         typeset mounted
747         typeset target

749         unset newroot_mounted

751         #
752         # /usr and /lib get lofs mounted in the zone on /native read-only
753         #
754         # $zoneroot/dev gets lofs mounted on /native/dev read/write to allow
755         # the use of native devices.
756         #
757         mount -F lofs -r /lib "$rootdir/a/native/lib" || return 1
758         newroot_mounted="$rootdir/a/native/lib"

760         if ! mount -F lofs -r /usr "$rootdir/a/native/usr"; then
761                 umount "$rootdir/a/native/lib"
762                 unset newroot_mounted
763                 return 1
764         fi

766         newroot_mounted="$newroot_mounted $rootdir/a/native/usr"

768         if ! mount -F lofs "$zoneroot/root/native/dev" \
769             "$rootdir/a/native/dev"; then
770                 umount_list $newroot_mounted
771                 unset newroot_mounted
772                 return 1
773         fi

775         newroot_mounted="$newroot_mounted $rootdir/a/native/dev"

777         #
778         # This is a bit ugly; to provide device access within the chrooted
779         # environment RPM will use for its install, we will create the same
780         # symlinks "$rootdir/dev" contains in the new dev directory, and will
781         # lofs mount the balance of "$rootdir/dev" into the same locations in
782         # /dev in the new filesystem we're installing to.
783         #
784         for dev in "$zoneroot"/root/dev/*
785         do
786                 if [[ "$dev" = "$zoneroot/root/dev/*" ]]; then
787                         log "ERROR: No files found in $zoneroot/root/dev"
```

```
788                         umount_list $newroot_mounted
789                         return 1
790                 fi

792                 target="$rootdir/a/dev/$(basename $dev)"

794                 #
795                 # If the device file is a symbolic link, create a new link
796                 # in the target directory with the same source.
797                 #
798                 # If the device file is any other file or directory, lofs
799                 # mount it from the device directory into the target directory.
800                 #
801                 if [[ -h $dev ]]; then
802                         typeset source=$(LC_ALL=C file -h "$dev")

804                         #
805                         # Remove extraneous text from the output of file(1) so
806                         # we're left only with the target path of the symbolic
807                         # link.
808                         #
809                         source="${source##*link to }"

811                         [[ -a "$target" ]] && /bin/rm -f "$target"

813                         if ! ln -s "$source" "$target"; then
814                                 screenlog "$symlink_failed" "$source" "$target"
815                                 umount_list $newroot_mounted
816                                 unset newroot_mounted
817                                 return 1
818                         fi
819                 else
820                         [[ ! -a "$target" ]] && touch "$target"

822                         if ! mount -F lofs "$dev" "$target"; then
823                                 screenlog "$lofs_failed" "$dev" "$target"
824                                 umount_list $newroot_mounted
825                                 unset newroot_mounted
826                                 return 1
827                         fi

829                         newroot_mounted="$newroot_mounted $target"
830                 fi

832         done

834         return 0
835 }

837 #
838 # Replace the root directory of a zone with the duplicate previously created
839 # in the zone's /a directory.
840 #
841 replace_miniroot()
842 {
843         #
844         # The zoneadm halt will automatically unmount any file systems
845         # mounted via lofs in the zone, so that saves us from having to
846         # methodically unmount each one.
847         #
848         if ! zoneadm -z "$zonename" halt; then
849                 screenlog "$zone_haltfail" "$zonename"
850                 return 1
851         fi

853         unset miniroot_booted
```



```
854         unset newroot_mounted

856         [[ -d "$zoneroot/a" ]] && rm -rf "$zoneroot/a"
857         [[ -d "$zoneroot/oldroot" ]] && rm -rf "$zoneroot/oldroot"

859         #
860         # Copy the logfile or we'll lose all details of the install into the
861         # new root directory, so strip "$zoneroot" off the pathname of the
862         # current logfile and use it to generate the pathname of the log file
863         # in the new root directory.
864         #
865         [[ -n $logfile && -f "$logfile" ]] &&
866             cp "$logfile" "$rootdir/a${logfile##$rootdir}"

868         mv -f "$rootdir/a" "$zoneroot/a" || return 1
869         mv -f "$rootdir" "$zoneroot/oldroot" || return 1
870         mv -f "$zoneroot/a" "$rootdir" || return 1

872         #
873         # After the directory munging above, we've moved the new copy of the
874         # logfile atop the logfile we WERE writing to, so if we don't reopen
875         # the logfile here the shell will continue writing to the old logfile's
876         # inode, meaning we would lose all log information from this point on.
877         #
878         [[ -n $logfile ]] && exec 2>>"$logfile"

880         rm -rf "$zoneroot/oldroot"

882         #
883         # Remove the contents of the /dev directory created by the install.
884         #
885         # We don't technically need to do this, but the zone infrastructure
886         # will mount $zoneroot/dev atop $rootdir/dev anyway, hiding its
887         # contents so we may as well clean up after ourselves.
888         #
889         # The extra checks are some basic paranoia due to the potentially
890         # dangerous nature of this command but are not intended to catch all
891         # malicious cases
892         #
893         [[ "$rootdir" != "" && "$rootdir" != "/" ]] && rm -rf "$rootdir"/dev/*

895         return 0
896 }

898 setup_miniroot()
899 {
900         unset miniroot_booted

902         if ! "$cwd/lx_init_zone" "$rootdir" mini; then
903                 screenlog "$mini_initfail" "$zonename"
904                 return 1
905         fi

907         if ! copy_miniroot; then
908                 screenlog "$mini_copyfail" "$zonename"
909                 return 1
910         fi

912         #
913         # zoneadm gets upset if the zone root directory is group or world
914         # readable or executable, so make sure it isn't before proceeding.
915         #
916         chmod 0700 "$zoneroot"

918         msg=$(gettext "Booting zone miniroot...")
919         screenlog "$msg"
```

```
921         if ! zoneadm -z "$zonename" boot -f; then
922                 screenlog "$mini_bootfail" "$zonename"
923                 return 1
924         fi

926         miniroot_booted=1

928         #
929         # Now that the miniroot is booted, unset the compatible architecture
930         # list that find_packages was using for the miniroot so that it will
931         # get the list from rpm for the full install.
932         #
933         unset archs

935         #
936         # Mount all the filesystems needed to install the new root
937         # directory.
938         #
939         if ! newroot_lofimnt; then
940                 screenlog "$mini_mntfsfail" "$zonename"

942                 if [[ -n $newroot_mounted ]]; then
943                         umount_list $newroot_mounted
944                         unset newroot_mounted
945                 fi
946                 return 1
947         fi

949         #
950         # Attempt to initialize the RPM database for the new zone
951         #
952         if ! zlogin "$zonename" /bin/rpm --initdb --root /a; then
953                 screenlog "$rpm_initfail" "$zonename"
954                 return 1
955         fi

957         msg=$(gettext "Miniroot zone setup complete.")
958         screenlog "$msg"
959         return 0
960 }

962 finish_install()
963 {
964         #
965         # Perform some last cleanup tasks on the newly installed zone.
966         #
967         # Note that the zlogin commands aren't checked for errors, as the
968         # newly installed zone will still boot even if the commands fail.
969         #
970         typeset file

972         typeset defdir=$rootdir/var/lx_install/deferred_rpms

974         msg=$(gettext "Completing installation; this may take a few minutes.")
975         screenlog "$msg"

977         if [[ -d $defdir ]]; then
978                 rm -f $defdir/*.rpm
979                 rmdir $defdir
980         fi

982         # Run ldconfig in the new root
983         zlogin "$zonename" /usr/sbin/chroot /a \
984             /sbin/ldconfig -f /etc/ld.so.conf
```



```
986         #
987         # Create the /etc/shadow and /etc/gshadow files if they don't already
988         # exist
989         #
990         [[ -a "$rootdir/a/etc/shadow" ]] ||
991             zlogin "$zonename" /usr/sbin/chroot /a /usr/sbin/pwconv

993         [[ -a "$rootdir/a/etc/gshadow" ]] ||
994             zlogin "$zonename" /usr/sbin/chroot /a /usr/sbin/grpconv

996         #
997         # Make sure all init.d and rc[0-6].d links are set up properly.
998         #
999         for file in `ls "$rootdir/a/etc/init.d"`; do
1000                zlogin "$zonename" /usr/sbin/chroot /a \
1001                    /sbin/chkconfig --del $file > /dev/null 2>&1

1003                zlogin "$zonename" /usr/sbin/chroot /a \
1004                    /sbin/chkconfig --add $file > /dev/null 2>&1
1005        done

1007        replace_miniroot

1009        rmdir -ps "$media_mntdir"

1011        if ! "$cwd/lx_init_zone" "$rootdir"; then
1012                screenlog "$zone_initrootfail" "$zonename"
1013                return 1
1014        fi

1016        return 0
1017 }

1019 #
1020 # Duplicate the installed "miniroot" image in a subdirectory of the base
1021 # directory of the zone.
1022 #
1023 # This is done so that a new root directory can be created that will be used
1024 # as the root of a chrooted directory that RPM running on the zone will install
1025 # into.
1026 #
1027 copy_miniroot()
1028 {
1029        #
1030        # Create the directory $zoneroot/a if it doesn't already exist
1031        #
1032        [[ -d "$zoneroot/a" ]] ||
1033                { mkdir -p "$zoneroot/a" || return 1 ; }

1035        msg=$(gettext "Duplicating miniroot; this may take a few minutes...")
1036        screenlog "$msg"

1038        #
1039        # Duplicate the miniroot to /a, but don't copy over any /etc/rc.d or
1040        # lxsave_ files.
1041        #
1042        ( cd "$rootdir"; find . -print | egrep -v "/etc/rc\.d|lxsave_" | \
1043            cpio -pdm ../a )

1045        [[ -d "$rootdir/a" ]] && rm -rf "$rootdir/a" 2>/dev/null
1046        mv -f "$zoneroot/a" "$rootdir/a" || return 1

1048        return 0
1049 }

1051 #
```

```
# Read the first six lines of the  .discinfo file from the root of the passed
# disc directory (which should either be a mounted disc or ISO file.)
#
# The read lines will be used to set appropriate shell variables on success:
#
#     rd_line[0]:  Disc Set Serial Number (sets rd_serial)
#     rd_line[1]:  Distribution Release Name (sets rd_release)
#     rd_line[2]:  Distribution Architecture (sets rd_arch)
#     rd_line[3]:  Disc Number$[s] in Distribution (sets rd_cdnum)
#     rd_line[4]:  "base" directory for disc (currently unused)
#     rd_line[5]:  RPM directory for disc (sets rd_rpmdir)
#
# Returns 0 on success, 1 on failure.
#
read_discinfo()
{
        typeset rd_file="$1/.discinfo"

        unset rd_arch
        unset rd_cdnum
        unset rd_disctype
        unset rd_pers
        unset rd_release
        unset rd_rpmdir
        unset rd_serial

        #
        # If more than one argument was passed to read_discinfo, the second
        # is a flag meaning that we should NOT print a warning message if
        # we don't find a .discinfo file, as this is just a test to see if
        # a distribution ISO is already mounted on the passed mount point.
        #
        if [[ ! -f "$rd_file" ]]; then
                [[ $# -eq 1 ]] &&
                    screenlog "$discinfo_nofile" "$rd_file"
                return 1
        fi

        verbose "Attempting to read \"$rd_file\"..."

        if [[ ! -r "$rd_file" ]]; then
                screenlog "$discinfo_notreadable" "$rd_file"
                return 1
        fi

        typeset rd_line
        typeset linenum=0

        while read -r rd_line[$linenum]; do
                #
                # If .discinfo architecture isn't "i386," fail here as
                # we only support i386 distros at this time.
                #
                if [[ $linenum = 2 && "${rd_line[2]}" != "i386" ]]; then
                        screenlog "$discinfo_wrongarch" "$rd_file" \
                            "${rd_line[2]}"
                        return 1
                fi

                #
                # We've successfully read the first six lines of .discinfo
                # into $rd_line, so do the appropriate shell variable munging.
                #
                if ((linenum == 5)); then
                        rd_serial=${rd_line[0]}
                        rd_release=${rd_line[1]}
```



```
                        # CentOS names their releases "final"
                        [[ "$rd_release" = "final" ]] && rd_release="CentOS"

                        #
                        # Line four of the .discinfo file contains either a
                        # single disc number for a CD or a comma delimited list
                        # representing the CDs contained on a particular DVD.
                        #
                        rd_cdnum=${rd_line[3]}

                        if [[ "$rd_cdnum" = *,* ]]; then
                                rd_disctype="DVD"
                        else
                                rd_disctype="CD"
                        fi

                        rd_rpmdir=${rd_line[5]}

                        #
                        # If the specified RPM directory doesn't exist, this is
                        # not a valid binary RPM disc (it's most likely a
                        # source RPM disc), so don't add it to the list of
                        # valid ISO files.
                        #
                        [[ ! -d "$1/$rd_rpmdir" ]] && return 1

                        if [[ "$rd_cdnum" = "1" &&
                           "$rd_release" = "Red Hat"* ]]; then
                                typeset rh_glob

                                #
                                # If this is a Red Hat release, get its
                                # personality name from the name of the
                                # redhat-release RPM package.
                                #
                                # Start by looking for the file
                                # "redhat-release-*.rpm" in the directory
                                # RedHat/RPMS of the ISO we're examining by
                                # using ksh's "echo" command to handle
                                # filename globbing.
                                #
                                # If no matching file is found, echo will
                                # simply return the passed string.
                                #
                                rh_glob="$1/RedHat/RPMS/redhat-release-*.rpm"
                                rd_pers="$(echo $rh_glob)"

                                if [[ "$rd_pers" != "$rh_glob" ]]; then
                                        #
                                        # An appropriate file was found, so
                                        # extract the personality type from the
                                        # filename.
                                        #
                                        # For example, the presence of the file:
                                        #
                                        #   redhat-release-3WS-13.5.1.i386.rpm
                                        #
                                        # would indicate the ISO either
                                        # represents a "WS" personality CD or
                                        # a "WS" installation DVD.
                                        #
                                        # Start the extraction by deleting the
                                        # pathname up to the personality type.
                                        #
                                        rh_glob="*/redhat-release-[0-9]"
```

```
1184                                    rd_pers="${rd_pers##$rh_glob}"
1185
1186                                    #
1187                                    # Now remove the trailing portion of the
1188                                    # pathname to leave only the personality
1189                                    # type, such as "WS" or "ES."
1190                                    #
1191                                    rd_pers="${rd_pers%%-*\.rpm}"
1192                            else
1193                                    unset rd_pers
1194                            fi
1195                    fi
1196
1197                    return 0
1198            fi
1199
1200            ((linenum += 1))
1201    done < "$rd_file"
1202
1203    #
1204    # The file didn't have at least six lines, so indicate that parsing
1205    # failed.
1206    #
1207    return 1
1208 }
1209
1210 #
1211 # Mount install media within the zone.
1212 #
1213 # The media will be mounted at $zoneroot/root/media, either via a loopback
1214 # mount (if it's a managed removable disc) or directly (if the media is an ISO
1215 # file or if the specified filename is a block device.)
1216 #
1217 # Returns 0 on success, 1 on failure, 2 if no disc was available
1218 #
1219 mount_install_media()
1220 {
1221    typeset device="$1"
1222    typeset mount_err
1223
1224    unset removable
1225    unset zone_mounted
1226
1227    [[ -z $mntdir ]] && return 1
1228
1229    [[ -d $mntdir ]] || if ! mkdir -p $mntdir; then
1230            screenlog "$mk_mntfail" "$mntdir"
1231            unset mntdir
1232            return 1
1233    fi
1234
1235    if [[ "$install_media" = "disc" && "$managed_removable" = "1" ]]; then
1236            #
1237            # The removable disc device is an automatically managed one,
1238            # so just wait for the device mounter to notice a disc has been
1239            # inserted into the drive and for the disc to appear at the
1240            # mount point.
1241            #
1242            typeset mount_interval=2
1243            typeset mount_timeout=10
1244            typeset mount_timer=0
1245
1246            typeset nickname=$(basename $device)
1247
1248            eject -q "$nickname" > /dev/null 2>&1 || return 2
1249            removable="$nickname"
```



```
1250
1251            #
1252            # Double check that the device was mounted.  If it wasn't, that
1253            # usually means the disc in the drive isn't in a format we can
1254            # read or the physical disc is unreadable in some way.
1255            #
1256            # The mount_timer loop is needed because the "eject -q" above
1257            # may report a disc is available before the mounter associated
1258            # with the drive actually gets around to mounting the device,
1259            # so we need to give it a chance to do so.  The mount_interval
1260            # allows us to short-circuit the timer loop as soon as the
1261            # device is mounted.
1262            #
1263            while ((mount_timer < mount_timeout)); do
1264                    [[ -d "$device" ]] && break
1265
1266                    sleep $mount_interval
1267                    ((mount_timer += mount_interval))
1268            done
1269
1270            if [[ ! -d "$device" ]]; then
1271                    screenlog "\n$unknown_media" "$device"
1272                    return 2
1273            fi
1274
1275            mount -F lofs -r "$device" "$mntdir"
1276            mount_err=$?
1277    else
1278            #
1279            # Attempt to mount the media manually.
1280            #
1281            # First, make sure the passed device name really IS a device.
1282            #
1283            [[ -b "$device" ]] || return 2
1284
1285            #
1286            # Now check to see if the device is already mounted and lofi
1287            # mount the existing mount point into the zone if it is.
1288            #
1289            if get_mountdir "$device"; then
1290                    mount -F lofs -r "$mount_dir" "$mntdir"
1291                    mount_err=$?
1292            else
1293                    [[ "$install_media" = "disc" ]] && removable="$device"
1294
1295                    # It wasn't mounted, so go ahead and try to do so.
1296                    mount -F hsfs -r "$device" "$mntdir"
1297                    mount_err=$?
1298            fi
1299
1300            # A mount_err of 33 means no suitable media was found
1301            ((mount_err == 33)) && return 2
1302    fi
1303
1304    if ((mount_err != 0)); then
1305            screenlog "$mountfail" "$device" "$mntdir"
1306            unset mntdir
1307            return 1
1308    fi
1309
1310    zone_mounted="$mntdir"
1311    verbose "Mount of \"$device\" on \"$mntdir\" succeeded."
1312    return 0
1313 }
1314
1315 # Eject the disc mounted on the passed directory name
```

```
1316 eject_removable_disc()
1317 {
1318         screenlog ""
1319         verbose "  (Attempting to eject 'removable'... \c"

1321         if [[ -n $zone_mounted ]]; then
1322                 umount "$zone_mounted"
1323                 unset zone_mounted
1324         fi

1326         if ! eject "$removable"; then
1327                 verbose "failed.)\n"
1328                 screenlog "$eject_fail" "$removable"

1330                 msg=$(gettext "Please eject the disc manually.")
1331                 screenlog "$msg"
1332         else
1333                 verbose "done.)\n"
1334         fi

1336         unset removable
1337 }

1339 #
1340 # Ask for the user to provide a disc or ISO.
1341 #
1342 # Returns 0 on success, 1 on failure.
1343 #
1344 prompt_for_media()
1345 {
1346         # No prompting is allowed in silent mode.
1347         if [[ -n $silent_mode ]]; then
1348                 log "$silent_err_msg"
1349                 return 1
1350         fi

1352         if [[ "$1" != "" ]]; then
1353                 msg="$release_name, CD $1"
1354         else
1355                 typeset disc=$(gettext "disc")

1357                 msg=$(gettext "any")
1358                 msg="$msg $release_name $disc"
1359         fi

1361         if [[ "$install_media" = "disc" ]]; then
1362                 screenlog "$insert_discmsg" "$msg" "$release_name"

1364                 msg=$(gettext "drive and press <RETURN>.")
1365                 screenlog "  $msg"

1367                 [[ -n $removable ]] && eject_removable_disc
1368         else
1369                 if [[ -n $zone_mounted ]]; then
1370                         umount "$mntdir"
1371                         unset zone_mounted
1372                 fi

1374                 #
1375                 # This is only be printed in the case of a user
1376                 # specifying a device name as an install medium.
1377                 # This is handy for testing the installer or if the user
1378                 # has ISOs stored in some strange way that somehow
1379                 # breaks the "install from ISO" mechanism, as ISOs
1380                 # can be manually added using lofiadm(1M) command and
1381                 # the resulting lofi device name passed to the
```



```
1382                 # installer.
1383                 #
1384                 screenlog "$mount_proper_iso1" "$msg"
1385                 screenlog "  $mount_proper_iso2" "$release_name" "$mntdev"

1387                 msg=$(gettext "and press <RETURN>.")
1388                 screenlog "  $msg"
1389         fi

1391         read && return 0
1392
1393         return 1
1394 }

1396 #
1397 # Get a particular CD of a multi-disc set.
1398 #
1399 # This basically works by doing the following:
1400 #
1401 #     1) Mount the disc
1402 #     2) Read the disc's .discinfo file to see which CD it is or represents
1403 #     3) If it doesn't contain the desired CD, ask the user for a disc
1404 #        containing the CD we wanted.
1405 #
1406 # Returns 0 on success, 1 on failure.
1407 #
1408 get_cd()
1409 {
1410         typeset mntdev="$1"

1412         typeset cdnum
1413         typeset discname
1414         typeset enter
1415         typeset mount_err
1416         typeset prompted


1419         if [[ $# -eq 2 ]]; then
1420                 # Caller specified a particular CD to look for
1421                 cdnum="$2"
1422                 discname="$release_name, CD $cdnum"
1423         else
1424                 # Caller wanted any disc
1425                 discname="a $release_name disc"
1426         fi

1428         verboselog "\nChecking for $discname on device"
1429         verboselog "  \"$mntdev\"\n"

1431         while :; do
1432                 # Check to see if a distro disc is already mounted
1433                 mntdir="$media_mntdir"

1435                 unset rd_disctype
1436                 if ! read_discinfo "$mntdir" "test"; then
1437                         mount_install_media "$mntdev"
1438                         mount_err=$?

1440                         #
1441                         # If the mount succeeded, continue on in the main
1442                         # script
1443                         #
1444                         if ((mount_err == 0)); then
1445                                 read_discinfo "$mntdir"
1446                         elif ((mount_err == 2)); then
1447                                 # No medium was found, so prompt for one.
```

```
1448                                    prompt_for_media "$cdnum" && prompted=1 continue

1450                                    unset mntdir
1451                                    return 1
1452                            else
1453                                    # mount failed
1454                                    unset mntdir
1455                                    return 1
1456                            fi
1457                    fi

1459                    if [[ -n $distro_serial &&
1460                        "$rd_serial" != "$distro_serial" ]]; then
1461                            screenlog "$wrong_serial" "$install_disctype"
1462                            screenlog "  $wrong_ser_expect" "$rd_serial" \
1463                                "$distro_serial"

1465                            #
1466                            # If we're installing from ISOs, don't prompt the user
1467                            # if the wrong serial number is present, as there's
1468                            # nothing they can do about it.
1469                            #
1470                            [[ "$install_media" = "ISO" ]] && return 1

1472                            prompt_for_media "$cdnum" && continue

1474                            umount "$mntdir"
1475                            unset zone_mountdir
1476                            return 1
1477                    fi

1479                    #
1480                    # Make sure that the mounted media is CD $cdnum.
1481                    #
1482                    # If it is, return to the caller, otherwise eject the
1483                    # disc and try again.
1484                    #
1485                    if [[ "$rd_disctype" = "CD" ]]; then
1486                            verboselog "Found CD #$rd_cdnum," \
1487                                "Serial #$rd_serial"
1488                            verboselog "Release Name \"$rd_release\""

1490                            [[ -n $rd_pers ]] &&
1491                                verboselog "Detected RedHat Personality" \
1492                                    "\"$rd_pers\""

1494                            verboselog ""

1496                            # If we didn't care which CD it was, return success
1497                            [[ "$cdnum" = "" ]] && return 0

1499                            # Return if the CD number read is a match
1500                            [[ "$rd_cdnum" = "$cdnum" ]] && return 0
1501                    else
1502                            verboselog "\nFound DVD (representing CDs" \
1503                                "$rd_cdnum), Serial #$rd_serial"
1504                            verboselog "Release Name \"$rd_release\"\n"

1506                            [[ -n $rd_pers ]] &&
1507                                verboselog "Detected RedHat Personality" \
1508                                    "\"$rd_pers\""

1510                            verboselog ""

1512                            # If we didn't care which CD it was, return success
1513                            [[ "$cdnum" = "" ]] && return 0
```



```
1515                            #
1516                            # Since a DVD represents multiple CDs, make sure the
1517                            # DVD inserted represents the CD we want.
1518                            #
1519                            { echo "$rd_cdnum," | egrep -s "$cdnum," ; } &&
1520                                return 0
1521                    fi

1523                    if [[ -n $prompted ]]; then
1524                            if [[ "$rd_disctype" = "CD" ]]; then
1525                                    screenlog "$wrong_cd" "$rd_cdnum" "$cdnum"
1526                            else
1527                                    msg=$(gettext "Incorrect DVD inserted.")
1528                                    screenlog "$msg"

1530                                    log "(DVD represented CDs $rd_cdnum," \
1531                                        " wanted CD $cdnum)"
1532                            fi
1533                    fi

1535                    #
1536                    # If we're installing from ISOs, don't prompt the user if the
1537                    # wrong CD is mounted, as there's nothing they can do about it.
1538                    #
1539                    [[ "$install_media" = "ISO" ]] && return 1

1541                    prompt_for_media "$cdnum" && prompted=1 && continue

1543                    umount "$mntdir"
1544                    unset zone_mountdir
1545                    return 1
1546            done
1547    }

1549    #
1550    # Find out which distro the mounted disc belongs to by comparing the
1551    # mounted disc's serial number against those contained in the various
1552    # distro files.
1553    #
1554    # When a match is found, the shell variable "distro_file" will be set to
1555    # the name of the matching file.  Since that will have been the last file
1556    # sourced by the shell, there's no need for the caller to do it again; the
1557    # variable is only set in case it's of some use later.
1558    #
1559    # Returns 0 on success, 1 on failure.
1560    #
1561    get_disc_distro()
1562    {
1563            typeset distro
1564            typeset distro_files="$(echo $distro_dir/*.distro)"

1566            unset distro_file
1567
1568            [[ "$distro_files" = "$distro_dir/*.distro" ]] && return 1

1570            for distro in $distro_files; do
1571                    [[ ! -f "$distro" ]] && continue
1572
1573                    verbose "Checking for disc distro \"$distro\"..."

1575                    . "$distro" > /dev/null

1577                    [[ "$rd_serial" != "$distro_serial" ]] && continue

1579                    distro_file="$distro"
```

```
1580                 release_name="$rd_release $distro_version"
1581                 distro_ncds=${#distro_cdorder[@]}

1583                 return 0
1584         done

1586         return 1
1587 }

1589 #
1590 # Iterate through the install media to install the miniroot and full zone
1591 #
1592 # The install media may be physical discs, a lofi mounted ISO file, or
1593 # iso files located in a directory specified by the user.
1594 #
1595 # All installations, regardless of media type, use a CD as their basic media
1596 # unit.  DVDs or ISOs representing DVDs actually contain multiple "CDs" of
1597 # installation packages.
1598 #
1599 # The variable "distro_ncds," as set elsewhere, represents the number
1600 # of CDs required to install the distribution.  Whether the installation
1601 # actually requires multiple physical discs or ISOs depends upon their content.
1602 #
1603 # Returns 0 on success, 1 on failure.
1604 #
1605 iterate_media()
1606 {
1607         typeset cdnum=1
1608         typeset cds
1609         typeset disc_rpms
1610         typeset err_media
1611         typeset err_msg
1612         typeset install_type="$1"
1613         typeset ldevs
1614         typeset mountdev
1615         typeset rh_pers

1617         shift

1619         if [[ "$install_type" = "miniroot" ]]; then
1620                 typeset i

1622                 disc_rpms=$distro_miniroot_rpms
1623                 err_msg="$mini_mediafail"

1625                 # For miniroot installs, ask for CDs in numerical order
1626                 cds[0]="zero_pad"

1628                 for i in ${distro_cdorder[@]}; do
1629                         cds[$cdnum]=$cdnum
1630                         ((cdnum += 1))
1631                 done

1633                 cdnum=1
1634         else
1635                 disc_rpms=$distro_rpms
1636                 err_msg="$zone_mediafail"

1638                 #
1639                 # For full zone installs, ask for CDs in the order RPM needs
1640                 # to find the packages.
1641                 #
1642                 set -A cds "zero_pad" ${distro_cdorder[@]}
1643         fi

1645         if [[ "$install_media" = "ISO" ]]; then
```



```
1646                 set -A ldevs "zero_pad" "$@"
1647         else
1648                 mountdev="$1"
1649                 err_media="$release_name, CD ${cds[$cdnum]} (or DVD)"
1650         fi

1652         unset rpms_left_save

1654         while ((cdnum <= distro_ncds)); do
1655                 [[ -z ${cds[$cdnum]} ]] && ((cdnum += 1)) && continue

1657                 if [[ "$install_media" = "ISO" ]]; then
1658                         typeset isonum="${cds[$cdnum]}"

1660                         #
1661                         # If this routine was called with a single ISO device
1662                         # name, it must be a DVD, so refer to that one lofi
1663                         # device (and associated ISO pathname)
1664                         #
1665                         [[ $# -eq 1 ]] && isonum=1

1667                         err_media="ISO \"${iso_pathnames[$isonum]}\""
1668                         mountdev="${ldevs[$isonum]}"
1669                 fi

1671                 #
1672                 # If the disc needed in the install order isn't the one in
1673                 # the drive, ask for the correct one.
1674                 #
1675                 if ! get_cd "$mountdev" "${cds[$cdnum]}"; then
1676                         screenlog "$err_msg" "$zonename" "$err_media"
1677                         return 1
1678                 fi

1680                 # set the RedHat personality type, if applicable
1681                 [[ -n $rd_pers && -z $rh_pers ]] && rh_pers=$rd_pers

1683                 #
1684                 # We now know the actual type of media being used, so
1685                 # modify the "err_media" string accordingly.
1686                 #
1687                 if [[ "$install_media" = "disc" ]]; then
1688                         if [[ "$rd_disctype" = "DVD" ]]; then
1689                                 err_media="$release_name DVD"
1690                         else
1691                                 err_media="$release_name, CD ${cds[$cdnum]}"
1692                         fi
1693                 fi

1695                 find_packages "$mntdir" $disc_rpms

1697                 #
1698                 # Save a copy of $rpms_left.  Other functions clobber it.
1699                 #
1700                 rpms_left_save="${rpms_left[@]}"

1702                 if [[ -n $rpms_found ]]; then
1703                         if [[ "$install_type" = "miniroot" ]]; then
1704                                 verboselog "\nInstalling miniroot from"
1705                                 verboselog "  $err_media...\n"

1707                                 if ! install_miniroot "$mntdir" \
1708                                     "${rpms_found[@]}"; then
1709                                         screenlog "$err_msg" "$zonename" \
1710                                             "$err_media"
1711                                         return 1
```

```
1712                                fi
1713                        else
1714                                screenlog "\n$install_msg\n" "$zonename" \
1715                                    "$err_media"

1717                                if ! install_zone "$mntdir" \
1718                                    ${rpms_found[@]}; then
1719                                        screenlog "$err_msg" "$zonename" \
1720                                            "$err_media"
1721                                        return 1
1722                                fi
1723                        fi

1725                        #
1726                        # Mark installation from this CD (or ISO representing
1727                        # this CD) as completed.
1728                        #
1729                        if [[ "$rd_disctype" = "CD" ]]; then
1730                                unset cds[$cdnum]
1731                        fi
1732                fi

1734                # A DVD install takes a single disc, so stop iterating
1735                [[ "$rd_disctype" = "DVD" ]] && break

1737                # If there are no RPMs left, we're done.
1738                [[ -z $rpms_left_save ]] && break

1740                disc_rpms="$rpms_left_save"
1741                ((cdnum += 1))

1743                if [[ "$install_media" != "ISO" ]]; then
1744                        #
1745                        # modify the err_media variable to reflect the next
1746                        # CD in the sequence
1747                        #
1748                        err_media="$release_name, CD ${cds[$cdnum]}"
1749                else
1750                        # Unmount the last used ISO if appropriate
1751                        if [[ -n $zone_mounted ]]; then
1752                                umount "$zone_mounted"
1753                                unset zone_mounted
1754                        fi
1755                fi
1756        done

1758        if [[ -n $zone_mounted ]]; then
1759                umount "$zone_mounted"
1760                unset zone_mounted
1761        fi

1763        if [[ -n $rpms_left_save ]]; then
1764                #
1765                # Uh oh - there were RPMS we couldn't locate.  This COULD
1766                # indicate a failed installation, but we need to check for
1767                # a RedHat personality "missing" list first.
1768                #
1769                if [[ -n $rh_pers && "$rh_pers" != "AS" ]]; then
1770                        typeset missing

1772                        if [[ $rh_pers = "WS" ]]; then
1773                                missing="$distro_WS_missing"
1774                        elif [[ $rh_pers = "ES" ]]; then
1775                                missing="$distro_ES_missing"
1776                        fi
```



```
1778                        #
1779                        # If any packages left in "rpm_left_save" appear in the
1780                        # list of packages expected to be missing from this
1781                        # personality, remove them from the "rpm_left_save"
1782                        # list.
1783                        #
1784                        if [[ -n $missing ]]; then
1785                                typeset pkg

1787                                for pkg in $missing
1788                                do
1789                                        rpm_left_save=$(echo "$rpm_left_save " |
1790                                            sed "s/$pkg //g")

1792                                        #
1793                                        # If all of the packages in
1794                                        # "rpm_left_save" appeared in this
1795                                        # personality's list of "expected
1796                                        # missing" packages, then the
1797                                        # installation completed successfully.
1798                                        #
1799                                        [[ -z ${rpm_left_save%%+( )} ]] &&
1800                                            return 0
1801                                done
1802                        fi
1803                fi

1805                log "\nERROR: Unable to locate some needed packages:\n" \
1806                    "  ${rpms_left_save%%+( )}\n"
1807                screenlog "$err_msg" "$zonename"
1808                return 1
1809        fi

1811        return 0
1812 }

1814 #
1815 # Install a zone from installation media
1816 #
1817 # Returns 0 on success, 1 on failure
1818 #
1819 install_from_media()
1820 {
1821        msg=$(gettext "Installing miniroot for zone '%s'.")
1822        screenlog "$msg" "$zonename"

1824        iterate_media "miniroot" $@ || return 1

1826        if ! setup_miniroot; then
1827                screenlog "$mini_setfail" "$zonename"
1828                return 1
1829        fi

1831        msg=$(gettext "Performing full install for zone '%s'.")

1833        screenlog "\n$msg" "$zonename"

1835        iterate_media "full" $@ || return 1

1837        #
1838        # Attempt to install deferred RPMS, if any
1839        #
1840        if [[ -n $deferred_rpms ]]; then
1841                if ! install_zone ""; then
1842                        return 1
1843                fi
```

```
1844         fi

1846         finish_install
1847         return $?
1848 }

1850 #
1851 # Add an entry to the valid distro list.
1852 #
1853 # The passed argument is the ISO type ("CD Set" or "DVD")
1854 #
1855 add_to_distro_list()
1856 {
1857         typeset name

1859         distro_file[${#distro_file[@]}]="$distro"

1861         name="$release_name"
1862         [[ -n $redhat_pers ]] && name="$name $redhat_pers"

1864         select_name[${#select_name[@]}]="$name ($1)"
1865         release[${#release[@]}]="$release_name"
1866         iso_set[${#iso_set[@]}]="${iso_names[@]}"
1867         verboselog "Distro \"$name\" ($1) found."
1868 }

1870 #
1871 # Find out which distros we have ISO files to support
1872 #
1873 # Do this by cycling through the distro directory and reading each distro
1874 # file in turn looking for:
1875 #
1876 #     1) The number of discs in a distribution
1877 #     2) The serial number of the distribution
1878 #     3) The name of the distribution
1879 #
1880 # Based on this, we can determine based on the ISO files available which
1881 # distributions, if any, we have a complete set of files to support.
1882 #
1883 # The function returns the supported isos in the array "iso_set."
1884 #
1885 validate_iso_distros()
1886 {
1887         typeset cd
1888         typeset disctype
1889         typeset index
1890         typeset iso
1891         typeset ncds
1892         typeset pers
1893         typeset pers_cd
1894         typeset pers_index
1895         typeset serial

1897         typeset distro_files="$(echo $distro_dir/*.distro)"
1898         typeset nisos=${#iso_filename[@]}

1900         unset distro_file
1901         unset iso_set
1902         unset release
1903         unset select_name

1905         if [[ "$distro_files" = "$distro_dir/*.distro" ]]; then
1906                 msg=$(gettext "Unable to find any distro files!")
1907                 screenlog "$msg"
1908                 return
1909         fi
```



```
1911         for distro in $distro_files; do
1912                 #
1913                 # We're done if we've already processed all available ISO files
1914                 # or if there were none in the first place.
1915                 #
1916                 ((${#iso_filename[@]} == 0)) && break

1918                 [[ ! -f $distro ]] && continue

1920                 . "$distro" > /dev/null
1921                 ncds=${#distro_cdorder[@]}

1923                 unset iso_names
1924                 unset pers
1925                 unset pers_cd

1927                 verbose "\nChecking ISOs against distro file \"$distro\"..."

1929                 index=0

1931                 while ((index < nisos)); do
1932                         #
1933                         # If the filename has been nulled out, it's already
1934                         # been found as part of a distro, so continue to the
1935                         # next one.
1936                         #
1937                         if [[ -z ${iso_filename[$index]} ]]; then
1938                                 ((index += 1))
1939                                 continue
1940                         fi

1942                         iso="${iso_filename[$index]}"
1943                         serial="${iso_serial[$index]}"
1944                         release_name="${iso_release[$index]}"
1945                         redhat_pers="${iso_pers[$index]}"

1947                         verbose "  ISO \"$iso\":"

1949                         #
1950                         # If the serial number doesn't match that for
1951                         # this distro, check other ISOs
1952                         #
1953                         if [[ "$serial" != "$distro_serial" ]]; then
1954                                 ((index += 1))
1955                                 continue
1956                         fi

1958                         verbose "    Serial #$serial"
1959                         verbose "    Release Name \"$release_name\""

1961                         [[ -n ${iso_pers[$index]} ]] &&
1962                             verbose "    RedHat Personality \"$redhat_pers\""

1964                         if [[ "${iso_disctype[$index]}" = "CD" ]]; then
1965                                 disctype="CD #"
1966                                 cd="${iso_cdnum[$index]}"
1967                         else
1968                                 disctype="DVD, representing CDs #"
1969                                 cd=0
1970                         fi

1972                         verbose "    ${disctype}${iso_cdnum[$index]}\n"

1974                         #
1975                         # Once we've matched a particular distro, don't check
```

```
1976                        # this ISO to see if it's part of any other.
1977                        #
1978                        unset iso_filename[$index]

1980                        iso_names[$cd]="$iso"

1982                        #
1983                        # A DVD-based distro consists of one and ONLY one disc,
1984                        # so process it now.
1985                        #
1986                        if [[ "${iso_disctype[$index]}" = "DVD" ]]; then
1987                                typeset dvd_discs=",${iso_cdnum[$index]}"

1989                                cd=1
1990                                while ((cd <= ncds)); do
1991                                        dvd_discs=$(echo "$dvd_discs" |
1992                                            sed "s/,$cd//")
1993                                        ((cd += 1))
1994                                done

1996                                #
1997                                # If no CDs are left in $dvd_discs, the DVD
1998                                # was a complete distribution, so add it to
1999                                # the valid distro list.
2000                                #
2001                                if [[ -z $dvd_discs ]]; then
2002                                        add_to_distro_list "DVD"
2003                                        unset iso_names[$cd]
2004                                fi
2005                        elif [[ -n ${iso_pers[$index]} ]]; then
2006                                #
2007                                # If this is a RedHat personality CD, save off
2008                                # some extra information about it so we can
2009                                # discern between mutiple personality discs
2010                                # later, if needed.
2011                                #
2012                                pers[${#pers[@]}]=${iso_pers[$index]}
2013                                pers_cd[${#pers_cd[@]}]="$iso"
2014                        fi

2016                        ((index += 1))
2017                done

2019                #
2020                # Check to see if we have ISOs representing a full CD set.
2021                # If we don't, don't mark this as an available distro.
2022                #
2023                (( ${#iso_names[@]} != $ncds )) && continue

2025                relase_name="$release_name $distro_version"
2026
2027                if [[ -z ${pers[@]} ]]; then
2028                        #
2029                        # If there were no personality discs, just add this
2030                        # ISO set to the distro list.
2031                        #
2032                        unset redhat_pers
2033                        add_to_distro_list "CD Set"
2034                else
2035                        #
2036                        # If a valid CD-based distro was found and there are
2037                        # RedHat personality discs for that distro present,
2038                        # create entries for each personality in the available
2039                        # distro list.
2040                        #
2041                        pers_index=0
```



```
2043                        while ((pers_index < ${#pers[@]})); do
2044                                redhat_pers=${pers[$pers_index]}

2046                                if [[ -n ${pers_cd[$pers_index]} ]]; then
2047                                        #
2048                                        # RedHat personality discs are always
2049                                        # disc 1 of a CD set, so if we found a
2050                                        # valid personality disc for this set,
2051                                        # set the disc 1 entry for this distro
2052                                        # to the ISO for the proper personality
2053                                        # disc.
2054                                        #
2055                                        iso_names[1]="${pers_cd[$pers_index]}"
2056                                        add_to_distro_list "CD Set"
2057                                fi

2059                                ((pers_index += 1))
2060                        done
2061                fi
2062        done
2063 }

2065 #
2066 # Do a lofi add for the passed filename and set lofi_dev to the lofi
2067 # device name lofiadm created for it (e.g. "/dev/lofi/1".)
2068 #
2069 # If the passed filename already has a lofi device name, simply set lofi_dir
2070 # to the existing device name.
2071 #
2072 # Returns 0 on success, 1 on failure.
2073 #
2074 lofi_add()
2075 {
2076         typeset filename="$1"

2078         lofi_dev=$(lofiadm "$filename" 2>/dev/null) && return 0
2079         lofi_dev=$(lofiadm -a "$filename") && return 0

2081         screenlog "$lofi_failed" "$filename"
2082         return 1
2083 }

2085 #
2086 # Delete the lofi device name passed in.
2087 #
2088 # Returns 0 on success, 1 on failure.
2089 #
2090 lofi_del()
2091 {
2092         typeset dev="$1"

2094         [[ "$dev" != /dev/lofi/* ]] && return 1

2096         if lofiadm -d "$dev" 2>/dev/null; then
2097                 [[ -n $lofi_dev ]] && unset lofi_dev
2098                 return 0
2099         fi

2101         return 1
2102 }

2104 #
2105 # Mount the lofi device name passed in.
2106 #
2107 # Set the variable mntdir to the directory on which the lofi device is
```

```
2108 # mounted.
2109 #
2110 # Returns 0 on success, 1 on failure.
2111 #
2112 lofi_mount()
2113 {
2114         typeset lofidev="$1"
2115         typeset mntpoint="$2"

2117         #
2118         # Check to see if the lofi device is already mounted and return
2119         # the existing mount point if it is.
2120         #
2121         get_mountdir "$lofidev" && { mntdir="$mount_dir" ; return 0 ; }

2123         unset mntdir
2124         if [[ ! -d  "$mntpoint" ]]; then
2125                 if ! mkdir -p "$mntpoint"; then
2126                         log "Could not create mountpoint \"$mntpoint\"!\n"
2127                         return 1
2128                 fi
2129                 lofi_created="$mntpoint"
2130         fi

2132         verbose "Attempting mount of device \"$lofidev\""
2133         verbose "  on directory \"$mntpoint\"... \c"

2135         if ! mount -F hsfs -r "$lofidev" "$mntpoint" 2>/dev/null; then
2136                 verbose "FAILED."
2137                 [[ -n $lofi_created ]] && rmdir -ps "$lofi_created" &&
2138                     unset lofi_created
2139                 return 1
2140         fi

2142         mntdir="$mntpoint"
2143         verbose "succeeded."
2144         return 0
2145 }

2147 #
2148 # Unmount the lofi device name passed in, and remove the device mount point
2149 # after unmounting the device.
2150 #
2151 # Returns 0 on success, 1 on failure.
2152 #
2153 lofi_umount()
2154 {
2155         typeset mntdev="$1"

2157         #
2158         # If the directory name passed wasn't mounted to begin with,
2159         # just return success.
2160         #
2161         get_mountdir "$mntdev" || return 0

2163         verbose "Unmounting device \"$mntdev\"... \c"

2165         if ! umount "$mntdev" ; then
2166                 verbose "FAILED."
2167                 return 1
2168         fi

2170         verbose "succeeded."
2171         return 0
2172 }
```



```
2174 # Scan the passed list of ISOs.
2175 scan_isos()
2176 {
2177         typeset iso
2178         typeset index=0

2180         unset iso_serial
2181         unset iso_release
2182         unset iso_cdnum
2183         unset iso_disctype
2184         unset iso_filename
2185         unset iso_pers

2187         for iso in "$@"; do
2188                 verbose "Checking possible ISO\n  \"$iso\"..."

2190                 if lofi_add "$iso"; then
2191                         verbose "  added as lofi device \"$lofi_dev\""
2192                         if lofi_mount "$lofi_dev" "/tmp/lxiso"; then
2193                                 if read_discinfo "$mntdir"; then
2194                                         iso_release[$index]="$rd_release"
2195                                         iso_serial[$index]="$rd_serial"
2196                                         iso_cdnum[$index]="$rd_cdnum"
2197                                         iso_disctype[$index]="$rd_disctype"

2199                                         [[ -n $rd_pers ]] &&
2200                                             iso_pers[$index]="$rd_pers"

2202                                         iso_filename[$index]="$iso"
2203                                         ((index += 1))
2204                                 fi
2205                                 lofi_umount "$lofi_dev"
2206                         else
2207                                 verbose "  not a usable ISO image."
2208                                 log "Unable to mount \"$lofi_dev\" (\"$iso\")"
2209                         fi

2211                         lofi_del "$lofi_dev"
2212                 else
2213                         verbose "  not a valid ISO image."
2214                 fi
2215         done
2216 }

2218 #
2219 # Prompt the user with the first argument, then make a menu selection
2220 # from the balance.
2221 #
2222 # This is effectively similar to the ksh "select" function, except it
2223 # outputs to stdout.
2224 #
2225 # Shell variables set:
2226 #    choice    - set to the menu number selected
2227 #    selection - set to the menu text selected
2228 #
2229 pick_one()
2230 {
2231         typeset menu_items
2232         typeset menu_index
2233         typeset reply

2235         typeset prompt="$1"
2236         shift

2238         unset choice
```

```
2240         set -A menu_items "$@"

2242         until [[ -n $choice ]]; do
2243                 menu_index=1
2244
2245                 echo "\n$prompt\n"

2247                 for f in "${menu_items[@]}"; do
2248                         echo "$menu_index) $f"
2249                         ((menu_index += 1))
2250                 done

2252                 echo "\n$(gettext "Please select") (1-$#): " "\c"
2253                 read reply
2254                 echo

2256                 [[ -z $reply ]] && echo && continue

2258                 #
2259                 # Reprint menu selections if the answer was not a number in
2260                 # range of the menu items available
2261                 #
2262                 [[ $reply != +([0-9]) ]] && continue
2263                 ((reply < 1)) || ((reply > $#)) && continue

2265                 choice=$reply
2266                 selection=${menu_items[((choice - 1))]}
2267         done
2268 }

2270 #
2271 # Select a distribution to install from the arguments passed and set
2272 # "ndsitro" to the value chosen - 1 (so it may be used as an array index.)
2273 #
2274 # The routine will automatically return with ndisto set to 0 if only one
2275 # argument is passed.
2276 #
2277 select_distro()
2278 {
2279         unset choice
2280         unset ndistro

2282         if (($# > 1)); then
2283                 if [[ -n $silent_mode ]]; then
2284                         typeset dist

2286                         log "ERROR: multiple distrubutions present in ISO" \
2287                             "directory but silent install"
2288                         log "  mode specified.  Distros available:"
2289                         for dist in "$@"; do
2290                                 log "    \"$dist\""
2291                         done
2292                         return 1
2293                 fi

2295                 pick_one \
2296                     "$(gettext "Which distro would you like to install?")" \
2297                     "$@"
2298         fi

2300         #
2301         # Covers both the cases of when only one distro name is passed
2302         # to the routine as well as when an EOF is sent to the distribution
2303         # selection prompt.
2304         #
2305         if [[ -z $choice ]]; then
```



```
2306                 screenlog "$install_dist" "$1"
2307                 ndistro=0
2308         else
2309                 screenlog "$install_dist" "$selection"
2310                 ndistro=$((choice - 1))
2311         fi

2313         return 0
2314 }

2316 #
2317 # Install a zone from discs or manually lofi-mounted ISOs.
2318 #
2319 # Return 0 on success, 1 on failure
2320 #
2321 do_disc_install()
2322 {
2323         typeset path="$1"

2325         typeset eject_final="N"
2326         typeset install_status

2328         #
2329         # Get a disc, it doesn't matter which one.
2330         #
2331         # We don't know which distro this may be yet, so we can't yet
2332         # ask for the first disc in the install order.
2333         #
2334         if ! get_cd "$path"; then
2335                 if [[ -z $silent_mode ]]; then
2336                         typeset distro_disc=\
2337                             $(gettext "a supported Linux distribution disc")

2339                         screenlog "\n$distro_mediafail" "$distro_disc ($path)"
2340                 fi
2341                 return 1
2342         fi

2344         if [[ -n $silent_mode && "$rd_disctype" = "CD" ]]; then
2345                 log "$silent_err_msg"
2346                 return 1
2347         fi

2349         if ! get_disc_distro "$mntdir"; then
2350                 msg=$(gettext "Unable to find a supported Linux release on")
2351                 screenlog "$msg"
2352                 screenlog "  $media_spec" "$path"
2353                 umount "$mntdir" > /dev/null 2>&1
2354                 return 1
2355         fi

2357         check_mbfree $zoneroot $distro_mb_required || return 1
2358         build_rpm_list $install_packages

2360         echo

2362         if [[ "$install_media" = "disc" ]]; then
2363                 #
2364                 # If we're in interactive mode, ask the user if they want the
2365                 # disc ejected when the installation is complete.
2366                 #
2367                 # Silent mode installs will require the user to manually run
2368                 # eject(1).
2369                 #
2370                 if [[ -n $removable && -z $silent_mode ]]; then
2371                         typeset ans
```

```
2372                        typeset disc
2373                        typeset status
2374                        typeset which=""

2376                        disc="$rd_disctype"
2377                        [[ "$disc" = "CD" ]] && which=$(gettext "final ")

2379                        #
2380                        # Ask the user if they want the install disc ejected
2381                        # when the installation is complete.  Any answer but
2382                        # "n" or "N" is taken to mean yes, eject it.
2383                        #
2384                        eject_final="Y"
2385                        status=$(gettext "WILL")

2387                        screenlog "$eject_final_msg" "$which" "$disc"
2388                        screenlog "  $eject_final_prompt" "$zonename" "[y]/n"

2390                        read ans && [[ "$ans" = [Nn]* ]] && eject_final="N" &&
2391                            status=$(gettext "will NOT")

2393                        screenlog "\n$eject_final_status\n" "$which" "$disc" \
2394                            "$status"
2395                fi

2397                screenlog "$install_ndiscs" "$distro_ncds"

2399                msg=$(gettext "install %s.")
2400                screenlog "$msg" "$release_name"
2401        else
2402                screenlog "$install_nisos" "$distro_ncds"

2404                msg=$(gettext "DVD) to install %s.")
2405                screenlog "$msg" "$release_name"
2406        fi

2408        install_from_media "$path"
2409        install_status=$?

2411        [[ "$eject_final" = "Y" ]] && eject_removable_disc

2413        return $install_status
2414 }

2416 #
2417 # Install a zone using the list of ISO files passed as arguments to this
2418 # function.
2419 #
2420 # Return 0 on success, 1 on failure.
2421 #
2422 do_iso_install()
2423 {
2424        typeset install_status
2425        typeset iso_path
2426        typeset ldev

2428        msg=$(gettext "Checking for valid Linux distribution ISO images...")
2429        screenlog "\n$msg"

2431        scan_isos "$@"

2433        if [[ -z ${iso_filename[@]} ]]; then
2434                msg=$(gettext "No valid ISO images available or mountable.")
2435                screenlog "\n$msg"
2436                return 1
2437        fi
```



```
2438
2439        validate_iso_distros

2441        if [[ -z ${release[@]} ]]; then
2442                msg=$(gettext "No supported Linux distributions found.")
2443                screenlog "\n$msg"
2444                return 1
2445        fi

2447        select_distro "${select_name[@]}" || return 1
2448        unset select_name

2450        . ${distro_file[$ndistro]} > /dev/null
2451        distro_ncds=${#distro_cdorder[@]}

2453        check_mbfree $zoneroot $distro_mb_required || return 1
2454        build_rpm_list $install_packages

2456        unset lofi_devs

2458        verboselog ""
2459        for iso_path in ${iso_set[$ndistro]}; do
2460                if ! lofi_add "$iso_path"; then
2461                        for ldev in $lofi_devs; do
2462                                lofi_del "$ldev"
2463                        done
2464                        return 1
2465                fi

2467                verboselog "Added \"$iso_path\""
2468                verboselog "  as \"$lofi_dev\""
2469                lofi_devs="$lofi_devs $lofi_dev"
2470        done

2472        release_name="${release[$ndistro]}"

2474        set -A iso_pathnames "zero_pad" ${iso_set[$ndistro]}
2475        install_from_media $lofi_devs
2476        install_status=$?

2478        for ldev in $lofi_devs; do
2479                lofi_del "$ldev"
2480        done

2482        unset lofi_devs
2483        return $install_status
2484 }

2486 # Clean up on interrupt
2487 trap_cleanup()
2488 {
2489        cd "$cwd"

2491        msg=$(gettext "Interrupt received, cleaning up partial install...")
2492        screenlog "$msg"

2494        [[ -n $miniroot_booted ]] && zoneadm -z "$zonename" halt &&
2495            unset miniroot_booted && unset newroot_mounted

2497        #
2498        # OK, why a sync here?  Because certain commands may have written data
2499        # to mounted file systems before the interrupt, and given just the right
2500        # timing there may be buffered data not yet sent to the disk or the
2501        # system may still be writing data to the disk.  Either way, the umount
2502        # will then fail because the system will still see the mounted
2503        # filesystems as busy.
```

```
2504         #
2505         sync

2507         if [[ -n $newroot_mounted ]]; then
2508                 umount_list $newroot_mounted
2509                 unset newroot_mounted
2510         fi

2512         if [[ -n $zone_mounted ]]; then
2513                 umount "$zone_mounted"
2514                 unset zone_mounted
2515         fi

2517         #
2518         # Normally, this isn't needed but there is a window where mntdir is set
2519         # before zone_mounted, so account for that case.
2520         #
2521         if [[ -n $mntdir ]]; then
2522                 umount "$mntdir"
2523                 unset mntdir
2524         fi

2526         [[ -n $lofi_dev ]] && lofi_del "$lofi_dev"

2528         if [[ -n $lofi_devs ]]; then
2529                 typeset ldev

2531                 for ldev in $lofi_devs
2532                 do
2533                         lofi_del "$ldev"
2534                 done

2536                 unset lofi_devs
2537         fi

2539         [[ -n $lofi_created ]] && rmdir -ps "$lofi_created" &&
2540             unset lofi_created

2542         msg=$(gettext "Installation aborted.")
2543         screenlog "$msg"
2544         exit $ZONE_SUBPROC_FATAL
2545 }

2547 #
2548 # Start of main script
2549 #
2550 cwd=$(dirname "$0")
2551 distro_dir="$cwd/distros"

2553 unset deferred_saved
2554 unset distro_path
2555 unset logfile
2556 unset msg
2557 unset newroot_mounted
2558 unset silent_err_msg
2559 unset silent_mode
2560 unset verbose_mode
2561 unset zone_mounted
2562 unset zoneroot
2563 unset zonename

2565 #
2566 # Exit values used by the script, as #defined in <sys/zone.h>
2567 #
2568 #       ZONE_SUBPROC_OK
2569 #       ===============
```



```
2570 #       Installation was successful
2571 #
2572 #       ZONE_SUBPROC_USAGE
2573 #       ==================
2574 #       Improper arguments were passed, so print a usage message before exiting
2575 #
2576 #       ZONE_SUBPROC_NOTCOMPLETE
2577 #       ========================
2578 #       Installation did not complete, but another installation attempt can be
2579 #       made without an uninstall
2580 #
2581 #       ZONE_SUBPROC_FATAL
2582 #       ==================
2583 #       Installation failed and an uninstall will be required before another
2584 #       install can be attempted
2585 #
2586 ZONE_SUBPROC_OK=0
2587 ZONE_SUBPROC_USAGE=253
2588 ZONE_SUBPROC_NOTCOMPLETE=254
2589 ZONE_SUBPROC_FATAL=255

2591 #
2592 # Process and set up various global option variables:
2593 #
2594 #    distro_path - Path containing files that make up the distribution
2595 #                  (e.g. a directory containing ISO files or a disc device)
2596 #    logfile     - Name (if any) of the install log file
2597 #    zoneroot    - Root directory for the zone to install
2598 #    zonename    - Name of the zone to install
2599 #
2600 while getopts 'svxd:l:r:z:' opt; do
2601         case $opt in
2602                 s) silent_mode=1; unset verbose_mode;;
2603                 v) verbose_mode=1; unset silent_mode;;
2604                 x) set -x;;
2605                 d) distro_path="$OPTARG";;
2606                 l) logfile="$OPTARG";;
2607                 r) zoneroot="$OPTARG";;
2608                 z) zonename="$OPTARG";;
2609         esac
2610 done
2611 shift OPTIND-1

2613 distro_path=${distro_path:=/cdrom/cdrom0}

2615 install_packages="$@"

2617 [[ -n $silent_mode ]] && exec 1>/dev/null

2619 if [[ -z $zonename ]]; then
2620         msg=$(gettext "ERROR:  Cannot install - no zone name was specified")
2621         screenlog "$msg"
2622         echo
2623         exit $ZONE_SUBPROC_NOTCOMPLETE
2624 fi

2626 if [[ -z $zoneroot ]]; then
2627         msg=$(gettext "ERROR:  Cannot install - no zone root directory was")
2628         screenlog "$msg"

2630         msg=$(gettext "specified.")
2631         screenlog "  $msg"
2632         echo
2633         exit $ZONE_SUBPROC_NOTCOMPLETE
2634 fi
```

```
# Make sure the specified zone root directory exists
[[ -d "$zoneroot" ]] || mkdir -m 0700 -p "$zoneroot"

if [[ ! -d "$zoneroot" ]]; then
        screenlog "$zone_rootfail" "$zoneroot"
        echo
        exit $ZONE_SUBPROC_NOTCOMPLETE
fi

rootdir="$zoneroot/root"

# Make sure the specified zone root subdirectory exists
[[ -d "$rootdir" ]] || mkdir -p "$rootdir"

if [[ ! -d "$rootdir" ]]; then
        screenlog "$zone_rootsub" "$rootdir"
        echo
        exit $ZONE_SUBPROC_NOTCOMPLETE
fi

media_mntdir="$rootdir/media"

if [[ -n $logfile ]]; then
        # If a log file was specified, log information regarding the install
        log "\nInstallation started ‘date’"
        log "Installing from path \"$distro_path\""
else
        # Redirect stderr to /dev/null if silent mode is specified.
        [[ -n $silent_mode ]] && exec 2>/dev/null
fi

distro_path=${distro_path:=$default_distro_path}

# From this point on, call trap_cleanup() on interrupt (^C)
trap trap_cleanup INT

verbose "Installing zone \"$zonename\" at root \"$zoneroot\""
release_name="supported Linux distribution"

#
# Based on the pathname, attempt to determine whether this will be a disc or
# lofi-based install or one using ISOs.
#
if [[ "$distro_path" = /cdrom/* || "$distro_path" = /media/* ||
    "$distro_path" = /dev/dsk/* || "$distro_path" = /dev/lofi/* ]]; then
        if [[ "$distro_path" = /dev/lofi/* ]]; then
                silent_err_msg="$silent_nolofi"
                install_media="lofi"
        else
                silent_err_msg="$silent_nodisc"
                install_media="disc"
        fi

        if [[ "$distro_path" = /cdrom/* || "$distro_path" = /media/* ]]; then
                managed_removable=1
        else
                managed_removable=0
        fi

        log "Installing zone \"$zonename\" at root \"$zoneroot\""
        verboselog "  Attempting ${install_media}-based install via:"
        verboselog "    \"$distro_path\""

        do_disc_install "$distro_path"
else
        typeset dir_start
```



```
        typeset dir_file

        dir_start=$(dirname "$distro_path" | cut -c 1)

        [[ "$dir_start" != "/" ]] && distro_path="${PWD:=$(pwd)}/$distro_path"

        if [[ ! -d "$distro_path" ]]; then
                screenlog "$no_distropath" "$distro_path"
                echo
                exit $ZONE_SUBPROC_NOTCOMPLETE
        fi

        log "Installing zone \"$zonename\" at root \"$zoneroot\""
        verboselog "  Attempting ISO-based install from directory:"
        verboselog "    \"$distro_path\""

        unset iso_files

        for dir_file in $distro_path/*; do
                #
                # Skip this file if it’s not a regular file or isn’t readable
                #
                [[ ! -f $dir_file || ! -r $dir_file ]] && continue

                #
                # If it’s an hsfs file, it’s an ISO, so add it to the possible
                # distro ISO list
                #
                filetype=$(LC_ALL=C fstyp $dir_file 2>/dev/null) &&
                    [[ "$filetype" = "hsfs" ]] &&
                    iso_files="$iso_files $dir_file"
        done

        install_media="ISO"
        do_iso_install $iso_files
fi

if [[ $? -ne 0 ]]; then
        cd "$cwd"

        [[ -n $miniroot_booted ]] && zoneadm -z "$zonename" halt &&
            unset miniroot_booted && unset newroot_mounted

        if [[ -n $zone_mounted ]]; then
                umount "$zone_mounted"
                unset zone_mounted
        fi

        if [[ -n $newroot_mounted ]]; then
                umount_list $newroot_mounted
                unset newroot_mounted
        fi

        screenlog "\n$install_failed\n" "$release_name" "$zonename" "‘date‘"

        msg=$(gettext "Cleaning up after failed install...")
        screenlog "$msg"

        #
        # The extra checks are some basic paranoia due to the potentially
        # dangerous nature of these commands but are not intended to catch all
        # malicious cases.
        #
        [[ -d "$zoneroot/a" ]] && rm -rf "$zoneroot/a"

        exit $ZONE_SUBPROC_FATAL
```

```
2768 fi

2770 screenlog "$install_done" "$release_name" "$zonename" "‘date‘"

2772 exit $ZONE_SUBPROC_OK
2773 #endif /* ! codereview */
```

```
**********************************************************
   17594 Tue Jan 14 16:17:15 2014
new/usr/src/lib/brand/lx/zone/lx_init_zone.ksh
Bring back LX zones.
**********************************************************
```

```
#!/bin/ksh -p
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
# Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "%Z%%M% %I%     %E% SMI"
#


#
# This script contains various routines used to post-process a zone for use
# with BrandZ after it has been installed from RPM media or a tar image.
#
# Briefly, there are three main jobs we need to do:
#
# 1) Create any needed directories and symlinks BrandZ needs but that the
#    Linux install may not create
#
# 2) Modify rc scripts to shut off services that don't apply to a zone
#    or that wish to access hardware directly
#
# 3) Modify various Linux system files for use within a zone environment
#


#
# Restrict executables to /bin and /usr/bin
#
PATH=/bin:/usr/bin
export PATH

#
# Sends output to a log file via redirection of stderr.
#
# This script assumes its caller has already performed the redirection to the
# logfile.
#
log()
{
        echo "$@" >&2
}

#
# Setup i18n output
```



```
#
TEXTDOMAIN="SUNW_OST_OSCMD"
export TEXTDOMAIN

cmd_failed=$(gettext "%s failed!  Aborting installation...")
cmd2_failed=$(gettext "%s of '%s' to '%s' failed!")
create_failed=$(gettext "Could not create new file '%s'!")
disable_failed=$(gettext "Attempt to disable entries in '%s' failed!")
install_aborted=$(gettext "Aborting installation...")
install_noroot=$(gettext "Installation root directory '%s' does not exist.")
ln_fail=$(gettext "Unable to symlink '%s' to '%s'!")
mkdir_fail=$(gettext "Unable to create the directory '%s'")
mod_failed=$(gettext -n "Attempt to modify entries in '%s' failed!")

usage=$(gettext "usage: %s <install_root> [mini]")

#
# Output an internationalized string followed by a carriage return
#
i18n_echo()
{
        typeset fmt="$1"
        shift

        printf "$fmt\n" "$@"
}

#
# Routine to make a full path out of a supplied path
#
fullpath()
{
        typeset path="$1"

        echo $path | egrep -s "^/" || path="${PWD:=$(pwd)}/$path"
        echo $path
}

#
# Routine to create directories and handle errors
#
makedir()
{
        typeset dirname=$(fullpath "$1")
        typeset mode=""

        [[ $# -eq 2 ]] && mode="-m $2"

        [[ -d "$dirname" ]] && return

        if ! mkdir $mode -p "$dirname"; then
                log "Unable to create the directory \"$dirname\"!"
                i18n_echo "$mkdir_fail" "$dirname"
                echo $(gettext "Aborting installation...")
                exit 1
        fi
}

#
# Routine to create initial symlinks and handle errors
#
symlink()
{
        typeset src="$1"
        typeset dst=$(fullpath "$2")
```

```
128         [[ -e "$dst" || -h "$dst" ]] && rm -f "$dst"
129
130         if ! ln -s "$src" "$dst"; then
131                 log "Unable to symlink \"$src\" to \"$dst\"!"
132                 i18n_echo "$ln_fail" "$src" "$dst"
133                 echo $(gettext "Aborting installation...")
134                 exit 1
135         fi
136 }

138 #
139 # Install a file using "ln -s"
140 #
141 # Returns 0 on success, 1 on failure.
142 #
143 install_ln()
144 {
145         typeset source="$1"
146         typeset target=$(fullpath "$2")

148         log "    Installing \"$target\""

150         mv -f "$target" "$target.$tag" 2>/dev/null

152         if ! ln -s "$source" "$target"; then
153                 log ""
154                 log "Attempt to install $target FAILED."
155                 return 1
156         fi

158         return 0
159 }


162 #
163 # Enable NFS servers and the NFS lock daemon for a particular zone.
164 #
165 enable_nfs_services()
166 {
167         log "Non-miniroot install; enabing NFS servers and NFS lock daemon"

169         #
170         # Setup files required for NFS:
171         #
172         #       /native/etc/netconfig
173         #       /native/etc/default/nfs
174         #
175         # These two files are treated as read-only in lx branded zones.
176         # To enfore this restriction we will read-only lofs mount them
177         # into the zone from the global zone.  For these lofs mounts to
178         # work we'll need to create empty directories now that will serve
179         # as mount points later.
180         #
181         #       /sbin/rpc.statd
182         #       /sbin/rpc.lockd
183         #
184         # These files are symlinks to scripts supplied by the lx brand
185         # that will start up the solaris nfs daemons.
186         #
187         if { ! makedir native/etc/netconfig ||
188             ! makedir native/etc/default/nfs ; }; then
189                 log "Aborting NFS setup..."
190                 log ""
191                 return
192         fi
```



```
194         if { ! install_ln ../native/usr/lib/brand/lx/lx_lockd sbin/rpc.lockd ||
195             ! install_ln ../native/usr/lib/brand/lx/lx_statd \
196             sbin/rpc.statd ; }; then
197                 log "Aborting NFS setup..."
198                 log ""
199                 return
200         fi

202         #
203         # update /etc/services for NFS
204         #
205         log ""
206         log "Adding lockd entry to \"$install_root/etc/services\"..."

208         cp -p $install_root/etc/services $install_root/etc/services.$tag

210         #
211         # Brackets in the sed script below contain a space followed by a tab
212         #
213         cat $install_root/etc/services.$tag |
214             sed 's:\(111\/..p[  ][      ]*\):\1rpcbind :' |
215             cat > $install_root/etc/services

217         cat >> $install_root/etc/services <<-EOF
218         lockd           4045/udp                # NFS lock daemon/manager
219         lockd           4045/tcp                # NFS lock daemon/manager
220         EOF

222         #
223         # Modify /etc/init.d/nfslock to enable the USERLAND_LOCKD option and to
224         # find some commands in alternate locations.
225         #
226         log ""
227         log "Modifying \"$install_root/etc/init.d/nfslock\"..."
228         cp -p etc/init.d/nfslock etc/init.d/nfslock.$tag
229         cat etc/init.d/nfslock.$tag |
230             sed '
231                 s/USERLAND_LOCKD=$/USERLAND_LOCKD="yes"/
232                 s/killproc rpc.statd/killproc statd/
233                 s/status rpc.statd/status statd/
234                 s/pidof rpc.statd/pidof statd/
235             ' |
236             cat > etc/init.d/nfslock
237 }

239 #
240 # The main script starts here.
241 #
242 # The syntax is:
243 #
244 #     lx_init_zone <rootdir> [mini]
245 #
246 # Where:
247 #       <rootdir> is the root of the zone directory to be modified
248 #
249 #       [mini]  is an optional second argument that signifies whether this is
250 #               to be a miniroot install; if it is, NFS services are not enabled
251 #               in the processed zone
252 #
253 unset is_miniroot
254 unset install_root

256 install_root="$1"

258 tag="lxsave_$(date +%m.%d.%Y@%T)"
```

```
260 if (($# < 1 || $# > 2)); then
261         i18n_echo "$usage" "$0"
262         exit 1
263 fi

265 (($# == 2)) && is_miniroot=1

267 if [[ ! -d "$install_root" ]]; then
268         i18n_echo "$install_noroot" "$install_root"
269         echo $(gettext "** Installation aborted **")
270         exit 1
271 fi

273 cd "$install_root"

275 log ""
276 log "Initial lx_brand environment modification started ‘date‘"
277 log "Making needed directories in \"$install_root\"."
278 echo $(gettext "Setting up the initial lx brand environment.")

280 #
281 # Make various directories in /native that are needed to boot an lx branded
282 # zone.
283 #
284 makedir native/dev
285 makedir native/etc/default
286 makedir native/etc/svc/volatile
287 makedir native/lib
288 makedir native/proc
289 makedir native/tmp 1777
290 makedir native/usr
291 makedir native/var

293 #
294 # Make various other directories needed for the lx brand
295 #
296 makedir mnt
297 makedir opt
298 makedir usr/local/bin
299 makedir usr/local/include
300 makedir usr/local/lib
301 makedir usr/local/sbin
302 makedir usr/local/share
303 makedir usr/local/src

305 makedir dev 0755
306 makedir tmp 1777
307 makedir proc 0555
308 makedir boot 0755

310 #
311 # zlogin requires that these utilities live in places other than their
312 # Linux defaults, so create appropriate links for them here.
313 #
314 # XX - The need for these links may go away in the future if zlogin is
315 #      appropriately modified
316 #
317 symlink /bin/sh sbin/sh
318 symlink /bin/su usr/bin/su
319 symlink /native/usr/lib/ld.so.1 usr/lib/ld.so.1

321 libpam_so="$(echo lib/libpam.so.0.*)"
322 libpam_misc="$(echo lib/libpam_misc.so.0.*)"
323 libpamc_so="$(echo lib/libpamc.so.0.*)"

325 symlink "/$libpam_so" lib/libpam.so.0
```



```
326 symlink "/$libpam_misc" lib/libpam_misc.so.0
327 symlink "/$libpamc_so" lib/libpamc.so.0

329 log ""
330 log "Modifying system configuration in \"$install_root\""

332 #
333 # Create a /var/ld/ld.config that will point to /native/lib for our Solaris
334 # libraries.
335 #
336 log "Creating \"$install_root/var/ld/ld.config\"..."

338 makedir var/ld

340 if ! crle -c var/ld/ld.config -l /native/lib:/native/usr/lib \
341      -s /native/lib/secure:/native/usr/lib/secure; then
342         log "\tCreation of \"$install_root/var/ld/ld.config\" failed!"
343         i18n_echo "$cmd_failed" "crle"
344         exit 1
345 fi

347 log ""
348 log "Modifying \"$install_root/etc/fstab\"..."

350 mv -f etc/fstab etc/fstab.$tag 2>/dev/null

352 cat > etc/fstab <<- EOF
353         none            /                       ufs     defaults        1 1
354         none            /proc                   proc    defaults        0 0
355 EOF

357 if [[ $? -ne 0 ]]; then
358         log "Could not create new \"$install_root/etc/fstab\"!"
359         i18n_echo "$create_failed" "$install_root/etc/fstab"
360         exit 1
361 fi

363 #
364 # The default /etc/inittab spawns mingetty on each of the virtual consoles
365 # as well as xdm on the X console.  Since we don’t have virtual consoles nor
366 # an X console, spawn a single mingetty on /dev/console instead.
367 #
368 # Don’t bother changing the file if it looks like we already did.
369 #
370 if ! egrep -s "Disabled by lx brand" etc/inittab; then
371         log "Modifying: \"$install_root/etc/inittab\"..."

373         tmpfile=/tmp/inittab.$$

375         sed ’s/^[1-6]:/# Disabled by lx brand: &/
376             s/^id:5:initdefault:/id:3:initdefault: # Modified by lx brand: &/’ \
377             etc/inittab > $tmpfile

379         #
380         # Don’t bother with further alterations if the sed above failed...
381         #
382         if [[ $? -eq 0 ]]; then
383                 egrep -s "console login for lx brand" etc/inittab
384                 if [[ $? -ne 0 ]]; then
385                         cat >> $tmpfile <<- EOF

387                                 #
388                                 # console login for lx brand
389                                 #
390                                 1:2345:respawn:/sbin/mingetty console
391                         EOF
```

```
393                         #
394                         # Only install the new inittab if the append
395                         # above succeeded.
396                         #
397                         if [[ $? -eq 0 ]]; then
398                                 #
399                                 # Attempt to save off the original inittab
400                                 # before moving over the modified version.
401                                 #
402                                 mv -f etc/inittab etc/inittab.$tag 2>/dev/null

404                                 mv -f $tmpfile etc/inittab

406                                 if [[ $? -ne 0 ]]; then
407                                         log "mv of \"$tmpfile\" to" \
408                                             "\"$installroot/etc/inittab\"" \
409                                             "failed!"
410                                         i18n_echo "$cmd2_failed" "mv" \
411                                             "$tmpfile" \
412                                             "$installroot/etc/inittab"
413                                         i18n_echo "$install_aborted"
414                                         exit 1
415                                 else
416                                         chmod 644 etc/inittab
417                                 fi
418                         fi
419                 fi

421         else
422                 log "Attempt to disable entries in" \
423                     "\"$install_root/etc/inittab\" failed!"
424                 i18n_echo "$disable_failed" "$install_root/etc/inittab"
425                 i18n_echo "$install_aborted"
426                 exit 1
427         fi
428 fi

430 if [[ ! -e "$install_root/etc/hosts" ]]; then
431         log ""
432         log "Creating: \"$install_root/etc/hosts\"..."

434         cat > "$install_root/etc/hosts" <<-_EOF_
435                 127.0.0.1               localhost
436         _EOF_
437 fi

439 #
440 # User must configure various brand-specific items to enable networking, so
441 # boot the system non-networked.
442 #
443 log ""
444 log "Modifying: \"$install_root/etc/sysconfig/network\"..."

446 mv -f etc/sysconfig/network etc/sysconfig/network.$tag 2>/dev/null

448 cat > etc/sysconfig/network <<- EOF
449         NETWORKING="no"
450         #
451         # To enable networking, change the "no" above to "yes" and
452         # uncomment and fill in the following parameters.
453         #
454         # If you are specifying a hostname by name rather than by IP address,
455         # be sure the system can resolve the name properly via the use of a
456         # name service and/or the proper name files, as specified by
457         # nsswitch.conf.  See nsswitch.conf(5) for further details.
```



```
458         #
459         # HOSTNAME=your_hostname_here
460         #
461 EOF

463 if [[ $? -ne 0 ]]; then
464         log "Could not create new \"$install_root/etc/sysconfig/network\"!"
465         i18n_echo "$create_failed" "$install_root/etc/sysconfig/network"
466         i18n_echo "$install_aborted"
467         exit 1
468 fi

470 if [[ -a etc/sysconfig/syslog ]]; then
471         #
472         # By default, syslogd will attempt to create a socket in /dev/log, but
473         # /dev is not be writable.  Instead, modify /etc/sysconfig/syslog to
474         # tell it to use /var/run/syslog instead, and make /dev/log a symlink
475         # to /var/run/syslog.
476         #
477         log ""
478         log "Modifying: \"$install_root/etc/sysconfig/syslog\"..."

480         tmpfile=/tmp/lx_sc.syslog.$$

482         sed 's@\(SYSLOGD_OPTIONS="-m 0\)"@\1 -p /var/run/syslog"@' \
483             etc/sysconfig/syslog > $tmpfile

485         #
486         # Only install the new sysconfig/syslog if the edit above succeeded.
487         #
488         if [[ $? -eq 0 ]]; then
489                 #
490                 # Attempt to save off the original syslog before moving over
491                 # the modified version.
492                 #
493                 mv -f etc/sysconfig/syslog etc/sysconfig/syslog.$tag 2>/dev/null

495                 if ! mv -f $tmpfile etc/sysconfig/syslog; then
496                         log "mv of \"$tmpfile\" to" \
497                             "\"$installroot/etc/sysconfig/syslog\" failed!"
498                         i18n_echo "$cmd2_failed" "mv" "$tmpfile" \
499                             "$installroot/etc/sysconfig/syslog"
500                         i18n_echo "$install_aborted"
501                         exit 1
502                 else
503                         chmod 755 etc/sysconfig/syslog
504                 fi
505         else
506                 log "Attempt to modify entries in" \
507                     "\"$install_root/sysconfig/syslog\" failed!"
508                 i18n_echo "$mod_failed" "$install_root/sysconfig/syslog"
509                 i18n_echo "$install_aborted"
510                 exit 1
511         fi
512 fi

514 if [[ $? -ne 0 ]]; then
515         log "Could not create new \"$install_root/etc/sysconfig/syslog\"!"
516         i18n_echo "$create_failed" "$install_root/etc/sysconfig/syslog"
517         i18n_echo "$install_aborted"
518         exit 1
519 fi

521 #
522 # /etc/rc.d/init.d/keytable tries to load a physical keyboard map, which won't
523 # work in a zone. If we remove etc/sysconfig/keyboard, it won't try this at all.
```

```
524 #
525 mv -f etc/sysconfig/keyboard etc/sysconfig/keyboard.$tag 2>/dev/null

527 #
528 # /etc/rc.d/init.d/gpm tries to configure the console mouse for cut-and-paste
529 # text operations, which we don't support.  Removing this file disables the
530 # mouse configuration.
531 #
532 mv -f etc/sysconfig/mouse etc/sysconfig/mouse.$tag 2>/dev/null

534 #
535 # The following scripts attempt to start services or otherwise configure
536 # the system in ways incompatible with zones, so don't execute them at boot
537 # time.
538 #
539 log ""
540 log "Modifying \"$install_root/etc/rc.d/init.d\" to disable any"
541 log "  services not supported by BrandZ:"
542 unsupported_services="
543         kudzu
544         microcode_ctl
545         network
546         random
547         pcmcia
548         isdn
549         iptables
550         ip6tables
551         iscsi
552         psacct
553         gpm
554         irda
555         smartd
556         rawdevices
557         netdump
558         hpoj
559         mdmonitor
560         mdmpd
561         irqbalance
562 "

564 for file in $unsupported_services; do
565         if [[ -a "etc/rc.d/init.d/$file" ]]; then

567             if mv -f "etc/rc.d/init.d/$file" "etc/rc.d/init.d/$file.$tag"; then
568                     log "    + Moved script \"etc/rc.d/init.d/$file\" to"
569                     log "          \"etc/rc.d/init.d/$file.$tag\""
570             fi
571         fi

573         rc_files="$(echo etc/rc.d/rc[0-6].d/[SK]+([0-9])$file)"

575         if [[ "$rc_files" != "etc/rc.d/rc[0-6].d/[SK]+([0-9])$file" ]]; then
576                 for file in $rc_files; do
577                         if [[ -h "$file" ]]; then
578                                 rm -f "$file" &&
579                                     log "    + Removed symbolic link \"$file\""
580                         else
581                                 rm -f "$file" &&
582                                     log "    + Removed script \"$file\""
583                         fi
584                 done
585         fi
586 done

588 #
589 # There is a lot of stuff in the standard halt and reboot scripts that we
```



```
590 # have no business running in a zone.  Fortunately, the stuff we want to
591 # skip is all in one contiguous chunk.
592 #
593 # Don't bother to modify the file if it looks like we already did.
594 #
595 if ! egrep -s "Disabled by lx brand" etc/rc.d/init.d/halt; then
596         log ""
597         log "Modifying  \"$install_root/etc/rc.d/init.d/halt\" for operation"
598         log "  within a zone..."
599         awk 'BEGIN {skip = ""}
600             /^# Save mixer/ {skip = "# Disabled by lx brand: "}
601             /halt.local/ {skip = ""}
602             /./ {print skip $0}' etc/rc.d/init.d/halt > /tmp/halt.$$

604         if [[ $? -eq 0 ]]; then
605                 mv -f etc/rc.d/init.d/halt etc/rc.d/init.d/halt.$tag 2>/dev/null
606                 mv -f /tmp/halt.$$ etc/rc.d/init.d/halt
607                 chmod 755 etc/rc.d/init.d/halt
608         else
609                 log "Attempt to modify \"$install_root/etc/rc.d/init.d/halt\"" \
610                     "FAILED"
611                 log "Continuing with balance of zone setup..."
612         fi
613 fi

615 #
616 # Fix up /etc/rc.d/rc.sysinit:
617 #
618 # 1) /sbin/hwclock requires the iopl() system call, which BrandZ won't support.
619 #    Since the hardware clock cannot be set from within a zone, we comment out
620 #    the line.
621 #
622 # 2) Disable dmesg commands, since we don't implement klogctl
623 #
624 # 3) Disable initlog and the mount of /dev/pts
625 #
626 # 4) Don't touch /dev/tty* in order to start virtual terminals, as that won't
627 #    work from within a zone.
628 #
629 # 5) Don't try to check the root filesystem (/) as there is no associated
630 #    physical device, and any attempt to run fsck will fail.
631 #
632 # Don't modify the rc.sysinit file if it looks like we already did.
633 #
634 if ! egrep -s "Disabled by lx brand" etc/rc.d/rc.sysinit; then
635         log ""
636         log "Modifying: \"$install_root/etc/rc.d/rc.sysinit\"..."
637         log ""

639         tmpfile=/tmp/lx_rc.sysinit.$$

641         sed 's@^/sbin/hwclock@# Disabled by lx brand: &@
642             s@^HOSTTYPE=@HOSTTYPE=\"s390\" # Spoofed for lx brand: &@
643             s@/bin/dmesg -n@: # Disabled by lx brand: &@
644             s@^dmesg -s@# Disabled by lx brand: &@
645             s@initlog -c \"fsck@: # Disabled by lx brand: &@
646             s@^.*mount .* /dev/pts$@# Disabled by lx brand: &@' \
647             etc/rc.d/rc.sysinit > $tmpfile

649         #
650         # Only install the new rc.sysinit if the edit above succeeded.
651         #
652         if [[ $? -eq 0 ]]; then
653                 #
654                 # Attempt to save off the original rc.sysinit
655                 # before moving over the modified version.
```

```
656                 #
657                 mv -f etc/rc.d/rc.sysinit etc/rc.d/rc.sysinit.$tag 2>/dev/null

659                 if ! mv -f $tmpfile etc/rc.d/rc.sysinit; then
660                         log "mv of \"$tmpfile\" to" \
661                             "\"$installroot/etc/rc.d/rc.sysinit\" failed!"
662                         i18n_echo "$cmd2_failed" "mv" "$tmpfile" \
663                             "$installroot/etc/rc.d/rc.sysinit"
664                         i18n_echo "$install_aborted"
665                         exit 1
666                 else
667                         chmod 755 etc/rc.d/rc.sysinit
668                 fi
669         else
670                 log "Attempt to modify entries in" \
671                     "\"$install_root/rc.d/rc.sysinit\" failed!"
672                 i18n_echo "$mod_failed" "$install_root/rc.d/rc.sysinit"
673                 i18n_echo "$install_aborted"
674                 exit 1
675         fi
676 fi

678 if [[ -z $is_miniroot ]]; then
679         enable_nfs_services || log "NFS services were not properly enabled."
680 fi

682 log ""
683 log "System configuration modifications complete ‘date‘"
684 log ""
685 i18n_echo "System configuration modifications complete."
686 exit 0
687 #endif /* ! codereview */
```

```
**********************************************************
   15415 Tue Jan 14 16:17:16 2014
new/usr/src/lib/brand/lx/zone/lx_install.ksh
Bring back LX zones.
**********************************************************
```

```
#!/bin/ksh -p
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

# Restrict executables to /bin, /usr/bin, /usr/sbin and /usr/sfw/bin
PATH=/bin:/usr/bin:/usr/sbin:/usr/sfw/bin

export PATH

# Setup i18n output
TEXTDOMAIN="SUNW_OST_OSCMD"
export TEXTDOMAIN

# Log passed arguments to file descriptor 2
log()
{
        [[ -n $logfile ]] && echo "$@" >&2
}

#
# Send the provided printf()-style arguments to the screen and to the
# logfile.
#
screenlog()
{
        typeset fmt="$1"
        shift

        printf "$fmt\n" "$@"
        [[ -n $logfile ]] && printf "$fmt\n" "$@" >&2
}

# Print and log provided text if the shell variable "verbose_mode" is set
verbose()
{
        [[ -n $verbose_mode ]] && echo "$@"
        [[ -n $logfile ]] && [[ -n $verbose_mode ]] && echo "$@" >&2
}
```



```
unsupported_cpu=\
$(gettext "ERROR: Cannot install branded zone: processor must be %s-compatible")

cmd_not_found=$(gettext "Required command '%s' cannot be found!")
cmd_not_exec=$(gettext "Required command '%s' not executable!")
zone_initfail=$(gettext "Attempt to initialize zone '%s' FAILED.")
path_abs=$(gettext "Pathname specified to -d '%s' must be absolute.")

cmd_h=$(gettext "%s -z <zone name> %s -h")
cmd_full=\
$(gettext "%s -z <zone name> %s [-v | -s] [-d <dir>|<device>] [<cluster> ... ]")

both_modes=$(gettext "%s: error: cannot select both silent and verbose modes")

not_found=$(gettext "%s: error: file or directory not found.")

wrong_type=\
$(gettext "%s: error: must be a gzip, bzip2, .Z or uncompressed tar archive.")

not_readable=$(gettext "Cannot read file '%s'")

no_install=$(gettext "Could not create install directory '%s'")
no_log=$(gettext "Could not create log directory '%s'")
no_logfile=$(gettext "Could not create log file '%s'")

root_full=$(gettext "Zonepath root %s exists and contains data; remove or move a

install_zone=$(gettext "Installing zone '%s' at root directory '%s'")
install_from=$(gettext "from archive '%s'")

install_fail=$(gettext "Installation of zone '%s' FAILED.")
see_log=$(gettext "See the log file:\n  '%s'\nfor details.")

install_abort=$(gettext "Installation of zone '%s' aborted.")
install_good=$(gettext "Installation of zone '%s' completed successfully.")

# Check if commands passed in exist and are executable.
check_cmd()
{
        for cmd in "$@"; do
                if [[ ! -f $cmd ]]; then
                        screenlog "$cmd_not_found" "$cmd"
                        screenlog "$install_abort" "$zonename"
                        exit $ZONE_SUBPROC_NOTCOMPLETE
                fi

                if [[ ! -x $cmd ]]; then
                        screenlog "$cmd_not_exec" "$cmd"
                        screenlog "$install_abort" "$zonename"
                        exit $ZONE_SUBPROC_NOTCOMPLETE
                fi
        done
}

# Post process as tarball-installed zone for use by BrandZ.
init_tarzone()
{
        typeset rootdir="$1"

        if ! $branddir/lx_init_zone "$rootdir"; then
                screenlog "$zone_initfail" "$zonename"
                return 1
        fi
}

# Clean up on interrupt
```

```
128 trap_cleanup()
129 {
130         msg=$(gettext "Installation cancelled due to interrupt.")

132         screenlog "$msg"
133         exit $int_code
134 }

136 #
137 # Output the usage message.
138 #
139 # This is done this way due to limitations in the way gettext strings are
140 # extracted from shell scripts and processed.  Use of this somewhat awkward
141 # syntax allows us to produce longer lines of text than otherwise would be
142 # possible without wrapping lines across more than one line of code.
143 #
144 usage()
145 {
146         int_code=$ZONE_SUBPROC_USAGE

148         echo $(gettext "Usage:")
149         printf "  $cmd_h\n" "zoneadm" "install"
150         printf "  $cmd_full\n" "zoneadm" "install"

152         echo

154         echo $(gettext "The installer will attempt to use the default system") \
155             $(gettext "removable disc device if <archive dir> is not") \
156             $(gettext "specified.") | fmt -80

158         echo

160         echo $(gettext "<cluster> specifies which package cluster you wish") \
161             $(gettext "to install.") | fmt -80

163         echo
164         echo $(gettext "The 'desktop' cluster will be installed by default.")
165         echo
166         echo $(gettext "The available clusters are:")
167         echo "    + core"
168         echo "    + server"
169         echo "    + desktop"
170         echo "    + development"
171         echo "    + all"
172         echo

174         echo $(gettext "Each cluster includes all of the clusters preceding") \
175             $(gettext "it, so the 'server' cluster includes the 'core'") \
176             $(gettext "cluster, the 'desktop' cluster includes the 'core'") \
177             $(gettext "and 'server' clusters, and so on.") | fmt -80

179         echo
180         echo $(gettext "Examples")
181         echo "========"

183         echo $(gettext "Example 1: Install a base Linux system from CDs or a") \
184             $(gettext "DVD using the system default removable disc device:") |
185             fmt -80

187         echo
188         echo "    # zoneadm -z myzone install"
189         echo

191         echo $(gettext "Example 2: Install the 'server' cluster from CDs or") \
192             $(gettext "a DVD via an alternative removable disc device:") |
193             fmt -80
```



```
195         echo
196         echo "    # zoneadm -z myzone install -d /cdrom/cdrom1 server"
197         echo

199         echo $(gettext "Example 3: Install the desktop Linux environment") \
200             $(gettext "from an ISO image made available as '/dev/lofi/1' by") \
201             $(gettext "use of lofiadm(1M):") | fmt -80

203         echo
204         echo "    # zoneadm -z myzone install -d /dev/lofi/1 desktop"
205         echo

207         echo $(gettext "Example 4: Install the entire Linux environment from") \
208             $(gettext "ISO images located in the directory") \
209             "'/export/centos_3.8/isos':" | fmt -80

211         echo
212         echo "    # zoneadm -z myzone install -d /export/centos_3.8/isos all"
213         echo

215         echo $(gettext "Example 5: Install from a compressed tar archive of") \
216             $(gettext "an existing Linux installation (a tar ball) with") \
217             $(gettext "verbose output regarding the progress of the") \
218             $(gettext "installation:") | fmt -80

220         echo
221         echo "    # zoneadm -z myzone install -v -d /tmp/linux_full.tar.gz"
222         echo

224         echo $(gettext "Example 6: Install from a compressed tar archive of") \
225             $(gettext "an existing Linux installation (a tar ball) with NO") \
226             $(gettext "output regarding the progress of the installation") \
227             $(gettext "(silent mode.)") | fmt -80

229         echo

231         echo $(gettext "NOTE: Silent mode is only recommended for use by") \
232             $(gettext "shell scripts and other non-interactive programs:") |
233             fmt -80

235         echo
236         echo "    # zoneadm -z myzone install -d /tmp/linux_full.tar.gz -s"
237         echo

239         exit $int_code
240 }

242 #
243 # The main body of the script starts here.
244 #
245 # This script should never be called directly by a user but rather should
246 # only be called by zoneadm to install a BrandZ Linux zone.
247 #

249 #
250 # Exit values used by the script, as #defined in <sys/zone.h>
251 #
252 #       ZONE_SUBPROC_OK
253 #       ===============
254 #       Installation was successful
255 #
256 #       ZONE_SUBPROC_USAGE
257 #       ==================
258 #       Improper arguments were passed, so print a usage message before exiting
259 #
```

```
#       ZONE_SUBPROC_NOTCOMPLETE
#       ========================
#       Installation did not complete, but another installation attempt can be
#       made without an uninstall
#
#       ZONE_SUBPROC_FATAL
#       ==================
#       Installation failed and an uninstall will be required before another
#       install can be attempted
#
ZONE_SUBPROC_OK=0
ZONE_SUBPROC_USAGE=253
ZONE_SUBPROC_NOTCOMPLETE=254
ZONE_SUBPROC_FATAL=255

#
# An unspecified exit or interrupt should exit with ZONE_SUBPROC_NOTCOMPLETE,
# meaning a user will not need to do an uninstall before attempting another
# install.
#
int_code=$ZONE_SUBPROC_NOTCOMPLETE

trap trap_cleanup INT

# If we weren't passed at least two arguments, exit now.
[[ $# -lt 2 ]] && usage

#
# This script is always started with a full path so we can extract the
# brand directory name here.
#
branddir=$(dirname "$0")
zonename="$1"
zoneroot="$2"

install_root="$zoneroot/root"
logdir="$install_root/var/log"

shift; shift    # remove zonename and zoneroot from arguments array

unset gtaropts
unset install_opts
unset install_src
unset msg
unset silent_mode
unset verbose_mode

while getopts "d:hsvX" opt
do
        case "$opt" in
                h)      usage;;
                s)      silent_mode=1;;
                v)      verbose_mode=1;;
                d)      install_src="$OPTARG" ;;
                X)      install_opts="$install_opts -x" ;;
                *)      usage;;
        esac
done
shift OPTIND-1

# Providing more than one passed argument generates a usage message
if [[ $# -gt 1 ]]; then
        msg=$(gettext "ERROR: Too many arguments provided:")

        screenlog "$msg"
        screenlog "  \"%s\"" "$@"
```



```
        screenlog ""
        usage
fi

# Validate any free-form arguments
if [[ $# -eq 1 && "$1" != "core" && "$1" != "server" && "$1" != "desktop" &&
    "$1" != "development" && "$1" != "all" ]]; then
        msg=$(gettext "ERROR: Unknown cluster name specified: %s")

        screenlog "$msg" "\"$1\""
        screenlog ""
        usage
fi

# The install can't be both verbose AND silent...
if [[ -n $silent_mode && -n $verbose_mode ]]; then
        screenlog "$both_modes" "zoneadm install"
        screenlog ""
        usage
fi

#
# Validate that we're running on a i686-compatible CPU; abort the zone
# installation now if we're not.
#
procinfo=$(LC_ALL=C psrinfo -vp | grep family)

#
# All x86 processors in CPUID families 6, 15, 16 or 17 should be
# i686-compatible, assuming third party processor vendors follow AMD and
# Intel's lead.
#
if [[ "$procinfo" != *" x86 "* ]] ||
    [[ "$procinfo" != *" family 6 "* && "$procinfo" != *" family 15 "* &&
    "$procinfo" != *" family 16 "* && "$procinfo" != *" family 17 "* ]] ; then
        screenlog "$unsupported_cpu" "i686"
        exit $int_code
fi

if [[ -n $install_src ]]; then
        #
        # Validate $install_src.
        #
        # If install_src is a directory, assume it contains ISO images to
        # install from, otherwise treat the argument as if it points to a tar
        # ball file.
        #
        if [[ "'echo $install_src | cut -c 1'" != "/" ]]; then
                screenlog "$path_abs" "$install_src"
                exit $int_code
        fi

        if [[ ! -a "$install_src" ]]; then
                screenlog "$not_found" "$install_src"
                screenlog "$install_abort" "$zonename"
                exit $int_code
        fi

        if [[ ! -r "$install_src" ]]; then
                screenlog "$not_readable" "$install_src"
                screenlog "$install_abort" "$zonename"
                exit $int_code
        fi

        #
        # If install_src is a block device, a directory, a possible device
```

```
392         # created via lofiadm(1M), or the directory used by a standard volume
393         # management daemon, pass it on to the secondary install script.
394         #
395         # Otherwise, validate the passed filename to prepare for a tar ball
396         # install.
397         #
398         if [[ ! -b "$install_src" && ! -d "$install_src" &&
399             "$install_src" != /dev/lofi/* && "$install_src" != /cdrom/* &&
400             "$install_src" != /media/* ]]; then
401                 if [[ ! -f "$install_src" ]]; then
402                         screenlog "$wrong_type" "$install_src"
403                         screenlog "$install_abort" "$zonename"
404                         exit $int_code
405                 fi

407                 filetype=‘{ LC_ALL=C file $install_src |
408                     awk ’{print $2}’ ; } 2>/dev/null‘

410                 if [[ "$filetype" = "gzip" ]]; then
411                         verbose "\"$install_src\": \"gzip\" archive"
412                         gtaropts="-xz"
413                 elif [[ "$filetype" = "bzip2" ]]; then
414                         verbose "\"$install_src\": \"bzip2\" archive"
415                         gtaropts="-xj"
416                 elif [[ "$filetype" = "compressed" ]]; then
417                         verbose "\"$install_src\": Lempel-Ziv" \
418                             "compressed (\".Z\") archive."
419                         gtaropts="-xZ"
420                 elif [[ "$filetype" = "USTAR" ]]; then
421                         verbose "\"$install_src\":" \
422                             "uncompressed (\"tar\") archive."
423                         gtaropts="-x"
424                 else
425                         screenlog "$wrong_type" "$install_src"
426                         screenlog "$install_abort" "$zonename"
427                         exit $int_code
428                 fi
429         fi
430 fi

432 #
433 # Start silent operation and pass the flag to prepare pass the flag to
434 # the ISO installer, if needed.
435 #
436 if [[ -n $silent_mode ]]
437 then
438         exec 1>/dev/null
439         install_opts="$install_opts -s"
440 fi

442 #
443 # If verbose mode was specified, pass the verbose flag to lx_distro_install
444 # for ISO or disc installations and to gtar for tarball-based installs.
445 #
446 if [[ -n $verbose_mode ]]
447 then
448         echo $(gettext "Verbose output mode enabled.")
449         install_opts="$install_opts -v"
450         [[ -n $gtaropts ]] && gtaropts="${gtaropts}v"
451 fi

453 [[ -n $gtaropts ]] && gtaropts="${gtaropts}f"

455 if [[ ! -d "$install_root" ]]
456 then
457         if ! mkdir -p "$install_root" 2>/dev/null; then
```



```
458                 screenlog "$no_install" "$install_root"
459                 exit $int_code
460         fi
461 fi

463 #
464 # Check for a non-empty root.
465 #
466 cnt=‘ls $install_root | wc -l‘
467 if [ $cnt -ne 0 ]; then
468         screenlog "$root_full" "$install_root"
469         exit $int_code
470 fi

472 if [[ ! -d "$logdir" ]]
473 then
474         if ! mkdir -p "$logdir" 2>/dev/null; then
475                 screenlog "$no_log" "$logdir"
476                 exit $int_code
477         fi
478 fi

480 logfile="${logdir}/$zonename.install.$$.log"

482 if ! > $logfile; then
483         screenlog "$no_logfile" "$logfile"
484         exit $int_code
485 fi

487 # Redirect stderr to the log file to automatically log any error messages
488 exec 2>>"$logfile"

490 #
491 # From here on out, an unspecified exit or interrupt should exit with
492 # ZONE_SUBPROC_FATAL, meaning a user will need to do an uninstall before
493 # attempting another install, as we’ve modified the directories we were going
494 # to install to in some way.
495 #
496 int_code=$ZONE_SUBPROC_FATAL

498 log "Installation started for zone \"$zonename\" ‘/usr/bin/date‘"

500 if [[ -n $gtaropts ]]; then
501         check_cmd /usr/sfw/bin/gtar $branddir/lx_init_zone

503         screenlog "$install_zone" "$zonename" "$zoneroot"
504         screenlog "$install_from" "$install_src"
505         echo
506         echo $(gettext "This process may take several minutes.")
507         echo

509         if ! ( cd "$install_root" && gtar "$gtaropts" "$install_src" ) ; then
510                 log "Error: extraction from tar archive failed."
511         else
512                 if ! [[ -d "${install_root}/bin" &&
513                     -d "${install_root}/sbin" ]]; then
514                         log "Error: improper or incomplete tar archive."
515                 else
516                         $branddir/lx_init_zone "$install_root" &&
517                             init_tarzone "$install_root"

519                         #
520                         # Emit the same code from here whether we’re
521                         # interrupted or exiting normally.
522                         #
523                         int_code=$?
```

```
524                 fi
525         fi

527         if [[ $int_code -eq ZONE_SUBPROC_OK ]]; then
528                 log "Tar install completed for zone ’$zonename’ ‘date‘."
529         else
530                 log "Tar install failed for zone \"$zonename\" ‘date‘."

532         fi
533 else
534         check_cmd $branddir/lx_distro_install

536         $branddir/lx_distro_install -z "$zonename" -r "$zoneroot" \
537             -d "$install_src" -l "$logfile" $install_opts "$@"

539         #
540         # Emit the same code from here whether we’re interrupted or exiting
541         # normally.
542         #
543         int_code=$?

545         [[ $int_code -eq $ZONE_SUBPROC_USAGE ]] && usage
546 fi

548 if [[ $int_code -ne $ZONE_SUBPROC_OK ]]; then
549         screenlog ""
550         screenlog "$install_fail" "$zonename"
551         screenlog ""

553         #
554         # Only make a reference to the log file if one will exist after
555         # zoneadm exits.
556         #
557         [[ $int_code -ne $ZONE_SUBPROC_NOTCOMPLETE ]] &&
558             screenlog "$see_log" "$logfile"

560         exit $int_code
561 fi

563 #
564 # After the install completes, we’ve likely moved a new copy of the logfile into
565 # place atop the logfile we WERE writing to, so if we don’t reopen the logfile
566 # here the shell will continue writing to the old logfile’s inode, meaning we
567 # would lose all log information from this point on.
568 #
569 exec 2>>"$logfile"

571 screenlog ""
572 screenlog "$install_good" "$zonename"
573 screenlog ""

575 echo $(gettext "Details saved to log file:")
576 echo "    \"$logfile\""
577 echo

579 exit $ZONE_SUBPROC_OK
580 #endif /* ! codereview */
```

```
**********************************************************
    3090 Tue Jan 14 16:17:16 2014
new/usr/src/lib/brand/lx/zone/platform.xml
Bring back LX zones.
**********************************************************
```

```
<?xml version="1.0"?>

<!--
 CDDL HEADER START

 The contents of this file are subject to the terms of the
 Common Development and Distribution License (the "License").
 You may not use this file except in compliance with the License.

 You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 or http://www.opensolaris.org/os/licensing.
 See the License for the specific language governing permissions
 and limitations under the License.

 When distributing Covered Code, include this CDDL HEADER in each
 file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 If applicable, add the following below this CDDL HEADER, with the
 fields enclosed by brackets "[]" replaced with your own identifying
 information: Portions Copyright [yyyy] [name of copyright owner]

 CDDL HEADER END

 Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 Use is subject to license terms.

 ident  "%Z%%M% %I%     %E% SMI"

 DO NOT EDIT THIS FILE.
-->

<!DOCTYPE platform PUBLIC "-//Sun Microsystems Inc//Zones Platform//EN"
    "file:///usr/share/lib/xml/dtd/zone_platform.dtd.1">

<platform name="lx" allow-exclusive-ip="false">
        <!-- Global filesystems to mount when booting the zone -->
        <global_mount special="/dev" directory="/native/dev" type="dev"
            opt="attrdir=%R/dev" />
        <global_mount special="/lib" directory="/native/lib"
            opt="ro" type="lofs" />
        <global_mount special="/usr/lib" directory="/native/usr/lib"
            opt="ro" type="lofs" />
        <global_mount special="/usr/lib/brand/lx/etc_default_nfs"
            directory="/native/etc/default/nfs" type="lofs" opt="ro" />
        <global_mount special="/usr/lib/brand/lx/etc_netconfig"
            directory="/native/etc/netconfig" type="lofs" opt="ro" />

        <!-- Local filesystems to mount when booting the zone -->
        <mount special="/native/dev" directory="/dev" type="lofs" />
        <mount special="proc" directory="/native/proc" type="proc" />
        <mount special="swap" directory="/native/etc/svc/volatile"
            type="tmpfs" />
        <mount special="swap" directory="/native/tmp" type="tmpfs" />

        <!-- Devices to create under /dev -->
        <device match="null" />
        <device match="pts/*" />
        <device match="random" />
        <device match="tcp" />
        <device match="tcp6" />
        <device match="tty" />
        <device match="udp" />
```



```
        <device match="udp6" />
        <device match="urandom" />
        <device match="zero" />

        <!-- Renamed devices to create under /dev -->
        <device match="brand/lx/ptmx" name="ptmx" />
        <device match="zcons/%z/zoneconsole" name="console" />

        <!-- Audio devices to create under /dev -->
        <device match="brand/lx/dsp" name="dsp" />
        <device match="brand/lx/mixer" name="mixer" />

        <!-- Symlinks to create under /dev -->
        <symlink source="fd" target="../proc/self/fd" />
        <symlink source="log" target="/var/run/syslog" />
        <symlink source="stderr" target="../proc/self/fd/2" />
        <symlink source="stdin" target="../proc/self/fd/0" />
        <symlink source="stdout" target="../proc/self/fd/1" />
        <symlink source="systty" target="console" />

        <!-- Create a mount point for for the /dev/initctl fifo -->
        <device match="null" name="initctl" />

</platform>
#endif /* ! codereview */
```

```
***********************************************************
    1154 Tue Jan 14 16:17:16 2014
new/usr/src/pkg/manifests/SUNWlx.mf
Bring back LX zones.
***********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
# Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
#

set name=pkg.fmri value=pkg:/SUNWlx@0.5.11,5.11-0.133
set name=pkg.renamed value=true
# Was renamed to system/zones/brand/lx, both now obsolete.

set name=pkg.fmri value=pkg:/SUNWlx@0.5.11,5.11-0.143
set name=pkg.obsolete value=true
set name=variant.arch value=i386
set name=variant.opensolaris.zone value=global value=nonglobal
depend fmri=pkg:/system/zones/brand/lx@0.5.11,5.11-0.133 type=require
#endif /* ! codereview */
```

```
**********************************************************
    5606 Tue Jan 14 16:17:16 2014
new/usr/src/pkg/manifests/system-zones-brand-lx.mf
Final fixups and bugfixes
LX zone support should now build and packages of relevance produced.
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
# Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
#

#
# This package will install successfully into any zone, global or
# non-global.  The files, directories, links, and hardlinks, however,
# will only be installed into the global zone.
#
<include global_zone_only_component>
set name=pkg.fmri value=pkg:/system/zones/brand/lx@$(PKGVERS)
set name=pkg.description value="Support for the 'lx' Brand"
set name=pkg.summary value="lx Brand"
set name=info.classification \
    value="org.opensolaris.category.2008:Applications/System Utilities"
set name=pkg.fmri value=pkg:/system/zones/brand/lx@0.5.11,5.11-0.143
set name=pkg.obsolete value=true
set name=variant.arch value=i386
dir path=etc group=sys
dir path=etc/zones group=sys
dir path=usr group=sys
dir path=usr/kernel group=sys
dir path=usr/kernel/brand group=sys
dir path=usr/kernel/brand/$(ARCH64) group=sys
dir path=usr/kernel/drv group=sys
dir path=usr/kernel/drv/$(ARCH64) group=sys
dir path=usr/kernel/dtrace group=sys
dir path=usr/kernel/dtrace/$(ARCH64) group=sys
dir path=usr/kernel/fs group=sys
dir path=usr/kernel/fs/$(ARCH64) group=sys
dir path=usr/kernel/strmod group=sys
dir path=usr/kernel/strmod/$(ARCH64) group=sys
dir path=usr/lib
dir path=usr/lib/brand
dir path=usr/lib/brand/lx
dir path=usr/lib/brand/lx/$(ARCH64)
```



```
dir path=usr/lib/brand/lx/distros
dir path=usr/lib/devfsadm group=sys
dir path=usr/lib/devfsadm/linkmod group=sys
driver name=lx_audio
driver name=lx_ptm perms="lx_ptmajor 0666 root sys"
driver name=lx_systrace perms="* 0644 root sys"
file path=etc/zones/SUNWlx.xml mode=0444
file path=etc/zones/SUNWlx26.xml mode=0444
file path=usr/kernel/brand/$(ARCH64)/lx_brand group=sys mode=0755
file path=usr/kernel/brand/lx_brand group=sys mode=0755
file path=usr/kernel/drv/$(ARCH64)/lx_audio group=sys
file path=usr/kernel/drv/$(ARCH64)/lx_ptm group=sys
file path=usr/kernel/drv/$(ARCH64)/lx_systrace group=sys
file path=usr/kernel/drv/lx_audio group=sys
file path=usr/kernel/drv/lx_audio.conf group=sys
file path=usr/kernel/drv/lx_ptm group=sys
file path=usr/kernel/drv/lx_ptm.conf group=sys
file path=usr/kernel/drv/lx_systrace group=sys
file path=usr/kernel/drv/lx_systrace.conf group=sys
file path=usr/kernel/fs/$(ARCH64)/lx_afs group=sys mode=0755
file path=usr/kernel/fs/$(ARCH64)/lx_proc group=sys mode=0755
file path=usr/kernel/fs/lx_afs group=sys mode=0755
file path=usr/kernel/fs/lx_proc group=sys mode=0755
file path=usr/kernel/strmod/$(ARCH64)/ldlinux group=sys mode=0755
file path=usr/kernel/strmod/ldlinux group=sys mode=0755
file path=usr/lib/brand/lx/$(ARCH64)/lx_librtld_db.so.1
file path=usr/lib/brand/lx/$(ARCH64)/lx_nametoaddr.so.1
file path=usr/lib/brand/lx/$(ARCH64)/lx_thunk.so.1
file path=usr/lib/brand/lx/config.xml mode=0444
file path=usr/lib/brand/lx/distros/centos35.distro mode=0444
file path=usr/lib/brand/lx/distros/centos36.distro mode=0444
file path=usr/lib/brand/lx/distros/centos37.distro mode=0444
file path=usr/lib/brand/lx/distros/centos38.distro mode=0444
file path=usr/lib/brand/lx/distros/rhel35.distro mode=0444
file path=usr/lib/brand/lx/distros/rhel36.distro mode=0444
file path=usr/lib/brand/lx/distros/rhel37.distro mode=0444
file path=usr/lib/brand/lx/distros/rhel38.distro mode=0444
file path=usr/lib/brand/lx/distros/rhel_centos_common mode=0444
file path=usr/lib/brand/lx/etc_default_nfs group=sys mode=0444
file path=usr/lib/brand/lx/etc_netconfig group=sys mode=0444
file path=usr/lib/brand/lx/lx_distro_install mode=0755
file path=usr/lib/brand/lx/lx_init_zone mode=0755
file path=usr/lib/brand/lx/lx_install mode=0755
file path=usr/lib/brand/lx/lx_librtld_db.so.1
file path=usr/lib/brand/lx/lx_lockd mode=0755
file path=usr/lib/brand/lx/lx_nametoaddr.so.1
file path=usr/lib/brand/lx/lx_native mode=0755
file path=usr/lib/brand/lx/lx_statd mode=0755
file path=usr/lib/brand/lx/lx_support mode=0755
file path=usr/lib/brand/lx/lx_thunk mode=0755
file path=usr/lib/brand/lx/lx_thunk.so.1
file path=usr/lib/brand/lx/platform.xml mode=0444
file path=usr/lib/devfsadm/linkmod/SUNW_lx_link_$(ARCH).so group=sys
file path=usr/lib/lx_brand.so.1
hardlink path=usr/kernel/dtrace/$(ARCH64)/lx_systrace \
    target=../../../kernel/drv/$(ARCH64)/lx_systrace
hardlink path=usr/kernel/dtrace/lx_systrace \
    target=../../kernel/drv/lx_systrace
legacy pkg=SUNWlxr arch=$(ARCH) category=system \
    desc="Support for the 'lx' Brand" \
    hotline="Please contact your local service provider" \
    name="lx Brand (Root)" vendor="Sun Microsystems, Inc." \
    version=11.11,REV=2009.11.11
legacy pkg=SUNWlxu arch=$(ARCH) category=system \
    desc="Support for the 'lx' Brand" \
    hotline="Please contact your local service provider" \
```

```
123     name="lx Brand (Usr)" vendor="Sun Microsystems, Inc." \
124     version=11.11,REV=2009.11.11
125 license cr_Sun license=cr_Sun
126 license lic_CDDL license=lic_CDDL
127 link path=usr/lib/brand/lx/64 target=$(ARCH64)
128 #endif /* ! codereview */
```

```
**********************************************************
   44221 Tue Jan 14 16:17:17 2014
new/usr/src/uts/common/Makefile.files
Bring back LX zones.
**********************************************************
```

```
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #

  22 #
  23 # Copyright (c) 1991, 2010, Oracle and/or its affiliates. All rights reserved.
  24 # Copyright (c) 2012 Nexenta Systems, Inc. All rights reserved.
  25 # Copyright (c) 2013 by Delphix. All rights reserved.
  26 # Copyright (c) 2013 by Saso Kiselkov. All rights reserved.
  27 #

  29 #
  30 # This Makefile defines all file modules for the directory uts/common
  31 # and its children. These are the source files which may be considered
  32 # common to all SunOS systems.

  34 i386_CORE_OBJS += \
  35                 atomic.o        \
  36                 avintr.o        \
  37                 pic.o

  39 sparc_CORE_OBJS +=

  41 COMMON_CORE_OBJS +=             \
  42                 beep.o          \
  43                 bitset.o        \
  44                 bp_map.o        \
  45                 brand.o         \
  46                 cpucaps.o       \
  47                 cmt.o           \
  48                 cmt_policy.o    \
  49                 cpu.o           \
  50                 cpu_event.o     \
  51                 cpu_intr.o      \
  52                 cpu_pm.o        \
  53                 cpupart.o       \
  54                 cap_util.o      \
  55                 disp.o          \
  56                 group.o         \
  57                 kstat_fr.o      \
  58                 iscsiboot_prop.o        \
  59                 lgrp.o          \
  60                 lgrp_topo.o     \
  61                 mmapobj.o       \
```



```
  62                 mutex.o         \
  63                 page_lock.o     \
  64                 page_retire.o   \
  65                 panic.o         \
  66                 param.o         \
  67                 pg.o            \
  68                 pghw.o          \
  69                 putnext.o       \
  70                 rctl_proc.o     \
  71                 rwlock.o        \
  72                 seg_kmem.o      \
  73                 softint.o       \
  74                 string.o        \
  75                 strtol.o        \
  76                 strtoul.o       \
  77                 strtoll.o       \
  78                 strtoull.o      \
  79                 thread_intr.o   \
  80                 vm_page.o       \
  81                 vm_pagelist.o   \
  82                 zlib_obj.o      \
  83                 clock_tick.o

  85 CORE_OBJS +=    $(COMMON_CORE_OBJS) $($(MACH)_CORE_OBJS)

  87 ZLIB_OBJS =     zutil.o zmod.o zmod_subr.o \
  88                 adler32.o crc32.o deflate.o inffast.o \
  89                 inflate.o inftrees.o trees.o

  91 GENUNIX_OBJS += \
  92                 access.o        \
  93                 acl.o           \
  94                 acl_common.o    \
  95                 adjtime.o       \
  96                 alarm.o         \
  97                 aio_subr.o      \
  98                 auditsys.o      \
  99                 audit_core.o    \
 100                 audit_zone.o    \
 101                 audit_memory.o  \
 102                 autoconf.o      \
 103                 avl.o           \
 104                 bdev_dsort.o    \
 105                 bio.o           \
 106                 bitmap.o        \
 107                 blabel.o        \
 108                 brandsys.o      \
 109                 bz2blocksort.o  \
 110                 bz2compress.o   \
 111                 bz2decompress.o \
 112                 bz2randtable.o  \
 113                 bz2bzlib.o      \
 114                 bz2crctable.o   \
 115                 bz2huffman.o    \
 116                 callb.o         \
 117                 callout.o       \
 118                 chdir.o         \
 119                 chmod.o         \
 120                 chown.o         \
 121                 cladm.o         \
 122                 class.o         \
 123                 clock.o         \
 124                 clock_highres.o \
 125                 clock_realtime.o\
 126                 close.o         \
 127                 compress.o      \
```

```
128                 condvar.o       \
129                 conf.o          \
130                 console.o       \
131                 contract.o      \
132                 copyops.o       \
133                 core.o          \
134                 corectl.o       \
135                 cred.o          \
136                 cs_stubs.o      \
137                 dacf.o          \
138                 dacf_clnt.o     \
139                 damap.o \
140                 cyclic.o        \
141                 ddi.o           \
142                 ddifm.o         \
143                 ddi_hp_impl.o   \
144                 ddi_hp_ndi.o    \
145                 ddi_intr.o      \
146                 ddi_intr_impl.o \
147                 ddi_intr_irm.o  \
148                 ddi_nodeid.o    \
149                 ddi_periodic.o  \
150                 devcfg.o        \
151                 devcache.o      \
152                 device.o        \
153                 devid.o         \
154                 devid_cache.o   \
155                 devid_scsi.o    \
156                 devid_smp.o     \
157                 devpolicy.o     \
158                 disp_lock.o     \
159                 dnlc.o          \
160                 driver.o        \
161                 dumpsubr.o      \
162                 driver_lyr.o    \
163                 dtrace_subr.o   \
164                 errorq.o        \
165                 etheraddr.o     \
166                 evchannels.o    \
167                 exacct.o        \
168                 exacct_core.o   \
169                 exec.o          \
170                 exit.o          \
171                 fbio.o          \
172                 fcntl.o         \
173                 fdbuffer.o      \
174                 fdsync.o        \
175                 fem.o           \
176                 ffs.o           \
177                 fio.o           \
178                 flock.o         \
179                 fm.o            \
180                 fork.o          \
181                 vpm.o           \
182                 fs_reparse.o    \
183                 fs_subr.o       \
184                 fsflush.o       \
185                 ftrace.o        \
186                 getcwd.o        \
187                 getdents.o      \
188                 getloadavg.o    \
189                 getpagesizes.o  \
190                 getpid.o        \
191                 gfs.o           \
192                 rusagesys.o     \
193                 gid.o           \
```



```
194                 groups.o        \
195                 grow.o          \
196                 hat_refmod.o    \
197                 id32.o          \
198                 id_space.o      \
199                 inet_ntop.o     \
200                 instance.o      \
201                 ioctl.o         \
202                 ip_cksum.o      \
203                 issetugid.o     \
204                 ippconf.o       \
205                 kcpc.o          \
206                 kdi.o           \
207                 kiconv.o        \
208                 klpd.o          \
209                 kmem.o          \
210                 ksyms_snapshot.o        \
211                 l_strplumb.o    \
212                 labelsys.o      \
213                 link.o          \
214                 list.o          \
215                 lockstat_subr.o \
216                 log_sysevent.o  \
217                 logsubr.o       \
218                 lookup.o        \
219                 lseek.o         \
220                 ltos.o          \
221                 lwp.o           \
222                 lwp_create.o    \
223                 lwp_info.o      \
224                 lwp_self.o      \
225                 lwp_sobj.o      \
226                 lwp_timer.o     \
227                 lwpsys.o        \
228                 main.o          \
229                 mmapobjsys.o    \
230                 memcntl.o       \
231                 memstr.o        \
232                 lgrpsys.o       \
233                 mkdir.o         \
234                 mknod.o         \
235                 mount.o         \
236                 move.o          \
237                 msacct.o        \
238                 multidata.o     \
239                 nbmlock.o       \
240                 ndifm.o         \
241                 nice.o          \
242                 netstack.o      \
243                 ntptime.o       \
244                 nvpair.o        \
245                 nvpair_alloc_system.o   \
246                 nvpair_alloc_fixed.o    \
247                 fnvpair.o       \
248                 octet.o         \
249                 open.o          \
250                 p_online.o      \
251                 pathconf.o      \
252                 pathname.o      \
253                 pause.o         \
254                 serializer.o    \
255                 pci_intr_lib.o  \
256                 pci_cap.o       \
257                 pcifm.o         \
258                 pgrp.o          \
259                 pgrpsys.o       \
```

```
260                 pid.o           \
261                 pkp_hash.o      \
262                 policy.o        \
263                 poll.o          \
264                 pool.o          \
265                 pool_pset.o     \
266                 port_subr.o     \
267                 ppriv.o         \
268                 printf.o        \
269                 priocntl.o      \
270                 priv.o          \
271                 priv_const.o    \
272                 proc.o          \
273                 procset.o       \
274                 processor_bind.o        \
275                 processor_info.o        \
276                 profil.o        \
277                 project.o       \
278                 qsort.o         \
279                 rctl.o          \
280                 rctlsys.o       \
281                 readlink.o      \
282                 refstr.o        \
283                 rename.o        \
284                 resolvepath.o   \
285                 retire_store.o  \
286                 process.o       \
287                 rlimit.o        \
288                 rmap.o          \
289                 rw.o            \
290                 rwstlock.o      \
291                 sad_conf.o      \
292                 sid.o           \
293                 sidsys.o        \
294                 sched.o         \
295                 schedctl.o      \
296                 sctp_crc32.o    \
297                 seg_dev.o       \
298                 seg_kp.o        \
299                 seg_kpm.o       \
300                 seg_map.o       \
301                 seg_vn.o        \
302                 seg_spt.o       \
303                 semaphore.o     \
304                 sendfile.o      \
305                 session.o       \
306                 share.o         \
307                 shuttle.o       \
308                 sig.o           \
309                 sigaction.o     \
310                 sigaltstack.o   \
311                 signotify.o     \
312                 sigpending.o    \
313                 sigprocmask.o   \
314                 sigqueue.o      \
315                 sigsendset.o    \
316                 sigsuspend.o    \
317                 sigtimedwait.o  \
318                 sleepq.o        \
319                 sock_conf.o     \
320                 space.o         \
321                 sscanf.o        \
322                 stat.o          \
323                 statfs.o        \
324                 statvfs.o       \
325                 stol.o          \
```



```
326                 str_conf.o      \
327                 strcalls.o      \
328                 stream.o        \
329                 streamio.o      \
330                 strext.o        \
331                 strsubr.o       \
332                 strsun.o        \
333                 subr.o          \
334                 sunddi.o        \
335                 sunmdi.o        \
336                 sunndi.o        \
337                 sunpci.o        \
338                 sunpm.o         \
339                 sundlpi.o       \
340                 suntpi.o        \
341                 swap_subr.o     \
342                 swap_vnops.o    \
343                 symlink.o       \
344                 sync.o          \
345                 sysclass.o      \
346                 sysconfig.o     \
347                 sysent.o        \
348                 sysfs.o         \
349                 systeminfo.o    \
350                 task.o          \
351                 taskq.o         \
352                 tasksys.o       \
353                 time.o          \
354                 timer.o         \
355                 times.o         \
356                 timers.o        \
357                 thread.o        \
358                 tlabel.o        \
359                 tnf_res.o       \
360                 turnstile.o     \
361                 tty_common.o    \
362                 u8_textprep.o   \
363                 uadmin.o        \
364                 uconv.o         \
365                 ucredsys.o      \
366                 uid.o           \
367                 umask.o         \
368                 umount.o        \
369                 uname.o         \
370                 unix_bb.o       \
371                 unlink.o        \
372                 urw.o           \
373                 utime.o         \
374                 utssys.o        \
375                 uucopy.o        \
376                 vfs.o           \
377                 vfs_conf.o      \
378                 vmem.o          \
379                 vm_anon.o       \
380                 vm_as.o         \
381                 vm_meter.o      \
382                 vm_pageout.o    \
383                 vm_pvn.o        \
384                 vm_rm.o         \
385                 vm_seg.o        \
386                 vm_subr.o       \
387                 vm_swap.o       \
388                 vm_usage.o      \
389                 vnode.o         \
390                 vuid_queue.o    \
391                 vuid_store.o    \
```

```
392                 waitq.o         \
393                 watchpoint.o    \
394                 yield.o         \
395                 scsi_confdata.o \
396                 xattr.o         \
397                 xattr_common.o  \
398                 xdr_mblk.o      \
399                 xdr_mem.o       \
400                 xdr.o           \
401                 xdr_array.o     \
402                 xdr_refer.o     \
403                 xhat.o          \
404                 zone.o

406 #
407 #       Stubs for the stand-alone linker/loader
408 #
409 sparc_GENSTUBS_OBJS =   \
410         kobj_stubs.o

412 i386_GENSTUBS_OBJS =

414 COMMON_GENSTUBS_OBJS =

416 GENSTUBS_OBJS += $(COMMON_GENSTUBS_OBJS) $($(MACH)_GENSTUBS_OBJS)

418 #
419 #       DTrace and DTrace Providers
420 #
421 DTRACE_OBJS += dtrace.o dtrace_isa.o dtrace_asm.o

423 SDT_OBJS += sdt_subr.o

425 PROFILE_OBJS += profile.o

427 SYSTRACE_OBJS += systrace.o

429 LX_SYSTRACE_OBJS += lx_systrace.o

431 #endif /* ! codereview */
432 LOCKSTAT_OBJS += lockstat.o

434 FASTTRAP_OBJS += fasttrap.o fasttrap_isa.o

436 DCPC_OBJS += dcpc.o

438 #
439 #       Driver (pseudo-driver) Modules
440 #
441 IPP_OBJS +=     ippctl.o

443 AUDIO_OBJS += audio_client.o audio_ddi.o audio_engine.o \
444             audio_fltdata.o audio_format.o audio_ctrl.o \
445             audio_grc3.o audio_output.o audio_input.o \
446             audio_oss.o audio_sun.o

448 AUDIOEMU10K_OBJS += audioemu10k.o

450 AUDIOENS_OBJS += audioens.o

452 AUDIOVIA823X_OBJS += audiovia823x.o

454 AUDIOVIA97_OBJS += audiovia97.o

456 AUDIO1575_OBJS += audio1575.o
```



```
458 AUDIO810_OBJS += audio810.o

460 AUDIOCMI_OBJS += audiocmi.o

462 AUDIOCMIHD_OBJS += audiocmihd.o

464 AUDIOHD_OBJS += audiohd.o

466 AUDIOIXP_OBJS += audioixp.o

468 AUDIOLS_OBJS += audiols.o

470 AUDIOP16X_OBJS += audiop16x.o

472 AUDIOPCI_OBJS += audiopci.o

474 AUDIOSOLO_OBJS += audiosolo.o

476 AUDIOTS_OBJS += audiots.o

478 AC97_OBJS += ac97.o ac97_ad.o ac97_alc.o ac97_cmi.o

480 BLKDEV_OBJS += blkdev.o

482 CARDBUS_OBJS += cardbus.o cardbus_hp.o cardbus_cfg.o

484 CONSKBD_OBJS += conskbd.o

486 CONSMS_OBJS +=  consms.o

488 OLDPTY_OBJS +=  tty_ptyconf.o

490 PTC_OBJS +=     tty_pty.o

492 PTSL_OBJS +=    tty_pts.o

494 PTM_OBJS +=     ptm.o

496 LX_PTM_OBJS +=  lx_ptm.o

498 LX_AUDIO_OBJS += lx_audio.o
499 #endif /* ! codereview */

501 MII_OBJS +=     mii.o mii_cicada.o mii_natsemi.o mii_intel.o mii_qualsemi.o \
502                 mii_marvell.o mii_realtek.o mii_other.o

504 PTS_OBJS +=     pts.o

506 PTY_OBJS +=     ptms_conf.o

508 SAD_OBJS +=     sad.o

510 MD4_OBJS +=     md4.o md4_mod.o

512 MD5_OBJS +=     md5.o md5_mod.o

514 SHA1_OBJS +=    sha1.o sha1_mod.o

516 SHA2_OBJS +=    sha2.o sha2_mod.o

518 IPGPC_OBJS +=   classifierddi.o classifier.o filters.o trie.o table.o \
519                 ba_table.o

521 DSCPMK_OBJS +=  dscpmk.o dscpmkddi.o

523 DLCOSMK_OBJS += dlcosmk.o dlcosmkddi.o
```

```
525 FLOWACCT_OBJS +=        flowacctddi.o flowacct.o

527 TOKENMT_OBJS += tokenmt.o tokenmtddi.o

529 TSWTCL_OBJS +=  tswtcl.o tswtclddi.o

531 ARP_OBJS +=     arpddi.o

533 ICMP_OBJS +=    icmpddi.o

535 ICMP6_OBJS +=   icmp6ddi.o

537 RTS_OBJS +=     rtsddi.o

539 IP_ICMP_OBJS =  icmp.o icmp_opt_data.o
540 IP_RTS_OBJS =   rts.o rts_opt_data.o
541 IP_TCP_OBJS =   tcp.o tcp_fusion.o tcp_opt_data.o tcp_sack.o tcp_stats.o \
542                 tcp_misc.o tcp_timers.o tcp_time_wait.o tcp_tpi.o tcp_output.o \
543                 tcp_input.o tcp_socket.o tcp_bind.o tcp_cluster.o tcp_tunables.o
544 IP_UDP_OBJS =   udp.o udp_opt_data.o udp_tunables.o udp_stats.o
545 IP_SCTP_OBJS =  sctp.o sctp_opt_data.o sctp_output.o \
546                 sctp_init.o sctp_input.o sctp_cookie.o \
547                 sctp_conn.o sctp_error.o sctp_snmp.o \
548                 sctp_tunables.o sctp_shutdown.o sctp_common.o \
549                 sctp_timer.o sctp_heartbeat.o sctp_hash.o \
550                 sctp_bind.o sctp_notify.o sctp_asconf.o \
551                 sctp_addr.o tn_ipopt.o tnet.o ip_netinfo.o \
552                 sctp_misc.o
553 IP_ILB_OBJS =   ilb.o ilb_nat.o ilb_conn.o ilb_alg_hash.o ilb_alg_rr.o

555 IP_OBJS +=      igmp.o ipmp.o ip.o ip6.o ip6_asp.o ip6_if.o ip6_ire.o \
556                 ip6_rts.o ip_if.o ip_ire.o ip_listutils.o ip_mroute.o \
557                 ip_multi.o ip2mac.o ip_ndp.o ip_rts.o ip_srcid.o \
558                 ipddi.o ipdrop.o mi.o nd.o tunables.o optcom.o snmpcom.o \
559                 ipsec_loader.o spd.o ipclassifier.o inet_common.o ip_squeue.o \
560                 squeue.o ip_sadb.o ip_ftable.o proto_set.o radix.o ip_dummy.o \
561                 ip_helper_stream.o ip_tunables.o \
562                 ip_output.o ip_input.o ip6_input.o ip6_output.o ip_arp.o \
563                 conn_opt.o ip_attr.o ip_dce.o \
564                 $(IP_ICMP_OBJS) \
565                 $(IP_RTS_OBJS) \
566                 $(IP_TCP_OBJS) \
567                 $(IP_UDP_OBJS) \
568                 $(IP_SCTP_OBJS) \
569                 $(IP_ILB_OBJS)

571 IP6_OBJS +=     ip6ddi.o

573 HOOK_OBJS +=    hook.o

575 NETI_OBJS +=    neti_impl.o neti_mod.o neti_stack.o

577 KEYSOCK_OBJS += keysockddi.o keysock.o keysock_opt_data.o

579 IPNET_OBJS +=   ipnet.o ipnet_bpf.o

581 SPDSOCK_OBJS += spdsockddi.o spdsock.o spdsock_opt_data.o

583 IPSECESP_OBJS += ipsecespddi.o ipsecesp.o

585 IPSECAH_OBJS += ipsecahddi.o ipsecah.o sadb.o

587 SPPP_OBJS +=    sppp.o sppp_dlpi.o sppp_mod.o s_common.o

589 SPPPTUN_OBJS += sppptun.o sppptun_mod.o
```



```
591 SPPPASYN_OBJS += spppasyn.o spppasyn_mod.o

593 SPPPCOMP_OBJS += spppcomp.o spppcomp_mod.o deflate.o bsd-comp.o vjcompress.o \
594                 zlib.o

596 TCP_OBJS +=     tcpddi.o

598 TCP6_OBJS +=    tcp6ddi.o

600 NCA_OBJS +=     ncaddi.o

602 SDP_SOCK_MOD_OBJS += sockmod_sdp.o socksdp.o socksdpsubr.o

604 SCTP_SOCK_MOD_OBJS += sockmod_sctp.o socksctp.o socksctpsubr.o

606 PFP_SOCK_MOD_OBJS += sockmod_pfp.o

608 RDS_SOCK_MOD_OBJS += sockmod_rds.o

610 RDS_OBJS +=     rdsddi.o rdssubr.o rds_opt.o rds_ioctl.o

612 RDSIB_OBJS +=   rdsib.o rdsib_ib.o rdsib_cm.o rdsib_ep.o rdsib_buf.o \
613                 rdsib_debug.o rdsib_sc.o

615 RDSV3_OBJS +=   af_rds.o rdsv3_ddi.o bind.o loop.o threads.o connection.o \
616                 transport.o cong.o sysctl.o message.o rds_recv.o send.o \
617                 stats.o info.o page.o rdma_transport.o ib_ring.o ib_rdma.o \
618                 ib_recv.o ib.o ib_send.o ib_sysctl.o ib_stats.o ib_cm.o \
619                 rdsv3_sc.o rdsv3_debug.o rdsv3_impl.o rdma.o rdsv3_af_thr.o

621 ISER_OBJS +=    iser.o iser_cm.o iser_cq.o iser_ib.o iser_idm.o \
622                 iser_resource.o iser_xfer.o

624 UDP_OBJS +=     udpddi.o

626 UDP6_OBJS +=    udp6ddi.o

628 SY_OBJS +=      gentty.o

630 TCO_OBJS +=     ticots.o

632 TCOO_OBJS +=    ticotsord.o

634 TCL_OBJS +=     ticlts.o

636 TL_OBJS +=      tl.o

638 DUMP_OBJS +=    dump.o

640 BPF_OBJS +=     bpf.o bpf_filter.o bpf_mod.o bpf_dlt.o bpf_mac.o

642 CLONE_OBJS +=   clone.o

644 CN_OBJS +=      cons.o

646 DLD_OBJS +=     dld_drv.o dld_proto.o dld_str.o dld_flow.o

648 DLS_OBJS +=     dls.o dls_link.o dls_mod.o dls_stat.o dls_mgmt.o

650 GLD_OBJS +=     gld.o gldutil.o

652 MAC_OBJS +=     mac.o  mac_bcast.o mac_client.o mac_datapath_setup.o mac_flow.o
653                 mac_hio.o mac_mod.o mac_ndd.o mac_provider.o mac_sched.o \
654                 mac_protect.o mac_soft_ring.o mac_stat.o mac_util.o
```

```
MAC_6TO4_OBJS +=        mac_6to4.o

MAC_ETHER_OBJS +=       mac_ether.o

MAC_IPV4_OBJS +=        mac_ipv4.o

MAC_IPV6_OBJS +=        mac_ipv6.o

MAC_WIFI_OBJS +=        mac_wifi.o

MAC_IB_OBJS +=          mac_ib.o

IPTUN_OBJS +=   iptun_dev.o iptun_ctl.o iptun.o

AGGR_OBJS +=    aggr_dev.o aggr_ctl.o aggr_grp.o aggr_port.o \
                aggr_send.o aggr_recv.o aggr_lacp.o

SOFTMAC_OBJS += softmac_main.o softmac_ctl.o softmac_capab.o \
                softmac_dev.o softmac_stat.o softmac_pkt.o softmac_fp.o

NET80211_OBJS += net80211.o net80211_proto.o net80211_input.o \
                 net80211_output.o net80211_node.o net80211_crypto.o \
                 net80211_crypto_none.o net80211_crypto_wep.o net80211_ioctl.o \
                 net80211_crypto_tkip.o net80211_crypto_ccmp.o  \
                 net80211_ht.o

VNIC_OBJS +=    vnic_ctl.o vnic_dev.o

SIMNET_OBJS +=  simnet.o

IB_OBJS +=      ibnex.o ibnex_ioctl.o ibnex_hca.o

IBCM_OBJS +=    ibcm_impl.o ibcm_sm.o ibcm_ti.o ibcm_utils.o ibcm_path.o \
                ibcm_arp.o ibcm_arp_link.o

IBDM_OBJS +=    ibdm.o

IBDMA_OBJS +=   ibdma.o

IBMF_OBJS +=    ibmf.o ibmf_impl.o ibmf_dr.o ibmf_wqe.o ibmf_ud_dest.o ibmf_mod.
                ibmf_send.o ibmf_recv.o ibmf_handlers.o ibmf_trans.o \
                ibmf_timers.o ibmf_msg.o ibmf_utils.o ibmf_rmpp.o \
                ibmf_saa.o ibmf_saa_impl.o ibmf_saa_utils.o ibmf_saa_events.o

IBTL_OBJS +=    ibtl_impl.o ibtl_util.o ibtl_mem.o ibtl_handlers.o ibtl_qp.o \
                ibtl_cq.o ibtl_wr.o ibtl_hca.o ibtl_chan.o ibtl_cm.o \
                ibtl_mcg.o ibtl_ibnex.o ibtl_srq.o ibtl_part.o

TAVOR_OBJS +=   tavor.o tavor_agents.o tavor_cfg.o tavor_ci.o tavor_cmd.o \
                tavor_cq.o tavor_event.o tavor_ioctl.o tavor_misc.o \
                tavor_mr.o tavor_qp.o tavor_qpmod.o tavor_rsrc.o \
                tavor_srq.o tavor_stats.o tavor_umap.o tavor_wr.o

HERMON_OBJS +=  hermon.o hermon_agents.o hermon_cfg.o hermon_ci.o hermon_cmd.o \
                hermon_cq.o hermon_event.o hermon_ioctl.o hermon_misc.o \
                hermon_mr.o hermon_qp.o hermon_qpmod.o hermon_rsrc.o \
                hermon_srq.o hermon_stats.o hermon_umap.o hermon_wr.o \
                hermon_fcoib.o hermon_fm.o

DAPLT_OBJS +=   daplt.o

SOL_OFS_OBJS += sol_cma.o sol_ib_cma.o sol_uobj.o \
                sol_ofs_debug_util.o sol_ofs_gen_util.o \
                sol_kverbs.o

SOL_UCMA_OBJS +=        sol_ucma.o
```



```
SOL_UVERBS_OBJS +=      sol_uverbs.o sol_uverbs_comp.o sol_uverbs_event.o \
                        sol_uverbs_hca.o sol_uverbs_qp.o

SOL_UMAD_OBJS += sol_umad.o

KSTAT_OBJS +=   kstat.o

KSYMS_OBJS +=   ksyms.o

INSTANCE_OBJS += inst_sync.o

IWSCN_OBJS +=   iwscons.o

LOFI_OBJS +=    lofi.o LzmaDec.o

FSSNAP_OBJS +=  fssnap.o

FSSNAPIF_OBJS += fssnap_if.o

MM_OBJS +=      mem.o

PHYSMEM_OBJS += physmem.o

OPTIONS_OBJS += options.o

WINLOCK_OBJS += winlockio.o

PM_OBJS +=      pm.o
SRN_OBJS +=     srn.o

PSEUDO_OBJS +=  pseudonex.o

RAMDISK_OBJS += ramdisk.o

LLC1_OBJS += llc1.o

USBKBM_OBJS += usbkbm.o

USBWCM_OBJS += usbwcm.o

BOFI_OBJS += bofi.o

HID_OBJS += hid.o

HWA_RC_OBJS += hwarc.o

USBSKEL_OBJS += usbskel.o

USBVC_OBJS += usbvc.o usbvc_v4l2.o

HIDPARSER_OBJS += hidparser.o

USB_AC_OBJS += usb_ac.o

USB_AS_OBJS += usb_as.o

USB_AH_OBJS += usb_ah.o

USBMS_OBJS += usbms.o

USBPRN_OBJS += usbprn.o

UGEN_OBJS += ugen.o

USBSER_OBJS += usbser.o usbser_rseq.o
```

```
789 USBSACM_OBJS += usbsacm.o

791 USBSER_KEYSPAN_OBJS += usbser_keyspan.o keyspan_dsd.o keyspan_pipe.o

793 USBS49_FW_OBJS += keyspan_49fw.o

795 USBSPRL_OBJS += usbser_pl2303.o pl2303_dsd.o

797 WUSB_CA_OBJS += wusb_ca.o

799 USBFTDI_OBJS += usbser_uftdi.o uftdi_dsd.o

801 USBECM_OBJS += usbecm.o

803 WC_OBJS += wscons.o vcons.o

805 VCONS_CONF_OBJS += vcons_conf.o

807 SCSI_OBJS +=    scsi_capabilities.o scsi_confsubr.o scsi_control.o \
808                 scsi_data.o scsi_fm.o scsi_hba.o scsi_reset_notify.o \
809                 scsi_resource.o scsi_subr.o scsi_transport.o scsi_watch.o \
810                 smp_transport.o

812 SCSI_VHCI_OBJS +=               scsi_vhci.o mpapi_impl.o scsi_vhci_tpgs.o

814 SCSI_VHCI_F_SYM_OBJS +=         sym.o

816 SCSI_VHCI_F_TPGS_OBJS +=        tpgs.o

818 SCSI_VHCI_F_ASYM_SUN_OBJS +=    asym_sun.o

820 SCSI_VHCI_F_SYM_HDS_OBJS +=     sym_hds.o

822 SCSI_VHCI_F_TAPE_OBJS +=        tape.o

824 SCSI_VHCI_F_TPGS_TAPE_OBJS +=   tpgs_tape.o

826 SGEN_OBJS +=    sgen.o

828 SMP_OBJS +=     smp.o

830 SATA_OBJS +=    sata.o

832 USBA_OBJS +=    hcdi.o  usba.o  usbai.o hubdi.o parser.o genconsole.o \
833                 usbai_pipe_mgmt.o usbai_req.o usbai_util.o usbai_register.o \
834                 usba_devdb.o usba10_calls.o usba_ugen.o whcdi.o wa.o
835 USBA_WITHOUT_WUSB_OBJS +=       hcdi.o  usba.o  usbai.o hubdi.o parser.o gencons
836                 usbai_pipe_mgmt.o usbai_req.o usbai_util.o usbai_register.o \
837                 usba_devdb.o usba10_calls.o usba_ugen.o

839 USBA10_OBJS +=  usba10.o

841 RSM_OBJS +=     rsm.o   rsmka_pathmanager.o     rsmka_util.o

843 RSMOPS_OBJS +=  rsmops.o

845 S1394_OBJS +=   t1394.o t1394_errmsg.o s1394.o s1394_addr.o s1394_asynch.o \
846                 s1394_bus_reset.o s1394_cmp.o s1394_csr.o s1394_dev_disc.o \
847                 s1394_fa.o s1394_fcp.o \
848                 s1394_hotplug.o s1394_isoch.o s1394_misc.o h1394.o nx1394.o

850 HCI1394_OBJS += hci1394.o hci1394_async.o hci1394_attach.o hci1394_buf.o \
851                 hci1394_csr.o hci1394_detach.o hci1394_extern.o \
852                 hci1394_ioctl.o hci1394_isoch.o hci1394_isr.o \
853                 hci1394_ixl_comp.o hci1394_ixl_isr.o hci1394_ixl_misc.o \
```



```
854                 hci1394_ixl_update.o hci1394_misc.o hci1394_ohci.o \
855                 hci1394_q.o hci1394_s1394if.o hci1394_tlabel.o \
856                 hci1394_tlist.o hci1394_vendor.o

858 AV1394_OBJS +=  av1394.o av1394_as.o av1394_async.o av1394_cfgrom.o \
859                 av1394_cmp.o av1394_fcp.o av1394_isoch.o av1394_isoch_chan.o \
860                 av1394_isoch_recv.o av1394_isoch_xmit.o av1394_list.o \
861                 av1394_queue.o

863 DCAM1394_OBJS += dcam.o dcam_frame.o dcam_param.o dcam_reg.o \
864                 dcam_ring_buff.o

866 SCSA1394_OBJS += hba.o sbp2_driver.o sbp2_bus.o

868 SBP2_OBJS += cfgrom.o sbp2.o

870 PMODEM_OBJS += pmodem.o pmodem_cis.o cis.o cis_callout.o cis_handlers.o cis_para

872 DSW_OBJS +=     dsw.o dsw_dev.o ii_tree.o

874 NCALL_OBJS +=   ncall.o \
875                 ncall_stub.o

877 RDC_OBJS +=     rdc.o \
878                 rdc_dev.o \
879                 rdc_io.o \
880                 rdc_clnt.o \
881                 rdc_prot_xdr.o \
882                 rdc_svc.o \
883                 rdc_bitmap.o \
884                 rdc_health.o \
885                 rdc_subr.o \
886                 rdc_diskq.o

888 RDCSRV_OBJS +=  rdcsrv.o

890 RDCSTUB_OBJS += rdc_stub.o

892 SDBC_OBJS +=    sd_bcache.o \
893                 sd_bio.o \
894                 sd_conf.o \
895                 sd_ft.o \
896                 sd_hash.o \
897                 sd_io.o \
898                 sd_misc.o \
899                 sd_pcu.o \
900                 sd_tdaemon.o \
901                 sd_trace.o \
902                 sd_iob_impl0.o \
903                 sd_iob_impl1.o \
904                 sd_iob_impl2.o \
905                 sd_iob_impl3.o \
906                 sd_iob_impl4.o \
907                 sd_iob_impl5.o \
908                 sd_iob_impl6.o \
909                 sd_iob_impl7.o \
910                 safestore.o \
911                 safestore_ram.o

913 NSCTL_OBJS +=   nsctl.o \
914                 nsc_cache.o \
915                 nsc_disk.o \
916                 nsc_dev.o \
917                 nsc_freeze.o \
918                 nsc_gen.o \
919                 nsc_mem.o \
```

```
920                 nsc_ncallio.o \
921                 nsc_power.o \
922                 nsc_resv.o \
923                 nsc_rmspin.o \
924                 nsc_solaris.o \
925                 nsc_trap.o \
926                 nsc_list.o
927 UNISTAT_OBJS += spuni.o \
928                 spcs_s_k.o

930 NSKERN_OBJS +=  nsc_ddi.o \
931                 nsc_proc.o \
932                 nsc_raw.o \
933                 nsc_thread.o \
934                 nskernd.o

936 SV_OBJS +=      sv.o

938 PMCS_OBJS += pmcs_attach.o pmcs_ds.o pmcs_intr.o pmcs_nvram.o pmcs_sata.o \
939                 pmcs_scsa.o pmcs_smhba.o pmcs_subr.o pmcs_fwlog.o

941 PMCS8001FW_C_OBJS +=    pmcs_fw_hdr.o
942 PMCS8001FW_OBJS +=              $(PMCS8001FW_C_OBJS) SPCBoot.o ila.o firmware.o

944 #
945 #       Build up defines and paths.

947 ST_OBJS +=      st.o    st_conf.o

949 EMLXS_OBJS +=   emlxs_clock.o emlxs_dfc.o emlxs_dhchap.o emlxs_diag.o \
950                 emlxs_download.o emlxs_dump.o emlxs_els.o emlxs_event.o \
951                 emlxs_fcf.o emlxs_fcp.o emlxs_fct.o emlxs_hba.o emlxs_ip.o \
952                 emlxs_mbox.o emlxs_mem.o emlxs_msg.o emlxs_node.o \
953                 emlxs_pkt.o emlxs_sli3.o emlxs_sli4.o emlxs_solaris.o \
954                 emlxs_thread.o

956 EMLXS_FW_OBJS +=        emlxs_fw.o

958 OCE_OBJS +=     oce_buf.o oce_fm.o oce_gld.o oce_hw.o oce_intr.o oce_main.o \
959                 oce_mbx.o oce_mq.o oce_queue.o oce_rx.o oce_stat.o oce_tx.o \
960                 oce_utils.o

962 FCT_OBJS += discovery.o fct.o

964 QLT_OBJS += 2400.o 2500.o 8100.o qlt.o qlt_dma.o

966 SRPT_OBJS += srpt_mod.o srpt_ch.o srpt_cm.o srpt_ioc.o srpt_stp.o

968 FCOE_OBJS += fcoe.o fcoe_eth.o fcoe_fc.o

970 FCOET_OBJS += fcoet.o fcoet_eth.o fcoet_fc.o

972 FCOEI_OBJS += fcoei.o fcoei_eth.o fcoei_lv.o

974 ISCSIT_SHARED_OBJS += \
975                 iscsit_common.o

977 ISCSIT_OBJS +=  $(ISCSIT_SHARED_OBJS) \
978                 iscsit.o iscsit_tgt.o iscsit_sess.o iscsit_login.o \
979                 iscsit_text.o iscsit_isns.o iscsit_radiusauth.o \
980                 iscsit_radiuspacket.o iscsit_auth.o iscsit_authclient.o

982 PPPT_OBJS +=    alua_ic_if.o pppt.o pppt_msg.o pppt_tgt.o

984 STMF_OBJS += lun_map.o stmf.o
```



```
986 STMF_SBD_OBJS += sbd.o sbd_scsi.o sbd_pgr.o sbd_zvol.o

988 SYSMSG_OBJS +=  sysmsg.o

990 SES_OBJS +=     ses.o ses_sen.o ses_safte.o ses_ses.o

992 TNF_OBJS +=     tnf_buf.o       tnf_trace.o     tnf_writer.o    trace_init.o \
993                 trace_funcs.o   tnf_probe.o     tnf.o

995 LOGINDMUX_OBJS += logindmux.o

997 DEVINFO_OBJS += devinfo.o

999 DEVPOLL_OBJS += devpoll.o

1001 DEVPOOL_OBJS += devpool.o

1003 I8042_OBJS +=   i8042.o

1005 KB8042_OBJS +=  \
1006                 at_keyprocess.o \
1007                 kb8042.o        \
1008                 kb8042_keytables.o

1010 MOUSE8042_OBJS += mouse8042.o

1012 FDC_OBJS +=     fdc.o

1014 ASY_OBJS +=     asy.o

1016 ECPP_OBJS +=    ecpp.o

1018 VUIDM3P_OBJS += vuidmice.o vuidm3p.o

1020 VUIDM4P_OBJS += vuidmice.o vuidm4p.o

1022 VUIDM5P_OBJS += vuidmice.o vuidm5p.o

1024 VUIDPS2_OBJS += vuidmice.o vuidps2.o

1026 HPCSVC_OBJS += hpcsvc.o

1028 PCIE_MISC_OBJS += pcie.o pcie_fault.o pcie_hp.o pciehpc.o pcishpc.o pcie_pwr.o p

1030 PCIHPNEXUS_OBJS += pcihp.o

1032 OPENEEPR_OBJS += openprom.o

1034 RANDOM_OBJS += random.o

1036 PSHOT_OBJS += pshot.o

1038 GEN_DRV_OBJS += gen_drv.o

1040 TCLIENT_OBJS += tclient.o

1042 TPHCI_OBJS += tphci.o

1044 TVHCI_OBJS += tvhci.o

1046 EMUL64_OBJS += emul64.o emul64_bsd.o

1048 FCP_OBJS += fcp.o

1050 FCIP_OBJS += fcip.o
```

```
1052 FCSM_OBJS += fcsm.o

1054 FCTL_OBJS += fctl.o

1056 FP_OBJS += fp.o

1058 QLC_OBJS += ql_api.o ql_debug.o ql_hba_fru.o ql_init.o ql_iocb.o ql_ioctl.o \
1059         ql_isr.o ql_mbx.o ql_nx.o ql_xioctl.o ql_fw_table.o

1061 QLC_FW_2200_OBJS += ql_fw_2200.o

1063 QLC_FW_2300_OBJS += ql_fw_2300.o

1065 QLC_FW_2400_OBJS += ql_fw_2400.o

1067 QLC_FW_2500_OBJS += ql_fw_2500.o

1069 QLC_FW_6322_OBJS += ql_fw_6322.o

1071 QLC_FW_8100_OBJS += ql_fw_8100.o

1073 QLGE_OBJS += qlge.o qlge_dbg.o qlge_flash.o qlge_fm.o qlge_gld.o qlge_mpi.o

1075 ZCONS_OBJS += zcons.o

1077 NV_SATA_OBJS += nv_sata.o

1079 SI3124_OBJS += si3124.o

1081 AHCI_OBJS += ahci.o

1083 PCIIDE_OBJS += pci-ide.o

1085 PCEPP_OBJS += pcepp.o

1087 CPC_OBJS += cpc.o

1089 CPUID_OBJS += cpuid_drv.o

1091 SYSEVENT_OBJS += sysevent.o

1093 BL_OBJS += bl.o

1095 DRM_OBJS += drm_sunmod.o drm_kstat.o drm_agpsupport.o \
1096             drm_auth.o drm_bufs.o drm_context.o drm_dma.o \
1097             drm_drawable.o drm_drv.o drm_fops.o drm_ioctl.o drm_irq.o \
1098             drm_lock.o drm_memory.o drm_msg.o drm_pci.o drm_scatter.o \
1099             drm_cache.o drm_gem.o drm_mm.o ati_pcigart.o

1101 FM_OBJS += devfm.o devfm_machdep.o

1103 RTLS_OBJS +=    rtls.o

1105 #
1106 #                       exec modules
1107 #
1108 AOUTEXEC_OBJS +=aout.o

1110 ELFEXEC_OBJS += elf.o elf_notes.o old_notes.o

1112 INTPEXEC_OBJS +=intp.o

1114 SHBINEXEC_OBJS +=shbin.o

1116 JAVAEXEC_OBJS +=java.o
```



```
1118 #
1119 #                       file system modules
1120 #
1121 AUTOFS_OBJS +=  auto_vfsops.o auto_vnops.o auto_subr.o auto_xdr.o auto_sys.o

1123 CACHEFS_OBJS += cachefs_cnode.o         cachefs_cod.o \
1124                 cachefs_dir.o           cachefs_dlog.o  cachefs_filegrp.o \
1125                 cachefs_fscache.o       cachefs_ioctl.o cachefs_log.o \
1126                 cachefs_module.o \
1127                 cachefs_noopc.o         cachefs_resource.o \
1128                 cachefs_strict.o \
1129                 cachefs_subr.o          cachefs_vfsops.o \
1130                 cachefs_vnops.o

1132 DCFS_OBJS +=    dc_vnops.o

1134 DEVFS_OBJS +=   devfs_subr.o    devfs_vfsops.o  devfs_vnops.o

1136 DEV_OBJS  +=    sdev_subr.o     sdev_vfsops.o   sdev_vnops.o    \
1137                 sdev_ptsops.o   sdev_zvolops.o  sdev_comm.o     \
1138                 sdev_profile.o  sdev_ncache.o   sdev_netops.o   \
1139                 sdev_ipnetops.o \
1140                 sdev_vtops.o

1142 CTFS_OBJS +=    ctfs_all.o ctfs_cdir.o ctfs_ctl.o ctfs_event.o \
1143                 ctfs_latest.o ctfs_root.o ctfs_sym.o ctfs_tdir.o ctfs_tmpl.o

1145 OBJFS_OBJS +=   objfs_vfs.o     objfs_root.o    objfs_common.o \
1146                 objfs_odir.o    objfs_data.o

1148 FDFS_OBJS +=    fdops.o

1150 FIFO_OBJS +=    fifosubr.o      fifovnops.o

1152 PIPE_OBJS +=    pipe.o

1154 HSFS_OBJS +=    hsfs_node.o     hsfs_subr.o     hsfs_vfsops.o   hsfs_vnops.o \
1155                 hsfs_susp.o     hsfs_rrip.o     hsfs_susp_subr.o

1157 LOFS_OBJS +=    lofs_subr.o     lofs_vfsops.o   lofs_vnops.o

1159 NAMEFS_OBJS +=  namevfs.o       namevno.o

1161 NFS_OBJS +=     nfs_client.o    nfs_common.o    nfs_dump.o \
1162                 nfs_subr.o      nfs_vfsops.o    nfs_vnops.o \
1163                 nfs_xdr.o       nfs_sys.o       nfs_strerror.o \
1164                 nfs3_vfsops.o   nfs3_vnops.o    nfs3_xdr.o \
1165                 nfs_acl_vnops.o nfs_acl_xdr.o   nfs4_vfsops.o \
1166                 nfs4_vnops.o    nfs4_xdr.o      nfs4_idmap.o \
1167                 nfs4_shadow.o   nfs4_subr.o \
1168                 nfs4_attr.o     nfs4_rnode.o    nfs4_client.o \
1169                 nfs4_acache.o   nfs4_common.o   nfs4_client_state.o \
1170                 nfs4_callback.o nfs4_recovery.o nfs4_client_secinfo.o \
1171                 nfs4_client_debug.o     nfs_stats.o \
1172                 nfs4_acl.o      nfs4_stub_vnops.o       nfs_cmd.o

1174 NFSSRV_OBJS +=  nfs_server.o    nfs_srv.o       nfs3_srv.o \
1175                 nfs_acl_srv.o   nfs_auth.o      nfs_auth_xdr.o \
1176                 nfs_export.o    nfs_log.o       nfs_log_xdr.o \
1177                 nfs4_srv.o      nfs4_state.o    nfs4_srv_attr.o \
1178                 nfs4_srv_ns.o   nfs4_db.o       nfs4_srv_deleg.o \
1179                 nfs4_deleg_ops.o nfs4_srv_readdir.o nfs4_dispatch.o

1181 SMBSRV_SHARED_OBJS += \
1182                 smb_inet.o \
1183                 smb_match.o \
```

```
1184                smb_msgbuf.o \
1185                smb_oem.o \
1186                smb_string.o \
1187                smb_utf8.o \
1188                smb_door_legacy.o \
1189                smb_xdr.o \
1190                smb_token.o \
1191                smb_token_xdr.o \
1192                smb_sid.o \
1193                smb_native.o \
1194                smb_netbios_util.o

1196 SMBSRV_OBJS +=  $(SMBSRV_SHARED_OBJS)                  \
1197                smb_acl.o                               \
1198                smb_alloc.o                             \
1199                smb_close.o                             \
1200                smb_common_open.o                       \
1201                smb_common_transact.o                   \
1202                smb_create.o                            \
1203                smb_delete.o                            \
1204                smb_directory.o                         \
1205                smb_dispatch.o                          \
1206                smb_echo.o                              \
1207                smb_fem.o                               \
1208                smb_find.o                              \
1209                smb_flush.o                             \
1210                smb_fsinfo.o                            \
1211                smb_fsops.o                             \
1212                smb_init.o                              \
1213                smb_kdoor.o                             \
1214                smb_kshare.o                            \
1215                smb_kutil.o                             \
1216                smb_lock.o                              \
1217                smb_lock_byte_range.o                   \
1218                smb_locking_andx.o                      \
1219                smb_logoff_andx.o                       \
1220                smb_mangle_name.o                       \
1221                smb_mbuf_marshaling.o                   \
1222                smb_mbuf_util.o                         \
1223                smb_negotiate.o                         \
1224                smb_net.o                               \
1225                smb_node.o                              \
1226                smb_nt_cancel.o                         \
1227                smb_nt_create_andx.o                    \
1228                smb_nt_transact_create.o                \
1229                smb_nt_transact_ioctl.o                 \
1230                smb_nt_transact_notify_change.o         \
1231                smb_nt_transact_quota.o                 \
1232                smb_nt_transact_security.o              \
1233                smb_odir.o                              \
1234                smb_ofile.o                             \
1235                smb_open_andx.o                         \
1236                smb_opipe.o                             \
1237                smb_oplock.o                            \
1238                smb_pathname.o                          \
1239                smb_print.o                             \
1240                smb_process_exit.o                      \
1241                smb_query_fileinfo.o                    \
1242                smb_read.o                              \
1243                smb_rename.o                            \
1244                smb_sd.o                                \
1245                smb_seek.o                              \
1246                smb_server.o                            \
1247                smb_session.o                           \
1248                smb_session_setup_andx.o                \
1249                smb_set_fileinfo.o                      \
```



```
1250                smb_signing.o                           \
1251                smb_tree.o                              \
1252                smb_trans2_create_directory.o           \
1253                smb_trans2_dfs.o                        \
1254                smb_trans2_find.o                       \
1255                smb_tree_connect.o                      \
1256                smb_unlock_byte_range.o                 \
1257                smb_user.o                              \
1258                smb_vfs.o                               \
1259                smb_vops.o                              \
1260                smb_vss.o                               \
1261                smb_write.o                             \
1262                smb_write_raw.o

1264 PCFS_OBJS +=    pc_alloc.o      pc_dir.o        pc_node.o       pc_subr.o \
1265                 pc_vfsops.o     pc_vnops.o

1267 PROC_OBJS +=    prcontrol.o     prioctl.o       prsubr.o        prusrio.o \
1268                 prvfsops.o      prvnops.o

1270 MNTFS_OBJS +=   mntvfsops.o     mntvnops.o

1272 SHAREFS_OBJS += sharetab.o      sharefs_vfsops.o        sharefs_vnops.o

1274 SPEC_OBJS +=    specsubr.o      specvfsops.o    specvnops.o

1276 SOCK_OBJS +=    socksubr.o      sockvfsops.o    sockparams.o    \
1277                 socksyscalls.o  socktpi.o       sockstr.o \
1278                 sockcommon_vnops.o      sockcommon_subr.o \
1279                 sockcommon_sops.o       sockcommon.o    \
1280                 sock_notsupp.o  socknotify.o \
1281                 nl7c.o          nl7curi.o       nl7chttp.o      nl7clogd.o \
1282                 nl7cnca.o       sodirect.o      sockfilter.o

1284 TMPFS_OBJS +=   tmp_dir.o       tmp_subr.o      tmp_tnode.o     tmp_vfsops.o \
1285                 tmp_vnops.o

1287 UDFS_OBJS +=    udf_alloc.o     udf_bmap.o      udf_dir.o       \
1288                 udf_inode.o     udf_subr.o      udf_vfsops.o    \
1289                 udf_vnops.o

1291 UFS_OBJS +=     ufs_alloc.o     ufs_bmap.o      ufs_dir.o       ufs_xattr.o \
1292                 ufs_inode.o     ufs_subr.o      ufs_tables.o    ufs_vfsops.o \
1293                 ufs_vnops.o     quota.o         quotacalls.o    quota_ufs.o \
1294                 ufs_filio.o     ufs_lockfs.o    ufs_thread.o    ufs_trans.o \
1295                 ufs_acl.o       ufs_panic.o     ufs_directio.o  ufs_log.o \
1296                 ufs_extvnops.o  ufs_snap.o      lufs.o          lufs_thread.o \
1297                 lufs_log.o      lufs_map.o      lufs_top.o      lufs_debug.o
1298 VSCAN_OBJS +=   vscan_drv.o     vscan_svc.o vscan_door.o

1300 NSMB_OBJS +=    smb_conn.o      smb_dev.o       smb_iod.o       smb_pass.o \
1301                 smb_rq.o        smb_sign.o      smb_smb.o       smb_subrs.o \
1302                 smb_time.o      smb_tran.o      smb_trantcp.o   smb_usr.o \
1303                 subr_mchain.o

1305 SMBFS_COMMON_OBJS += smbfs_ntacl.o
1306 SMBFS_OBJS +=   smbfs_vfsops.o  smbfs_vnops.o   smbfs_node.o    \
1307                 smbfs_acl.o     smbfs_client.o  smbfs_smb.o     \
1308                 smbfs_subr.o    smbfs_subr2.o   \
1309                 smbfs_rwlock.o  smbfs_xattr.o   \
1310                 $(SMBFS_COMMON_OBJS)


1313 #
1314 #                       LVM modules
1315 #
```

```
1316 MD_OBJS += md.o md_error.o md_ioctl.o md_mddb.o md_names.o \
1317         md_med.o md_rename.o md_subr.o

1319 MD_COMMON_OBJS = md_convert.o md_crc.o md_revchk.o

1321 MD_DERIVED_OBJS = metamed_xdr.o meta_basic_xdr.o

1323 SOFTPART_OBJS += sp.o sp_ioctl.o

1325 STRIPE_OBJS += stripe.o stripe_ioctl.o

1327 HOTSPARES_OBJS += hotspares.o

1329 RAID_OBJS += raid.o raid_ioctl.o raid_replay.o raid_resync.o raid_hotspare.o

1331 MIRROR_OBJS += mirror.o mirror_ioctl.o mirror_resync.o

1333 NOTIFY_OBJS += md_notify.o

1335 TRANS_OBJS += mdtrans.o trans_ioctl.o trans_log.o

1337 ZFS_COMMON_OBJS +=              \
1338         arc.o                   \
1339         bplist.o                \
1340         bpobj.o                 \
1341         bptree.o                \
1342         dbuf.o                  \
1343         ddt.o                   \
1344         ddt_zap.o               \
1345         dmu.o                   \
1346         dmu_diff.o              \
1347         dmu_send.o              \
1348         dmu_object.o            \
1349         dmu_objset.o            \
1350         dmu_traverse.o          \
1351         dmu_tx.o                \
1352         dnode.o                 \
1353         dnode_sync.o            \
1354         dsl_bookmark.o          \
1355         dsl_dir.o               \
1356         dsl_dataset.o           \
1357         dsl_deadlist.o          \
1358         dsl_destroy.o           \
1359         dsl_pool.o              \
1360         dsl_synctask.o          \
1361         dsl_userhold.o          \
1362         dmu_zfetch.o            \
1363         dsl_deleg.o             \
1364         dsl_prop.o              \
1365         dsl_scan.o              \
1366         zfeature.o              \
1367         gzip.o                  \
1368         lz4.o                   \
1369         lzjb.o                  \
1370         metaslab.o              \
1371         range_tree.o            \
1372         refcount.o              \
1373         rrwlock.o               \
1374         sa.o                    \
1375         sha256.o                \
1376         spa.o                   \
1377         spa_config.o            \
1378         spa_errlog.o            \
1379         spa_history.o           \
1380         spa_misc.o              \
1381         space_map.o             \
```



```
1382         space_reftree.o         \
1383         txg.o                   \
1384         uberblock.o             \
1385         unique.o                \
1386         vdev.o                  \
1387         vdev_cache.o            \
1388         vdev_file.o             \
1389         vdev_label.o            \
1390         vdev_mirror.o           \
1391         vdev_missing.o          \
1392         vdev_queue.o            \
1393         vdev_raidz.o            \
1394         vdev_root.o             \
1395         zap.o                   \
1396         zap_leaf.o              \
1397         zap_micro.o             \
1398         zfs_byteswap.o          \
1399         zfs_debug.o             \
1400         zfs_fm.o                \
1401         zfs_fuid.o              \
1402         zfs_sa.o                \
1403         zfs_znode.o             \
1404         zil.o                   \
1405         zio.o                   \
1406         zio_checksum.o          \
1407         zio_compress.o          \
1408         zio_inject.o            \
1409         zle.o                   \
1410         zrlock.o

1412 ZFS_SHARED_OBJS +=              \
1413         zfeature_common.o       \
1414         zfs_comutil.o           \
1415         zfs_deleg.o             \
1416         zfs_fletcher.o          \
1417         zfs_namecheck.o         \
1418         zfs_prop.o              \
1419         zpool_prop.o            \
1420         zprop_common.o

1422 ZFS_OBJS +=                     \
1423         $(ZFS_COMMON_OBJS)      \
1424         $(ZFS_SHARED_OBJS)      \
1425         vdev_disk.o             \
1426         zfs_acl.o               \
1427         zfs_ctldir.o            \
1428         zfs_dir.o               \
1429         zfs_ioctl.o             \
1430         zfs_log.o               \
1431         zfs_onexit.o            \
1432         zfs_replay.o            \
1433         zfs_rlock.o             \
1434         zfs_vfsops.o            \
1435         zfs_vnops.o             \
1436         zvol.o

1438 ZUT_OBJS +=                     \
1439         zut.o

1441 #
1442 #                       streams modules
1443 #
1444 BUFMOD_OBJS     +=      bufmod.o

1446 CONNLD_OBJS +=  connld.o
```

```
1448 DEDUMP_OBJS +=  dedump.o

1450 DRCOMPAT_OBJS +=        drcompat.o

1452 LDLINUX_OBJS += ldlinux.o

1454 LDTERM_OBJS +=  ldterm.o uwidth.o

1456 PCKT_OBJS +=    pckt.o

1458 PFMOD_OBJS +=   pfmod.o

1460 PTEM_OBJS +=    ptem.o

1462 REDIRMOD_OBJS += strredirm.o

1464 TIMOD_OBJS +=   timod.o

1466 TIRDWR_OBJS +=  tirdwr.o

1468 TTCOMPAT_OBJS +=ttcompat.o

1470 LOG_OBJS +=     log.o

1472 PIPEMOD_OBJS += pipemod.o

1474 RPCMOD_OBJS +=  rpcmod.o        clnt_cots.o     clnt_clts.o \
1475                 clnt_gen.o      clnt_perr.o     mt_rpcinit.o    rpc_calmsg.o \
1476                 rpc_prot.o      rpc_sztypes.o   rpc_subr.o      rpcb_prot.o \
1477                 svc.o           svc_clts.o      svc_gen.o       svc_cots.o \
1478                 rpcsys.o        xdr_sizeof.o    clnt_rdma.o     svc_rdma.o \
1479                 xdr_rdma.o      rdma_subr.o     xdrrdma_sizeof.o

1481 KLMMOD_OBJS +=  klmmod.o \
1482                 nlm_impl.o \
1483                 nlm_rpc_handle.o \
1484                 nlm_dispatch.o \
1485                 nlm_rpc_svc.o \
1486                 nlm_client.o \
1487                 nlm_service.o \
1488                 nlm_prot_clnt.o \
1489                 nlm_prot_xdr.o \
1490                 nlm_rpc_clnt.o \
1491                 nsm_addr_clnt.o \
1492                 nsm_addr_xdr.o \
1493                 sm_inter_clnt.o \
1494                 sm_inter_xdr.o

1496 KLMOPS_OBJS +=  klmops.o

1498 TLIMOD_OBJS +=  tlimod.o        t_kalloc.o      t_kbind.o       t_kclose.o \
1499                 t_kconnect.o    t_kfree.o       t_kgtstate.o    t_kopen.o \
1500                 t_krcvudat.o    t_ksndudat.o    t_kspoll.o      t_kunbind.o \
1501                 t_kutil.o

1503 RLMOD_OBJS += rlmod.o

1505 TELMOD_OBJS += telmod.o

1507 CRYPTMOD_OBJS += cryptmod.o

1509 KB_OBJS +=      kbd.o           keytables.o

1511 #
1512 #                       ID mapping module
1513 #
```



```
1514 IDMAP_OBJS +=   idmap_mod.o     idmap_kapi.o    idmap_xdr.o     idmap_cache.o

1516 #
1517 #                       scheduling class modules
1518 #
1519 SDC_OBJS +=             sysdc.o

1521 RT_OBJS +=              rt.o
1522 RT_DPTBL_OBJS +=        rt_dptbl.o

1524 TS_OBJS +=              ts.o
1525 TS_DPTBL_OBJS +=        ts_dptbl.o

1527 IA_OBJS +=              ia.o

1529 FSS_OBJS +=             fss.o

1531 FX_OBJS +=              fx.o
1532 FX_DPTBL_OBJS +=        fx_dptbl.o

1534 #
1535 #                       Inter-Process Communication (IPC) modules
1536 #
1537 IPC_OBJS +=     ipc.o

1539 IPCMSG_OBJS +=  msg.o

1541 IPCSEM_OBJS +=  sem.o

1543 IPCSHM_OBJS +=  shm.o

1545 #
1546 #                       bignum module
1547 #
1548 COMMON_BIGNUM_OBJS += bignum_mod.o bignumimpl.o

1550 BIGNUM_OBJS += $(COMMON_BIGNUM_OBJS) $(BIGNUM_PSR_OBJS)

1552 #
1553 #                       kernel cryptographic framework
1554 #
1555 KCF_OBJS +=     kcf.o kcf_callprov.o kcf_cbufcall.o kcf_cipher.o kcf_crypto.o \
1556                 kcf_cryptoadm.o kcf_ctxops.o kcf_digest.o kcf_dual.o \
1557                 kcf_keys.o kcf_mac.o kcf_mech_tabs.o kcf_miscapi.o \
1558                 kcf_object.o kcf_policy.o kcf_prov_lib.o kcf_prov_tabs.o \
1559                 kcf_sched.o kcf_session.o kcf_sign.o kcf_spi.o kcf_verify.o \
1560                 kcf_random.o modes.o ecb.o cbc.o ctr.o ccm.o gcm.o \
1561                 fips_random.o

1563 CRYPTOADM_OBJS += cryptoadm.o

1565 CRYPTO_OBJS +=  crypto.o

1567 DPROV_OBJS +=   dprov.o

1569 DCA_OBJS +=     dca.o dca_3des.o dca_debug.o dca_dsa.o dca_kstat.o dca_rng.o \
1570                 dca_rsa.o

1572 AESPROV_OBJS += aes.o aes_impl.o aes_modes.o

1574 ARCFOURPROV_OBJS += arcfour.o arcfour_crypt.o

1576 BLOWFISHPROV_OBJS += blowfish.o blowfish_impl.o

1578 ECCPROV_OBJS += ecc.o ec.o ec2_163.o ec2_mont.o ecdecode.o ecl_mult.o \
1579                 ecp_384.o ecp_jac.o ec2_193.o ecl.o ecp_192.o ecp_521.o \
```

```
1580                 ecp_jm.o ec2_233.o ecl_curve.o ecp_224.o ecp_aff.o \
1581                 ecp_mont.o ec2_aff.o ec_naf.o ecl_gf.o ecp_256.o mp_gf2m.o \
1582                 mpi.o mplogic.o mpmontg.o mpprime.o oid.o \
1583                 secitem.o ec2_test.o ecp_test.o

1585 RSAPROV_OBJS += rsa.o rsa_impl.o pkcs1.o

1587 SWRANDPROV_OBJS += swrand.o

1589 #
1590 #                       kernel SSL
1591 #
1592 KSSL_OBJS +=    kssl.o kssl_ioctl.o

1594 KSSL_SOCKFIL_MOD_OBJS += ksslfilter.o ksslapi.o ksslrec.o

1596 #
1597 #                       misc. modules
1598 #

1600 C2AUDIT_OBJS += adr.o audit.o audit_event.o audit_io.o \
1601                 audit_path.o audit_start.o audit_syscalls.o audit_token.o \
1602                 audit_mem.o

1604 PCIC_OBJS +=    pcic.o

1606 RPCSEC_OBJS +=  secmod.o        sec_clnt.o      sec_svc.o       sec_gen.o \
1607                 auth_des.o      auth_kern.o     auth_none.o     auth_loopb.o\
1608                 authdesprt.o    authdesubr.o    authu_prot.o \
1609                 key_call.o      key_prot.o      svc_authu.o     svcauthdes.o

1611 RPCSEC_GSS_OBJS +=      rpcsec_gssmod.o rpcsec_gss.o rpcsec_gss_misc.o \
1612                 rpcsec_gss_utils.o svc_rpcsec_gss.o

1614 CONSCONFIG_OBJS += consconfig.o

1616 CONSCONFIG_DACF_OBJS  += consconfig_dacf.o consplat.o

1618 TEM_OBJS += tem.o tem_safe.o 6x10.o 7x14.o 12x22.o

1620 KBTRANS_OBJS +=                        \
1621                 kbtrans.o               \
1622                 kbtrans_keytables.o     \
1623                 kbtrans_polled.o        \
1624                 kbtrans_streams.o       \
1625                 usb_keytables.o

1627 KGSSD_OBJS +=   gssd_clnt_stubs.o gssd_handle.o gssd_prot.o \
1628                 gss_display_name.o gss_release_name.o gss_import_name.o \
1629                 gss_release_buffer.o gss_release_oid_set.o gen_oids.o gssdmod.o

1631 KGSSD_DERIVED_OBJS = gssd_xdr.o

1633 KGSS_DUMMY_OBJS += dmech.o

1635 KSOCKET_OBJS += ksocket.o ksocket_mod.o

1637 CRYPTO= cksumtypes.o decrypt.o encrypt.o encrypt_length.o etypes.o \
1638         nfold.o verify_checksum.o prng.o block_size.o make_checksum.o\
1639         checksum_length.o hmac.o default_state.o mandatory_sumtype.o

1641 # crypto/des
1642 CRYPTO_DES= f_cbc.o f_cksum.o f_parity.o weak_key.o d3_cbc.o ef_crypto.o

1644 CRYPTO_DK= checksum.o derive.o dk_decrypt.o dk_encrypt.o
```



```
1646 CRYPTO_ARCFOUR= k5_arcfour.o

1648 # crypto/enc_provider
1649 CRYPTO_ENC= des.o des3.o arcfour_provider.o aes_provider.o

1651 # crypto/hash_provider
1652 CRYPTO_HASH= hash_kef_generic.o hash_kmd5.o hash_crc32.o hash_ksha1.o

1654 # crypto/keyhash_provider
1655 CRYPTO_KEYHASH= descbc.o k5_kmd5des.o k_hmac_md5.o

1657 # crypto/crc32
1658 CRYPTO_CRC32= crc32.o

1660 # crypto/old
1661 CRYPTO_OLD= old_decrypt.o old_encrypt.o

1663 # crypto/raw
1664 CRYPTO_RAW= raw_decrypt.o raw_encrypt.o

1666 K5_KRB= kfree.o copy_key.o \
1667         parse.o init_ctx.o \
1668         ser_adata.o ser_addr.o \
1669         ser_auth.o ser_cksum.o \
1670         ser_key.o ser_princ.o \
1671         serialize.o unparse.o \
1672         ser_actx.o

1674 K5_OS=  timeofday.o toffset.o \
1675         init_os_ctx.o c_ustime.o

1677 SEAL=   seal.o unseal.o

1679 MECH=   delete_sec_context.o \
1680         import_sec_context.o \
1681         gssapi_krb5.o \
1682         k5seal.o k5unseal.o k5sealv3.o \
1683         ser_sctx.o \
1684         sign.o \
1685         util_crypt.o  \
1686         util_validate.o  util_ordering.o  \
1687         util_seqnum.o util_set.o util_seed.o \
1688         wrap_size_limit.o verify.o



1692 MECH_GEN= util_token.o


1695 KGSS_KRB5_OBJS += krb5mech.o \
1696         $(MECH) $(SEAL) $(MECH_GEN) \
1697         $(CRYPTO) $(CRYPTO_DES) $(CRYPTO_DK) $(CRYPTO_ARCFOUR) \
1698         $(CRYPTO_ENC) $(CRYPTO_HASH) \
1699         $(CRYPTO_KEYHASH) $(CRYPTO_CRC32) \
1700         $(CRYPTO_OLD) \
1701         $(CRYPTO_RAW) $(K5_KRB) $(K5_OS)

1703 DES_OBJS +=     des_crypt.o des_impl.o des_ks.o des_soft.o

1705 DLBOOT_OBJS +=  bootparam_xdr.o nfs_dlinet.o scan.o

1707 KRTLD_OBJS +=   kobj_bootflags.o getoptstr.o \
1708                 kobj.o kobj_kdi.o kobj_lm.o kobj_subr.o

1710 MOD_OBJS +=     modctl.o modsubr.o modsysfile.o modconf.o modhash.o
```

```
1712 STRPLUMB_OBJS += strplumb.o

1714 CPR_OBJS +=     cpr_driver.o cpr_dump.o \
1715                 cpr_main.o cpr_misc.o cpr_mod.o cpr_stat.o \
1716                 cpr_uthread.o

1718 PROF_OBJS +=    prf.o

1720 SE_OBJS += se_driver.o

1722 SYSACCT_OBJS += acct.o

1724 ACCTCTL_OBJS += acctctl.o

1726 EXACCTSYS_OBJS += exacctsys.o

1728 KAIO_OBJS += aio.o

1730 PCMCIA_OBJS += pcmcia.o cs.o cis.o cis_callout.o cis_handlers.o cis_params.o

1732 BUSRA_OBJS += busra.o

1734 PCS_OBJS += pcs.o

1736 PSET_OBJS +=    pset.o

1738 OHCI_OBJS += ohci.o ohci_hub.o ohci_polled.o

1740 UHCI_OBJS += uhci.o uhciutil.o uhcitgt.o uhcihub.o uhcipolled.o

1742 EHCI_OBJS += ehci.o ehci_hub.o ehci_xfer.o ehci_intr.o ehci_util.o ehci_polled.o

1744 HUBD_OBJS += hubd.o

1746 USB_MID_OBJS += usb_mid.o

1748 USB_IA_OBJS += usb_ia.o

1750 UWBA_OBJS += uwba.o uwbai.o

1752 SCSA2USB_OBJS += scsa2usb.o usb_ms_bulkonly.o usb_ms_cbi.o

1754 HWAHC_OBJS += hwahc.o hwahc_util.o

1756 WUSB_DF_OBJS += wusb_df.o
1757 WUSB_FWMOD_OBJS += wusb_fwmod.o

1759 IPF_OBJS += ip_fil_solaris.o fil.o solaris.o ip_state.o ip_frag.o ip_nat.o \
1760             ip_proxy.o ip_auth.o ip_pool.o ip_htable.o ip_lookup.o \
1761             ip_log.o misc.o ip_compat.o ip_nat6.o drand48.o

1763 IPD_OBJS += ipd.o

1765 IBD_OBJS +=     ibd.o ibd_cm.o

1767 EIBNX_OBJS +=   enx_main.o enx_hdlrs.o enx_ibt.o enx_log.o enx_fip.o \
1768                 enx_misc.o enx_q.o enx_ctl.o

1770 EOIB_OBJS +=    eib_adm.o eib_chan.o eib_cmn.o eib_ctl.o eib_data.o \
1771                 eib_fip.o eib_ibt.o eib_log.o eib_mac.o eib_main.o \
1772                 eib_rsrc.o eib_svc.o eib_vnic.o

1774 DLPISTUB_OBJS += dlpistub.o

1776 SDP_OBJS +=     sdpddi.o
```



```
1778 TRILL_OBJS +=   trill.o

1780 CTF_OBJS += ctf_create.o ctf_decl.o ctf_error.o ctf_hash.o ctf_labels.o \
1781         ctf_lookup.o ctf_open.o ctf_types.o ctf_util.o ctf_subr.o ctf_mod.o

1783 SMBIOS_OBJS += smb_error.o smb_info.o smb_open.o smb_subr.o smb_dev.o

1785 RPCIB_OBJS += rpcib.o

1787 KMDB_OBJS += kdrv.o

1789 AFE_OBJS += afe.o

1791 BGE_OBJS += bge_main2.o bge_chip2.o bge_kstats.o bge_log.o bge_ndd.o \
1792                 bge_atomic.o bge_mii.o bge_send.o bge_recv2.o bge_mii_5906.o

1794 DMFE_OBJS += dmfe_log.o dmfe_main.o dmfe_mii.o

1796 EFE_OBJS += efe.o

1798 ELXL_OBJS += elxl.o

1800 HME_OBJS += hme.o

1802 IXGB_OBJS += ixgb.o ixgb_atomic.o ixgb_chip.o ixgb_gld.o ixgb_kstats.o \
1803                 ixgb_log.o ixgb_ndd.o ixgb_rx.o ixgb_tx.o ixgb_xmii.o

1805 NGE_OBJS += nge_main.o nge_atomic.o nge_chip.o nge_ndd.o nge_kstats.o \
1806                 nge_log.o nge_rx.o nge_tx.o nge_xmii.o

1808 PCN_OBJS += pcn.o

1810 RGE_OBJS += rge_main.o rge_chip.o rge_ndd.o rge_kstats.o rge_log.o rge_rxtx.o

1812 URTW_OBJS += urtw.o

1814 ARN_OBJS += arn_hw.o arn_eeprom.o arn_mac.o arn_calib.o arn_ani.o arn_phy.o arn_
1815                 arn_main.o arn_recv.o arn_xmit.o arn_rc.o

1817 ATH_OBJS += ath_aux.o ath_main.o ath_osdep.o ath_rate.o

1819 ATU_OBJS += atu.o

1821 IPW_OBJS += ipw2100_hw.o ipw2100.o

1823 IWI_OBJS += ipw2200_hw.o ipw2200.o

1825 IWH_OBJS += iwh.o

1827 IWK_OBJS += iwk2.o

1829 IWP_OBJS += iwp.o

1831 MWL_OBJS += mwl.o

1833 MWLFW_OBJS += mwlfw_mode.o

1835 WPI_OBJS += wpi.o

1837 RAL_OBJS += rt2560.o ral_rate.o

1839 RUM_OBJS += rum.o

1841 RWD_OBJS += rt2661.o

1843 RWN_OBJS += rt2860.o
```

```
1845 UATH_OBJS += uath.o

1847 UATHFW_OBJS += uathfw_mod.o

1849 URAL_OBJS += ural.o

1851 RTW_OBJS += rtw.o smc93cx6.o rtwphy.o rtwphyio.o

1853 ZYD_OBJS += zyd.o zyd_usb.o zyd_hw.o zyd_fw.o

1855 MXFE_OBJS += mxfe.o

1857 MPTSAS_OBJS += mptsas.o mptsas_impl.o mptsas_init.o mptsas_raid.o mptsas_smhba.o

1859 SFE_OBJS += sfe.o sfe_util.o

1861 BFE_OBJS += bfe.o

1863 BRIDGE_OBJS += bridge.o

1865 IDM_SHARED_OBJS += base64.o

1867 IDM_OBJS +=     $(IDM_SHARED_OBJS) \
1868                 idm.o idm_impl.o idm_text.o idm_conn_sm.o idm_so.o

1870 VR_OBJS += vr.o

1872 ATGE_OBJS += atge_main.o atge_l1e.o atge_mii.o atge_l1.o atge_l1c.o

1874 YGE_OBJS = yge.o

1876 #
1877 #       Build up defines and paths.
1878 #
1879 LINT_DEFS       += -Dunix

1881 #
1882 #       This duality can be removed when the native and target compilers
1883 #       are the same (or at least recognize the same command line syntax!)
1884 #       It is a bug in the current compilation system that the assember
1885 #       can't process the -Y I, flag.
1886 #
1887 NATIVE_INC_PATH += $(INC_PATH) $(CCYFLAG)$(UTSBASE)/common
1888 AS_INC_PATH     += $(INC_PATH) -I$(UTSBASE)/common
1889 INCLUDE_PATH    += $(INC_PATH) $(CCYFLAG)$(UTSBASE)/common

1891 PCIEB_OBJS += pcieb.o

1893 #       Chelsio N110 10G NIC driver module
1894 #
1895 CH_OBJS = ch.o glue.o pe.o sge.o

1897 CH_COM_OBJS =   ch_mac.o ch_subr.o cspi.o espi.o ixf1010.o mc3.o mc4.o mc5.o \
1898                 mv88e1xxx.o mv88x201x.o my3126.o pm3393.o tp.o ulp.o \
1899                 vsc7321.o vsc7326.o xpak.o

1901 #
1902 #       Chelsio Terminator 4 10G NIC nexus driver module
1903 #
1904 CXGBE_FW_OBJS  =        t4_fw.o t4_cfg.o
1905 CXGBE_COM_OBJS =        t4_hw.o common.o
1906 CXGBE_NEX_OBJS =        t4_nexus.o t4_sge.o t4_mac.o t4_ioctl.o shared.o \
1907                         t4_l2t.o adapter.o osdep.o

1909 #
```



```
1910 #       Chelsio Terminator 4 10G NIC driver module
1911 #
1912 CXGBE_OBJS =    cxgbe.o

1914 #
1915 #       PCI strings file
1916 #
1917 PCI_STRING_OBJS = pci_strings.o

1919 NET_DACF_OBJS += net_dacf.o

1921 #
1922 #       Xframe 10G NIC driver module
1923 #
1924 XGE_OBJS = xge.o xgell.o

1926 XGE_HAL_OBJS =  xgehal-channel.o xgehal-fifo.o xgehal-ring.o  xgehal-config.o \
1927                 xgehal-driver.o  xgehal-mm.o xgehal-stats.o  xgehal-device.o \
1928                 xge-queue.o  xgehal-mgmt.o xgehal-mgmtaux.o

1930 #
1931 #       e1000/igb common objs
1932 #
1933 #       Historically e1000g and igb had separate copies of all of the common
1934 #       code. At this time while they are now sharing the same copy of it, they
1935 #       are building it into their own modules which is due to the differences
1936 #       in the osdep and debug portions of their code.
1937 #
1938 E1000API_OBJS += e1000_80003es2lan.o e1000_82540.o e1000_82541.o e1000_82542.o \
1939                 e1000_82543.o e1000_82571.o e1000_api.o e1000_ich8lan.o \
1940                 e1000_mac.o e1000_manage.o e1000_nvm.o e1000_phy.o \
1941                 e1000_82575.o e1000_i210.o e1000_mbx.o e1000_vf.o

1943 #
1944 #       e1000g module
1945 #
1946 E1000G_OBJS +=  e1000g_debug.o e1000g_main.o e1000g_alloc.o \
1947                 e1000g_tx.o e1000g_rx.o e1000g_stat.o \
1948                 e1000g_osdep.o e1000g_workarounds.o
1949


1951 #
1952 #       Intel 82575 1G NIC driver module
1953 #
1954 IGB_OBJS =      igb_buf.o igb_debug.o igb_gld.o igb_log.o igb_main.o \
1955                 igb_rx.o igb_stat.o igb_tx.o igb_osdep.o

1957 #
1958 #       Intel Pro/100 NIC driver module
1959 #
1960 IPRB_OBJS =     iprb.o

1962 #
1963 #       Intel 10GbE PCIE NIC driver module
1964 #
1965 IXGBE_OBJS =    ixgbe_82598.o ixgbe_82599.o ixgbe_api.o         \
1966                 ixgbe_common.o ixgbe_phy.o                      \
1967                 ixgbe_buf.o ixgbe_debug.o ixgbe_gld.o           \
1968                 ixgbe_log.o ixgbe_main.o                        \
1969                 ixgbe_osdep.o ixgbe_rx.o ixgbe_stat.o           \
1970                 ixgbe_tx.o  ixgbe_x540.o ixgbe_mbx.o

1972 #
1973 #       NIU 10G/1G driver module
1974 #
1975 NXGE_OBJS =     nxge_mac.o nxge_ipp.o nxge_rxdma.o               \
```

```
1976                 nxge_txdma.o nxge_txc.o nxge_main.o             \
1977                 nxge_hw.o nxge_fzc.o nxge_virtual.o             \
1978                 nxge_send.o nxge_classify.o nxge_fflp.o         \
1979                 nxge_fflp_hash.o nxge_ndd.o nxge_kstats.o       \
1980                 nxge_zcp.o nxge_fm.o nxge_espc.o nxge_hv.o      \
1981                 nxge_hio.o nxge_hio_guest.o nxge_intr.o

1983 NXGE_NPI_OBJS = \
1984                 npi.o npi_mac.o npi_ipp.o                       \
1985                 npi_txdma.o npi_rxdma.o npi_txc.o               \
1986                 npi_zcp.o npi_espc.o npi_fflp.o                 \
1987                 npi_vir.o

1989 NXGE_HCALL_OBJS =       \
1990                 nxge_hcall.o

1992 #
1993 # Virtio modules
1994 #

1996 # Virtio core
1997 VIRTIO_OBJS = virtio.o

1999 # Virtio block driver
2000 VIOBLK_OBJS = vioblk.o

2002 #
2003 #       kiconv modules
2004 #
2005 KICONV_EMEA_OBJS += kiconv_emea.o

2007 KICONV_JA_OBJS += kiconv_ja.o

2009 KICONV_KO_OBJS += kiconv_cck_common.o kiconv_ko.o

2011 KICONV_SC_OBJS += kiconv_cck_common.o kiconv_sc.o

2013 KICONV_TC_OBJS += kiconv_cck_common.o kiconv_tc.o

2015 #
2016 #       AAC module
2017 #
2018 AAC_OBJS = aac.o aac_ioctl.o

2020 #
2021 #       sdcard modules
2022 #
2023 SDA_OBJS =      sda_cmd.o sda_host.o sda_init.o sda_mem.o sda_mod.o sda_slot.o
2024 SDHOST_OBJS =   sdhost.o

2026 #
2027 #       hxge 10G driver module
2028 #
2029 HXGE_OBJS =     hxge_main.o hxge_vmac.o hxge_send.o              \
2030                 hxge_txdma.o hxge_rxdma.o hxge_virtual.o        \
2031                 hxge_fm.o hxge_fzc.o hxge_hw.o hxge_kstats.o    \
2032                 hxge_ndd.o hxge_pfc.o                           \
2033                 hpi.o hpi_vmac.o hpi_rxdma.o hpi_txdma.o        \
2034                 hpi_vir.o hpi_pfc.o

2036 #
2037 #       MEGARAID_SAS module
2038 #
2039 MEGA_SAS_OBJS = megaraid_sas.o

2041 #
```



```
2042 #       MR_SAS module
2043 #
2044 MR_SAS_OBJS = ld_pd_map.o mr_sas.o mr_sas_tbolt.o mr_sas_list.o

2046 #
2047 #       CPQARY3 module
2048 #
2049 CPQARY3_OBJS =  cpqary3.o cpqary3_noe.o cpqary3_talk2ctlr.o     \
2050                 cpqary3_isr.o cpqary3_transport.o cpqary3_mem.o \
2051                 cpqary3_scsi.o cpqary3_util.o cpqary3_ioctl.o   \
2052                 cpqary3_bd.o

2054 #
2055 #       ISCSI_INITIATOR module
2056 #
2057 ISCSI_INITIATOR_OBJS =  chap.o iscsi_io.o iscsi_thread.o        \
2058                         iscsi_ioctl.o iscsid.o iscsi.o          \
2059                         iscsi_login.o isns_client.o iscsiAuthClient.o   \
2060                         iscsi_lun.o iscsiAuthClientGlue.o       \
2061                         iscsi_net.o nvfile.o iscsi_cmd.o        \
2062                         iscsi_queue.o persistent.o iscsi_conn.o \
2063                         iscsi_sess.o radius_auth.o iscsi_crc.o  \
2064                         iscsi_stats.o radius_packet.o iscsi_doorclt.o   \
2065                         iscsi_targetparam.o utils.o kifconf.o

2067 #
2068 #       ntxn 10Gb/1Gb NIC driver module
2069 #
2070 NTXN_OBJS =     unm_nic_init.o unm_gem.o unm_nic_hw.o unm_ndd.o \
2071                         unm_nic_main.o unm_nic_isr.o unm_nic_ctx.o niu.o

2073 #
2074 #       Myricom 10Gb NIC driver module
2075 #
2076 MYRI10GE_OBJS = myri10ge.o myri10ge_lro.o

2078 #       nulldriver module
2079 #
2080 NULLDRIVER_OBJS =       nulldriver.o

2082 TPM_OBJS =      tpm.o tpm_hcall.o
```

```
**********************************************************
   39786 Tue Jan 14 16:17:17 2014
new/usr/src/uts/common/brand/lx/autofs/lx_autofs.c
LX zone support should now build and packages of relevance produced.
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <fs/fs_subr.h>
#include <sys/atomic.h>
#include <sys/cmn_err.h>
#include <sys/dirent.h>
#include <sys/fs/fifonode.h>
#include <sys/modctl.h>
#include <sys/mount.h>
#include <sys/policy.h>
#include <sys/sunddi.h>

#include <sys/sysmacros.h>
#include <sys/vfs.h>
#include <sys/vfs_opreg.h>

#include <sys/lx_autofs_impl.h>

/*
 * External functions
 */
extern uintptr_t                space_fetch(char *key);
extern int                      space_store(char *key, uintptr_t ptr);

/*
 * Globals
 */
static vfsops_t                 *lx_autofs_vfsops;
static vnodeops_t               *lx_autofs_vn_ops = NULL;
static int                      lx_autofs_fstype;
static major_t                  lx_autofs_major;
static minor_t                  lx_autofs_minor = 0;

/*
```



```
 * Support functions
 */
static void
i_strfree(char *str)
{
        kmem_free(str, strlen(str) + 1);
}

static char *
i_strdup(char *str)
{
        int     n = strlen(str);
        char    *ptr = kmem_alloc(n + 1, KM_SLEEP);
        bcopy(str, ptr, n + 1);
        return (ptr);
}

static int
i_str_to_int(char *str, int *val)
{
        long    res;

        if (str == NULL)
                return (-1);

        if ((ddi_strtol(str, NULL, 10, &res) != 0) ||
            (res < INT_MIN) || (res > INT_MAX))
                return (-1);

        *val = res;
        return (0);
}

static void
i_stack_init(list_t *lp)
{
        list_create(lp,
            sizeof (stack_elem_t), offsetof(stack_elem_t, se_list));
}

static void
i_stack_fini(list_t *lp)
{
        ASSERT(list_head(lp) == NULL);
        list_destroy(lp);
}

static void
i_stack_push(list_t *lp, caddr_t ptr1, caddr_t ptr2, caddr_t ptr3)
{
        stack_elem_t    *se;

        se = kmem_alloc(sizeof (*se), KM_SLEEP);
        se->se_ptr1 = ptr1;
        se->se_ptr2 = ptr2;
        se->se_ptr3 = ptr3;
        list_insert_head(lp, se);
}

static int
i_stack_pop(list_t *lp, caddr_t *ptr1, caddr_t *ptr2, caddr_t *ptr3)
{
        stack_elem_t    *se;

        if ((se = list_head(lp)) == NULL)
                return (-1);
```

```
127         list_remove(lp, se);
128         if (ptr1 != NULL)
129                 *ptr1 = se->se_ptr1;
130         if (ptr2 != NULL)
131                 *ptr2 = se->se_ptr2;
132         if (ptr3 != NULL)
133                 *ptr3 = se->se_ptr3;
134         kmem_free(se, sizeof (*se));
135         return (0);
136 }

138 static vnode_t *
139 fifo_peer_vp(vnode_t *vp)
140 {
141         fifonode_t *fnp = VTOF(vp);
142         fifonode_t *fn_dest = fnp->fn_dest;
143         return (FTOV(fn_dest));
144 }

146 static vnode_t *
147 i_vn_alloc(vfs_t *vfsp, vnode_t *uvp)
148 {
149         lx_autofs_vfs_t *data = vfsp->vfs_data;
150         vnode_t         *vp, *vp_old;

152         /* Allocate a new vnode structure in case we need it. */
153         vp = vn_alloc(KM_SLEEP);
154         vn_setops(vp, lx_autofs_vn_ops);
155         VN_SET_VFS_TYPE_DEV(vp, vfsp, uvp->v_type, uvp->v_rdev);
156         vp->v_data = uvp;
157         ASSERT(vp->v_count == 1);

159         /*
160          * Take a hold on the vfs structure.  This is how unmount will
161          * determine if there are any active vnodes in the file system.
162          */
163         VFS_HOLD(vfsp);

165         /*
166          * Check if we already have a vnode allocated for this underlying
167          * vnode_t.
168          */
169         mutex_enter(&data->lav_lock);
170         if (mod_hash_find(data->lav_vn_hash,
171             (mod_hash_key_t)uvp, (mod_hash_val_t *)&vp_old) != 0) {

173                 /*
174                  * Didn't find an existing node.
175                  * Add this node to the hash and return.
176                  */
177                 VERIFY(mod_hash_insert(data->lav_vn_hash,
178                     (mod_hash_key_t)uvp,
179                     (mod_hash_val_t)vp) == 0);
180                 mutex_exit(&data->lav_lock);
181                 return (vp);
182         }

184         /* Get a hold on the existing vnode and free up the one we allocated. */
185         VN_HOLD(vp_old);
186         mutex_exit(&data->lav_lock);

188         /* Free up the new vnode we allocated. */
189         VN_RELE(uvp);
190         VFS_RELE(vfsp);
191         vn_invalid(vp);
192         vn_free(vp);
```



```
194         return (vp_old);
195 }

197 static void
198 i_vn_free(vnode_t *vp)
199 {
200         vfs_t           *vfsp = vp->v_vfsp;
201         lx_autofs_vfs_t *data = vfsp->vfs_data;
202         vnode_t         *uvp = vp->v_data;
203         vnode_t *vp_tmp;

205         ASSERT(MUTEX_HELD((&data->lav_lock)));
206         ASSERT(MUTEX_HELD((&vp->v_lock)));

208         ASSERT(vp->v_count == 0);

210         /* We're about to free this vnode so take it out of the hash. */
211         (void) mod_hash_remove(data->lav_vn_hash,
212             (mod_hash_key_t)uvp, (mod_hash_val_t)&vp_tmp);

214         /*
215          * No one else can lookup this vnode any more so there's no need
216          * to hold locks.
217          */
218         mutex_exit(&data->lav_lock);
219         mutex_exit(&vp->v_lock);

221         /* Release the underlying vnode. */
222         VN_RELE(uvp);
223         VFS_RELE(vfsp);
224         vn_invalid(vp);
225         vn_free(vp);
226 }

228 static lx_autofs_lookup_req_t *
229 i_lalr_alloc(lx_autofs_vfs_t *data, int *dup_request, char *nm)
230 {
231         lx_autofs_lookup_req_t  *lalr, *lalr_dup;

233         /* Pre-allocate a new automounter request before grabbing locks. */
234         lalr = kmem_zalloc(sizeof (*lalr), KM_SLEEP);
235         mutex_init(&lalr->lalr_lock, NULL, MUTEX_DEFAULT, NULL);
236         cv_init(&lalr->lalr_cv, NULL, CV_DEFAULT, NULL);
237         lalr->lalr_ref = 1;
238         lalr->lalr_pkt.lap_protover = LX_AUTOFS_PROTO_VERSION;

240         /* Assign a unique id for this request. */
241         lalr->lalr_pkt.lap_id = id_alloc(data->lav_ids);

243         /*
244          * The token expected by the linux automount is the name of
245          * the directory entry to look up.  (And not the entire
246          * path that is being accessed.)
247          */
248         lalr->lalr_pkt.lap_name_len = strlen(nm);
249         if (lalr->lalr_pkt.lap_name_len >
250             (sizeof (lalr->lalr_pkt.lap_name) - 1)) {
251                 zcmn_err(getzoneid(), CE_NOTE,
252                     "invalid autofs lookup: \"%s\"", nm);
253                 id_free(data->lav_ids, lalr->lalr_pkt.lap_id);
254                 kmem_free(lalr, sizeof (*lalr));
255                 return (NULL);
256         }
257         (void) strlcpy(lalr->lalr_pkt.lap_name, nm,
258             sizeof (lalr->lalr_pkt.lap_name));
```

```
260         /* Check for an outstanding request for this path. */
261         mutex_enter(&data->lav_lock);
262         if (mod_hash_find(data->lav_path_hash,
263             (mod_hash_key_t)nm, (mod_hash_val_t *)&lalr_dup) == 0) {
264                 /*
265                  * There's already an outstanding request for this
266                  * path so we don't need a new one.
267                  */
268                 id_free(data->lav_ids, lalr->lalr_pkt.lap_id);
269                 kmem_free(lalr, sizeof (*lalr));
270                 lalr = lalr_dup;
271
272                 /* Bump the ref count on the old request. */
273                 atomic_add_int(&lalr->lalr_ref, 1);
274
275                 *dup_request = 1;
276         } else {
277                 /* Add it to the hashes. */
278                 VERIFY(mod_hash_insert(data->lav_id_hash,
279                     (mod_hash_key_t)(uintptr_t)lalr->lalr_pkt.lap_id,
280                     (mod_hash_val_t)lalr) == 0);
281                 VERIFY(mod_hash_insert(data->lav_path_hash,
282                     (mod_hash_key_t)i_strdup(nm),
283                     (mod_hash_val_t)lalr) == 0);
284
285                 *dup_request = 0;
286         }
287         mutex_exit(&data->lav_lock);
288
289         return (lalr);
290 }
291
292 static lx_autofs_lookup_req_t *
293 i_lalr_find(lx_autofs_vfs_t *data, int id)
294 {
295         lx_autofs_lookup_req_t  *lalr;
296
297         /* Check for an outstanding request for this id. */
298         mutex_enter(&data->lav_lock);
299         if (mod_hash_find(data->lav_id_hash, (mod_hash_key_t)(uintptr_t)id,
300             (mod_hash_val_t *)&lalr) != 0) {
301                 mutex_exit(&data->lav_lock);
302                 return (NULL);
303         }
304         atomic_add_int(&lalr->lalr_ref, 1);
305         mutex_exit(&data->lav_lock);
306         return (lalr);
307 }
308
309 static void
310 i_lalr_complete(lx_autofs_vfs_t *data, lx_autofs_lookup_req_t *lalr)
311 {
312         lx_autofs_lookup_req_t  *lalr_tmp;
313
314         /* Remove this request from the hashes so no one can look it up. */
315         mutex_enter(&data->lav_lock);
316         (void) mod_hash_remove(data->lav_id_hash,
317                     (mod_hash_key_t)(uintptr_t)lalr->lalr_pkt.lap_id,
318                     (mod_hash_val_t)&lalr_tmp);
319         (void) mod_hash_remove(data->lav_path_hash,
320                     (mod_hash_key_t)lalr->lalr_pkt.lap_name,
321                     (mod_hash_val_t)&lalr_tmp);
322         mutex_exit(&data->lav_lock);
323
324         /* Mark this requst as complete and wakeup anyone waiting on it. */
```



```
325         mutex_enter(&lalr->lalr_lock);
326         lalr->lalr_complete = 1;
327         cv_broadcast(&lalr->lalr_cv);
328         mutex_exit(&lalr->lalr_lock);
329 }
330
331 static void
332 i_lalr_release(lx_autofs_vfs_t *data, lx_autofs_lookup_req_t *lalr)
333 {
334         ASSERT(!MUTEX_HELD(&lalr->lalr_lock));
335         if (atomic_add_int_nv(&lalr->lalr_ref, -1) > 0)
336                 return;
337         ASSERT(lalr->lalr_ref == 0);
338         id_free(data->lav_ids, lalr->lalr_pkt.lap_id);
339         kmem_free(lalr, sizeof (*lalr));
340 }
341
342 static void
343 i_lalr_abort(lx_autofs_vfs_t *data, lx_autofs_lookup_req_t *lalr)
344 {
345         lx_autofs_lookup_req_t  *lalr_tmp;
346
347         /*
348          * This is a little tricky.  We're aborting the wait for this
349          * request.  So if anyone else is waiting for this request we
350          * can't free it, but if no one else is waiting for the request
351          * we should free it.
352          */
353         mutex_enter(&data->lav_lock);
354         if (atomic_add_int_nv(&lalr->lalr_ref, -1) > 0) {
355                 mutex_exit(&data->lav_lock);
356                 return;
357         }
358         ASSERT(lalr->lalr_ref == 0);
359
360         /* Remove this request from the hashes so no one can look it up. */
361         (void) mod_hash_remove(data->lav_id_hash,
362                     (mod_hash_key_t)(uintptr_t)lalr->lalr_pkt.lap_id,
363                     (mod_hash_val_t)&lalr_tmp);
364         (void) mod_hash_remove(data->lav_path_hash,
365                     (mod_hash_key_t)lalr->lalr_pkt.lap_name,
366                     (mod_hash_val_t)&lalr_tmp);
367         mutex_exit(&data->lav_lock);
368
369         /* It's ok to free this now because the ref count was zero. */
370         id_free(data->lav_ids, lalr->lalr_pkt.lap_id);
371         kmem_free(lalr, sizeof (*lalr));
372 }
373
374 static int
375 i_fifo_lookup(pid_t pgrp, int fd, file_t **fpp_wr, file_t **fpp_rd)
376 {
377         proc_t          *prp;
378         uf_info_t       *fip;
379         uf_entry_t      *ufp_wr, *ufp_rd = NULL;
380         file_t          *fp_wr, *fp_rd = NULL;
381         vnode_t         *vp_wr, *vp_rd;
382         int             i;
383
384         /*
385          * sprlock() is zone aware, so assuming this mount call was
386          * initiated by a process in a zone, if it tries to specify
387          * a pgrp outside of it's zone this call will fail.
388          *
389          * Also, we want to grab hold of the main automounter process
390          * and its going to be the group leader for pgrp, so its
```

```
391          * pid will be equal to pgrp.
392          */
393         prp = sprlock(pgrp);
394         if (prp == NULL)
395                 return (-1);
396         mutex_exit(&prp->p_lock);
397
398         /* Now we want to access the processes open file descriptors. */
399         fip = P_FINFO(prp);
400         mutex_enter(&fip->fi_lock);
401
402         /* Sanity check fifo write fd. */
403         if (fd >= fip->fi_nfiles) {
404                 mutex_exit(&fip->fi_lock);
405                 mutex_enter(&prp->p_lock);
406                 sprunlock(prp);
407                 return (-1);
408         }
409
410         /* Get a pointer to the write fifo. */
411         UF_ENTER(ufp_wr, fip, fd);
412         if (((fp_wr = ufp_wr->uf_file) == NULL) ||
413             ((vp_wr = fp_wr->f_vnode) == NULL) || (vp_wr->v_type != VFIFO)) {
414                 /* Invalid fifo fd. */
415                 UF_EXIT(ufp_wr);
416                 mutex_exit(&fip->fi_lock);
417                 mutex_enter(&prp->p_lock);
418                 sprunlock(prp);
419                 return (-1);
420         }
421
422         /*
423          * Now we need to find the read end of the fifo (for reasons
424          * explained below.)  We assume that the read end of the fifo
425          * is in the same process as the write end.
426          */
427         vp_rd = fifo_peer_vp(fp_wr->f_vnode);
428         for (i = 0; i < fip->fi_nfiles; i++) {
429                 UF_ENTER(ufp_rd, fip, i);
430                 if (((fp_rd = ufp_rd->uf_file) != NULL) &&
431                     (fp_rd->f_vnode == vp_rd))
432                         break;
433                 UF_EXIT(ufp_rd);
434         }
435         if (i == fip->fi_nfiles) {
436                 /* Didn't find it. */
437                 UF_EXIT(ufp_wr);
438                 mutex_exit(&fip->fi_lock);
439                 mutex_enter(&prp->p_lock);
440                 sprunlock(prp);
441                 return (-1);
442         }
443
444         /*
445          * We need to drop fi_lock before we can try to acquire f_tlock
446          * the good news is that the file pointers are protected because
447          * we're still holding uf_lock.
448          */
449         mutex_exit(&fip->fi_lock);
450
451         /*
452          * Here we bump the open counts on the fifos.  The reason
453          * that we do this is because when we go to write to the
454          * fifo we want to ensure that they are actually open (and
455          * not in the process of being closed) without having to
456          * stop the automounter.  (If the write end of the fifo
```



```
457          * were closed and we tried to write to it we would panic.
458          * If the read end of the fifo was closed and we tried to
459          * write to the other end, the process that invoked the
460          * lookup operation would get an unexpected SIGPIPE.)
461          */
462         mutex_enter(&fp_wr->f_tlock);
463         fp_wr->f_count++;
464         ASSERT(fp_wr->f_count >= 2);
465         mutex_exit(&fp_wr->f_tlock);
466
467         mutex_enter(&fp_rd->f_tlock);
468         fp_rd->f_count++;
469         ASSERT(fp_rd->f_count >= 2);
470         mutex_exit(&fp_rd->f_tlock);
471
472         /* Release all our locks. */
473         UF_EXIT(ufp_wr);
474         UF_EXIT(ufp_rd);
475         mutex_enter(&prp->p_lock);
476         sprunlock(prp);
477
478         /* Return the file pointers. */
479         *fpp_rd = fp_rd;
480         *fpp_wr = fp_wr;
481         return (0);
482 }
483
484 static uint_t
485 /*ARGSUSED*/
486 i_fifo_close_cb(mod_hash_key_t key, mod_hash_val_t *val, void *arg)
487 {
488         int     *id = (int *)arg;
489         /* Return the key and terminate the walk. */
490         *id = (uintptr_t)key;
491         return (MH_WALK_TERMINATE);
492 }
493
494 static void
495 i_fifo_close(lx_autofs_vfs_t *data)
496 {
497         /*
498          * Close the fifo to prevent any future requests from
499          * getting sent to the automounter.
500          */
501         mutex_enter(&data->lav_lock);
502         if (data->lav_fifo_wr != NULL) {
503                 (void) closef(data->lav_fifo_wr);
504                 data->lav_fifo_wr = NULL;
505         }
506         if (data->lav_fifo_rd != NULL) {
507                 (void) closef(data->lav_fifo_rd);
508                 data->lav_fifo_rd = NULL;
509         }
510         mutex_exit(&data->lav_lock);
511
512         /*
513          * Wakeup any threads currently waiting for the automounter
514          * note that it's possible for multiple threads to have entered
515          * this function and to be doing the work below simultaneously.
516          */
517         for (;;) {
518                 lx_autofs_lookup_req_t  *lalr;
519                 int                     id;
520
521                 /* Lookup the first entry in the hash. */
522                 id = -1;
```

```
523                 mod_hash_walk(data->lav_id_hash,
524                     i_fifo_close_cb, &id);
525                 if (id == -1) {
526                         /* No more id's in the hash. */
527                         break;
528                 }
529                 if ((lalr = i_lalr_find(data, id)) == NULL) {
530                         /* Someone else beat us to it. */
531                         continue;
532                 }

534                 /* Mark the request as compleate and release it. */
535                 i_lalr_complete(data, lalr);
536                 i_lalr_release(data, lalr);
537         }
538 }

540 static int
541 i_fifo_verify_rd(lx_autofs_vfs_t *data)
542 {
543         proc_t          *prp;
544         uf_info_t       *fip;
545         uf_entry_t      *ufp_rd = NULL;
546         file_t          *fp_rd =  NULL;
547         vnode_t         *vp_rd;
548         int             i;

550         ASSERT(MUTEX_HELD((&data->lav_lock)));

552         /* Check if we've already been shut down. */
553         if (data->lav_fifo_wr == NULL) {
554                 ASSERT(data->lav_fifo_rd == NULL);
555                 return (-1);
556         }
557         vp_rd = fifo_peer_vp(data->lav_fifo_wr->f_vnode);

559         /*
560          * sprlock() is zone aware, so assuming this mount call was
561          * initiated by a process in a zone, if it tries to specify
562          * a pgrp outside of it's zone this call will fail.
563          *
564          * Also, we want to grab hold of the main automounter process
565          * and its going to be the group leader for pgrp, so its
566          * pid will be equal to pgrp.
567          */
568         prp = sprlock(data->lav_pgrp);
569         if (prp == NULL)
570                 return (-1);
571         mutex_exit(&prp->p_lock);

573         /* Now we want to access the processes open file descriptors. */
574         fip = P_FINFO(prp);
575         mutex_enter(&fip->fi_lock);

577         /*
578          * Now we need to find the read end of the fifo (for reasons
579          * explained below.)  We assume that the read end of the fifo
580          * is in the same process as the write end.
581          */
582         for (i = 0; i < fip->fi_nfiles; i++) {
583                 UF_ENTER(ufp_rd, fip, i);
584                 if (((fp_rd = ufp_rd->uf_file) != NULL) &&
585                     (fp_rd->f_vnode == vp_rd))
586                         break;
587                 UF_EXIT(ufp_rd);
588         }
```



```
589         if (i == fip->fi_nfiles) {
590                 /* Didn't find it. */
591                 mutex_exit(&fip->fi_lock);
592                 mutex_enter(&prp->p_lock);
593                 sprunlock(prp);
594                 return (-1);
595         }

597         /*
598          * Seems the automounter still has the read end of the fifo
599          * open, we're done here.  Release all our locks and exit.
600          */
601         mutex_exit(&fip->fi_lock);
602         UF_EXIT(ufp_rd);
603         mutex_enter(&prp->p_lock);
604         sprunlock(prp);

606         return (0);
607 }

609 static int
610 i_fifo_write(lx_autofs_vfs_t *data, lx_autofs_pkt_t *lap)
611 {
612         struct uio      uio;
613         struct iovec    iov;
614         file_t          *fp_wr, *fp_rd;
615         int             error;

617         /*
618          * The catch here is we need to make sure _we_ don't close
619          * the the fifo while writing to it.  (Another thread could come
620          * along and realize the automounter process is gone and close
621          * the fifo.  To do this we bump the open count before we
622          * write to the fifo.
623          */
624         mutex_enter(&data->lav_lock);
625         if (data->lav_fifo_wr == NULL) {
626                 ASSERT(data->lav_fifo_rd == NULL);
627                 mutex_exit(&data->lav_lock);
628                 return (ENOENT);
629         }
630         fp_wr = data->lav_fifo_wr;
631         fp_rd = data->lav_fifo_rd;

633         /* Bump the open count on the write fifo. */
634         mutex_enter(&fp_wr->f_tlock);
635         fp_wr->f_count++;
636         mutex_exit(&fp_wr->f_tlock);

638         /* Bump the open count on the read fifo. */
639         mutex_enter(&fp_rd->f_tlock);
640         fp_rd->f_count++;
641         mutex_exit(&fp_rd->f_tlock);

643         mutex_exit(&data->lav_lock);

645         iov.iov_base = (caddr_t)lap;
646         iov.iov_len = sizeof (*lap);
647         uio.uio_iov = &iov;
648         uio.uio_iovcnt = 1;
649         uio.uio_loffset = 0;
650         uio.uio_segflg = (short)UIO_SYSSPACE;
651         uio.uio_resid = sizeof (*lap);
652         uio.uio_llimit = 0;
653         uio.uio_fmode = FWRITE | FNDELAY | FNONBLOCK;
```

```
655         error = VOP_WRITE(fp_wr->f_vnode, &uio, 0, kcred, NULL);
656         (void) closef(fp_wr);
657         (void) closef(fp_rd);

659         /*
660          * After every write we verify that the automounter still has
661          * these files open.
662          */
663         mutex_enter(&data->lav_lock);
664         if (i_fifo_verify_rd(data) != 0) {
665                 /*
666                  * Something happened to the automounter.
667                  * Close down the communication pipe we setup.
668                  */
669                 mutex_exit(&data->lav_lock);
670                 i_fifo_close(data);
671                 if (error != 0)
672                         return (error);
673                 return (ENOENT);
674         }
675         mutex_exit(&data->lav_lock);

677         return (error);
678 }

680 static int
681 i_bs_readdir(vnode_t *dvp, list_t *dir_stack, list_t *file_stack)
682 {
683         struct iovec    iov;
684         struct uio      uio;
685         dirent64_t      *dp, *dbuf;
686         vnode_t         *vp;
687         size_t          dlen, dbuflen;
688         int             eof, error, ndirents = 64;
689         char            *nm;

691         dlen = ndirents * (sizeof (*dbuf));
692         dbuf = kmem_alloc(dlen, KM_SLEEP);

694         uio.uio_iov = &iov;
695         uio.uio_iovcnt = 1;
696         uio.uio_segflg = UIO_SYSSPACE;
697         uio.uio_fmode = 0;
698         uio.uio_extflg = UIO_COPY_CACHED;
699         uio.uio_loffset = 0;
700         uio.uio_llimit = MAXOFFSET_T;

702         eof = 0;
703         error = 0;
704         while (!error && !eof) {
705                 uio.uio_resid = dlen;
706                 iov.iov_base = (char *)dbuf;
707                 iov.iov_len = dlen;

709                 (void) VOP_RWLOCK(dvp, V_WRITELOCK_FALSE, NULL);
710                 if (VOP_READDIR(dvp, &uio, kcred, &eof, NULL, 0) != 0) {
711                         VOP_RWUNLOCK(dvp, V_WRITELOCK_FALSE, NULL);
712                         kmem_free(dbuf, dlen);
713                         return (-1);
714                 }
715                 VOP_RWUNLOCK(dvp, V_WRITELOCK_FALSE, NULL);

717                 if ((dbuflen = dlen - uio.uio_resid) == 0) {
718                         /* We're done. */
719                         break;
720                 }
```



```
722                 for (dp = dbuf; ((intptr_t)dp < (intptr_t)dbuf + dbuflen);
723                         dp = (dirent64_t *)((intptr_t)dp + dp->d_reclen)) {

725                         nm = dp->d_name;

727                         if (strcmp(nm, ".") == 0 || strcmp(nm, "..") == 0)
728                                 continue;

730                         if (VOP_LOOKUP(dvp, nm, &vp, NULL, 0, NULL, kcred,
731                             NULL, NULL, NULL) != 0) {
732                                 kmem_free(dbuf, dlen);
733                                 return (-1);
734                         }
735                         if (vp->v_type == VDIR) {
736                                 if (dir_stack != NULL) {
737                                         i_stack_push(dir_stack, (caddr_t)dvp,
738                                             (caddr_t)vp, i_strdup(nm));
739                                 } else {
740                                         VN_RELE(vp);
741                                 }
742                         } else {
743                                 if (file_stack != NULL) {
744                                         i_stack_push(file_stack, (caddr_t)dvp,
745                                             (caddr_t)vp, i_strdup(nm));
746                                 } else {
747                                         VN_RELE(vp);
748                                 }
749                         }
750                 }
751         }
752         kmem_free(dbuf, dlen);
753         return (0);
754 }

756 static void
757 i_bs_destroy(vnode_t *dvp, char *path)
758 {
759         list_t  search_stack;
760         list_t  dir_stack;
761         list_t  file_stack;
762         vnode_t *pdvp, *vp;
763         char    *dpath, *fpath;
764         int     ret;

766         if (VOP_LOOKUP(dvp, path, &vp, NULL, 0, NULL, kcred,
767             NULL, NULL, NULL) != 0) {
768                 /* A directory entry with this name doesn't actually exist. */
769                 return;
770         }

772         if ((vp->v_type & VDIR) == 0) {
773                 /* Easy, the directory entry is a file so delete it. */
774                 VN_RELE(vp);
775                 (void) VOP_REMOVE(dvp, path, kcred, NULL, 0);
776                 return;
777         }

779         /*
780          * The directory entry is a subdirectory, now we have a bit more
781          * work to do.  (We'll have to recurse into the sub directory.)
782          * It would have been much easier to do this recursively but kernel
783          * stacks are notoriously small.
784          */
785         i_stack_init(&search_stack);
786         i_stack_init(&dir_stack);
```

```
787         i_stack_init(&file_stack);

789         /* Save our newfound subdirectory into a list. */
790         i_stack_push(&search_stack, (caddr_t)dvp, (caddr_t)vp, i_strdup(path));

792         /* Do a recursive depth first search into the subdirectories. */
793         while (i_stack_pop(&search_stack,
794             (caddr_t *)&pdvp, (caddr_t *)&dvp, &dpath) == 0) {

796                 /* Get a list of the subdirectories in this directory. */
797                 if (i_bs_readdir(dvp, &search_stack, NULL) != 0)
798                         goto exit;

800                 /* Save the current directory a separate stack. */
801                 i_stack_push(&dir_stack, (caddr_t)pdvp, (caddr_t)dvp, dpath);
802         }

804         /*
805          * Now dir_stack contains a list of directories, the deepest paths
806          * are at the top of the list.  So let's go through and process them.
807          */
808         while (i_stack_pop(&dir_stack,
809             (caddr_t *)&pdvp, (caddr_t *)&dvp, &dpath) == 0) {

811                 /* Get a list of the files in this directory. */
812                 if (i_bs_readdir(dvp, NULL, &file_stack) != 0) {
813                         VN_RELE(dvp);
814                         i_strfree(dpath);
815                         goto exit;
816                 }

818                 /* Delete all the files in this directory. */
819                 while (i_stack_pop(&file_stack,
820                     NULL, (caddr_t *)&vp, &fpath) == 0) {
821                         VN_RELE(vp)
822                         ret = VOP_REMOVE(dvp, fpath, kcred, NULL, 0);
823                         i_strfree(fpath);
824                         if (ret != 0) {
825                                 i_strfree(dpath);
826                                 goto exit;
827                         }
828                 }

830                 /* Delete this directory. */
831                 VN_RELE(dvp);
832                 ret = VOP_RMDIR(pdvp, dpath, pdvp, kcred, NULL, 0);
833                 i_strfree(dpath);
834                 if (ret != 0)
835                         goto exit;
836         }

838 exit:
839         while (
840             (i_stack_pop(&search_stack, NULL, (caddr_t *)&vp, &path) == 0) ||
841             (i_stack_pop(&dir_stack, NULL, (caddr_t *)&vp, &path) == 0) ||
842             (i_stack_pop(&file_stack, NULL, (caddr_t *)&vp, &path) == 0)) {
843                 VN_RELE(vp);
844                 i_strfree(path);
845         }
846         i_stack_fini(&search_stack);
847         i_stack_fini(&dir_stack);
848         i_stack_fini(&file_stack);
849 }

851 static vnode_t *
852 i_bs_create(vnode_t *dvp, char *bs_name)
```



```
853 {
854         vnode_t *vp;
855         vattr_t vattr;

857         /*
858          * After looking at the mkdir syscall path it seems we don't need
859          * to initialize all of the vattr_t structure.
860          */
861         bzero(&vattr, sizeof (vattr));
862         vattr.va_type = VDIR;
863         vattr.va_mode = 0755; /* u+rwx,og=rx */
864         vattr.va_mask = AT_TYPE|AT_MODE;

866         if (VOP_MKDIR(dvp, bs_name, &vattr, &vp, kcred, NULL, 0, NULL) != 0)
867                 return (NULL);
868         return (vp);
869 }

871 static int
872 i_automounter_call(vnode_t *dvp, char *nm)
873 {
874         lx_autofs_lookup_req_t  *lalr;
875         lx_autofs_vfs_t         *data;
876         int                     error, dup_request;

878         /* Get a pointer to the vfs mount data. */
879         data = dvp->v_vfsp->vfs_data;

881         /* The automounter only support queries in the root directory. */
882         if (dvp != data->lav_root)
883                 return (ENOENT);

885         /*
886          * Check if the current process is in the automounters process
887          * group.  (If it is, the current process is either the autmounter
888          * itself or one of it's forked child processes.)  If so, don't
889          * redirect this lookup back into the automounter because we'll
890          * hang.
891          */
892         mutex_enter(&pidlock);
893         if (data->lav_pgrp == curproc->p_pgrp) {
894                 mutex_exit(&pidlock);
895                 return (ENOENT);
896         }
897         mutex_exit(&pidlock);

899         /* Verify that the automount process pipe still exists. */
900         mutex_enter(&data->lav_lock);
901         if (data->lav_fifo_wr == NULL) {
902                 ASSERT(data->lav_fifo_rd == NULL);
903                 mutex_exit(&data->lav_lock);
904                 return (ENOENT);
905         }
906         mutex_exit(&data->lav_lock);

908         /* Allocate an automounter request structure. */
909         if ((lalr = i_lalr_alloc(data, &dup_request, nm)) == NULL)
910                 return (ENOENT);

912         /*
913          * If we were the first one to allocate this request then we
914          * need to send it to the automounter.
915          */
916         if ((!dup_request) &&
917             ((error = i_fifo_write(data, &lalr->lalr_pkt)) != 0)) {
918                 /*
```

```
                 * Unable to send the request to the automounter.
                 * Unblock any other threads waiting on the request
                 * and release the request.
                 */
                i_lalr_complete(data, lalr);
                i_lalr_release(data, lalr);
                return (error);
        }

        /* Wait for someone to signal us that this request has compleated. */
        mutex_enter(&lalr->lalr_lock);
        while (!lalr->lalr_complete) {
                if (cv_wait_sig(&lalr->lalr_cv, &lalr->lalr_lock) == 0) {
                        /* We got a signal, abort this lookup. */
                        mutex_exit(&lalr->lalr_lock);
                        i_lalr_abort(data, lalr);
                        return (EINTR);
                }
        }
        mutex_exit(&lalr->lalr_lock);
        i_lalr_release(data, lalr);

        return (0);
}

static int
i_automounter_ioctl(vnode_t *vp, int cmd, intptr_t arg)
{
        lx_autofs_vfs_t *data = (lx_autofs_vfs_t *)vp->v_vfsp->vfs_data;

        /*
         * Be strict.
         * We only accept ioctls from the automounter process group.
         */
        mutex_enter(&pidlock);
        if (data->lav_pgrp != curproc->p_pgrp) {
                mutex_exit(&pidlock);
                return (ENOENT);
        }
        mutex_exit(&pidlock);

        if ((cmd == LX_AUTOFS_IOC_READY) || (cmd == LX_AUTOFS_IOC_FAIL)) {
                lx_autofs_lookup_req_t  *lalr;
                int                     id = arg;

                /*
                 * We don't actually care if the request failed or succeeded.
                 * We do the same thing either way.
                 */
                if ((lalr = i_lalr_find(data, id)) == NULL)
                        return (ENXIO);

                /* Mark the request as compleate and release it. */
                i_lalr_complete(data, lalr);
                i_lalr_release(data, lalr);
                return (0);
        }
        if (cmd == LX_AUTOFS_IOC_CATATONIC) {
                /* The automounter is shutting down. */
                i_fifo_close(data);
                return (0);
        }
        return (ENOTSUP);
}

static int
```



```
i_parse_mntopt(vfs_t *vfsp, lx_autofs_vfs_t *data)
{
        char            *fd_str, *pgrp_str, *minproto_str, *maxproto_str;
        int             fd, pgrp, minproto, maxproto;
        file_t          *fp_wr, *fp_rd;

        /* Require all options to be present. */
        if ((vfs_optionisset(vfsp, LX_MNTOPT_FD, &fd_str) != 1) ||
            (vfs_optionisset(vfsp, LX_MNTOPT_PGRP, &pgrp_str) != 1) ||
            (vfs_optionisset(vfsp, LX_MNTOPT_MINPROTO, &minproto_str) != 1) ||
            (vfs_optionisset(vfsp, LX_MNTOPT_MAXPROTO, &maxproto_str) != 1))
                return (EINVAL);

        /* Get the values for each parameter. */
        if ((i_str_to_int(fd_str, &fd) != 0) ||
            (i_str_to_int(pgrp_str, &pgrp) != 0) ||
            (i_str_to_int(minproto_str, &minproto) != 0) ||
            (i_str_to_int(maxproto_str, &maxproto) != 0))
                return (EINVAL);

        /*
         * We support v2 of the linux kernel automounter protocol.
         * Make sure the mount request we got indicates support
         * for this version of the protocol.
         */
        if ((minproto > 2) || (maxproto < 2))
                return (EINVAL);

        /*
         * Now we need to lookup the fifos we'll be using
         * to talk to the userland automounter process.
         */
        if (i_fifo_lookup(pgrp, fd, &fp_wr, &fp_rd) != 0)
                return (EINVAL);

        /* Save the mount options and fifo pointers. */
        data->lav_fd = fd;
        data->lav_pgrp = pgrp;
        data->lav_fifo_rd = fp_rd;
        data->lav_fifo_wr = fp_wr;
        return (0);
}

/*
 * VFS entry points
 */
static int
lx_autofs_mount(vfs_t *vfsp, vnode_t *mvp, struct mounta *uap, cred_t *cr)
{
        lx_autofs_vfs_t *data;
        dev_t           dev;
        char            name[40];
        int             error;

        if (secpolicy_fs_mount(cr, mvp, vfsp) != 0)
                return (EPERM);

        if (mvp->v_type != VDIR)
                return (ENOTDIR);

        if ((uap->flags & MS_OVERLAY) == 0 &&
            (mvp->v_count > 1 || (mvp->v_flag & VROOT)))
                return (EBUSY);

        /* We don't support mountes in the global zone. */
        if (getzoneid() == GLOBAL_ZONEID)
```

```
1051                return (EPERM);

1053        /* We don't support mounting on top of ourselves. */
1054        if (vn_matchops(mvp, lx_autofs_vn_ops))
1055                return (EPERM);

1057        /* Allocate a vfs struct. */
1058        data = kmem_zalloc(sizeof (lx_autofs_vfs_t), KM_SLEEP);

1060        /* Parse mount options. */
1061        if ((error = i_parse_mntopt(vfsp, data)) != 0) {
1062                kmem_free(data, sizeof (lx_autofs_vfs_t));
1063                return (error);
1064        }

1066        /* Initialize the backing store. */
1067        i_bs_destroy(mvp, LX_AUTOFS_BS_DIR);
1068        if ((data->lav_bs_vp = i_bs_create(mvp, LX_AUTOFS_BS_DIR)) == NULL) {
1069                kmem_free(data, sizeof (lx_autofs_vfs_t));
1070                return (EBUSY);
1071        }
1072        data->lav_bs_name = LX_AUTOFS_BS_DIR;

1074        /* We have to hold the underlying vnode we're mounted on. */
1075        data->lav_mvp = mvp;
1076        VN_HOLD(mvp);

1078        /* Initialize vfs fields */
1079        vfsp->vfs_bsize = DEV_BSIZE;
1080        vfsp->vfs_fstype = lx_autofs_fstype;
1081        vfsp->vfs_data = data;

1083        /* Invent a dev_t (sigh) */
1084        do {
1085                dev = makedevice(lx_autofs_major,
1086                    atomic_add_32_nv(&lx_autofs_minor, 1) & L_MAXMIN32);
1087        } while (vfs_devismounted(dev));
1088        vfsp->vfs_dev = dev;
1089        vfs_make_fsid(&vfsp->vfs_fsid, dev, lx_autofs_fstype);

1091        /* Create an id space arena for automounter requests. */
1092        (void) snprintf(name, sizeof (name), "lx_autofs_id_%d",
1093            getminor(vfsp->vfs_dev));
1094        data->lav_ids = id_space_create(name, 1, INT_MAX);

1096        /* Create hashes to keep track of automounter requests. */
1097        mutex_init(&data->lav_lock, NULL, MUTEX_DEFAULT, NULL);
1098        (void) snprintf(name, sizeof (name), "lx_autofs_path_hash_%d",
1099            getminor(vfsp->vfs_dev));
1100        data->lav_path_hash = mod_hash_create_strhash(name,
1101            LX_AUTOFS_VFS_PATH_HASH_SIZE, mod_hash_null_valdtor);
1102        (void) snprintf(name, sizeof (name), "lx_autofs_id_hash_%d",
1103            getminor(vfsp->vfs_dev));
1104        data->lav_id_hash = mod_hash_create_idhash(name,
1105            LX_AUTOFS_VFS_ID_HASH_SIZE, mod_hash_null_valdtor);

1107        /* Create a hash to keep track of vnodes. */
1108        (void) snprintf(name, sizeof (name), "lx_autofs_vn_hash_%d",
1109            getminor(vfsp->vfs_dev));
1110        data->lav_vn_hash = mod_hash_create_ptrhash(name,
1111            LX_AUTOFS_VFS_VN_HASH_SIZE, mod_hash_null_valdtor,
1112            sizeof (vnode_t));

1114        /* Create root vnode */
1115        data->lav_root = i_vn_alloc(vfsp, data->lav_bs_vp);
1116        data->lav_root->v_flag |=
```



```
1117                VROOT | VNOCACHE | VNOMAP | VNOSWAP | VNOMOUNT;

1119        return (0);
1120 }

1122 static int
1123 lx_autofs_unmount(vfs_t *vfsp, int flag, struct cred *cr)
1124 {
1125        lx_autofs_vfs_t *data;

1127        if (secpolicy_fs_unmount(cr, vfsp) != 0)
1128                return (EPERM);

1130        /* We do not currently support forced unmounts. */
1131        if (flag & MS_FORCE)
1132                return (ENOTSUP);

1134        /*
1135         * We should never have a reference count of less than 2: one for the
1136         * caller, one for the root vnode.
1137         */
1138        ASSERT(vfsp->vfs_count >= 2);

1140        /* If there are any outstanding vnodes, we can't unmount. */
1141        if (vfsp->vfs_count > 2)
1142                return (EBUSY);

1144        /* Check for any remaining holds on the root vnode. */
1145        data = vfsp->vfs_data;
1146        ASSERT(data->lav_root->v_vfsp == vfsp);
1147        if (data->lav_root->v_count > 1)
1148                return (EBUSY);

1150        /* Close the fifo to the automount process. */
1151        if (data->lav_fifo_wr != NULL)
1152                (void) closef(data->lav_fifo_wr);
1153        if (data->lav_fifo_rd != NULL)
1154                (void) closef(data->lav_fifo_rd);

1156        /*
1157         * We have to release our hold on our root vnode before we can
1158         * delete the backing store.  (Since the root vnode is linked
1159         * to the backing store.)
1160         */
1161        VN_RELE(data->lav_root);

1163        /* Cleanup the backing store. */
1164        i_bs_destroy(data->lav_mvp, data->lav_bs_name);
1165        VN_RELE(data->lav_mvp);

1167        /* Cleanup out remaining data structures. */
1168        mod_hash_destroy_strhash(data->lav_path_hash);
1169        mod_hash_destroy_idhash(data->lav_id_hash);
1170        mod_hash_destroy_ptrhash(data->lav_vn_hash);
1171        id_space_destroy(data->lav_ids);
1172        kmem_free(data, sizeof (lx_autofs_vfs_t));

1174        return (0);
1175 }

1177 static int
1178 lx_autofs_root(vfs_t *vfsp, vnode_t **vpp)
1179 {
1180        lx_autofs_vfs_t *data = vfsp->vfs_data;

1182        *vpp = data->lav_root;
```

```
1183         VN_HOLD(*vpp);

1185         return (0);
1186 }

1188 static int
1189 lx_autofs_statvfs(vfs_t *vfsp, statvfs64_t *sp)
1190 {
1191         lx_autofs_vfs_t *data = vfsp->vfs_data;
1192         vnode_t         *urvp = data->lav_root->v_data;
1193         dev32_t         d32;
1194         int             error;

1196         if ((error = VFS_STATVFS(urvp->v_vfsp, sp)) != 0)
1197                 return (error);

1199         /* Update some of values before returning. */
1200         (void) cmpldev(&d32, vfsp->vfs_dev);
1201         sp->f_fsid = d32;
1202         (void) strlcpy(sp->f_basetype, vfssw[vfsp->vfs_fstype].vsw_name,
1203             sizeof (sp->f_basetype));
1204         sp->f_flag = vf_to_stf(vfsp->vfs_flag);
1205         bzero(sp->f_fstr, sizeof (sp->f_fstr));
1206         return (0);
1207 }

1209 static const fs_operation_def_t lx_autofs_vfstops[] = {
1210         { VFSNAME_MOUNT,        { .vfs_mount = lx_autofs_mount } },
1211         { VFSNAME_UNMOUNT,      { .vfs_unmount = lx_autofs_unmount } },
1212         { VFSNAME_ROOT,         { .vfs_root = lx_autofs_root } },
1213         { VFSNAME_STATVFS,      { .vfs_statvfs = lx_autofs_statvfs } },
1214         { NULL, NULL }
1215 };

1217 /*
1218  * VOP entry points - simple passthrough
1219  *
1220  * For most VOP entry points we can simply pass the request on to
1221  * the underlying filesystem we're mounted on.
1222  */
1223 static int
1224 lx_autofs_close(vnode_t *vp, int flag, int count, offset_t offset, cred_t *cr,
1225     caller_context_t *ctp)
1226 {
1227         vnode_t *uvp = vp->v_data;
1228         return (VOP_CLOSE(uvp, flag, count, offset, cr, ctp));
1229 }

1231 static int
1232 lx_autofs_readdir(vnode_t *vp, uio_t *uiop, cred_t *cr, int *eofp,
1233     caller_context_t *ctp, int flags)
1234 {
1235         vnode_t *uvp = vp->v_data;
1236         return (VOP_READDIR(uvp, uiop, cr, eofp, ctp, flags));
1237 }

1239 static int
1240 lx_autofs_access(vnode_t *vp, int mode, int flags, cred_t *cr,
1241     caller_context_t *ctp)
1242 {
1243         vnode_t *uvp = vp->v_data;
1244         return (VOP_ACCESS(uvp, mode, flags, cr, ctp));
1245 }

1247 static int
1248 lx_autofs_rwlock(struct vnode *vp, int write_lock, caller_context_t *ctp)
```



```
1249 {
1250         vnode_t *uvp = vp->v_data;
1251         return (VOP_RWLOCK(uvp, write_lock, ctp));
1252 }

1254 static void
1255 lx_autofs_rwunlock(struct vnode *vp, int write_lock, caller_context_t *ctp)
1256 {
1257         vnode_t *uvp = vp->v_data;
1258         VOP_RWUNLOCK(uvp, write_lock, ctp);
1259 }

1261 /*ARGSUSED*/
1262 static int
1263 lx_autofs_rmdir(vnode_t *dvp, char *nm, vnode_t *cdir, cred_t *cr,
1264     caller_context_t *ctp, int flags)
1265 {
1266         vnode_t *udvp = dvp->v_data;

1268         /*
1269          * cdir is the calling processes current directory.
1270          * If cdir is lx_autofs vnode then get its real underlying
1271          * vnode ptr.  (It seems like the only thing cdir is
1272          * ever used for is to make sure the user doesn't delete
1273          * their current directory.)
1274          */
1275         if (vn_matchops(cdir, lx_autofs_vn_ops)) {
1276                 vnode_t *ucdir = cdir->v_data;
1277                 return (VOP_RMDIR(udvp, nm, ucdir, cr, ctp, flags));
1278         }

1280         return (VOP_RMDIR(udvp, nm, cdir, cr, ctp, flags));
1281 }

1283 /*
1284  * VOP entry points - special passthrough
1285  *
1286  * For some VOP entry points we will first pass the request on to
1287  * the underlying filesystem we're mounted on.  If there's an error
1288  * then we immediately return the error, but if the request succeeds
1289  * we have to do some extra work before returning.
1290  */
1291 static int
1292 lx_autofs_open(vnode_t **vpp, int flag, cred_t *cr, caller_context_t *ctp)
1293 {
1294         vnode_t         *ovp = *vpp;
1295         vnode_t         *uvp = ovp->v_data;
1296         int             error;

1298         if ((error = VOP_OPEN(&uvp, flag, cr, ctp)) != 0)
1299                 return (error);

1301         /* Check for clone opens. */
1302         if (uvp == ovp->v_data)
1303                 return (0);

1305         /* Deal with clone opens by returning a new vnode. */
1306         *vpp = i_vn_alloc(ovp->v_vfsp, uvp);
1307         VN_RELE(ovp);
1308         return (0);
1309 }

1311 static int
1312 lx_autofs_getattr(vnode_t *vp, vattr_t *vap, int flags, cred_t *cr,
1313     caller_context_t *ctp)
1314 {
```

```
1315         vnode_t         *uvp = vp->v_data;
1316         int             error;

1318         if ((error = VOP_GETATTR(uvp, vap, flags, cr, ctp)) != 0)
1319                 return (error);

1321         /* Update the attributes with our filesystem id. */
1322         vap->va_fsid = vp->v_vfsp->vfs_dev;
1323         return (0);
1324 }

1326 static int
1327 lx_autofs_mkdir(vnode_t *dvp, char *nm, struct vattr *vap, vnode_t **vpp,
1328     cred_t *cr, caller_context_t *ctp, int flags, vsecattr_t *vsecp)
1329 {
1330         vnode_t         *udvp = dvp->v_data;
1331         vnode_t         *uvp = NULL;
1332         int             error;

1334         if ((error = VOP_MKDIR(udvp, nm, vap, &uvp, cr,
1335             ctp, flags, vsecp)) != 0)
1336                 return (error);

1338         /* Update the attributes with our filesystem id. */
1339         vap->va_fsid = dvp->v_vfsp->vfs_dev;

1341         /* Allocate a new vnode. */
1342         *vpp = i_vn_alloc(dvp->v_vfsp, uvp);
1343         return (0);
1344 }

1346 /*
1347  * VOP entry points - custom
1348  */
1349 /*ARGSUSED*/
1350 static void
1351 lx_autofs_inactive(struct vnode *vp, struct cred *cr, caller_context_t *ctp)
1352 {
1353         lx_autofs_vfs_t *data = vp->v_vfsp->vfs_data;

1355         /*
1356          * We need to hold the vfs lock because if we're going to free
1357          * this vnode we have to prevent anyone from looking it up
1358          * in the vnode hash.
1359          */
1360         mutex_enter(&data->lav_lock);
1361         mutex_enter(&vp->v_lock);

1363         if (vp->v_count < 1) {
1364                 panic("lx_autofs_inactive: bad v_count");
1365                 /*NOTREACHED*/
1366         }

1368         /* Drop the temporary hold by vn_rele now. */
1369         if (--vp->v_count > 0) {
1370                 mutex_exit(&vp->v_lock);
1371                 mutex_exit(&data->lav_lock);
1372                 return;
1373         }

1375         /*
1376          * No one should have been blocked on this lock because we're
1377          * about to free this vnode.
1378          */
1379         i_vn_free(vp);
1380 }
```



```
1382 static int
1383 lx_autofs_lookup(vnode_t *dvp, char *nm, vnode_t **vpp, struct pathname *pnp,
1384     int flags, vnode_t *rdir, cred_t *cr, caller_context_t *ctp,
1385     int *direntflags, pathname_t *realpnp)
1386 {
1387         vnode_t                 *udvp = dvp->v_data;
1388         vnode_t                 *uvp = NULL;
1389         int                     error;

1391         /* First try to lookup if this path component already exitst. */
1392         if ((error = VOP_LOOKUP(udvp, nm, &uvp, pnp, flags, rdir, cr, ctp,
1393             direntflags, realpnp)) == 0) {
1394                 *vpp = i_vn_alloc(dvp->v_vfsp, uvp);
1395                 return (0);
1396         }

1398         /* Only query the automounter if the path does not exist. */
1399         if (error != ENOENT)
1400                 return (error);

1402         /* Refer the lookup to the automounter. */
1403         if ((error = i_automounter_call(dvp, nm)) != 0)
1404                 return (error);

1406         /* Retry the lookup operation. */
1407         if ((error = VOP_LOOKUP(udvp, nm, &uvp, pnp, flags, rdir, cr, ctp,
1408             direntflags, realpnp)) == 0) {
1409                 *vpp = i_vn_alloc(dvp->v_vfsp, uvp);
1410                 return (0);
1411         }
1412         return (error);
1413 }

1415 /*ARGSUSED*/
1416 static int
1417 lx_autofs_ioctl(vnode_t *vp, int cmd, intptr_t arg, int mode, cred_t *cr,
1418     int *rvalp, caller_context_t *ctp)
1419 {
1420         vnode_t                 *uvp = vp->v_data;

1422         /* Intercept certain ioctls. */
1423         switch ((uint_t)cmd) {
1424         case LX_AUTOFS_IOC_READY:
1425         case LX_AUTOFS_IOC_FAIL:
1426         case LX_AUTOFS_IOC_CATATONIC:
1427         case LX_AUTOFS_IOC_EXPIRE:
1428         case LX_AUTOFS_IOC_PROTOVER:
1429         case LX_AUTOFS_IOC_SETTIMEOUT:
1430                 return (i_automounter_ioctl(vp, cmd, arg));
1431         }

1433         /* Pass any remaining ioctl on. */
1434         return (VOP_IOCTL(uvp, cmd, arg, mode, cr, rvalp, ctp));
1435 }

1437 /*
1438  * VOP entry points definitions
1439  */
1440 static const fs_operation_def_t lx_autofs_tops_root[] = {
1441         { VOPNAME_OPEN,         { .vop_open = lx_autofs_open } },
1442         { VOPNAME_CLOSE,        { .vop_close = lx_autofs_close } },
1443         { VOPNAME_IOCTL,        { .vop_ioctl = lx_autofs_ioctl } },
1444         { VOPNAME_RWLOCK,       { .vop_rwlock = lx_autofs_rwlock } },
1445         { VOPNAME_RWUNLOCK,     { .vop_rwunlock = lx_autofs_rwunlock } },
1446         { VOPNAME_GETATTR,      { .vop_getattr = lx_autofs_getattr } },
```

```
1447         { VOPNAME_ACCESS,       { .vop_access = lx_autofs_access } },
1448         { VOPNAME_READDIR,      { .vop_readdir = lx_autofs_readdir } },
1449         { VOPNAME_LOOKUP,       { .vop_lookup = lx_autofs_lookup } },
1450         { VOPNAME_INACTIVE,     { .vop_inactive = lx_autofs_inactive } },
1451         { VOPNAME_MKDIR,        { .vop_mkdir = lx_autofs_mkdir } },
1452         { VOPNAME_RMDIR,        { .vop_rmdir = lx_autofs_rmdir } },
1453         { NULL }
1454 };

1456 /*
1457  * lx_autofs_init() gets invoked via the mod_install() call in
1458  * this modules _init() routine.  Therefor, the code that cleans
1459  * up the structures we allocate below is actually found in
1460  * our _fini() routine.
1461  */
1462 /* ARGSUSED */
1463 static int
1464 lx_autofs_init(int fstype, char *name)
1465 {
1466         int             error;

1468         if ((lx_autofs_major =
1469             (major_t)space_fetch(LX_AUTOFS_SPACE_KEY_UDEV)) == 0) {

1471                 if ((lx_autofs_major = getudev()) == (major_t)-1) {
1472                         cmn_err(CE_WARN, "lx_autofs_init: "
1473                             "can't get unique device number");
1474                         return (EAGAIN);
1475                 }

1477                 if (space_store(LX_AUTOFS_SPACE_KEY_UDEV,
1478                     (uintptr_t)lx_autofs_major) != 0) {
1479                         cmn_err(CE_WARN, "lx_autofs_init: "
1480                             "can't save unique device number");
1481                         return (EAGAIN);
1482                 }
1483         }

1485         lx_autofs_fstype = fstype;
1486         if ((error = vfs_setfsops(
1487             fstype, lx_autofs_vfstops, &lx_autofs_vfsops)) != 0) {
1488                 cmn_err(CE_WARN, "lx_autofs_init: bad vfs ops template");
1489                 return (error);
1490         }

1492         if ((error = vn_make_ops("lx_autofs vnode ops",
1493             lx_autofs_tops_root, &lx_autofs_vn_ops)) != 0) {
1494                 VERIFY(vfs_freevfsops_by_type(fstype) == 0);
1495                 lx_autofs_vn_ops = NULL;
1496                 return (error);
1497         }

1499         return (0);
1500 }


1503 /*
1504  * Module linkage
1505  */
1506 static mntopt_t lx_autofs_mntopt[] = {
1507         { LX_MNTOPT_FD,         NULL,   0,      MO_HASVALUE },
1508         { LX_MNTOPT_PGRP,       NULL,   0,      MO_HASVALUE },
1509         { LX_MNTOPT_MINPROTO,   NULL,   0,      MO_HASVALUE },
1510         { LX_MNTOPT_MAXPROTO,   NULL,   0,      MO_HASVALUE }
1511 };
```



```
1513 static mntopts_t lx_autofs_mntopts = {
1514         sizeof (lx_autofs_mntopt) / sizeof (mntopt_t),
1515         lx_autofs_mntopt
1516 };

1518 static vfsdef_t vfw = {
1519         VFSDEF_VERSION,
1520         LX_AUTOFS_NAME,
1521         lx_autofs_init,
1522         VSW_HASPROTO | VSW_VOLATILEDEV,
1523         &lx_autofs_mntopts
1524 };

1526 extern struct mod_ops mod_fsops;

1528 static struct modlfs modlfs = {
1529         &mod_fsops, "linux autofs filesystem", &vfw
1530 };

1532 static struct modlinkage modlinkage = {
1533         MODREV_1, (void *)&modlfs, NULL
1534 };

1536 int
1537 _init(void)
1538 {
1539         return (mod_install(&modlinkage));
1540 }

1542 int
1543 _info(struct modinfo *modinfop)
1544 {
1545         return (mod_info(&modlinkage, modinfop));
1546 }

1548 int
1549 _fini(void)
1550 {
1551         int             error;

1553         if ((error = mod_remove(&modlinkage)) != 0)
1554                 return (error);

1556         if (lx_autofs_vn_ops != NULL) {
1557                 vn_freevnodeops(lx_autofs_vn_ops);
1558                 lx_autofs_vn_ops = NULL;
1559         }

1561         /*
1562          * In our init routine, if we get an error after calling
1563          * vfs_setfsops() we cleanup by calling vfs_freevfsops_by_type().
1564          * But we don't need to call vfs_freevfsops_by_type() here
1565          * because the fs framework did this for us as part of the
1566          * mod_remove() call above.
1567          */
1568         return (0);
1569 }
1570 #endif /* ! codereview */
```

```
**********************************************************
    9240 Tue Jan 14 16:17:17 2014
new/usr/src/uts/common/brand/lx/dtrace/lx_systrace.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */


#include <sys/modctl.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/stat.h>
#include <sys/conf.h>
#include <sys/frame.h>
#include <sys/dtrace.h>
#include <sys/dtrace_impl.h>

#include <sys/lx_impl.h>

#define LX_SYSTRACE_SHIFT       16
#define LX_SYSTRACE_ISENTRY(x)  ((int)(x) >> LX_SYSTRACE_SHIFT)
#define LX_SYSTRACE_SYSNUM(x)   ((int)(x) & ((1 << LX_SYSTRACE_SHIFT) - 1))
#define LX_SYSTRACE_ENTRY(id)   ((1 << LX_SYSTRACE_SHIFT) | (id))
#define LX_SYSTRACE_RETURN(id)  (id)

#define LX_SYSTRACE_ENTRY_AFRAMES       2
#define LX_SYSTRACE_RETURN_AFRAMES      4

typedef struct lx_systrace_sysent {
        const char *lss_name;
        dtrace_id_t lss_entry;
        dtrace_id_t lss_return;
} lx_systrace_sysent_t;

static dev_info_t *lx_systrace_devi;
static dtrace_provider_id_t lx_systrace_id;
static kmutex_t lx_systrace_lock;
static uint_t lx_systrace_nenabled;

static int lx_systrace_nsysent;
static lx_systrace_sysent_t *lx_systrace_sysent;
```



```
/*ARGSUSED*/
static void
lx_systrace_entry(ulong_t sysnum, ulong_t arg0, ulong_t arg1, ulong_t arg2,
    ulong_t arg3, ulong_t arg4, ulong_t arg5)
{
        dtrace_id_t id;

        if (sysnum >= lx_systrace_nsysent)
                return;

        if ((id = lx_systrace_sysent[sysnum].lss_entry) == DTRACE_IDNONE)
                return;

        dtrace_probe(id, arg0, arg1, arg2, arg3, arg4);
}

/*ARGSUSED*/
static void
lx_systrace_return(ulong_t sysnum, ulong_t arg0, ulong_t arg1, ulong_t arg2,
    ulong_t arg3, ulong_t arg4, ulong_t arg5)
{
        dtrace_id_t id;

        if (sysnum >= lx_systrace_nsysent)
                return;

        if ((id = lx_systrace_sysent[sysnum].lss_return) == DTRACE_IDNONE)
                return;

        dtrace_probe(id, arg0, arg1, arg2, arg3, arg4);
}

/*ARGSUSED*/
static void
lx_systrace_provide(void *arg, const dtrace_probedesc_t *desc)
{
        int i;

        if (desc != NULL)
                return;

        for (i = 0; i < lx_systrace_nsysent; i++) {
                if (dtrace_probe_lookup(lx_systrace_id, NULL,
                    lx_systrace_sysent[i].lss_name, "entry") != 0)
                        continue;

                (void) dtrace_probe_create(lx_systrace_id, NULL,
                    lx_systrace_sysent[i].lss_name, "entry",
                    LX_SYSTRACE_ENTRY_AFRAMES,
                    (void *)((uintptr_t)LX_SYSTRACE_ENTRY(i)));

                (void) dtrace_probe_create(lx_systrace_id, NULL,
                    lx_systrace_sysent[i].lss_name, "return",
                    LX_SYSTRACE_RETURN_AFRAMES,
                    (void *)((uintptr_t)LX_SYSTRACE_RETURN(i)));

                lx_systrace_sysent[i].lss_entry = DTRACE_IDNONE;
                lx_systrace_sysent[i].lss_return = DTRACE_IDNONE;
        }
}

/*ARGSUSED*/
static int
lx_systrace_enable(void *arg, dtrace_id_t id, void *parg)
{
        int sysnum = LX_SYSTRACE_SYSNUM((uintptr_t)parg);
```

```
129         ASSERT(sysnum < lx_systrace_nsysent);
131         mutex_enter(&lx_systrace_lock);
132         if (lx_systrace_nenabled++ == 0)
133                 lx_brand_systrace_enable();
134         mutex_exit(&lx_systrace_lock);
136         if (LX_SYSTRACE_ISENTRY((uintptr_t)parg)) {
137                 lx_systrace_sysent[sysnum].lss_entry = id;
138         } else {
139                 lx_systrace_sysent[sysnum].lss_return = id;
140         }
141         return (0);
142 }
144 /*ARGSUSED*/
145 static void
146 lx_systrace_disable(void *arg, dtrace_id_t id, void *parg)
147 {
148         int sysnum = LX_SYSTRACE_SYSNUM((uintptr_t)parg);
150         ASSERT(sysnum < lx_systrace_nsysent);
152         if (LX_SYSTRACE_ISENTRY((uintptr_t)parg)) {
153                 lx_systrace_sysent[sysnum].lss_entry = DTRACE_IDNONE;
154         } else {
155                 lx_systrace_sysent[sysnum].lss_return = DTRACE_IDNONE;
156         }
158         mutex_enter(&lx_systrace_lock);
159         if (--lx_systrace_nenabled == 0)
160                 lx_brand_systrace_disable();
161         mutex_exit(&lx_systrace_lock);
162 }
164 /*ARGSUSED*/
165 static void
166 lx_systrace_destroy(void *arg, dtrace_id_t id, void *parg)
167 {
168 }
170 /*ARGSUSED*/
171 static uint64_t
172 lx_systrace_getarg(void *arg, dtrace_id_t id, void *parg, int argno,
173     int aframes)
174 {
175         struct frame *fp = (struct frame *)dtrace_getfp();
176         uintptr_t *stack;
177         uint64_t val = 0;
178         int i;
180         if (argno >= 6)
181                 return (0);
183         /*
184          * Walk the four frames down the stack to the entry or return callback.
185          * Our callback calls dtrace_probe() which calls dtrace_dif_variable()
186          * which invokes this function to get the extended arguments. We get
187          * the frame pointer in via call to dtrace_getfp() above which makes for
188          * four frames.
189          */
190         for (i = 0; i < 4; i++) {
191                 fp = (struct frame *)fp->fr_savfp;
192         }
```



```
194         stack = (uintptr_t *)&fp[1];
196         /*
197          * Skip the first argument to the callback -- the system call number.
198          */
199         argno++;
201 #ifdef __amd64
202         /*
203          * On amd64, the first 6 arguments are passed in registers while
204          * subsequent arguments are on the stack.
205          */
206         argno -= 6;
207 #endif
209         DTRACE_CPUFLAG_SET(CPU_DTRACE_NOFAULT);
210         val = stack[argno];
211         DTRACE_CPUFLAG_CLEAR(CPU_DTRACE_NOFAULT);
213         return (val);
214 }
217 static const dtrace_pattr_t lx_systrace_attr = {
218 { DTRACE_STABILITY_EVOLVING, DTRACE_STABILITY_EVOLVING, DTRACE_CLASS_COMMON },
219 { DTRACE_STABILITY_PRIVATE, DTRACE_STABILITY_PRIVATE, DTRACE_CLASS_UNKNOWN },
220 { DTRACE_STABILITY_PRIVATE, DTRACE_STABILITY_PRIVATE, DTRACE_CLASS_ISA },
221 { DTRACE_STABILITY_EVOLVING, DTRACE_STABILITY_EVOLVING, DTRACE_CLASS_COMMON },
222 { DTRACE_STABILITY_PRIVATE, DTRACE_STABILITY_PRIVATE, DTRACE_CLASS_ISA },
223 };
225 static dtrace_pops_t lx_systrace_pops = {
226         lx_systrace_provide,
227         NULL,
228         lx_systrace_enable,
229         lx_systrace_disable,
230         NULL,
231         NULL,
232         NULL,
233         lx_systrace_getarg,
234         NULL,
235         lx_systrace_destroy
236 };
238 static int
239 lx_systrace_attach(dev_info_t *devi, ddi_attach_cmd_t cmd)
240 {
241         int i;
243         switch (cmd) {
244         case DDI_ATTACH:
245                 break;
246         case DDI_RESUME:
247                 return (DDI_SUCCESS);
248         default:
249                 return (DDI_FAILURE);
250         }
252         if (ddi_create_minor_node(devi, "lx_systrace", S_IFCHR,
253             0, DDI_PSEUDO, NULL) == DDI_FAILURE ||
254             dtrace_register("lx-syscall", &lx_systrace_attr,
255             DTRACE_PRIV_KERNEL, 0, &lx_systrace_pops, NULL,
256             &lx_systrace_id) != 0) {
257                 ddi_remove_minor_node(devi, NULL);
258                 return (DDI_FAILURE);
259         }
```

```
261         ddi_report_dev(devi);
262         lx_systrace_devi = devi;
263
264         /*
265          * Count up the lx_brand system calls.
266          */
267         for (i = 0; lx_sysent[i].sy_callc != NULL; i++)
268                 continue;
269
270         /*
271          * Initialize our corresponding table.
272          */
273         lx_systrace_sysent = kmem_zalloc(i * sizeof (lx_systrace_sysent_t),
274             KM_SLEEP);
275         lx_systrace_nsysent = i;
276
277         for (i = 0; i < lx_systrace_nsysent; i++) {
278                 lx_systrace_sysent[i].lss_name = lx_sysent[i].sy_name;
279                 lx_systrace_sysent[i].lss_entry = DTRACE_IDNONE;
280                 lx_systrace_sysent[i].lss_return = DTRACE_IDNONE;
281         }
282
283         /*
284          * Install probe triggers.
285          */
286         lx_systrace_entry_ptr = lx_systrace_entry;
287         lx_systrace_return_ptr = lx_systrace_return;
288
289         return (DDI_SUCCESS);
290 }
291
292 /*ARGSUSED*/
293 static int
294 lx_systrace_detach(dev_info_t *devi, ddi_detach_cmd_t cmd)
295 {
296         switch (cmd) {
297         case DDI_DETACH:
298                 break;
299         case DDI_SUSPEND:
300                 return (DDI_SUCCESS);
301         default:
302                 return (DDI_FAILURE);
303         }
304
305         if (dtrace_unregister(lx_systrace_id) != 0)
306                 return (DDI_FAILURE);
307
308         /*
309          * Free table.
310          */
311         kmem_free(lx_systrace_sysent, lx_systrace_nsysent *
312             sizeof (lx_systrace_sysent_t));
313         lx_systrace_sysent = NULL;
314         lx_systrace_nsysent = 0;
315
316         /*
317          * Reset probe triggers.
318          */
319         lx_systrace_entry_ptr = NULL;
320         lx_systrace_return_ptr = NULL;
321
322         return (DDI_SUCCESS);
323 }
324
325 /*ARGSUSED*/
```



```
326 static int
327 lx_systrace_open(dev_t *devp, int flag, int otyp, cred_t *cred_p)
328 {
329         return (0);
330 }
331
332 static struct cb_ops lx_systrace_cb_ops = {
333         lx_systrace_open,       /* open */
334         nodev,                  /* close */
335         nulldev,                /* strategy */
336         nulldev,                /* print */
337         nodev,                  /* dump */
338         nodev,                  /* read */
339         nodev,                  /* write */
340         nodev,                  /* ioctl */
341         nodev,                  /* devmap */
342         nodev,                  /* mmap */
343         nodev,                  /* segmap */
344         nochpoll,               /* poll */
345         ddi_prop_op,            /* cb_prop_op */
346         0,                      /* streamtab */
347         D_NEW | D_MP            /* Driver compatibility flag */
348 };
349
350 static struct dev_ops lx_systrace_ops = {
351         DEVO_REV,               /* devo_rev */
352         0,                      /* refcnt */
353         ddi_getinfo_1to1,       /* get_dev_info */
354         nulldev,                /* identify */
355         nulldev,                /* probe */
356         lx_systrace_attach,     /* attach */
357         lx_systrace_detach,     /* detach */
358         nodev,                  /* reset */
359         &lx_systrace_cb_ops,    /* driver operations */
360         NULL,                   /* bus operations */
361         nodev,                  /* dev power */
362         ddi_quiesce_not_needed,         /* quiesce */
363 };
364
365 /*
366  * Module linkage information for the kernel.
367  */
368 static struct modldrv modldrv = {
369         &mod_driverops,         /* module type (this is a pseudo driver) */
370         "Linux Brand System Call Tracing", /* name of module */
371         &lx_systrace_ops        /* driver ops */
372 };
373
374 static struct modlinkage modlinkage = {
375         MODREV_1,
376         (void *)&modldrv,
377         NULL
378 };
379
380 int
381 _init(void)
382 {
383         return (mod_install(&modlinkage));
384 }
385
386 int
387 _info(struct modinfo *modinfop)
388 {
389         return (mod_info(&modlinkage, modinfop));
390 }
```

```
392 int
393 _fini(void)
394 {
395         return (mod_remove(&modlinkage));
396 }
397 #endif /* ! codereview */
```

```
**********************************************************
     976 Tue Jan 14 16:17:17 2014
new/usr/src/uts/common/brand/lx/dtrace/lx_systrace.conf
Bring back LX zones.
**********************************************************
```

```
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #
  21 #
  22 # Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  23 # Use is subject to license terms.
  24 #
  25 #ident  "%Z%%M% %I%     %E% SMI"

  27 name="lx_systrace" parent="pseudo" instance=0;
  28 #endif /* ! codereview */
```

```
**********************************************************
    6169 Tue Jan 14 16:17:18 2014
new/usr/src/uts/common/brand/lx/io/ldlinux.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/ddi.h>
#include <sys/cmn_err.h>
#include <sys/modctl.h>
#include <sys/ptms.h>
#include <sys/stropts.h>
#include <sys/strsun.h>
#include <sys/sunddi.h>

#include <sys/ldlinux.h>


/*
 * ldlinuxopen - open routine gets called when the module gets pushed onto the
 * stream.
 */
/* ARGSUSED */
static int
ldlinuxopen(
        queue_t    *q,          /* pointer to the read side queue */
        dev_t   *devp,          /* pointer to stream tail's dev */
        int     oflag,          /* the user open(2) supplied flags */
        int     sflag,          /* open state flag */
        cred_t *credp)          /* credentials */
{
        struct ldlinux *tp;     /* ldlinux entry for this module */
        mblk_t *mop;
        struct stroptions *sop;
        struct termios *termiosp;
        int len;

        if (sflag != MODOPEN)
                return (EINVAL);

        if (q->q_ptr != NULL) {
```



```
                /* It's already attached. */
                return (0);
        }

        mop = allocb(sizeof (struct stroptions), BPRI_MED);
        if (mop == NULL)
                return (ENOSR);
        mop->b_datap->db_type = M_SETOPTS;
        mop->b_wptr += sizeof (struct stroptions);
        sop = (struct stroptions *)mop->b_rptr;
        sop->so_flags = SO_ISTTY;

        /*
         * Allocate state structure.
         */
        tp = kmem_alloc(sizeof (*tp), KM_SLEEP);

        /* Stash a pointer to our private data in q_ptr. */
        q->q_ptr = WR(q)->q_ptr = tp;

        /*
         * Get termios defaults.  These are stored as
         * a property in the "options" node.
         */
        if (ddi_getlongprop(DDI_DEV_T_ANY, ddi_root_node(), 0, "ttymodes",
            (caddr_t)&termiosp, &len) == DDI_PROP_SUCCESS &&
            len == sizeof (struct termios)) {
                if (termiosp->c_lflag & ICANON) {
                        tp->veof = termiosp->c_cc[VEOF];
                        tp->veol = termiosp->c_cc[VEOL];
                        tp->vmin = 1;
                        tp->vtime = 0;
                } else {
                        tp->veof = 0;
                        tp->veol = 0;
                        tp->vmin = termiosp->c_cc[VMIN];
                        tp->vtime = termiosp->c_cc[VTIME];
                }
                kmem_free(termiosp, len);
        } else {
                /*
                 * winge winge winge...
                 */
                cmn_err(CE_WARN,
                    "ldlinuxopen: Couldn't get ttymodes property!");
                bzero(tp, sizeof (*tp));
        }

        tp->state = 0;

        /*
         * Commit to the open and send the M_SETOPTS off to the stream head.
         */
        qprocson(q);
        putnext(q, mop);

        return (0);
}


/*
 * ldlinuxclose - This routine gets called when the module gets
 * popped off of the stream.
 */
/* ARGSUSED */
static int
```

```
ldlinuxclose(queue_t *q, int flag, cred_t *credp)
{
        struct ldlinux *tp;

        qprocsoff(q);
        tp = q->q_ptr;
        kmem_free(tp, sizeof (*tp));
        q->q_ptr = WR(q)->q_ptr = NULL;
        return (0);
}


static void
do_ioctl(queue_t *q, mblk_t *mp)
{
        struct ldlinux  *tp = q->q_ptr;
        struct iocblk   *iocp = (struct iocblk *)mp->b_rptr;
        struct lx_cc    *cb;
        mblk_t          *tmp;
        int             error;

        switch (iocp->ioc_cmd) {
        case TIOCSETLD:
                /* prepare caller supplied data for access */
                error = miocpullup(mp, sizeof (struct lx_cc));
                if (error != 0) {
                        miocnak(q, mp, 0, error);
                        return;
                }

                /* get a pointer to the caller supplied data */
                cb = (struct lx_cc *)mp->b_cont->b_rptr;

                /* save caller supplied data in our per-stream cache */
                tp->veof = cb->veof;
                tp->veol = cb->veol;
                tp->vmin = cb->vmin;
                tp->vtime = cb->vtime;

                /* initialize and send a reply indicating that we're done */
                miocack(q, mp, 0, 0);
                return;

        case TIOCGETLD:
                /* allocate a reply message */
                if ((tmp = allocb(sizeof (struct lx_cc), BPRI_MED)) == NULL) {
                        miocnak(q, mp, 0, ENOSR);
                        return;
                }

                /* initialize the reply message */
                mioc2ack(mp, tmp, sizeof (struct lx_cc), 0);

                /* get a pointer to the reply data */
                cb = (struct lx_cc *)mp->b_cont->b_rptr;

                /* copy data from our per-stream cache into the reply data */
                cb->veof = tp->veof;
                cb->veol = tp->veol;
                cb->vmin = tp->vmin;
                cb->vtime = tp->vtime;

                /* send the reply indicating that we're done */
                qreply(q, mp);
                return;
```



```
        case PTSSTTY:
                tp->state |= ISPTSTTY;
                break;

        default:
                break;
        }

        putnext(q, mp);
}


/*
 * ldlinuxput - Module read and write queue put procedure.
 */
static void
ldlinuxput(queue_t *q, mblk_t *mp)
{
        struct ldlinux *tp = q->q_ptr;

        switch (DB_TYPE(mp)) {
        default:
                break;
        case M_IOCTL:
                if ((q->q_flag & QREADR) == 0) {
                        do_ioctl(q, mp);
                        return;
                }
                break;

        case M_FLUSH:
                /*
                 * Handle read and write flushes.
                 */
                if ((((q->q_flag & QREADR) != 0) && (*mp->b_rptr & FLUSHR)) ||
                    (((q->q_flag & QREADR) == 0) && (*mp->b_rptr & FLUSHW))) {
                        if ((tp->state & ISPTSTTY) && (*mp->b_rptr & FLUSHBAND))
                                flushband(q, *(mp->b_rptr + 1), FLUSHDATA);
                        else
                                flushq(q, FLUSHDATA);
                }
                break;
        }
        putnext(q, mp);
}


static struct module_info ldlinux_info = {
        LDLINUX_MODID,
        LDLINUX_MOD,
        0,
        INFPSZ,
        0,
        0
};

static struct qinit ldlinuxinit = {
        (int (*)()) ldlinuxput,
        NULL,
        ldlinuxopen,
        ldlinuxclose,
        NULL,
        &ldlinux_info
};

static struct streamtab ldlinuxinfo = {
```

```
260         &ldlinuxinit,
261         &ldlinuxinit
262 };

264 /*
265  * Module linkage information for the kernel.
266  */
267 static struct fmodsw fsw = {
268         LDLINUX_MOD,
269         &ldlinuxinfo,
270         D_MTQPAIR | D_MP
271 };

273 static struct modlstrmod modlstrmod = {
274         &mod_strmodops, "termios extensions for lx brand", &fsw
275 };

277 static struct modlinkage modlinkage = {
278         MODREV_1, &modlstrmod, NULL
279 };

281 int
282 _init()
283 {
284         return (mod_install(&modlinkage));
285 }

287 int
288 _fini()
289 {
290         return (mod_remove(&modlinkage));
291 }

293 int
294 _info(struct modinfo *modinfop)
295 {
296         return (mod_info(&modlinkage, modinfop));
297 }
298 #endif /* ! codereview */
```

```
**********************************************************
   53819 Tue Jan 14 16:17:18 2014
new/usr/src/uts/common/brand/lx/io/lx_audio.c
LX zone support should now build and packages of relevance produced.
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */


#include <sys/audio.h>
#include <sys/conf.h>
#include <sys/debug.h>
#include <sys/disp.h>
#include <sys/ddi.h>
#include <sys/file.h>
#include <sys/id_space.h>
#include <sys/kmem.h>
#include <sys/lx_audio.h>
#include <sys/mixer.h>
#include <sys/modhash.h>
#include <sys/stat.h>
#include <sys/sunddi.h>
#include <sys/sunldi.h>
#include <sys/sysmacros.h>
#include <sys/stropts.h>
#include <sys/types.h>
#include <sys/zone.h>

/* Properties used by the lx_audio driver */
#define LXA_PROP_INPUTDEV               "inputdev"
#define LXA_PROP_OUTPUTDEV              "outputdev"

/* default device paths used by this driver */
#define LXA_DEV_DEFAULT                 "/dev/audio"
#define LXA_DEV_CUSTOM_DIR              "/dev/sound/"

/* maximum possible number of concurrent opens of this driver */
#define LX_AUDIO_MAX_OPENS              1024

/*
 * these are default fragment size and fragment count values.
 * these values were chosen to make quake work well on my
 * laptop: 2Ghz Pentium M + NVIDIA GeForce Go 6400.
```



```
 *
 * for reference:
 * - 1 sec of stereo output at 44Khz is about 171 Kb of data
 * - 1 sec of mono output at 8Khz is about 8Kb of data
 */
#define LXA_OSS_FRAG_SIZE               (1024)  /* 1/8 sec at 8Khz mono */
#define LXA_OSS_FRAG_CNT                (1024 * 2)

/* maximum ammount of fragment memory we'll allow a process to mmap */
#define LXA_OSS_FRAG_MEM                (1024 * 1024 * 2) /* 2Mb */

/* forward declarations */
typedef struct lxa_state lxa_state_t;
typedef struct lxa_zstate lxa_zstate_t;

/*
 * Structure and enum declarations
 */
typedef enum {
        LXA_TYPE_INVALID        = 0,
        LXA_TYPE_AUDIO          = 1,    /* audio device */
        LXA_TYPE_AUDIOCTL       = 2     /* audio control/mixer device */
} lxa_dev_type_t;

struct lxa_zstate {
        char                    *lxa_zs_zonename;

        /*
         * we could store the input/output audio device setting here,
         * but instead we're keeing them as device node properties
         * so that a user can easily see the audio configuration for
         * a zone via prtconf.
         */

        /*
         * OSS doesn't support multiple opens of the audio device.
         * (multiple opens of the mixer device are supported.)
         * so here we'll keep a pointer to any open input/output
         * streams.  (OSS does support two opens if one is for input
         * and the other is for output.)
         */
        lxa_state_t             *lxa_zs_istate;
        lxa_state_t             *lxa_zs_ostate;

        /*
         * we need to cache channel gain and balance.  channel gain and
         * balance map to PCM volume in OSS, which are supposedly a property
         * of the underlying hardware.  but in solaris, channels are
         * implemented in software and only exist when an audio device
         * is actually open.  (each open returns a unique channel.)  OSS
         * apps will expect consistent PCM volume set/get operations to
         * work even if no audio device is open.  hence, if no underlying
         * device is open we need to cache the gain and balance setting.
         */
        lxa_mixer_levels_t      lxa_zs_pcm_levels;
};

struct lxa_state {
        lxa_zstate_t    *lxas_zs;       /* zone state pointer */

        dev_t           lxas_dev_old;   /* dev_t used to open the device */
        dev_t           lxas_dev_new;   /* new dev_t assigned to an open */
        int             lxas_flags;     /* original flags passed to open */
        lxa_dev_type_t  lxas_type;      /* type of device that was opened */

        int             lxas_devs_same; /* input and output device the same? */
```

```
128         /* input device variables */
129         ldi_handle_t    lxas_idev_lh;           /* ldi handle for access */
130         int             lxas_idev_flags;        /* flags used for open */

132         /* output device variables */
133         ldi_handle_t    lxas_odev_lh;           /* ldi handle for access */
134         int             lxas_odev_flags;        /* flags used for open */

136         /*
137          * since we support multiplexing of devices we need to remember
138          * certain parameters about the devices
139          */
140         uint_t          lxas_hw_features;
141         uint_t          lxas_sw_features;

143         uint_t          lxas_frag_size;
144         uint_t          lxas_frag_cnt;

146         /*
147          * members needed to support mmap device access.  note that to
148          * simplify things we only support one mmap access per open.
149          */
150         ddi_umem_cookie_t       lxas_umem_cookie;
151         char                    *lxas_umem_ptr;
152         size_t                  lxas_umem_len;
153         kthread_t               *lxas_mmap_thread;
154         int                     lxas_mmap_thread_running;
155         int                     lxas_mmap_thread_exit;
156         int                     lxas_mmap_thread_frag;
157 };

159 /*
160  * Global variables
161  */
162 dev_info_t      *lxa_dip = NULL;
163 kmutex_t        lxa_lock;
164 id_space_t      *lxa_minor_id = NULL;
165 mod_hash_t      *lxa_state_hash = NULL;
166 mod_hash_t      *lxa_zstate_hash = NULL;
167 size_t          lxa_state_hash_size = 15;
168 size_t          lxa_zstate_hash_size = 15;
169 size_t          lxa_registered_zones = 0;

171 /*
172  * function declarations
173  */
174 static void lxa_mmap_output_disable(lxa_state_t *);

176 /*
177  * functions
178  */
179 static void
180 lxa_state_close(lxa_state_t *lxa_state)
181 {
182         lxa_zstate_t            *lxa_zs = lxa_state->lxas_zs;
183         minor_t                 minor = getminor(lxa_state->lxas_dev_new);

185         /* disable any mmap output that might still be going on */
186         lxa_mmap_output_disable(lxa_state);

188         /*
189          * if this was the active input/output device, unlink it from
190          * the global zone state so that other opens of the audio device
191          * can now succeed.
192          */
```



```
193         mutex_enter(&lxa_lock);
194         if (lxa_zs->lxa_zs_istate == lxa_state)
195                 lxa_zs->lxa_zs_istate = NULL;
196         if (lxa_zs->lxa_zs_ostate == lxa_state) {
197                 lxa_zs->lxa_zs_ostate = NULL;
198         }
199         mutex_exit(&lxa_lock);

201         /* remove this state structure from the hash (if it's there) */
202         (void) mod_hash_remove(lxa_state_hash,
203             (mod_hash_key_t)(uintptr_t)minor, (mod_hash_val_t *)&lxa_state);

205         /* close any audio device that we have open */
206         if (lxa_state->lxas_idev_lh != NULL)
207                 (void) ldi_close(lxa_state->lxas_idev_lh,
208                     lxa_state->lxas_idev_flags, kcred);
209         if (lxa_state->lxas_odev_lh != NULL)
210                 (void) ldi_close(lxa_state->lxas_odev_lh,
211                     lxa_state->lxas_odev_flags, kcred);

213         /* free up any memory allocated by mmaps */
214         if (lxa_state->lxas_umem_cookie != NULL)
215                 ddi_umem_free(lxa_state->lxas_umem_cookie);

217         /* release the id associated with this state structure */
218         id_free(lxa_minor_id, minor);

220         kmem_free(lxa_state, sizeof (*lxa_state));
221 }

223 static char *
224 getzonename(void)
225 {
226         return (curproc->p_zone->zone_name);
227 }

229 static char *
230 lxa_devprop_name(char *zname, char *pname)
231 {
232         char    *zpname;
233         int     n;

235         ASSERT((pname != NULL) && (zname != NULL));

237         /* prepend the zone name to the property name */
238         n = snprintf(NULL, 0, "%s_%s", zname, pname) + 1;
239         zpname = kmem_alloc(n, KM_SLEEP);
240         (void) snprintf(zpname, n, "%s_%s", zname, pname);

242         return (zpname);
243 }

245 static int
246 lxa_devprop_verify(char *pval)
247 {
248         int     n;

250         ASSERT(pval != NULL);

252         if (strcmp(pval, "default") == 0)
253                 return (0);

255         /* make sure the value is an integer */
256         for (n = 0; pval[n] != '\0'; n++) {
257                 if ((pval[n] < '0') && (pval[n] > '9')) {
258                         return (-1);
```

```
259                 }
260         }

262         return (0);
263 }

265 static char *
266 lxa_devprop_lookup(char *zname, char *pname, lxa_dev_type_t lxa_type)
267 {
268         char            *zprop_name, *pval;
269         char            *dev_path;
270         int             n, rv;

272         ASSERT((pname != NULL) && (zname != NULL));
273         ASSERT((lxa_type == LXA_TYPE_AUDIO) || (lxa_type == LXA_TYPE_AUDIOCTL));

275         zprop_name = lxa_devprop_name(zname, pname);

277         /* attempt to lookup the property */
278         rv = ddi_prop_lookup_string(DDI_DEV_T_ANY, lxa_dip,
279             DDI_PROP_DONTPASS | DDI_PROP_NOTPROM, zprop_name, &pval);
280         strfree(zprop_name);

282         if (rv != DDI_PROP_SUCCESS)
283                 return (NULL);

285         if (lxa_devprop_verify(pval) != 0) {
286                 ddi_prop_free(pval);
287                 return (NULL);
288         }

290         if (strcmp(pval, "none") == 0) {
291                 /* there is no audio device specified */
292                 return (NULL);
293         } else if (strcmp(pval, "default") == 0) {
294                 /* use the default audio device on the system */
295                 dev_path = strdup(LXA_DEV_DEFAULT);
296         } else {
297                 /* a custom audio device was specified, generate a path */
298                 n = snprintf(NULL, 0, "%s%s", LXA_DEV_CUSTOM_DIR, pval) + 1;
299                 dev_path = kmem_alloc(n, KM_SLEEP);
300                 (void) snprintf(dev_path, n, "%s%s", LXA_DEV_CUSTOM_DIR, pval);
301         }
302         ddi_prop_free(pval);

304         /*
305          * if this is an audio control device so we need to append
306          * "ctl" to the path
307          */
308         if (lxa_type == LXA_TYPE_AUDIOCTL) {
309                 char    *tmp;
310                 n = snprintf(NULL, 0, "%s%s", dev_path, "ctl") + 1;
311                 tmp = kmem_alloc(n, KM_SLEEP);
312                 (void) snprintf(tmp, n, "%s%s", dev_path, "ctl");
313                 strfree(dev_path);
314                 dev_path = tmp;
315         }

317         return (dev_path);
318 }

320 static int
321 lxa_dev_getfeatures(lxa_state_t *lxa_state)
322 {
323         audio_info_t    ai_idev, ai_odev;
324         int             n, rv;
```



```
326         /* set a default fragment size */
327         lxa_state->lxas_frag_size = LXA_OSS_FRAG_SIZE;
328         lxa_state->lxas_frag_cnt = LXA_OSS_FRAG_CNT;

330         /* get info for the currently open audio devices */
331         if ((lxa_state->lxas_idev_lh != NULL) &&
332             ((rv = ldi_ioctl(lxa_state->lxas_idev_lh,
333             AUDIO_GETINFO, (intptr_t)&ai_idev, FKIOCTL, kcred, &n)) != 0))
334                 return (rv);
335         if ((lxa_state->lxas_odev_lh != NULL) &&
336             ((rv = ldi_ioctl(lxa_state->lxas_odev_lh,
337             AUDIO_GETINFO, (intptr_t)&ai_odev, FKIOCTL, kcred, &n)) != 0))
338                 return (rv);

340         /* if we're only open for reading or writing then it's easy */
341         if (lxa_state->lxas_idev_lh == NULL) {
342                 lxa_state->lxas_sw_features = ai_odev.sw_features;
343                 lxa_state->lxas_hw_features = ai_odev.hw_features;
344                 return (0);
345         } else if (lxa_state->lxas_odev_lh == NULL) {
346                 lxa_state->lxas_sw_features = ai_idev.sw_features;
347                 lxa_state->lxas_hw_features = ai_idev.hw_features;
348                 return (0);
349         }

351         /*
352          * well if we're open for reading and writing but the underlying
353          * device is the same then it's also pretty easy
354          */
355         if (lxa_state->lxas_devs_same) {
356                 if ((ai_odev.sw_features != ai_idev.sw_features) ||
357                     (ai_odev.hw_features != ai_idev.hw_features)) {
358                         zcmn_err(getzoneid(), CE_WARN, "lx_audio error: "
359                             "audio device reported inconsistent features");
360                         return (EIO);
361                 }
362                 lxa_state->lxas_sw_features = ai_odev.sw_features;
363                 lxa_state->lxas_hw_features = ai_odev.hw_features;
364                 return (0);
365         }

367         /*
368          * figure out which software features we're going to support.
369          * we will report a feature as supported if both the input
370          * and output device support it.
371          */
372         lxa_state->lxas_sw_features = 0;
373         n = ai_idev.sw_features & ai_odev.sw_features;
374         if (n & AUDIO_SWFEATURE_MIXER)
375                 lxa_state->lxas_sw_features |= AUDIO_SWFEATURE_MIXER;

377         /*
378          * figure out which hardware features we're going to support.
379          * for a first pass we will report a feature as supported if
380          * both the input and output device support it.
381          */
382         lxa_state->lxas_hw_features = 0;
383         n = ai_idev.hw_features & ai_odev.hw_features;
384         if (n & AUDIO_HWFEATURE_MSCODEC)
385                 lxa_state->lxas_hw_features |= AUDIO_HWFEATURE_MSCODEC;

387         /*
388          * if we made it here then we have different audio input and output
389          * devices.  this will allow us to report support for additional
390          * hardware features that may not supported by just the input or
```

```
391          * output device alone.
392          */
394         /* always report tha we support both playback and recording */
395         lxa_state->lxas_hw_features =
396             AUDIO_HWFEATURE_PLAY | AUDIO_HWFEATURE_RECORD;
398         /* always report full duplex support */
399         lxa_state->lxas_hw_features = AUDIO_HWFEATURE_DUPLEX;
401         /* never report that we have input to output loopback support */
402         ASSERT((lxa_state->lxas_hw_features & AUDIO_HWFEATURE_IN2OUT) == 0);
403         return (0);
404 }
406 static int
407 lxa_dev_open(lxa_state_t *lxa_state)
408 {
409         char            *idev, *odev;
410         int             flags, rv;
411         ldi_handle_t    lh;
412         ldi_ident_t     li = NULL;
414         ASSERT((lxa_state->lxas_type == LXA_TYPE_AUDIO) ||
415             (lxa_state->lxas_type == LXA_TYPE_AUDIOCTL));
417         /*
418          * check if we have configuration properties for this zone.
419          * if we don't then audio isn't supported in this zone.
420          */
421         idev = lxa_devprop_lookup(getzonename(), LXA_PROP_INPUTDEV,
422             lxa_state->lxas_type);
423         odev = lxa_devprop_lookup(getzonename(), LXA_PROP_OUTPUTDEV,
424             lxa_state->lxas_type);
426         /* make sure there is at least one device to read from or write to */
427         if ((idev == NULL) && (odev == NULL))
428                 return (ENODEV);
430         /* see if the input and output devices are actually the same device */
431         if (((idev != NULL) && (odev != NULL)) &&
432             (strcmp(idev, odev) == 0))
433                 lxa_state->lxas_devs_same = 1;
435         /* we don't respect FEXCL */
436         flags = lxa_state->lxas_flags & ~FEXCL;
437         if (lxa_state->lxas_type == LXA_TYPE_AUDIO) {
438                 /*
439                  * if we're opening audio devices then we need to muck
440                  * with the FREAD/FWRITE flags.
441                  *
442                  * certain audio device may only support input or output
443                  * (but not both.)  so if we're multiplexing input/output
444                  * to different devices we need to make sure we don't try
445                  * and open the output device for reading and the input
446                  * device for writing.
447                  *
448                  * if we're using the same device for input/output we still
449                  * need to do this because some audio devices won't let
450                  * themselves be opened multiple times for read access.
451                  */
452                 lxa_state->lxas_idev_flags = flags & ~FWRITE;
453                 lxa_state->lxas_odev_flags = flags & ~FREAD;
455                 /* make sure we have devices to read from and write to */
456                 if (((flags & FREAD) && (idev == NULL)) ||
```



```
457                     ((flags & FWRITE) && (odev == NULL))) {
458                         rv = ENODEV;
459                         goto out;
460                 }
461         } else {
462                 lxa_state->lxas_idev_flags = lxa_state->lxas_odev_flags = flags;
463         }
465         /* get an ident to open the devices */
466         if (ldi_ident_from_dev(lxa_state->lxas_dev_new, &li) != 0) {
467                 rv = ENODEV;
468                 goto out;
469         }
471         /* open the input device */
472         lxa_state->lxas_idev_lh = NULL;
473         if (((lxa_state->lxas_type == LXA_TYPE_AUDIOCTL) ||
474             (lxa_state->lxas_idev_flags & FREAD)) &&
475             (idev != NULL)) {
476                 rv = ldi_open_by_name(idev, lxa_state->lxas_idev_flags,
477                     kcred, &lh, li);
478                 if (rv != 0) {
479                         zcmn_err(getzoneid(), CE_WARN, "lxa_open_dev: "
480                             "unable to open audio device: %s", idev);
481                         zcmn_err(getzoneid(), CE_WARN, "lxa_open_dev: "
482                             "possible zone audio configuration error");
483                         goto out;
484                 }
485                 lxa_state->lxas_idev_lh = lh;
486         }
488         /* open the output device */
489         lxa_state->lxas_odev_lh = NULL;
490         if (((lxa_state->lxas_type == LXA_TYPE_AUDIOCTL) ||
491             (lxa_state->lxas_odev_flags & FWRITE)) &&
492             (odev != NULL)) {
493                 rv = ldi_open_by_name(odev, lxa_state->lxas_odev_flags,
494                     kcred, &lh, li);
495                 if (rv != 0) {
496                         /*
497                          * If this open failed and we previously opened an
498                          * input device, it is the responsibility of the
499                          * caller to close that device after we return
500                          * failure here.
501                          */
502                         zcmn_err(getzoneid(), CE_WARN, "lxa_open_dev: "
503                             "unable to open audio device: %s", odev);
504                         zcmn_err(getzoneid(), CE_WARN, "lxa_open_dev: "
505                             "possible zone audio configuration error");
506                         goto out;
507                 }
508                 lxa_state->lxas_odev_lh = lh;
509         }
511         /* free up stuff */
512 out:
513         if (li != NULL)
514                 ldi_ident_release(li);
515         if (idev != NULL)
516                 strfree(idev);
517         if (odev != NULL)
518                 strfree(odev);
520         return (rv);
521 }
```

```
523 void
524 lxa_mmap_thread_exit(lxa_state_t *lxa_state)
525 {
526         mutex_enter(&lxa_lock);
527         lxa_state->lxas_mmap_thread = NULL;
528         lxa_state->lxas_mmap_thread_frag = 0;
529         lxa_state->lxas_mmap_thread_running = 0;
530         lxa_state->lxas_mmap_thread_exit = 0;
531         mutex_exit(&lxa_lock);
532         thread_exit();
533         /*NOTREACHED*/
534 }

536 void
537 lxa_mmap_thread(lxa_state_t *lxa_state)
538 {
539         struct uio      uio, uio_null;
540         iovec_t         iovec, iovec_null;
541         uint_t          bytes_per_sec, usec_per_frag, ticks_per_frag;
542         int             rv, junk, eof, retry;
543         audio_info_t    ai;

545         /* we better be setup for writing to the output device */
546         ASSERT((lxa_state->lxas_flags & FWRITE) != 0);
547         ASSERT(lxa_state->lxas_odev_lh != NULL);

549         /* setup a uio to output one fragment */
550         uio.uio_iov = &iovec;
551         uio.uio_iovcnt = 1;
552         uio.uio_offset = 0;
553         uio.uio_segflg = UIO_SYSSPACE;
554         uio.uio_fmode = 0;
555         uio.uio_extflg = 0;
556         uio.uio_llimit = MAXOFFSET_T;

558         /* setup a uio to output a eof (a fragment with a length of 0) */
559         uio_null.uio_iov = &iovec_null;
560         uio_null.uio_iov->iov_len = 0;
561         uio_null.uio_iov->iov_base = NULL;
562         uio_null.uio_iovcnt = 1;
563         uio_null.uio_offset = 0;
564         uio_null.uio_segflg = UIO_SYSSPACE;
565         uio_null.uio_fmode = 0;
566         uio_null.uio_extflg = 0;
567         uio_null.uio_llimit = MAXOFFSET_T;
568         uio_null.uio_resid = 0;

570 lxa_mmap_thread_top:
571         ASSERT(!MUTEX_HELD(&lxa_lock));

573         /* first drain any pending audio output */
574         if ((rv = ldi_ioctl(lxa_state->lxas_odev_lh,
575             AUDIO_DRAIN, NULL, FKIOCTL, kcred, &junk)) != 0) {
576                 cmn_err(CE_WARN, "lxa_mmap_thread: "
577                     "AUDIO_DRAIN failed, aborting audio output");
578                 lxa_mmap_thread_exit(lxa_state);
579                 /*NOTREACHED*/
580         }

582         /*
583          * we depend on the ai.play.eof value to keep track of
584          * audio output progress so reset it here.
585          */
586         AUDIO_INITINFO(&ai);
587         ai.play.eof = 0;
588         if ((rv = ldi_ioctl(lxa_state->lxas_odev_lh,
```



```
589             AUDIO_SETINFO, (intptr_t)&ai, FKIOCTL, kcred, &junk)) != 0) {
590                 cmn_err(CE_WARN, "lxa_mmap_thread: "
591                     "AUDIO_SETINFO failed, aborting audio output");
592                 lxa_mmap_thread_exit(lxa_state);
593                 /*NOTREACHED*/
594         }

596         /*
597          * we're going to need to know the sampling rate and number
598          * of output channels to estimate how long we can sleep between
599          * requests.
600          */
601         if ((rv = ldi_ioctl(lxa_state->lxas_odev_lh, AUDIO_GETINFO,
602             (intptr_t)&ai, FKIOCTL, kcred, &junk)) != 0) {
603                 cmn_err(CE_WARN, "lxa_mmap_thread: "
604                     "AUDIO_GETINFO failed, aborting audio output");
605                 lxa_mmap_thread_exit(lxa_state);
606                 /*NOTREACHED*/
607         }

609         /* estimate how many ticks it takes to output a fragment of data */
610         bytes_per_sec = (ai.play.sample_rate * ai.play.channels *
611             ai.play.precision) / 8;
612         usec_per_frag = MICROSEC * lxa_state->lxas_frag_size / bytes_per_sec;
613         ticks_per_frag = drv_usectohz(usec_per_frag);

615         /* queue up three fragments of of data into the output stream */
616         eof = 3;

618         /* sanity check the eof value */
619         ASSERT(ai.play.eof == 0);
620         ai.play.eof = 0;

622         /* we always start audio output at fragment 0 */
623         mutex_enter(&lxa_lock);
624         lxa_state->lxas_mmap_thread_frag = 0;

626         /*
627          * we shouldn't have allowed the mapping if it isn't a multiple
628          * of the fragment size
629          */
630         ASSERT((lxa_state->lxas_umem_len % lxa_state->lxas_frag_size) == 0);

632         while (!lxa_state->lxas_mmap_thread_exit) {
633                 size_t start, end;

635                 /*
636                  * calculate the start and ending offsets of the next
637                  * fragment to output
638                  */
639                 start = lxa_state->lxas_mmap_thread_frag *
640                     lxa_state->lxas_frag_size;
641                 end = start + lxa_state->lxas_frag_size;

643                 ASSERT(start < lxa_state->lxas_umem_len);
644                 ASSERT(end <= lxa_state->lxas_umem_len);

646                 /* setup the uio to output one fragment of audio */
647                 uio.uio_resid = end - start;
648                 uio.uio_iov->iov_len = end - start;
649                 uio.uio_iov->iov_base = &lxa_state->lxas_umem_ptr[start];

651                 /* increment the current fragment index */
652                 lxa_state->lxas_mmap_thread_frag =
653                     (lxa_state->lxas_mmap_thread_frag + 1) %
654                     (lxa_state->lxas_umem_len / lxa_state->lxas_frag_size);
```

```
656                 /* drop the audio lock before actually outputting data */
657                 mutex_exit(&lxa_lock);

659                 /*
660                  * write the fragment of audio data to the device stream
661                  * then write a eof to the stream to tell the device to
662                  * increment ai.play.eof when it's done processing the
663                  * fragment we just wrote
664                  */
665                 if ((rv = ldi_write(lxa_state->lxas_odev_lh,
666                     &uio, kcred)) != 0) {
667                         cmn_err(CE_WARN, "lxa_mmap_thread: "
668                             "ldi_write() failed (%d), "
669                             "resetting audio output", rv);
670                         goto lxa_mmap_thread_top;
671                 }
672                 if ((rv = ldi_write(lxa_state->lxas_odev_lh,
673                     &uio_null, kcred)) != 0) {
674                         cmn_err(CE_WARN, "lxa_mmap_thread: "
675                             "ldi_write(eof) failed (%d), "
676                             "resetting audio output", rv);
677                         goto lxa_mmap_thread_top;
678                 }

680                 /*
681                  * we want to avoid buffer underrun so ensure that
682                  * there is always at least one fragment of data in the
683                  * output stream.
684                  */
685                 mutex_enter(&lxa_lock);
686                 if (--eof > 0) {
687                         continue;
688                 }

690                 /*
691                  * now we wait until the audio device has finished outputting
692                  * at least one fragment of data.
693                  */
694                 retry = 0;
695                 while (!lxa_state->lxas_mmap_thread_exit && (eof == 0)) {
696                         uint_t ai_eof_old = ai.play.eof;

698                         mutex_exit(&lxa_lock);

700                         /*
701                          * delay for the number of ticks it takes
702                          * to output one fragment of data
703                          */
704                         if (ticks_per_frag > 0)
705                                 delay(ticks_per_frag);

707                         /* check if we've managed to output any fragments */
708                         if ((rv = ldi_ioctl(lxa_state->lxas_odev_lh,
709                             AUDIO_GETINFO, (intptr_t)&ai,
710                             FKIOCTL, kcred, &junk)) != 0) {
711                                 cmn_err(CE_WARN, "lxa_mmap_thread: "
712                                     "AUDIO_GETINFO failed (%d), "
713                                     "resetting audio output", rv);
714                                 /* re-start mmap audio output */
715                                 goto lxa_mmap_thread_top;
716                         }

718                         if (ai_eof_old == ai.play.eof) {
719                                 /* institute a random retry limit */
720                                 if (retry++ < 100) {
```



```
721                                         mutex_enter(&lxa_lock);
722                                         continue;
723                                 }
724                                 cmn_err(CE_WARN, "lxa_mmap_thread: "
725                                     "output stalled, "
726                                     "resetting audio output");
727                                 /* re-start mmap audio output */
728                                 goto lxa_mmap_thread_top;
729                         }

731                         if (ai.play.eof > ai_eof_old) {
732                                 eof = ai.play.eof - ai_eof_old;
733                         } else {
734                                 /* eof counter wrapped around */
735                                 ASSERT(ai_eof_old < ai.play.eof);
736                                 eof = ai.play.eof + (ai_eof_old - UINTMAX_MAX);
737                         }
738                         /* we're done with this loop so re-aquire the lock */
739                         ASSERT(eof != 0);
740                         mutex_enter(&lxa_lock);
741                 }
742         }
743         mutex_exit(&lxa_lock);
744         lxa_mmap_thread_exit(lxa_state);
745         /*NOTREACHED*/
746 }

748 static void
749 lxa_mmap_output_disable(lxa_state_t *lxa_state)
750 {
751         kt_did_t tid;

753         mutex_enter(&lxa_lock);

755         /* if the output thread isn't running there's nothing to do */
756         if (lxa_state->lxas_mmap_thread_running == 0) {
757                 mutex_exit(&lxa_lock);
758                 return;
759         }

761         /* tell the pcm mmap output thread to exit */
762         lxa_state->lxas_mmap_thread_exit = 1;

764         /* wait for the mmap output thread to exit */
765         tid = lxa_state->lxas_mmap_thread->t_did;
766         mutex_exit(&lxa_lock);
767         thread_join(tid);
768 }

770 static void
771 lxa_mmap_output_enable(lxa_state_t *lxa_state)
772 {
773         mutex_enter(&lxa_lock);

775         /* if the output thread is already running there's nothing to do */
776         if (lxa_state->lxas_mmap_thread_running != 0) {
777                 mutex_exit(&lxa_lock);
778                 return;
779         }

781         /* setup output state */
782         lxa_state->lxas_mmap_thread_running = 1;
783         lxa_state->lxas_mmap_thread_exit = 0;
784         lxa_state->lxas_mmap_thread_frag = 0;

786         /* kick off a thread to do the mmap pcm output */
```

```
787         lxa_state->lxas_mmap_thread = thread_create(NULL, 0,
788             (void (*)())lxa_mmap_thread, lxa_state,
789             0, &p0, TS_RUN, minclsyspri);
790         ASSERT(lxa_state->lxas_mmap_thread != NULL);

792         mutex_exit(&lxa_lock);
793 }

795 static int
796 lxa_ioc_mmap_output(lxa_state_t *lxa_state, intptr_t arg, int mode)
797 {
798         uint_t  trigger;

800         /* we only support output via mmap */
801         if ((lxa_state->lxas_flags & FWRITE) == 0)
802                 return (EINVAL);

804         /* if the user hasn't mmap the device then there's nothing to do */
805         if (lxa_state->lxas_umem_cookie == NULL)
806                 return (EINVAL);

808         /* copy in the request */
809         if (ddi_copyin((void *)arg, &trigger, sizeof (trigger), mode) != 0)
810                 return (EFAULT);

812         /* a zero value disables output */
813         if (trigger == 0) {
814                 lxa_mmap_output_disable(lxa_state);
815                 return (0);
816         }

818         /* a non-zero value enables output */
819         lxa_mmap_output_enable(lxa_state);
820         return (0);
821 }

823 static int
824 lxa_ioc_mmap_ptr(lxa_state_t *lxa_state, intptr_t arg, int mode)
825 {
826         int     ptr;

828         /* we only support output via mmap */
829         if ((lxa_state->lxas_flags & FWRITE) == 0)
830                 return (EINVAL);

832         /* if the user hasn't mmap the device then there's nothing to do */
833         if (lxa_state->lxas_umem_cookie == NULL)
834                 return (EINVAL);

836         /* if the output thread isn't running then there's nothing to do */
837         if (lxa_state->lxas_mmap_thread_running == 0)
838                 return (EINVAL);

840         mutex_enter(&lxa_lock);
841         ptr = lxa_state->lxas_mmap_thread_frag * lxa_state->lxas_frag_size;
842         mutex_exit(&lxa_lock);

844         if (ddi_copyout(&ptr, (void *)arg, sizeof (ptr), mode) != 0)
845                 return (EFAULT);

847         return (0);
848 }

850 static int
851 lxa_ioc_get_frag_info(lxa_state_t *lxa_state, intptr_t arg, int mode)
852 {
```



```
853         lxa_frag_info_t fi;

855         fi.lxa_fi_size = lxa_state->lxas_frag_size;
856         fi.lxa_fi_cnt = lxa_state->lxas_frag_cnt;

858         if (ddi_copyout(&fi, (void *)arg, sizeof (fi), mode) != 0)
859                 return (EFAULT);

861         return (0);
862 }

864 static int
865 lxa_ioc_set_frag_info(lxa_state_t *lxa_state, intptr_t arg, int mode)
866 {
867         lxa_frag_info_t fi;

869         /* if the device is mmaped we can't change the fragment settings */
870         if (lxa_state->lxas_umem_cookie != NULL)
871                 return (EINVAL);

873         /* copy in the request */
874         if (ddi_copyin((void *)arg, &fi, sizeof (fi), mode) != 0)
875                 return (EFAULT);

877         /* do basic bounds checking */
878         if ((fi.lxa_fi_cnt == 0) || (fi.lxa_fi_size < 16))
879                 return (EINVAL);

881         /* don't accept size values less than 16 */

883         lxa_state->lxas_frag_size = fi.lxa_fi_size;
884         lxa_state->lxas_frag_cnt = fi.lxa_fi_cnt;

886         return (0);
887 }

889 static int
890 lxa_audio_drain(lxa_state_t *lxa_state)
891 {
892         int     junk;

894         /* only applies to output buffers */
895         if (lxa_state->lxas_odev_lh == NULL)
896                 return (EINVAL);

898         /* can't fail so ignore the return value */
899         (void) ldi_ioctl(lxa_state->lxas_odev_lh, AUDIO_DRAIN, NULL,
900             FKIOCTL, kcred, &junk);
901         return (0);
902 }

904 /*
905  * lxa_audio_info_merge() usage notes:
906  *
907  * - it's important to make sure NOT to get the ai_idev and ai_odev
908  *   parameters mixed up when calling lxa_audio_info_merge().
909  *
910  * - it's important for the caller to make sure that AUDIO_GETINFO
911  *   was called for the input device BEFORE the output device.  (see
912  *   the comments for merging the monitor_gain setting to see why.)
913  */
914 static void
915 lxa_audio_info_merge(lxa_state_t *lxa_state,
916     audio_info_t *ai_idev, audio_info_t *ai_odev, audio_info_t *ai_merged)
917 {
918         /* if we're not setup for output return the intput device info */
```

```
919         if (lxa_state->lxas_odev_lh == NULL) {
920                 *ai_merged = *ai_idev;
921                 return;
922         }
923
924         /* if we're not setup for input return the output device info */
925         if (lxa_state->lxas_idev_lh == NULL) {
926                 *ai_merged = *ai_odev;
927                 return;
928         }
929
930         /* get record values from the input device */
931         ai_merged->record = ai_idev->record;
932
933         /* get play values from the output device */
934         ai_merged->play = ai_odev->play;
935
936         /* muting status only matters for the output device */
937         ai_merged->output_muted = ai_odev->output_muted;
938
939         /* we don't support device reference counts, always return 1 */
940         ai_merged->ref_cnt = 1;
941
942         /*
943          * for supported hw/sw features report the combined feature
944          * set we calcuated out earlier.
945          */
946         ai_merged->hw_features = lxa_state->lxas_hw_features;
947         ai_merged->sw_features = lxa_state->lxas_sw_features;
948
949         if (!lxa_state->lxas_devs_same) {
950                 /*
951                  * if the input and output devices are different
952                  * physical devices then we don't support input to
953                  * output loopback so we always report the input
954                  * to output loopback gain to be zero.
955                  */
956                 ai_merged->monitor_gain = 0;
957         } else {
958                 /*
959                  * the intput and output devices are actually the
960                  * same physical device.  hence it probably supports
961                  * intput to output loopback.  regardless we should
962                  * pass back the intput to output gain reported by
963                  * the device.  when we pick a value to passback we
964                  * use the output device value since that was
965                  * the most recently queried.  (we base this
966                  * decision on the assumption that io gain is
967                  * actually hardware setting in the device and
968                  * hence if it is changed on one open instance of
969                  * the device the change will be visable to all
970                  * other instances of the device.)
971                  */
972                 ai_merged->monitor_gain = ai_odev->monitor_gain;
973         }
974
975         /*
976          * for currently enabled software features always return the
977          * merger of the two.  (of course the enabled software features
978          * for the input and output devices should alway be the same,
979          * so if it isn't complain.)
980          */
981         if (ai_idev->sw_features_enabled != ai_odev->sw_features_enabled)
982                 zcmn_err(getzoneid(), CE_WARN, "lx_audio: "
983                     "unexpected sofware feature state");
984         ai_merged->sw_features_enabled =
```



```
985             ai_idev->sw_features_enabled & ai_odev->sw_features_enabled;
986 }
987
988 static int
989 lxa_audio_setinfo(lxa_state_t *lxa_state, int cmd, intptr_t arg,
990     int mode)
991 {
992         audio_info_t    ai, ai_null, ai_idev, ai_odev;
993         int             rv, junk;
994
995         /* copy in the request */
996         if (ddi_copyin((void *)arg, &ai, sizeof (ai), mode) != 0)
997                 return (EFAULT);
998
999         /*
1000         * if the caller is attempting to enable a software feature that
1001         * we didn't report as supported the return an error
1002         */
1003        if ((ai.sw_features_enabled != -1) &&
1004            (ai.sw_features_enabled & ~lxa_state->lxas_sw_features))
1005                return (EINVAL);
1006
1007        /*
1008         * if a process has mmaped this device then we don't allow
1009         * changes to the play.eof field (since mmap output depends
1010         * on this field.
1011         */
1012        if ((lxa_state->lxas_umem_cookie != NULL) &&
1013            (ai.play.eof != -1))
1014                return (EIO);
1015
1016        /* initialize the new requests */
1017        AUDIO_INITINFO(&ai_null);
1018        ai_idev = ai_odev = ai;
1019
1020        /* remove audio input settings from the output device request */
1021        ai_odev.record = ai_null.record;
1022
1023        /* remove audio output settings from the input device request */
1024        ai_idev.play = ai_null.play;
1025        ai_idev.output_muted = ai_null.output_muted;
1026
1027        /* apply settings to the intput device */
1028        if ((lxa_state->lxas_idev_lh != NULL) &&
1029            ((rv = ldi_ioctl(lxa_state->lxas_idev_lh, cmd,
1030            (intptr_t)&ai_idev, FKIOCTL, kcred, &junk)) != 0))
1031                return (rv);
1032
1033        /* apply settings to the output device */
1034        if ((lxa_state->lxas_odev_lh != NULL) &&
1035            ((rv = ldi_ioctl(lxa_state->lxas_odev_lh, cmd,
1036            (intptr_t)&ai_odev, FKIOCTL, kcred, &junk)) != 0))
1037                return (rv);
1038
1039        /*
1040         * a AUDIO_SETINFO call performs an implicit AUDIO_GETINFO to
1041         * return values (see the coments in audioio.h.) so we need
1042         * to combine the values returned from the input and output
1043         * device back into the users buffer.
1044         */
1045        lxa_audio_info_merge(lxa_state, &ai_idev, &ai_odev, &ai);
1046
1047        /* copyout the results */
1048        if (ddi_copyout(&ai, (void *)arg, sizeof (ai), mode) != 0) {
1049                return (EFAULT);
1050        }
```

```
1052         return (0);
1053 }

1055 static int
1056 lxa_audio_getinfo(lxa_state_t *lxa_state, intptr_t arg, int mode)
1057 {
1058         audio_info_t    ai, ai_idev, ai_odev;
1059         int             rv, junk;

1061         /* get the settings from the input device */
1062         if ((lxa_state->lxas_idev_lh != NULL) &&
1063             ((rv = ldi_ioctl(lxa_state->lxas_idev_lh, AUDIO_GETINFO,
1064             (intptr_t)&ai_idev, FKIOCTL, kcred, &junk)) != 0))
1065                 return (rv);

1067         /* get the settings from the output device */
1068         if ((lxa_state->lxas_odev_lh != NULL) &&
1069             ((rv = ldi_ioctl(lxa_state->lxas_odev_lh, AUDIO_GETINFO,
1070             (intptr_t)&ai_odev, FKIOCTL, kcred, &junk)) != 0))
1071                 return (rv);

1073         /*
1074          * we need to combine the values returned from the input
1075          * and output device back into a single user buffer.
1076          */
1077         lxa_audio_info_merge(lxa_state, &ai_idev, &ai_odev, &ai);

1079         /* copyout the results */
1080         if (ddi_copyout(&ai, (void *)arg, sizeof (ai), mode) != 0)
1081                 return (EFAULT);

1083         return (0);
1084 }

1086 static int
1087 lxa_mixer_ai_from_lh(ldi_handle_t lh, audio_info_t *ai)
1088 {
1089         int             rv, junk;

1091         ASSERT((lh != NULL) && (ai != NULL));

1093         /* get the device state and channel state */
1094         rv = ldi_ioctl(lh, AUDIO_GETINFO, (intptr_t)ai, FKIOCTL, kcred, &junk);

1096         return (rv);
1097 }

1099 static int
1100 lxa_mixer_get_ai(lxa_state_t *lxa_state, audio_info_t *ai)
1101 {
1102         audio_info_t    ai_idev, ai_odev;
1103         int             rv;

1105         /* if there is no input device, query the output device */
1106         if (lxa_state->lxas_idev_lh == NULL)
1107                 return (lxa_mixer_ai_from_lh(lxa_state->lxas_odev_lh, ai));

1109         /* if there is no ouput device, query the intput device */
1110         if (lxa_state->lxas_odev_lh == NULL)
1111                 return (lxa_mixer_ai_from_lh(lxa_state->lxas_idev_lh, ai));

1113         /*
1114          * now get the audio_info and channel information for the
1115          * underlying output device.
1116          */
```



```
1117         if ((rv = lxa_mixer_ai_from_lh(lxa_state->lxas_idev_lh,
1118             &ai_idev)) != 0)
1119                 return (rv);
1120         if ((rv = lxa_mixer_ai_from_lh(lxa_state->lxas_odev_lh,
1121             &ai_odev)) != 0)
1122                 return (rv);

1124         /* now merge the audio_info structures */
1125         lxa_audio_info_merge(lxa_state, &ai_idev, &ai_odev, ai);
1126         return (0);
1127 }

1129 static int
1130 lxa_mixer_get_common(lxa_state_t *lxa_state, int cmd, intptr_t arg, int mode)
1131 {
1132         lxa_mixer_levels_t      lxa_ml;
1133         audio_info_t            ai;
1134         int                     rv;

1136         ASSERT(lxa_state->lxas_type == LXA_TYPE_AUDIOCTL);

1138         if ((rv = lxa_mixer_get_ai(lxa_state, &ai)) != 0)
1139                 return (rv);

1141         switch (cmd) {
1142         case LXA_IOC_MIXER_GET_VOL:
1143                 lxa_ml.lxa_ml_gain = ai.play.gain;
1144                 lxa_ml.lxa_ml_balance = ai.play.balance;
1145                 break;
1146         case LXA_IOC_MIXER_GET_MIC:
1147                 lxa_ml.lxa_ml_gain = ai.record.gain;
1148                 lxa_ml.lxa_ml_balance = ai.record.balance;
1149                 break;
1150         }

1152         if (ddi_copyout(&lxa_ml, (void *)arg, sizeof (lxa_ml), mode) != 0)
1153                 return (EFAULT);
1154         return (0);
1155 }

1157 static int
1158 lxa_mixer_set_common(lxa_state_t *lxa_state, int cmd, intptr_t arg, int mode)
1159 {
1160         lxa_mixer_levels_t      lxa_ml;
1161         audio_info_t            ai;

1163         ASSERT(lxa_state->lxas_type == LXA_TYPE_AUDIOCTL);

1165         /* get the new mixer settings */
1166         if (ddi_copyin((void *)arg, &lxa_ml, sizeof (lxa_ml), mode) != 0)
1167                 return (EFAULT);

1169         /* sanity check the mixer settings */
1170         if (!LXA_MIXER_LEVELS_OK(&lxa_ml))
1171                 return (EINVAL);

1173         /* initialize an audio_info struct with the new settings */
1174         AUDIO_INITINFO(&ai);
1175         switch (cmd) {
1176         case LXA_IOC_MIXER_SET_VOL:
1177                 ai.play.gain = lxa_ml.lxa_ml_gain;
1178                 ai.play.balance = lxa_ml.lxa_ml_balance;
1179                 break;
1180         case LXA_IOC_MIXER_SET_MIC:
1181                 ai.record.gain = lxa_ml.lxa_ml_gain;
1182                 ai.record.balance = lxa_ml.lxa_ml_balance;
```

```
1183                 break;
1184         }

1186         return (lxa_audio_setinfo(lxa_state, AUDIO_SETINFO, (intptr_t)&ai,
1187             FKIOCTL));
1188 }

1190 static int
1191 lxa_mixer_get_pcm(lxa_state_t *lxa_state, intptr_t arg, int mode)
1192 {
1193         ASSERT(lxa_state->lxas_type == LXA_TYPE_AUDIOCTL);

1195         /* simply return the cached pcm mixer settings */
1196         mutex_enter(&lxa_lock);
1197         if (ddi_copyout(&lxa_state->lxas_zs->lxa_zs_pcm_levels, (void *)arg,
1198             sizeof (lxa_state->lxas_zs->lxa_zs_pcm_levels), mode) != 0) {
1199                 mutex_exit(&lxa_lock);
1200                 return (EFAULT);
1201         }
1202         mutex_exit(&lxa_lock);
1203         return (0);
1204 }

1206 static int
1207 lxa_mixer_set_pcm(lxa_state_t *lxa_state, intptr_t arg, int mode)
1208 {
1209         lxa_mixer_levels_t      lxa_ml;
1210         int                     rv;

1212         ASSERT(lxa_state->lxas_type == LXA_TYPE_AUDIOCTL);

1214         /* get the new mixer settings */
1215         if (ddi_copyin((void *)arg, &lxa_ml, sizeof (lxa_ml), mode) != 0)
1216                 return (EFAULT);

1218         /* sanity check the mixer settings */
1219         if (!LXA_MIXER_LEVELS_OK(&lxa_ml))
1220                 return (EINVAL);

1222         mutex_enter(&lxa_lock);

1224         /* if there is an active output channel, update it */
1225         if (lxa_state->lxas_zs->lxa_zs_ostate != NULL) {
1226                 audio_info_t    ai;

1228                 /* initialize an audio_info struct with the new settings */
1229                 AUDIO_INITINFO(&ai);
1230                 ai.play.gain = lxa_ml.lxa_ml_gain;
1231                 ai.play.balance = lxa_ml.lxa_ml_balance;

1233                 if ((rv = lxa_audio_setinfo(lxa_state->lxas_zs->lxa_zs_ostate,
1234                     AUDIO_SETINFO, (intptr_t)&ai, FKIOCTL)) != 0) {
1235                         mutex_exit(&lxa_lock);
1236                         return (rv);
1237                 }
1238         }

1240         /* update the cached mixer settings */
1241         lxa_state->lxas_zs->lxa_zs_pcm_levels = lxa_ml;

1243         mutex_exit(&lxa_lock);
1244         return (0);
1245 }

1247 static int
1248 lxa_zone_reg(intptr_t arg, int mode)
```



```
1249 {
1250         lxa_zone_reg_t  lxa_zr;
1251         lxa_zstate_t    *lxa_zs = NULL;
1252         char            *idev_name = NULL, *odev_name = NULL, *pval = NULL;
1253         int             i, junk;

1255         if (ddi_copyin((void *)arg, &lxa_zr, sizeof (lxa_zr), mode) != 0)
1256                 return (EFAULT);

1258         /* make sure that zone_name is a valid string */
1259         for (i = 0; i < sizeof (lxa_zr.lxa_zr_zone_name); i++)
1260                 if (lxa_zr.lxa_zr_zone_name[i] == '\0')
1261                         break;
1262         if (i == sizeof (lxa_zr.lxa_zr_zone_name))
1263                 return (EINVAL);

1265         /* make sure that inputdev is a valid string */
1266         for (i = 0; i < sizeof (lxa_zr.lxa_zr_inputdev); i++)
1267                 if (lxa_zr.lxa_zr_inputdev[i] == '\0')
1268                         break;
1269         if (i == sizeof (lxa_zr.lxa_zr_inputdev))
1270                 return (EINVAL);

1272         /* make sure it's a valid inputdev property value */
1273         if (lxa_devprop_verify(lxa_zr.lxa_zr_inputdev) != 0)
1274                 return (EINVAL);

1276         /* make sure that outputdev is a valid string */
1277         for (i = 0; i < sizeof (lxa_zr.lxa_zr_outputdev); i++)
1278                 if (lxa_zr.lxa_zr_outputdev[i] == '\0')
1279                         break;
1280         if (i == sizeof (lxa_zr.lxa_zr_outputdev))
1281                 return (EINVAL);

1283         /* make sure it's a valid outputdev property value */
1284         if (lxa_devprop_verify(lxa_zr.lxa_zr_outputdev) != 0)
1285                 return (EINVAL);

1287         /* get the property names */
1288         idev_name = lxa_devprop_name(lxa_zr.lxa_zr_zone_name,
1289             LXA_PROP_INPUTDEV);
1290         odev_name = lxa_devprop_name(lxa_zr.lxa_zr_zone_name,
1291             LXA_PROP_OUTPUTDEV);

1293         /*
1294          * allocate and initialize a zone state structure
1295          * since the audio device can't possibly be opened yet
1296          * (since we're setting it up now and the zone isn't booted
1297          * yet) assign some some resonable default pcm channel settings.
1298          * also, default to one mixer channel.
1299          */
1300         lxa_zs = kmem_zalloc(sizeof (*lxa_zs), KM_SLEEP);
1301         lxa_zs->lxa_zs_zonename = strdup(lxa_zr.lxa_zr_zone_name);
1302         lxa_zs->lxa_zs_pcm_levels.lxa_ml_gain = AUDIO_MID_GAIN;
1303         lxa_zs->lxa_zs_pcm_levels.lxa_ml_balance = AUDIO_MID_BALANCE;

1305         mutex_enter(&lxa_lock);

1307         /*
1308          * make sure this zone isn't already registered
1309          * a zone is registered with properties for that zone exist
1310          * or there is a zone state structure for that zone
1311          */
1312         if (ddi_prop_lookup_string(DDI_DEV_T_ANY, lxa_dip,
1313             DDI_PROP_DONTPASS | DDI_PROP_NOTPROM,
1314             idev_name, &pval) == DDI_PROP_SUCCESS) {
```

```
1315                goto err_unlock;
1316        }
1317        if (ddi_prop_lookup_string(DDI_DEV_T_ANY, lxa_dip,
1318            DDI_PROP_DONTPASS | DDI_PROP_NOTPROM,
1319            odev_name, &pval) == DDI_PROP_SUCCESS) {
1320                goto err_unlock;
1321        }
1322        if (mod_hash_find(lxa_zstate_hash,
1323            (mod_hash_key_t)lxa_zs->lxa_zs_zonename,
1324            (mod_hash_val_t *)&junk) == 0)
1325                goto err_unlock;

1327        /*
1328         * create the new properties and insert the zone state structure
1329         * into the global hash
1330         */
1331        if (ddi_prop_update_string(DDI_DEV_T_NONE, lxa_dip,
1332            idev_name, lxa_zr.lxa_zr_inputdev) != DDI_PROP_SUCCESS)
1333                goto err_prop_remove;
1334        if (ddi_prop_update_string(DDI_DEV_T_NONE, lxa_dip,
1335            odev_name, lxa_zr.lxa_zr_outputdev) != DDI_PROP_SUCCESS)
1336                goto err_prop_remove;
1337        if (mod_hash_insert(lxa_zstate_hash,
1338            (mod_hash_key_t)lxa_zs->lxa_zs_zonename,
1339            (mod_hash_val_t)lxa_zs) != 0)
1340                goto err_prop_remove;

1342        /* success! */
1343        lxa_registered_zones++;
1344        mutex_exit(&lxa_lock);

1346        /* cleanup */
1347        strfree(idev_name);
1348        strfree(odev_name);
1349        return (0);

1351 err_prop_remove:
1352        (void) ddi_prop_remove(DDI_DEV_T_NONE, lxa_dip, idev_name);
1353        (void) ddi_prop_remove(DDI_DEV_T_NONE, lxa_dip, odev_name);

1355 err_unlock:
1356        mutex_exit(&lxa_lock);

1358        if (lxa_zs != NULL) {
1359                strfree(lxa_zs->lxa_zs_zonename);
1360                kmem_free(lxa_zs, sizeof (*lxa_zs));
1361        }
1362        if (pval != NULL)
1363                ddi_prop_free(pval);
1364        if (idev_name != NULL)
1365                strfree(idev_name);
1366        if (odev_name != NULL)
1367                strfree(odev_name);
1368        return (EIO);
1369 }

1371 static int
1372 lxa_zone_unreg(intptr_t arg, int mode)
1373 {
1374        lxa_zone_reg_t  lxa_zr;
1375        lxa_zstate_t    *lxa_zs = NULL;
1376        char            *idev_name = NULL, *odev_name = NULL, *pval = NULL;
1377        int             rv, i;

1379        if (ddi_copyin((void *)arg, &lxa_zr, sizeof (lxa_zr), mode) != 0)
1380                return (EFAULT);
```



```
1382        /* make sure that zone_name is a valid string */
1383        for (i = 0; i < sizeof (lxa_zr.lxa_zr_zone_name); i++)
1384                if (lxa_zr.lxa_zr_zone_name[i] == '\0')
1385                        break;
1386        if (i == sizeof (lxa_zr.lxa_zr_zone_name))
1387                return (EINVAL);

1389        /* get the property names */
1390        idev_name = lxa_devprop_name(lxa_zr.lxa_zr_zone_name,
1391            LXA_PROP_INPUTDEV);
1392        odev_name = lxa_devprop_name(lxa_zr.lxa_zr_zone_name,
1393            LXA_PROP_OUTPUTDEV);

1395        mutex_enter(&lxa_lock);

1397        if (lxa_registered_zones <= 0) {
1398                rv = ENOENT;
1399                goto err_unlock;
1400        }

1402        /* make sure this zone is actually registered */
1403        if (ddi_prop_lookup_string(DDI_DEV_T_ANY, lxa_dip,
1404            DDI_PROP_DONTPASS | DDI_PROP_NOTPROM,
1405            idev_name, &pval) != DDI_PROP_SUCCESS) {
1406                rv = ENOENT;
1407                goto err_unlock;
1408        }
1409        ddi_prop_free(pval);
1410        pval = NULL;
1411        if (ddi_prop_lookup_string(DDI_DEV_T_ANY, lxa_dip,
1412            DDI_PROP_DONTPASS | DDI_PROP_NOTPROM,
1413            odev_name, &pval) != DDI_PROP_SUCCESS) {
1414                rv = ENOENT;
1415                goto err_unlock;
1416        }
1417        ddi_prop_free(pval);
1418        pval = NULL;
1419        if (mod_hash_find(lxa_zstate_hash,
1420            (mod_hash_key_t)lxa_zr.lxa_zr_zone_name,
1421            (mod_hash_val_t *)&lxa_zs) != 0) {
1422                rv = ENOENT;
1423                goto err_unlock;
1424        }
1425        ASSERT(strcmp(lxa_zr.lxa_zr_zone_name, lxa_zs->lxa_zs_zonename) == 0);

1427        /*
1428         * if the audio device is currently in use then refuse to
1429         * unregister the zone
1430         */
1431        if ((lxa_zs->lxa_zs_ostate != NULL) ||
1432            (lxa_zs->lxa_zs_ostate != NULL)) {
1433                rv = EBUSY;
1434                goto err_unlock;
1435        }

1437        /* success! cleanup zone config state */
1438        (void) ddi_prop_remove(DDI_DEV_T_NONE, lxa_dip, idev_name);
1439        (void) ddi_prop_remove(DDI_DEV_T_NONE, lxa_dip, odev_name);

1441        /*
1442         * note, the action of removing the zone state structure from the
1443         * hash will automatically free lxa_zs->lxa_zs_zonename.
1444         *
1445         * the reason for this is that we used lxa_zs->lxa_zs_zonename
1446         * as the hash key and by default mod_hash_create_strhash() uses
```

```
1447          * mod_hash_strkey_dtor() as a the hash key destructor.  (which
1448          * free's the key for us.
1449          */
1450         (void) mod_hash_remove(lxa_zstate_hash,
1451             (mod_hash_key_t)lxa_zr.lxa_zr_zone_name,
1452             (mod_hash_val_t *)&lxa_zs);
1453         lxa_registered_zones--;
1454         mutex_exit(&lxa_lock);

1456         /* cleanup */
1457         kmem_free(lxa_zs, sizeof (*lxa_zs));
1458         strfree(idev_name);
1459         strfree(odev_name);
1460         return (0);

1462 err_unlock:
1463         mutex_exit(&lxa_lock);

1465         if (pval != NULL)
1466                 ddi_prop_free(pval);
1467         if (idev_name != NULL)
1468                 strfree(idev_name);
1469         if (odev_name != NULL)
1470                 strfree(odev_name);
1471         return (rv);
1472 }

1474 static int
1475 lxa_ioctl_devctl(int cmd, intptr_t arg, int mode)
1476 {
1477         /* devctl ioctls are only allowed from the global zone */
1478         ASSERT(getzoneid() == 0);
1479         if (getzoneid() != 0)
1480                 return (EINVAL);

1482         switch (cmd) {
1483         case LXA_IOC_ZONE_REG:
1484                 return (lxa_zone_reg(arg, mode));
1485         case LXA_IOC_ZONE_UNREG:
1486                 return (lxa_zone_unreg(arg, mode));
1487         }

1489         return (EINVAL);
1490 }

1492 static int
1493 /*ARGSUSED*/
1494 lxa_open(dev_t *devp, int flags, int otyp, cred_t *credp)
1495 {
1496         lxa_dev_type_t  open_type = LXA_TYPE_INVALID;
1497         lxa_zstate_t    *lxa_zs;
1498         lxa_state_t     *lxa_state;
1499         minor_t         minor;
1500         int             rv;

1502         if (getminor(*devp) == LXA_MINORNUM_DEVCTL) {
1503                 /*
1504                  * this is a devctl node, it exists to administer this
1505                  * pseudo driver so it doesn't actually need access to
1506                  * any underlying audio devices.  hence there is nothing
1507                  * really to do here.  course, this driver should
1508                  * only be administered from the global zone.
1509                  */
1510                 ASSERT(getzoneid() == 0);
1511                 if (getzoneid() != 0)
1512                         return (EINVAL);
```



```
1513                 return (0);
1514         }

1516         /* lookup the zone state structure */
1517         if (mod_hash_find(lxa_zstate_hash, (mod_hash_key_t)getzonename(),
1518             (mod_hash_val_t *)&lxa_zs) != 0) {
1519                 return (EIO);
1520         }

1522         /* determine what type of device was opened */
1523         switch (getminor(*devp)) {
1524         case LXA_MINORNUM_DSP:
1525                 open_type = LXA_TYPE_AUDIO;
1526                 break;
1527         case LXA_MINORNUM_MIXER:
1528                 open_type = LXA_TYPE_AUDIOCTL;
1529                 break;
1530         default:
1531                 return (EINVAL);
1532         }
1533         ASSERT(open_type != LXA_TYPE_INVALID);

1535         /* all other opens are clone opens so get a new minor node */
1536         minor = id_alloc(lxa_minor_id);

1538         /* allocate and initialize the new lxa_state structure */
1539         lxa_state = kmem_zalloc(sizeof (*lxa_state), KM_SLEEP);
1540         lxa_state->lxas_zs = lxa_zs;
1541         lxa_state->lxas_dev_old = *devp;
1542         lxa_state->lxas_dev_new = makedevice(getmajor(*devp), minor);
1543         lxa_state->lxas_flags = flags;
1544         lxa_state->lxas_type = open_type;

1546         /* initialize the input and output device */
1547         if (((rv = lxa_dev_open(lxa_state)) != 0) ||
1548             ((rv = lxa_dev_getfeatures(lxa_state)) != 0)) {
1549                 lxa_state_close(lxa_state);
1550                 return (rv);
1551         }

1553         /*
1554          * save this audio statue structure into a hash indexed
1555          * by it's minor device number.  (this will provide a convient
1556          * way to lookup the state structure on future operations.)
1557          */
1558         if (mod_hash_insert(lxa_state_hash, (mod_hash_key_t)(uintptr_t)minor,
1559             (mod_hash_val_t)lxa_state) != 0) {
1560                 lxa_state_close(lxa_state);
1561                 return (EIO);
1562         }

1564         mutex_enter(&lxa_lock);

1566         /* apply the currently cached zone PCM mixer levels */
1567         if ((lxa_state->lxas_type == LXA_TYPE_AUDIO) &&
1568             (lxa_state->lxas_odev_lh != NULL)) {
1569                 audio_info_t ai;

1571                 AUDIO_INITINFO(&ai);
1572                 ai.play.gain = lxa_zs->lxa_zs_pcm_levels.lxa_ml_gain;
1573                 ai.play.balance = lxa_zs->lxa_zs_pcm_levels.lxa_ml_balance;

1575                 if ((rv = lxa_audio_setinfo(lxa_state,
1576                     AUDIO_SETINFO, (intptr_t)&ai, FKIOCTL)) != 0) {
1577                         mutex_exit(&lxa_lock);
1578                         lxa_state_close(lxa_state);
```

```
1579                        return (rv);
1580                }
1581        }

1583        /*
1584         * we only allow one active open of the input or output device.
1585         * check here for duplicate opens
1586         */
1587        if (lxa_state->lxas_type == LXA_TYPE_AUDIO) {
1588                if ((lxa_state->lxas_idev_lh != NULL) &&
1589                    (lxa_zs->lxa_zs_istate != NULL)) {
1590                        mutex_exit(&lxa_lock);
1591                        lxa_state_close(lxa_state);
1592                        return (EBUSY);
1593                }
1594                if ((lxa_state->lxas_odev_lh != NULL) &&
1595                    (lxa_zs->lxa_zs_ostate != NULL)) {
1596                        mutex_exit(&lxa_lock);
1597                        lxa_state_close(lxa_state);
1598                        return (EBUSY);
1599                }

1601                /* not a duplicate open, update the global zone state */
1602                if (lxa_state->lxas_idev_lh != NULL)
1603                        lxa_zs->lxa_zs_istate = lxa_state;
1604                if (lxa_state->lxas_odev_lh != NULL)
1605                        lxa_zs->lxa_zs_ostate = lxa_state;
1606        }
1607        mutex_exit(&lxa_lock);

1609        /* make sure to return our newly allocated dev_t */
1610        *devp = lxa_state->lxas_dev_new;
1611        return (0);
1612 }

1614 static int
1615 /*ARGSUSED*/
1616 lxa_close(dev_t dev, int flags, int otyp, cred_t *credp)
1617 {
1618        lxa_state_t     *lxa_state;
1619        minor_t         minor = getminor(dev);

1621        /* handle devctl minor nodes (these nodes don't have a handle */
1622        if (getminor(dev) == LXA_MINORNUM_DEVCTL)
1623                return (0);

1625        /* get the handle for this device */
1626        if (mod_hash_find(lxa_state_hash, (mod_hash_key_t)(uintptr_t)minor,
1627            (mod_hash_val_t *)&lxa_state) != 0)
1628                return (EINVAL);

1630        lxa_state_close(lxa_state);
1631        return (0);
1632 }

1634 static int
1635 /*ARGSUSED*/
1636 lxa_read(dev_t dev, struct uio *uiop, cred_t *credp)
1637 {
1638        lxa_state_t     *lxa_state;
1639        minor_t         minor = getminor(dev);
1640        int             rv;

1642        /* get the handle for this device */
1643        if (mod_hash_find(lxa_state_hash, (mod_hash_key_t)(uintptr_t)minor,
1644            (mod_hash_val_t *)&lxa_state) != 0)
```



```
1645                return (EINVAL);

1647        /*
1648         * if a process has mmaped this device then we don't allow
1649         * any more reads or writes to the device
1650         */
1651        if (lxa_state->lxas_umem_cookie != NULL)
1652                return (EIO);

1654        /* we can't do a read if there is no input device */
1655        if (lxa_state->lxas_idev_lh == NULL)
1656                return (EBADF);

1658        /* pass the request on */
1659        while (uiop->uio_resid != 0) {
1660                rv = ldi_read(lxa_state->lxas_idev_lh, uiop, kcred);
1661                if ((rv != 0) || (uiop->uio_fmode & (FNONBLOCK|FNDELAY))) {
1662                        break;
1663                }
1664        }
1665        return (rv);
1666 }

1668 static int
1669 /*ARGSUSED*/
1670 lxa_write(dev_t dev, struct uio *uiop, cred_t *credp)
1671 {
1672        lxa_state_t     *lxa_state;
1673        minor_t         minor = getminor(dev);
1674        int             rv;

1676        /* get the handle for this device */
1677        if (mod_hash_find(lxa_state_hash, (mod_hash_key_t)(uintptr_t)minor,
1678            (mod_hash_val_t *)&lxa_state) != 0)
1679                return (EINVAL);

1681        /*
1682         * if a process has mmaped this device then we don't allow
1683         * any more reads or writes to the device
1684         */
1685        if (lxa_state->lxas_umem_cookie != NULL)
1686                return (EIO);

1688        /* we can't do a write if there is no output device */
1689        if (lxa_state->lxas_odev_lh == NULL)
1690                return (EBADF);

1692        /* pass the request on */
1693        while (uiop->uio_resid != 0) {
1694                rv = ldi_write(lxa_state->lxas_odev_lh, uiop, kcred);
1695                if ((rv != 0) || (uiop->uio_fmode & (FNONBLOCK|FNDELAY))) {
1696                        break;
1697                }
1698        }
1699        return (rv);
1700 }

1702 static int
1703 /*ARGSUSED*/
1704 lxa_ioctl(dev_t dev, int cmd, intptr_t arg, int mode, cred_t *credp,
1705     int *rvalp)
1706 {
1707        lxa_state_t     *lxa_state;
1708        minor_t         minor = getminor(dev);

1710        /* handle devctl minor nodes (these nodes don't have a handle */
```

```
1711         if (getminor(dev) == LXA_MINORNUM_DEVCTL)
1712                 return (lxa_ioctl_devctl(cmd, arg, mode));
1713
1714         /* get the handle for this device */
1715         if (mod_hash_find(lxa_state_hash, (mod_hash_key_t)(uintptr_t)minor,
1716             (mod_hash_val_t *)&lxa_state) != 0)
1717                 return (EINVAL);
1718
1719         ASSERT((lxa_state->lxas_type == LXA_TYPE_AUDIO) ||
1720             (lxa_state->lxas_type == LXA_TYPE_AUDIOCTL));
1721
1722         switch (cmd) {
1723         case LXA_IOC_GETMINORNUM:
1724                 {
1725                         int minornum = getminor(lxa_state->lxas_dev_old);
1726                         if (ddi_copyout(&minornum, (void *)arg,
1727                             sizeof (minornum), mode) != 0)
1728                                 return (EFAULT);
1729                 }
1730                 return (0);
1731         }
1732
1733         if (lxa_state->lxas_type == LXA_TYPE_AUDIO) {
1734                 /* deal with native ioctl */
1735                 switch (cmd) {
1736                 case LXA_IOC_MMAP_OUTPUT:
1737                         return (lxa_ioc_mmap_output(lxa_state, arg, mode));
1738                 case LXA_IOC_MMAP_PTR:
1739                         return (lxa_ioc_mmap_ptr(lxa_state, arg, mode));
1740                 case LXA_IOC_GET_FRAG_INFO:
1741                         return (lxa_ioc_get_frag_info(lxa_state, arg, mode));
1742                 case LXA_IOC_SET_FRAG_INFO:
1743                         return (lxa_ioc_set_frag_info(lxa_state, arg, mode));
1744                 }
1745
1746                 /* deal with layered ioctls */
1747                 switch (cmd) {
1748                 case AUDIO_DRAIN:
1749                         return (lxa_audio_drain(lxa_state));
1750                 case AUDIO_SETINFO:
1751                         return (lxa_audio_setinfo(lxa_state,
1752                             AUDIO_SETINFO, arg, mode));
1753                 case AUDIO_GETINFO:
1754                         return (lxa_audio_getinfo(lxa_state, arg, mode));
1755                 }
1756         }
1757
1758         if (lxa_state->lxas_type == LXA_TYPE_AUDIOCTL) {
1759                 /* deal with native ioctl */
1760                 switch (cmd) {
1761                 case LXA_IOC_MIXER_GET_VOL:
1762                         return (lxa_mixer_get_common(lxa_state,
1763                             cmd, arg, mode));
1764                 case LXA_IOC_MIXER_SET_VOL:
1765                         return (lxa_mixer_set_common(lxa_state,
1766                             cmd, arg, mode));
1767                 case LXA_IOC_MIXER_GET_MIC:
1768                         return (lxa_mixer_get_common(lxa_state,
1769                             cmd, arg, mode));
1770                 case LXA_IOC_MIXER_SET_MIC:
1771                         return (lxa_mixer_set_common(lxa_state,
1772                             cmd, arg, mode));
1773                 case LXA_IOC_MIXER_GET_PCM:
1774                         return (lxa_mixer_get_pcm(lxa_state, arg, mode));
1775                 case LXA_IOC_MIXER_SET_PCM:
1776                         return (lxa_mixer_set_pcm(lxa_state, arg, mode));
```



```
1777                 }
1778
1779         }
1780
1781         return (EINVAL);
1782 }
1783
1784 static int
1785 /*ARGSUSED*/
1786 lxa_devmap(dev_t dev, devmap_cookie_t dhp,
1787     offset_t off, size_t len, size_t *maplen, uint_t model)
1788 {
1789         lxa_state_t             *lxa_state;
1790         minor_t                 minor = getminor(dev);
1791         ddi_umem_cookie_t       umem_cookie;
1792         void                    *umem_ptr;
1793         int                     rv;
1794
1795         /* get the handle for this device */
1796         if (mod_hash_find(lxa_state_hash, (mod_hash_key_t)(uintptr_t)minor,
1797             (mod_hash_val_t *)&lxa_state) != 0)
1798                 return (EINVAL);
1799
1800         /* we only support mmaping of audio devices */
1801         if (lxa_state->lxas_type != LXA_TYPE_AUDIO)
1802                 return (EINVAL);
1803
1804         /* we only support output via mmap */
1805         if ((lxa_state->lxas_flags & FWRITE) == 0)
1806                 return (EINVAL);
1807
1808         /* sanity check the amount of memory the user is allocating */
1809         if ((len == 0) ||
1810             (len > LXA_OSS_FRAG_MEM) ||
1811             ((len % lxa_state->lxas_frag_size) != 0))
1812                 return (EINVAL);
1813
1814         /* allocate and clear memory to mmap */
1815         umem_ptr = ddi_umem_alloc(len, DDI_UMEM_NOSLEEP, &umem_cookie);
1816         if (umem_ptr == NULL)
1817                 return (ENOMEM);
1818         bzero(umem_ptr, len);
1819
1820         /* setup the memory mappings */
1821         rv = devmap_umem_setup(dhp, lxa_dip, NULL, umem_cookie, 0, len,
1822             PROT_USER | PROT_READ | PROT_WRITE, 0, NULL);
1823         if (rv != 0) {
1824                 ddi_umem_free(umem_cookie);
1825                 return (EIO);
1826         }
1827
1828         mutex_enter(&lxa_lock);
1829
1830         /* we only support one mmap per open */
1831         if (lxa_state->lxas_umem_cookie != NULL) {
1832                 ASSERT(lxa_state->lxas_umem_ptr != NULL);
1833                 mutex_exit(&lxa_lock);
1834                 ddi_umem_free(umem_cookie);
1835                 return (EBUSY);
1836         }
1837         ASSERT(lxa_state->lxas_umem_ptr == NULL);
1838
1839         *maplen = len;
1840         lxa_state->lxas_umem_len = len;
1841         lxa_state->lxas_umem_ptr = umem_ptr;
1842         lxa_state->lxas_umem_cookie = umem_cookie;
```

```
1843         mutex_exit(&lxa_lock);
1844         return (0);
1845 }

1847 static int
1848 /*ARGSUSED*/
1849 lxa_attach(dev_info_t *dip, ddi_attach_cmd_t cmd)
1850 {
1851         int     instance = ddi_get_instance(dip);

1853         if (cmd != DDI_ATTACH)
1854                 return (DDI_FAILURE);

1856         ASSERT(instance == 0);
1857         if (instance != 0)
1858                 return (DDI_FAILURE);

1860         lxa_dip = dip;
1861         mutex_init(&lxa_lock, NULL, MUTEX_DEFAULT, NULL);

1863         /* create our minor nodes */
1864         if (ddi_create_minor_node(dip, LXA_MINORNAME_DEVCTL, S_IFCHR,
1865             LXA_MINORNUM_DEVCTL, DDI_PSEUDO, 0) != DDI_SUCCESS)
1866                 return (DDI_FAILURE);

1868         if (ddi_create_minor_node(dip, LXA_MINORNAME_DSP, S_IFCHR,
1869             LXA_MINORNUM_DSP, DDI_PSEUDO, 0) != DDI_SUCCESS)
1870                 return (DDI_FAILURE);

1872         if (ddi_create_minor_node(dip, LXA_MINORNAME_MIXER, S_IFCHR,
1873             LXA_MINORNUM_MIXER, DDI_PSEUDO, 0) != DDI_SUCCESS)
1874                 return (DDI_FAILURE);

1876         /* allocate our data structures */
1877         lxa_minor_id = id_space_create("lxa_minor_id",
1878             LXA_MINORNUM_COUNT, LX_AUDIO_MAX_OPENS);
1879         lxa_state_hash = mod_hash_create_idhash("lxa_state_hash",
1880             lxa_state_hash_size, mod_hash_null_valdtor);
1881         lxa_zstate_hash = mod_hash_create_strhash("lxa_zstate_hash",
1882             lxa_zstate_hash_size, mod_hash_null_valdtor);

1884         return (DDI_SUCCESS);
1885 }

1887 static int
1888 /*ARGSUSED*/
1889 lxa_detach(dev_info_t *dip, ddi_detach_cmd_t cmd)
1890 {
1891         if (cmd != DDI_DETACH)
1892                 return (DDI_FAILURE);

1894         ASSERT(!MUTEX_HELD(&lxa_lock));
1895         if (lxa_registered_zones > 0)
1896                 return (DDI_FAILURE);

1898         mod_hash_destroy_idhash(lxa_state_hash);
1899         mod_hash_destroy_idhash(lxa_zstate_hash);
1900         id_space_destroy(lxa_minor_id);
1901         lxa_state_hash = NULL;
1902         lxa_dip = NULL;

1904         return (DDI_SUCCESS);
1905 }

1907 static int
1908 /*ARGSUSED*/
```



```
1909 lxa_getinfo(dev_info_t *dip, ddi_info_cmd_t infocmd, void *arg, void **resultp)
1910 {
1911         switch (infocmd) {
1912         case DDI_INFO_DEVT2DEVINFO:
1913                 *resultp = lxa_dip;
1914                 return (DDI_SUCCESS);

1916         case DDI_INFO_DEVT2INSTANCE:
1917                 *resultp = (void *)0;
1918                 return (DDI_SUCCESS);
1919         }
1920         return (DDI_FAILURE);
1921 }

1923 /*
1924  * Driver flags
1925  */
1926 static struct cb_ops lxa_cb_ops = {
1927         lxa_open,               /* open */
1928         lxa_close,              /* close */
1929         nodev,                  /* strategy */
1930         nodev,                  /* print */
1931         nodev,                  /* dump */
1932         lxa_read,               /* read */
1933         lxa_write,              /* write */
1934         lxa_ioctl,              /* ioctl */
1935         lxa_devmap,             /* devmap */
1936         nodev,                  /* mmap */
1937         ddi_devmap_segmap,      /* segmap */
1938         nochpoll,               /* chpoll */
1939         ddi_prop_op,            /* prop_op */
1940         NULL,                   /* cb_str */
1941         D_NEW | D_MP | D_DEVMAP,
1942         CB_REV,
1943         NULL,
1944         NULL
1945 };

1947 static struct dev_ops lxa_ops = {
1948         DEVO_REV,
1949         0,
1950         lxa_getinfo,
1951         nulldev,
1952         nulldev,
1953         lxa_attach,
1954         lxa_detach,
1955         nodev,
1956         &lxa_cb_ops,
1957         NULL,
1958         NULL,
1959         ddi_quiesce_not_needed,         /* quiesce */
1960 };

1962 /*
1963  * Module linkage information for the kernel.
1964  */
1965 static struct modldrv modldrv = {
1966         &mod_driverops,         /* type of module */
1967         "linux audio driver",   /* description of module */
1968         &lxa_ops                /* driver ops */
1969 };

1971 static struct modlinkage modlinkage = {
1972         MODREV_1,
1973         &modldrv,
1974         NULL
```

```
1975 };

1977 /*
1978  * standard module entry points
1979  */
1980 int
1981 _init(void)
1982 {
1983         return (mod_install(&modlinkage));
1984 }

1986 int
1987 _fini(void)
1988 {
1989         return (mod_remove(&modlinkage));
1990 }

1992 int
1993 _info(struct modinfo *modinfop)
1994 {
1995         return (mod_info(&modlinkage, modinfop));
1996 }
1997 #endif /* ! codereview */
```

```
**********************************************************
     973 Tue Jan 14 16:17:18 2014
new/usr/src/uts/common/brand/lx/io/lx_audio.conf
Bring back LX zones.
**********************************************************
```

```
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #
  21 #
  22 # Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  23 # Use is subject to license terms.
  24 #
  25 #ident  "%Z%%M% %I%     %E% SMI"

  27 name="lx_audio" parent="pseudo" instance=0;
  28 #endif /* ! codereview */
```

```
**********************************************************
   31891 Tue Jan 14 16:17:18 2014
new/usr/src/uts/common/brand/lx/io/lx_ptm.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */


/*
 * This driver attempts to emulate some of the the behaviors of
 * Linux terminal devices (/dev/ptmx and /dev/pts/[0-9][0-9]*) on Solaris
 *
 * It does this by layering over the /dev/ptmx device and intercepting
 * opens to it.
 *
 * This driver makes the following assumptions about the way the ptm/pts
 * drivers on Solaris work:
 *
 *    - all opens of the /dev/ptmx device node return a unique dev_t.
 *
 *    - the dev_t minor node value for each open ptm instance corrospondes
 *      to it's associated slave terminal device number.  ie. the path to
 *      the slave terminal device associated with an open ptm instance
 *      who's dev_t minor node vaue is 5, is /dev/pts/5.
 *
 *    - the ptm driver always allocates the lowest numbered slave terminal
 *      device possible.
 */

#include <sys/conf.h>
#include <sys/ddi.h>
#include <sys/devops.h>
#include <sys/file.h>
#include <sys/filio.h>
#include <sys/kstr.h>
#include <sys/ldlinux.h>
#include <sys/lx_ptm.h>
#include <sys/modctl.h>
#include <sys/pathname.h>
#include <sys/ptms.h>
#include <sys/ptyvar.h>
#include <sys/stat.h>
#include <sys/stropts.h>
```



```
#include <sys/sunddi.h>
#include <sys/sunldi.h>
#include <sys/sysmacros.h>
#include <sys/types.h>

#define LP_PTM_PATH             "/dev/ptmx"
#define LP_PTS_PATH             "/dev/pts/"
#define LP_PTS_DRV_NAME         "pts"
#define LP_PTS_USEC_DELAY       (5 * 1000)      /* 5 ms */
#define LP_PTS_USEC_DELAY_MAX   (5 * MILLISEC)  /* 5 ms */

/*
 * this driver is layered on top of the ptm driver.  we'd like to
 * make this drivers minor name space a mirror of the ptm drivers
 * namespace, but we can't actually do this.  the reason is that the
 * ptm driver is opened via the clone driver.  there for no minor nodes
 * of the ptm driver are actually accessible via the filesystem.
 * since we're not a streams device we can't be opened by the clone
 * driver.  there for we need to have at least minor node accessible
 * via the filesystem so that consumers can open it.  we use the device
 * node with a minor number of 0 for this purpose.  what this means is
 * that minor node 0 can't be used to map ptm minor node 0.  since this
 * minor node is now reserved we need to shift our ptm minor node
 * mappings by one.  ie. a ptm minor node with a value of 0 will
 * corrospond to our minor node with a value of 1.  these mappings are
 * managed with the following macros.
 */
#define DEVT_TO_INDEX(x)        LX_PTM_DEV_TO_PTS(x)
#define INDEX_TO_MINOR(x)       ((x) + 1)

/*
 * grow our layered handle array by the same size increment that the ptm
 * driver uses to grow the pty device space - PTY_MAXDELTA
 */
#define LP_PTY_INC      128

/*
 * lx_ptm_ops contains state information about outstanding operations on the
 * underlying master terminal device.  Currently we only track information
 * for read operations.
 *
 * Note that this data has not been rolled directly into the lx_ptm_handle
 * structure because we can't put mutex's of condition variables into
 * lx_ptm_handle structure.  The reason is that the array of lx_ptm_handle
 * structures linked to from the global lx_ptm state can be resized
 * dynamically, and when it's resized, the new array is at a different
 * memory location and the old array memory is discarded.  Mutexs and cvs
 * are accessed based off their address, so if this array was re-sized while
 * there were outstanding operations on any mutexs or cvs in the array
 * then the system would tip over.  In the future the lx_ptm_handle structure
 * array should probably be replaced with either an array of pointers to
 * lx_ptm_handle structures or some other kind of data structure containing
 * pointers to lx_ptm_handle structures.  Then the lx_ptm_ops structure
 * could be folded directly into the lx_ptm_handle structures.  (This will
 * also require the definition of a new locking mechanism to protect the
 * contents of lx_ptm_handle structures.)
 */
typedef struct lx_ptm_ops {
        int                     lpo_rops;
        kcondvar_t              lpo_rops_cv;
        kmutex_t                lpo_rops_lock;
} lx_ptm_ops_t;

/*
 * Every open of the master terminal device in a zone results in a new
 * lx_ptm_handle handle allocation.  These handles are stored in an array
```

```
 * hanging off the lx_ptm_state structure.
 */
typedef struct lx_ptm_handle {
        /* Device handle to the underlying real /dev/ptmx master terminal. */
        ldi_handle_t            lph_handle;

        /* Flag to indicate if TIOCPKT mode has been enabled. */
        int                     lph_pktio;

        /* Number of times the slave device has been opened/closed. */
        int                     lph_eofed;

        /* Callback handler in the ptm driver to check if slave is open. */
        ptmptsopencb_t          lph_ppocb;

        /* Pointer to state for operations on underlying device. */
        lx_ptm_ops_t            *lph_lpo;
} lx_ptm_handle_t;

/*
 * Global state for the lx_ptm driver.
 */
typedef struct lx_ptm_state {
        /* lx_ptm device devinfo pointer */
        dev_info_t              *lps_dip;

        /* LDI ident used to open underlying real /dev/ptmx master terminals. */
        ldi_ident_t             lps_li;

        /* pts drivers major number */
        major_t                 lps_pts_major;

        /* rw lock used to manage access and growth of lps_lh_array */
        krwlock_t               lps_lh_rwlock;

        /* number of elements in lps_lh_array */
        uint_t                  lps_lh_count;

        /* Array of handles to underlying real /dev/ptmx master terminals. */
        lx_ptm_handle_t         *lps_lh_array;
} lx_ptm_state_t;

/* Pointer to the lx_ptm global state structure. */
static lx_ptm_state_t   lps;

/*
 * List of modules to be autopushed onto slave terminal devices when they
 * are opened in an lx branded zone.
 */
static char *lx_pts_mods[] = {
        "ptem",
        "ldterm",
        "ttcompat",
        LDLINUX_MOD,
        NULL
};

static void
lx_ptm_lh_grow(uint_t index)
{
        uint_t                  new_lh_count, old_lh_count;
        lx_ptm_handle_t         *new_lh_array, *old_lh_array;

        /*
         * allocate a new array.  we drop the rw lock on the array so that
         * readers can still access devices in case our memory allocation
```



```
         * blocks.
         */
        new_lh_count = MAX(lps.lps_lh_count + LP_PTY_INC, index + 1);
        new_lh_array =
            kmem_zalloc(sizeof (lx_ptm_handle_t) * new_lh_count, KM_SLEEP);

        /*
         * double check that we still actually need to increase the size
         * of the array
         */
        rw_enter(&lps.lps_lh_rwlock, RW_WRITER);
        if (index < lps.lps_lh_count) {
                /* someone beat us to it so there's nothing more to do */
                rw_exit(&lps.lps_lh_rwlock);
                kmem_free(new_lh_array,
                    sizeof (lx_ptm_handle_t) * new_lh_count);
                return;
        }

        /* copy the existing data into the new array */
        ASSERT((lps.lps_lh_count != 0) || (lps.lps_lh_array == NULL));
        ASSERT((lps.lps_lh_count == 0) || (lps.lps_lh_array != NULL));
        if (lps.lps_lh_count != 0) {
                bcopy(lps.lps_lh_array, new_lh_array,
                    sizeof (lx_ptm_handle_t) * lps.lps_lh_count);
        }

        /* save info on the old array */
        old_lh_array = lps.lps_lh_array;
        old_lh_count = lps.lps_lh_count;

        /* install the new array */
        lps.lps_lh_array = new_lh_array;
        lps.lps_lh_count = new_lh_count;

        rw_exit(&lps.lps_lh_rwlock);

        /* free the old array */
        if (old_lh_array != NULL) {
                kmem_free(old_lh_array,
                    sizeof (lx_ptm_handle_t) * old_lh_count);
        }
}

static void
lx_ptm_lh_insert(uint_t index, ldi_handle_t lh)
{
        lx_ptm_ops_t *lpo;

        ASSERT(lh != NULL);

        /* Allocate and initialize the ops structure */
        lpo = kmem_zalloc(sizeof (lx_ptm_ops_t), KM_SLEEP);
        mutex_init(&lpo->lpo_rops_lock, NULL, MUTEX_DEFAULT, NULL);
        cv_init(&lpo->lpo_rops_cv, NULL, CV_DEFAULT, NULL);

        rw_enter(&lps.lps_lh_rwlock, RW_WRITER);

        /* check if we need to grow the size of the layered handle array */
        if (index >= lps.lps_lh_count) {
                rw_exit(&lps.lps_lh_rwlock);
                lx_ptm_lh_grow(index);
                rw_enter(&lps.lps_lh_rwlock, RW_WRITER);
        }

        ASSERT(index < lps.lps_lh_count);
```

```
260         ASSERT(lps.lps_lh_array[index].lph_handle == NULL);
261         ASSERT(lps.lps_lh_array[index].lph_pktio == 0);
262         ASSERT(lps.lps_lh_array[index].lph_eofed == 0);
263         ASSERT(lps.lps_lh_array[index].lph_lpo == NULL);

265         /* insert the new handle and return */
266         lps.lps_lh_array[index].lph_handle = lh;
267         lps.lps_lh_array[index].lph_pktio = 0;
268         lps.lps_lh_array[index].lph_eofed = 0;
269         lps.lps_lh_array[index].lph_lpo = lpo;

271         rw_exit(&lps.lps_lh_rwlock);
272 }

274 static ldi_handle_t
275 lx_ptm_lh_remove(uint_t index)
276 {
277         ldi_handle_t    lh;

279         rw_enter(&lps.lps_lh_rwlock, RW_WRITER);

281         ASSERT(index < lps.lps_lh_count);
282         ASSERT(lps.lps_lh_array[index].lph_handle != NULL);
283         ASSERT(lps.lps_lh_array[index].lph_lpo->lpo_rops == 0);
284         ASSERT(!MUTEX_HELD(&lps.lps_lh_array[index].lph_lpo->lpo_rops_lock));

286         /* free the write handle */
287         kmem_free(lps.lps_lh_array[index].lph_lpo, sizeof (lx_ptm_ops_t));
288         lps.lps_lh_array[index].lph_lpo = NULL;

290         /* remove the handle and return it */
291         lh = lps.lps_lh_array[index].lph_handle;
292         lps.lps_lh_array[index].lph_handle = NULL;
293         lps.lps_lh_array[index].lph_pktio = 0;
294         lps.lps_lh_array[index].lph_eofed = 0;
295         rw_exit(&lps.lps_lh_rwlock);
296         return (lh);
297 }

299 static void
300 lx_ptm_lh_get_ppocb(uint_t index, ptmptsopencb_t *ppocb)
301 {
302         rw_enter(&lps.lps_lh_rwlock, RW_WRITER);

304         ASSERT(index < lps.lps_lh_count);
305         ASSERT(lps.lps_lh_array[index].lph_handle != NULL);

307         *ppocb = lps.lps_lh_array[index].lph_ppocb;
308         rw_exit(&lps.lps_lh_rwlock);
309 }

311 static void
312 lx_ptm_lh_set_ppocb(uint_t index, ptmptsopencb_t *ppocb)
313 {
314         rw_enter(&lps.lps_lh_rwlock, RW_WRITER);

316         ASSERT(index < lps.lps_lh_count);
317         ASSERT(lps.lps_lh_array[index].lph_handle != NULL);

319         lps.lps_lh_array[index].lph_ppocb = *ppocb;
320         rw_exit(&lps.lps_lh_rwlock);
321 }

323 static ldi_handle_t
324 lx_ptm_lh_lookup(uint_t index)
325 {
```



```
326         ldi_handle_t    lh;

328         rw_enter(&lps.lps_lh_rwlock, RW_READER);

330         ASSERT(index < lps.lps_lh_count);
331         ASSERT(lps.lps_lh_array[index].lph_handle != NULL);

333         /* return the handle */
334         lh = lps.lps_lh_array[index].lph_handle;
335         rw_exit(&lps.lps_lh_rwlock);
336         return (lh);
337 }

339 static lx_ptm_ops_t *
340 lx_ptm_lpo_lookup(uint_t index)
341 {
342         lx_ptm_ops_t    *lpo;

344         rw_enter(&lps.lps_lh_rwlock, RW_READER);

346         ASSERT(index < lps.lps_lh_count);
347         ASSERT(lps.lps_lh_array[index].lph_lpo != NULL);

349         /* return the handle */
350         lpo = lps.lps_lh_array[index].lph_lpo;
351         rw_exit(&lps.lps_lh_rwlock);
352         return (lpo);
353 }

355 static int
356 lx_ptm_lh_pktio_get(uint_t index)
357 {
358         int             pktio;

360         rw_enter(&lps.lps_lh_rwlock, RW_READER);

362         ASSERT(index < lps.lps_lh_count);
363         ASSERT(lps.lps_lh_array[index].lph_handle != NULL);

365         /* return the pktio state */
366         pktio = lps.lps_lh_array[index].lph_pktio;
367         rw_exit(&lps.lps_lh_rwlock);
368         return (pktio);
369 }

371 static void
372 lx_ptm_lh_pktio_set(uint_t index, int pktio)
373 {
374         rw_enter(&lps.lps_lh_rwlock, RW_WRITER);

376         ASSERT(index < lps.lps_lh_count);
377         ASSERT(lps.lps_lh_array[index].lph_handle != NULL);

379         /* set the pktio state */
380         lps.lps_lh_array[index].lph_pktio = pktio;
381         rw_exit(&lps.lps_lh_rwlock);
382 }

384 static int
385 lx_ptm_lh_eofed_get(uint_t index)
386 {
387         int             eofed;

389         rw_enter(&lps.lps_lh_rwlock, RW_READER);

391         ASSERT(index < lps.lps_lh_count);
```

```
392         ASSERT(lps.lps_lh_array[index].lph_handle != NULL);

394         /* return the eofed state */
395         eofed = lps.lps_lh_array[index].lph_eofed;
396         rw_exit(&lps.lps_lh_rwlock);
397         return (eofed);
398 }

400 static void
401 lx_ptm_lh_eofed_set(uint_t index)
402 {
403         rw_enter(&lps.lps_lh_rwlock, RW_WRITER);

405         ASSERT(index < lps.lps_lh_count);
406         ASSERT(lps.lps_lh_array[index].lph_handle != NULL);

408         /* set the eofed state */
409         lps.lps_lh_array[index].lph_eofed++;
410         rw_exit(&lps.lps_lh_rwlock);
411 }

413 static int
414 lx_ptm_read_start(dev_t dev)
415 {
416         lx_ptm_ops_t    *lpo = lx_ptm_lpo_lookup(DEVT_TO_INDEX(dev));

418         mutex_enter(&lpo->lpo_rops_lock);
419         ASSERT(lpo->lpo_rops >= 0);

421         /* Wait for other read operations to finish */
422         while (lpo->lpo_rops != 0) {
423                 if (cv_wait_sig(&lpo->lpo_rops_cv, &lpo->lpo_rops_lock) == 0) {
424                         mutex_exit(&lpo->lpo_rops_lock);
425                         return (-1);
426                 }
427         }

429         /* Start a read operation */
430         VERIFY(++lpo->lpo_rops == 1);
431         mutex_exit(&lpo->lpo_rops_lock);
432         return (0);
433 }

435 static void
436 lx_ptm_read_end(dev_t dev)
437 {
438         lx_ptm_ops_t    *lpo = lx_ptm_lpo_lookup(DEVT_TO_INDEX(dev));

440         mutex_enter(&lpo->lpo_rops_lock);
441         ASSERT(lpo->lpo_rops >= 0);

443         /* End a read operation */
444         VERIFY(--lpo->lpo_rops == 0);
445         cv_signal(&lpo->lpo_rops_cv);

447         mutex_exit(&lpo->lpo_rops_lock);
448 }

450 static int
451 lx_ptm_pts_isopen(dev_t dev)
452 {
453         ptmptsopencb_t  ppocb;

455         lx_ptm_lh_get_ppocb(DEVT_TO_INDEX(dev), &ppocb);
456         return (ppocb.ppocb_func(ppocb.ppocb_arg));
457 }
```



```
459 static void
460 lx_ptm_eof_read(ldi_handle_t lh)
461 {
462         struct uio      uio;
463         iovec_t         iov;
464         char            junk[1];

466         /*
467          * We can remove any EOF message from the head of the stream by
468          * doing a zero byte read from the stream.
469          */
470         iov.iov_len = 0;
471         iov.iov_base = junk;
472         uio.uio_iovcnt = 1;
473         uio.uio_iov = &iov;
474         uio.uio_resid = iov.iov_len;
475         uio.uio_offset = 0;
476         uio.uio_segflg = UIO_SYSSPACE;
477         uio.uio_fmode = 0;
478         uio.uio_extflg = 0;
479         uio.uio_llimit = MAXOFFSET_T;
480         (void) ldi_read(lh, &uio, kcred);
481 }

483 static int
484 lx_ptm_eof_drop_1(dev_t dev, int *rvalp)
485 {
486         ldi_handle_t    lh = lx_ptm_lh_lookup(DEVT_TO_INDEX(dev));
487         int             err, msg_size, msg_count;

489         *rvalp = 0;

491         /*
492          * Check if there is an EOF message (represented by a zero length
493          * data message) at the head of the stream.  Note that the
494          * I_NREAD ioctl is a streams framework ioctl so it will succeed
495          * even if there have been previous write errors on this stream.
496          */
497         if ((err = ldi_ioctl(lh, I_NREAD, (intptr_t)&msg_size,
498             FKIOCTL, kcred, &msg_count)) != 0)
499                 return (err);

501         if ((msg_count == 0) || (msg_size != 0)) {
502                 /* No EOF message found */
503                 return (0);
504         }

506         /* Record the fact that the slave device has been closed. */
507         lx_ptm_lh_eofed_set(DEVT_TO_INDEX(dev));

509         /* drop the EOF */
510         lx_ptm_eof_read(lh);
511         *rvalp = 1;
512         return (0);
513 }

515 static int
516 lx_ptm_eof_drop(dev_t dev, int *rvalp)
517 {
518         int rval, err;

520         if (rvalp != NULL)
521                 *rvalp = 0;
522         for (;;) {
523                 if ((err = lx_ptm_eof_drop_1(dev, &rval)) != 0)
```

```
524                         return (err);
525                 if (rval == 0)
526                         return (0);
527                 if (rvalp != NULL)
528                         *rvalp = 1;
529         }
530 }

532 static int
533 lx_ptm_data_check(dev_t dev, int ignore_eof, int *rvalp)
534 {
535         ldi_handle_t    lh = lx_ptm_lh_lookup(DEVT_TO_INDEX(dev));
536         int             err;

538         *rvalp = 0;
539         if (ignore_eof) {
540                 int     size, rval;

542                 if ((err = ldi_ioctl(lh, FIONREAD, (intptr_t)&size,
543                     FKIOCTL, kcred, &rval)) != 0)
544                         return (err);
545                 if (size != 0)
546                         *rvalp = 1;
547         } else {
548                 int     msg_size, msg_count;

550                 if ((err = ldi_ioctl(lh, I_NREAD, (intptr_t)&msg_size,
551                     FKIOCTL, kcred, &msg_count)) != 0)
552                         return (err);
553                 if (msg_count != 0)
554                         *rvalp = 1;
555         }
556         return (0);
557 }

559 static int
560 lx_ptm_attach(dev_info_t *dip, ddi_attach_cmd_t cmd)
561 {
562         int err;

564         if (cmd != DDI_ATTACH)
565                 return (DDI_FAILURE);

567         if (ddi_create_minor_node(dip, LX_PTM_MINOR_NODE, S_IFCHR,
568             ddi_get_instance(dip), DDI_PSEUDO, 0) != DDI_SUCCESS)
569                 return (DDI_FAILURE);

571         err = ldi_ident_from_dip(dip, &lps.lps_li);
572         if (err != 0) {
573                 ddi_remove_minor_node(dip, ddi_get_name(dip));
574                 return (DDI_FAILURE);
575         }

577         lps.lps_dip = dip;
578         lps.lps_pts_major = ddi_name_to_major(LP_PTS_DRV_NAME);

580         rw_init(&lps.lps_lh_rwlock, NULL, RW_DRIVER, NULL);
581         lps.lps_lh_count = 0;
582         lps.lps_lh_array = NULL;

584         return (DDI_SUCCESS);
585 }

587 /*ARGSUSED*/
588 static int
589 lx_ptm_detach(dev_info_t *dip, ddi_detach_cmd_t cmd)
```



```
590 {
591         if (cmd != DDI_DETACH)
592                 return (DDI_FAILURE);

594         ldi_ident_release(lps.lps_li);
595         lps.lps_dip = NULL;

597         ASSERT((lps.lps_lh_count != 0) || (lps.lps_lh_array == NULL));
598         ASSERT((lps.lps_lh_count == 0) || (lps.lps_lh_array != NULL));
599         if (lps.lps_lh_array != NULL) {
600                 kmem_free(lps.lps_lh_array,
601                     sizeof (lx_ptm_handle_t) * lps.lps_lh_count);
602                 lps.lps_lh_array = NULL;
603                 lps.lps_lh_count = 0;
604         }

606         return (DDI_SUCCESS);
607 }

609 /*ARGSUSED*/
610 static int
611 lx_ptm_open(dev_t *devp, int flag, int otyp, cred_t *credp)
612 {
613         struct strioctl iocb;
614         ptmptsopencb_t  ppocb = { NULL, NULL };
615         ldi_handle_t    lh;
616         major_t         maj, our_major = getmajor(*devp);
617         minor_t         min, lastmin;
618         uint_t          index, anchor = 1;
619         dev_t           ptm_dev;
620         int             err, rval = 0;

622         /*
623          * Don't support the FNDELAY flag and FNONBLOCK until we either
624          * find a Linux app that opens /dev/ptmx with the O_NDELAY
625          * or O_NONBLOCK flags explicitly, or until we create test cases
626          * to determine how reads of master terminal devices opened with
627          * these flags behave in different situations on Linux.  Supporting
628          * these flags will involve enhancing our read implementation
629          * and changing the way it deals with EOF notifications.
630          */
631         if (flag & (FNDELAY | FNONBLOCK))
632                 return (ENOTSUP);

634         /*
635          * we're layered on top of the ptm driver so open that driver
636          * first.  (note that we're opening /dev/ptmx in the global
637          * zone, not ourselves in the Linux zone.)
638          */
639         err = ldi_open_by_name(LP_PTM_PATH, flag, credp, &lh, lps.lps_li);
640         if (err != 0)
641                 return (err);

643         /* get the devt returned by the ptmx open */
644         err = ldi_get_dev(lh, &ptm_dev);
645         if (err != 0) {
646                 (void) ldi_close(lh, flag, credp);
647                 return (err);
648         }

650         /*
651          * we're a cloning driver so here's well change the devt that we
652          * return.  the ptmx is also a cloning driver so we'll just use
653          * it's minor number as our minor number (it already manages it's
654          * minor name space so no reason to duplicate the effort.)
655          */
```

```
656         index = getminor(ptm_dev);
657         *devp = makedevice(our_major, INDEX_TO_MINOR(index));

659         /* Get a callback function to query if the pts device is open. */
660         iocb.ic_cmd = PTMPTSOPENCB;
661         iocb.ic_timout = 0;
662         iocb.ic_len = sizeof (ppocb);
663         iocb.ic_dp = (char *)&ppocb;

665         err = ldi_ioctl(lh, I_STR, (intptr_t)&iocb, FKIOCTL, kcred, &rval);
666         if ((err != 0) || (rval != 0)) {
667                 (void) ldi_close(lh, flag, credp);
668                 return (EIO); /* XXX return something else here? */
669         }
670         ASSERT(ppocb.ppocb_func != NULL);

672         /*
673          * now setup autopush for the terminal slave device.  this is
674          * necessary so that when a Linux program opens the device we
675          * can push required strmod modules onto the stream.  in Solaris
676          * this is normally done by the application that actually
677          * allocates the terminal.
678          */
679         maj = lps.lps_pts_major;
680         min = index;
681         lastmin = 0;
682         err = kstr_autopush(SET_AUTOPUSH, &maj, &min, &lastmin,
683             &anchor, lx_pts_mods);
684         if (err != 0) {
685                 (void) ldi_close(lh, flag, credp);
686                 return (EIO); /* XXX return something else here? */
687         }

689         /* save off this layered handle for future accesses */
690         lx_ptm_lh_insert(index, lh);
691         lx_ptm_lh_set_ppocb(index, &ppocb);
692         return (0);
693 }

695 /*ARGSUSED*/
696 static int
697 lx_ptm_close(dev_t dev, int flag, int otyp, cred_t *credp)
698 {
699         ldi_handle_t    lh;
700         major_t         maj;
701         minor_t         min, lastmin;
702         uint_t          index;
703         int             err;

705         index = DEVT_TO_INDEX(dev);

707         /*
708          * we must cleanup all the state associated with this major/minor
709          * terminal pair before actually closing the ptm master device.
710          * this is required because once the close of the ptm device is
711          * complete major/minor terminal pair is immediatly available for
712          * re-use in any zone.
713          */

715         /* free up our saved reference for this layered handle */
716         lh = lx_ptm_lh_remove(index);

718         /* unconfigure autopush for the associated terminal slave device */
719         maj = lps.lps_pts_major;
720         min = index;
721         lastmin = 0;
```



```
722         do {
723                 /*
724                  * we loop here because we don't want to release this ptm
725                  * node if autopush can't be disabled on the associated
726                  * slave device because then bad things could happen if
727                  * another brand were to get this terminal allocated
728                  * to them.
729                  *
730                  * XXX should we ever give up?
731                  */
732                 err = kstr_autopush(CLR_AUTOPUSH, &maj, &min, &lastmin,
733                     0, NULL);
734         } while (err != 0);

736         err = ldi_close(lh, flag, credp);

738         /*
739          * note that we don't have to bother with changing the permissions
740          * on the associated slave device here.  the reason is that no one
741          * can actually open the device untill it's associated master
742          * device is re-opened, which will result in the permissions on
743          * it being reset.
744          */
745         return (err);
746 }

748 static int
749 lx_ptm_read_loop(dev_t dev, struct uio *uiop, cred_t *credp, int *loop)
750 {
751         ldi_handle_t    lh = lx_ptm_lh_lookup(DEVT_TO_INDEX(dev));
752         int             err, rval;
753         struct uio      uio = *uiop;

755         *loop = 0;

757         /*
758          * Here's another way that Linux master terminals behave differently
759          * from Solaris master terminals.  If you do a read on a Linux
760          * master terminal (that was opened witout NDELAY and NONBLOCK)
761          * who's corrosponding slave terminal is currently closed and
762          * has been opened and closed at least once, Linux return -1 and
763          * set errno to EIO where as Solaris blocks.
764          */
765         if (lx_ptm_lh_eofed_get(DEVT_TO_INDEX(dev))) {
766                 /* Slave has been opened and closed at least once. */
767                 if (lx_ptm_pts_isopen(dev) == 0) {
768                         /*
769                          * Slave is closed.  Make sure that data is avaliable
770                          * before attempting a read.
771                          */
772                         if ((err = lx_ptm_data_check(dev, 0, &rval)) != 0)
773                                 return (err);

775                         /* If there is no data available then return. */
776                         if (rval == 0)
777                                 return (EIO);
778                 }
779         }

781         /* Actually do the read operation. */
782         if ((err = ldi_read(lh, uiop, credp)) != 0)
783                 return (err);

785         /* If read returned actual data then return. */
786         if (uio.uio_resid != uiop->uio_resid)
787                 return (0);
```

```
789         /*
790          * This was a zero byte read (ie, an EOF).  This indicates
791          * that the slave terinal device has been closed.  Record
792          * the fact that the slave device has been closed and retry
793          * the read operation.
794          */
795         lx_ptm_lh_eofed_set(DEVT_TO_INDEX(dev));
796         *loop = 1;
797         return (0);
798 }

800 static int
801 lx_ptm_read(dev_t dev, struct uio *uiop, cred_t *credp)
802 {
803         int             pktio = lx_ptm_lh_pktio_get(DEVT_TO_INDEX(dev));
804         int             err, loop;
805         struct uio      uio;
806         struct iovec    iovp;

808         ASSERT(uiop->uio_iovcnt > 0);

810         /*
811          * If packet mode has been enabled (via TIOCPKT) we need to pad
812          * all read requests with a leading byte that indicates any
813          * relevant control status information.
814          */
815         if (pktio != 0) {
816                 /*
817                  * We'd like to write the control information into
818                  * the current buffer but we can't yet.  We don't
819                  * want to modify userspace memory here only to have
820                  * the read operation fail later.  So instead
821                  * what we'll do here is read one character from the
822                  * beginning of the memory pointed to by the uio
823                  * structure.  This will advance the output pointer
824                  * by one.  Then when the read completes successfully
825                  * we can update the byte that we passed over.  Before
826                  * we do the read make a copy of the current uiop and
827                  * iovec structs so we can write to them later.
828                  */
829                 uio = *uiop;
830                 iovp = *uiop->uio_iov;
831                 uio.uio_iov = &iovp;

833                 if (uwritec(uiop) == -1)
834                         return (EFAULT);
835         }

837         do {
838                 /*
839                  * Before we actually attempt a read operation we need
840                  * to make sure there's some buffer space to actually
841                  * read in some data.  We do this because if we're in
842                  * pktio mode and the caller only requested one byte,
843                  * then we've already used up that one byte and we
844                  * don't want to pass this read request.  Doing a 0
845                  * byte read (unless there is a problem with the stream
846                  * head) always returns succcess.  Normally when a streams
847                  * read returns 0 bytes we interpret that as an EOF on
848                  * the stream (ie, the slave side has been opened and
849                  * closed) and we ignore it and re-try the read operation.
850                  * So if we pass on a 0 byte read here lx_ptm_read_loop()
851                  * will tell us to loop around and we'll end up in an
852                  * infinite loop.
853                  */
```



```
854                 if (uiop->uio_resid == 0)
855                         break;

857                 /*
858                  * Serialize all reads.  We need to do this so that we can
859                  * properly emulate the behavior of master terminals on Linux.
860                  * In reality this serializaion should not pose any kind of
861                  * performance problem since it would be very strange to have
862                  * multiple threads trying to read from the same master
863                  * terminal device concurrently.
864                  */
865                 if (lx_ptm_read_start(dev) != 0)
866                         return (EINTR);

868                 err = lx_ptm_read_loop(dev, uiop, credp, &loop);
869                 lx_ptm_read_end(dev);
870                 if (err != 0)
871                         return (err);
872         } while (loop != 0);

874         if (pktio != 0) {
875                 uint8_t         pktio_data = TIOCPKT_DATA;

877                 /*
878                  * Note that the control status information we
879                  * pass back is faked up in the sense that we
880                  * don't actually report any events, we always
881                  * report a status of 0.
882                  */
883                 if (uiomove(&pktio_data, 1, UIO_READ, &uio) != 0)
884                         return (EFAULT);
885         }

887         return (0);
888 }

890 static int
891 lx_ptm_write(dev_t dev, struct uio *uiop, cred_t *credp)
892 {
893         ldi_handle_t            lh = lx_ptm_lh_lookup(DEVT_TO_INDEX(dev));
894         int             err;

896         err = ldi_write(lh, uiop, credp);

898         return (err);
899 }

901 static int
902 lx_ptm_ioctl(dev_t dev, int cmd, intptr_t arg, int mode, cred_t *credp,
903     int *rvalp)
904 {
905         ldi_handle_t    lh = lx_ptm_lh_lookup(DEVT_TO_INDEX(dev));
906         int             err;

908         /*
909          * here we need to make sure that we never allow the
910          * I_SETSIG and I_ESETSIG ioctls to pass through.  we
911          * do this because we can't support them.
912          *
913          * the native Solaris ptm device supports these ioctls because
914          * they are streams framework ioctls and all streams devices
915          * support them by default.  these ioctls cause the current
916          * process to be registered with a stream and receive signals
917          * when certain stream events occur.
918          *
919          * a problem arises with cleanup of these registrations
```

```
920          * for layered drivers.
921          *
922          * normally the streams framework is notified whenever a
923          * process closes any reference to a stream and it goes ahead
924          * and cleans up these registrations.  but actual device drivers
925          * are not notified when a process performs a close operation
926          * unless the process is closing the last opened reference to
927          * the device on the entire system.
928          *
929          * so while we could pass these ioctls on and allow processes
930          * to register for signal delivery, we would never receive
931          * any notification when those processes exit (or close a
932          * stream) and we wouldn't be able to unregister them.
933          *
934          * luckily these operations are streams specific and Linux
935          * doesn't support streams devices.  so it doesn't actually
936          * seem like we need to support these ioctls.  if it turns
937          * out that we do need to support them for some reason in
938          * the future, the current driver model will have to be
939          * enhanced to better support streams device layering.
940          */
941         if ((cmd == I_SETSIG) || (cmd == I_ESETSIG))
942                 return (EINVAL);

944         /*
945          * here we fake up support for TIOCPKT.  Linux applications expect
946          * /etc/ptmx to support this ioctl, but on Solaris it doesn't.
947          * (it is supported on older bsd style ptys.)  so we'll fake
948          * up support for it here.
949          *
950          * the reason that this ioctl is emulated here instead of in
951          * userland is that this ioctl affects the results returned
952          * from read() operations.  if this ioctl was emulated in
953          * userland the brand library would need to intercept all
954          * read operations and check to see if pktio was enabled
955          * for the fd being read from.  since this ioctl only needs
956          * to be supported on the ptmx device it makes more sense
957          * to support it here where we can easily update the results
958          * returned for read() operations performed on ourselves.
959          */
960         if (cmd == TIOCPKT) {
961                 int     pktio;

963                 if (ddi_copyin((void *)arg, &pktio, sizeof (pktio),
964                     mode) != DDI_SUCCESS)
965                         return (EFAULT);

967                 if (pktio == 0)
968                         lx_ptm_lh_pktio_set(DEVT_TO_INDEX(dev), 0);
969                 else
970                         lx_ptm_lh_pktio_set(DEVT_TO_INDEX(dev), 1);

972                 return (0);
973         }

975         err = ldi_ioctl(lh, cmd, arg, mode, credp, rvalp);

977         return (err);
978 }

980 static int
981 lx_ptm_poll_loop(dev_t dev, short events, int anyyet, short *reventsp,
982     struct pollhead **phpp, int *loop)
983 {
984         ldi_handle_t    lh = lx_ptm_lh_lookup(DEVT_TO_INDEX(dev));
985         short           reventsp2;
```



```
986         int             err, rval;

988         *loop = 0;

990         /*
991          * If the slave device has been opened and closed at least
992          * once and the slave device is currently closed, then poll
993          * always needs to returns immediatly.
994          */
995         if ((lx_ptm_lh_eofed_get(DEVT_TO_INDEX(dev)) != 0) &&
996             (lx_ptm_pts_isopen(dev) == 0)) {
997                 /* In this case always return POLLHUP */
998                 *reventsp = POLLHUP;

1000                /*
1001                 * Check if there really is data on the stream.
1002                 * If so set the correct return flags.
1003                 */
1004                if ((err = lx_ptm_data_check(dev, 1, &rval)) != 0) {
1005                        /* Something went wrong. */
1006                        return (err);
1007                }
1008                if (rval != 0)
1009                        *reventsp |= (events & (POLLIN | POLLRDNORM));

1011                /*
1012                 * Is the user checking for writability?  Note that for ptm
1013                 * devices Linux seems to ignore the POLLWRBAND write flag.
1014                 */
1015                if ((events & POLLWRNORM) == 0)
1016                        return (0);

1018                /*
1019                 * To check if the stream is writable we have to actually
1020                 * call poll, but make sure to set anyyet to 1 to prevent
1021                 * the streams framework from setting up callbacks.
1022                 */
1023                if ((err = ldi_poll(lh, POLLWRNORM, 1, &reventsp2, NULL)) != 0)
1024                        return (err);

1026                *reventsp |= (reventsp2 & POLLWRNORM);
1027        } else {
1028                int lockstate;

1030                /* The slave device is open, do the poll */
1031                if ((err = ldi_poll(lh, events, anyyet, reventsp, phpp)) != 0)
1032                        return (err);

1034                /*
1035                 * Drop any leading EOFs on the stream.
1036                 *
1037                 * Note that we have to use pollunlock() here to avoid
1038                 * recursive mutex enters in the poll framework.  The
1039                 * reason is that if there is an EOF message on the stream
1040                 * then the act of reading from the queue to remove the
1041                 * message can cause the ptm drivers event service
1042                 * routine to be invoked, and if there is no open
1043                 * slave device then the ptm driver may generate
1044                 * error messages and put them on the stream.  This
1045                 * in turn will generate a poll event and the poll
1046                 * framework will try to invoke any poll callbacks
1047                 * associated with the stream.  In the process of
1048                 * doing that the poll framework will try to aquire
1049                 * locks that we are already holding.  So we need to
1050                 * drop those locks here before we do our read.
1051                 */
```

```
1052                 lockstate = pollunlock();
1053                 err = lx_ptm_eof_drop(dev, &rval);
1054                 pollrelock(lockstate);
1055                 if (err)
1056                         return (err);

1058                 /* If no EOF was dropped then return */
1059                 if (rval == 0)
1060                         return (0);

1062                 /*
1063                  * An EOF was removed from the stream.  Retry the entire
1064                  * poll operation from the top because polls on the ptm
1065                  * device should behave differently now.
1066                  */
1067                 *loop = 1;
1068         }
1069         return (0);
1070 }

1072 static int
1073 lx_ptm_poll(dev_t dev, short events, int anyyet, short *reventsp,
1074     struct pollhead **phpp)
1075 {
1076         int loop, err;

1078         do {
1079                 /* Serialize ourself wrt read operations. */
1080                 if (lx_ptm_read_start(dev) != 0)
1081                         return (EINTR);

1083                 err = lx_ptm_poll_loop(dev,
1084                     events, anyyet, reventsp, phpp, &loop);
1085                 lx_ptm_read_end(dev);
1086                 if (err != 0)
1087                         return (err);
1088         } while (loop != 0);
1089         return (0);
1090 }

1092 static struct cb_ops lx_ptm_cb_ops = {
1093         lx_ptm_open,            /* open */
1094         lx_ptm_close,           /* close */
1095         nodev,                  /* strategy */
1096         nodev,                  /* print */
1097         nodev,                  /* dump */
1098         lx_ptm_read,            /* read */
1099         lx_ptm_write,           /* write */
1100         lx_ptm_ioctl,           /* ioctl */
1101         nodev,                  /* devmap */
1102         nodev,                  /* mmap */
1103         nodev,                  /* segmap */
1104         lx_ptm_poll,            /* chpoll */
1105         ddi_prop_op,            /* prop_op */
1106         NULL,                   /* cb_str */
1107         D_NEW | D_MP,
1108         CB_REV,
1109         NULL,
1110         NULL
1111 };

1113 static struct dev_ops lx_ptm_ops = {
1114         DEVO_REV,
1115         0,
1116         ddi_getinfo_1to1,
1117         nulldev,
```



```
1118         nulldev,
1119         lx_ptm_attach,
1120         lx_ptm_detach,
1121         nodev,
1122         &lx_ptm_cb_ops,
1123         NULL,
1124         NULL,
1125         ddi_quiesce_not_needed,         /* quiesce */
1126 };

1128 static struct modldrv modldrv = {
1129         &mod_driverops,                 /* type of module */
1130         "Linux master terminal driver", /* description of module */
1131         &lx_ptm_ops                     /* driver ops */
1132 };

1134 static struct modlinkage modlinkage = {
1135         MODREV_1,
1136         &modldrv,
1137         NULL
1138 };

1140 int
1141 _init(void)
1142 {
1143         return (mod_install(&modlinkage));
1144 }

1146 int
1147 _info(struct modinfo *modinfop)
1148 {
1149         return (mod_info(&modlinkage, modinfop));
1150 }

1152 int
1153 _fini(void)
1154 {
1155         return (mod_remove(&modlinkage));
1156 }
1157 #endif /* ! codereview */
```

```
**********************************************************
     971 Tue Jan 14 16:17:18 2014
new/usr/src/uts/common/brand/lx/io/lx_ptm.conf
Bring back LX zones.
**********************************************************
```

```
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #
  21 #
  22 # Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  23 # Use is subject to license terms.
  24 #
  25 #ident  "%Z%%M% %I%     %E% SMI"

  27 name="lx_ptm" parent="pseudo" instance=0;
  28 #endif /* ! codereview */
```

```
**********************************************************
   24113 Tue Jan 14 16:17:19 2014
new/usr/src/uts/common/brand/lx/os/lx_brand.c
LX zone support should now build and packages of relevance produced.
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/types.h>
#include <sys/kmem.h>
#include <sys/errno.h>
#include <sys/thread.h>
#include <sys/systm.h>
#include <sys/syscall.h>
#include <sys/proc.h>
#include <sys/modctl.h>
#include <sys/cmn_err.h>
#include <sys/model.h>
#include <sys/exec.h>
#include <sys/lx_impl.h>
#include <sys/machbrand.h>
#include <sys/lx_syscalls.h>
#include <sys/lx_pid.h>
#include <sys/lx_futex.h>
#include <sys/lx_brand.h>
#include <sys/termios.h>
#include <sys/sunddi.h>
#include <sys/ddi.h>
#include <sys/vnode.h>
#include <sys/pathname.h>
#include <sys/auxv.h>
#include <sys/priv.h>
#include <sys/regset.h>
#include <sys/privregs.h>
#include <sys/archsystm.h>
#include <sys/zone.h>
#include <sys/brand.h>

int     lx_debug = 0;

void    lx_init_brand_data(zone_t *);
void    lx_free_brand_data(zone_t *);
```



```
void    lx_setbrand(proc_t *);
int     lx_getattr(zone_t *, int, void *, size_t *);
int     lx_setattr(zone_t *, int, void *, size_t);
int     lx_brandsys(int, int64_t *, uintptr_t, uintptr_t, uintptr_t,
                uintptr_t, uintptr_t, uintptr_t);
int     lx_get_kern_version(void);
void    lx_set_kern_version(zone_t *, int);
void    lx_copy_procdata(proc_t *, proc_t *);

extern void lx_setrval(klwp_t *, int, int);
extern void lx_proc_exit(proc_t *, klwp_t *);
extern void lx_exec();
extern int lx_initlwp(klwp_t *);
extern void lx_forklwp(klwp_t *, klwp_t *);
extern void lx_exitlwp(klwp_t *);
extern void lx_freelwp(klwp_t *);
extern greg_t lx_fixsegreg(greg_t, model_t);
extern int lx_sched_affinity(int, uintptr_t, int, uintptr_t, int64_t *);

int lx_systrace_brand_enabled;

lx_systrace_f *lx_systrace_entry_ptr;
lx_systrace_f *lx_systrace_return_ptr;

static int lx_systrace_enabled;

static int lx_elfexec(struct vnode *vp, struct execa *uap, struct uarg *args,
    struct intpdata *idata, int level, long *execsz, int setid,
    caddr_t exec_file, struct cred *cred, int brand_action);

/* lx brand */
struct brand_ops lx_brops = {
        lx_init_brand_data,
        lx_free_brand_data,
        lx_brandsys,
        lx_setbrand,
        lx_getattr,
        lx_setattr,
        lx_copy_procdata,
        lx_proc_exit,
        lx_exec,
        lx_setrval,
        lx_initlwp,
        lx_forklwp,
        lx_freelwp,
        lx_exitlwp,
        lx_elfexec,
        NULL,
        NULL,
        NSIG,
};

struct brand_mach_ops lx_mops = {
        NULL,
        lx_brand_int80_callback,
        NULL,
        NULL,
        NULL,
        lx_fixsegreg,
};

struct brand lx_brand = {
        BRAND_VER_1,
        "lx",
        &lx_brops,
        &lx_mops
```

```
};

static struct modlbrand modlbrand = {
        &mod_brandops, "lx brand", &lx_brand
};

static struct modlinkage modlinkage = {
        MODREV_1, (void *)&modlbrand, NULL
};

void
lx_proc_exit(proc_t *p, klwp_t *lwp)
{
        zone_t *z = p->p_zone;

        ASSERT(p->p_brand != NULL);
        ASSERT(p->p_brand_data != NULL);

        /*
         * If init is dying and we aren't explicitly shutting down the zone
         * or the system, then Solaris is about to restart init.  The Linux
         * init is not designed to handle a restart, which it interprets as
         * a reboot.  To give it a sane environment in which to run, we
         * reboot the zone.
         */
        if (p->p_pid == z->zone_proc_initpid) {
                if (z->zone_boot_err == 0 &&
                    z->zone_restart_init &&
                    zone_status_get(z) < ZONE_IS_SHUTTING_DOWN &&
                    zone_status_get(global_zone) < ZONE_IS_SHUTTING_DOWN)
                        (void) zone_kadmin(A_REBOOT, 0, NULL, CRED());
        }
        lx_exitlwp(lwp);
        kmem_free(p->p_brand_data, sizeof (struct lx_proc_data));
        p->p_brand_data = NULL;
}

void
lx_setbrand(proc_t *p)
{
        kthread_t *t = p->p_tlist;
        int err;

        ASSERT(p->p_brand_data == NULL);
        ASSERT(ttolxlwp(curthread) == NULL);

        p->p_brand_data = kmem_zalloc(sizeof (struct lx_proc_data), KM_SLEEP);

        /*
         * This routine can only be called for single-threaded processes.
         * Since lx_initlwp() can only fail if we run out of PIDs for
         * multithreaded processes, we know that this can never fail.
         */
        err = lx_initlwp(t->t_lwp);
        ASSERT(err == 0);
}

/* ARGSUSED */
int
lx_setattr(zone_t *zone, int attr, void *buf, size_t bufsize)
{
        boolean_t val;
        int num;

        if (attr == LX_ATTR_RESTART_INIT) {
                if (bufsize > sizeof (boolean_t))
```



```
                        return (ERANGE);
                if (copyin(buf, &val, sizeof (val)) != 0)
                        return (EFAULT);
                if (val != B_TRUE && val != B_FALSE)
                        return (EINVAL);
                zone->zone_restart_init = val;
                return (0);
        } else if (attr == LX_KERN_VERSION_NUM) {
                if (bufsize > sizeof (int))
                        return (ERANGE);
                if (copyin(buf, &num, sizeof (num)) != 0)
                        return (EFAULT);
                lx_set_kern_version(zone, num);
                return (0);
        }
        return (EINVAL);
}

/* ARGSUSED */
int
lx_getattr(zone_t *zone, int attr, void *buf, size_t *bufsize)
{
        int num;
        if (attr == LX_ATTR_RESTART_INIT) {
                if (*bufsize < sizeof (boolean_t))
                        return (ERANGE);
                if (copyout(&zone->zone_restart_init, buf,
                    sizeof (boolean_t)) != 0)
                        return (EFAULT);
                *bufsize = sizeof (boolean_t);
                return (0);
        } else if (attr == LX_KERN_VERSION_NUM) {
                if (*bufsize < sizeof (int))
                        return (ERANGE);
                num = lx_get_kern_version();
                if (copyout(&num, buf, sizeof (int)) != 0)
                        return (EFAULT);
                *bufsize = sizeof (int);
                return (0);
        }
        return (-EINVAL);
}

/*
 * Enable ptrace system call tracing for the given LWP. This is done by
 * both setting the flag in that LWP's brand data (in the kernel) and setting
 * the process-wide trace flag (in the brand library of the traced process).
 */
static int
lx_ptrace_syscall_set(pid_t pid, id_t lwpid, int set)
{
        proc_t *p;
        kthread_t *t;
        klwp_t *lwp;
        lx_proc_data_t *lpdp;
        lx_lwp_data_t *lldp;
        uintptr_t addr;
        int ret, flag = 1;

        if ((p = sprlock(pid)) == NULL)
                return (ESRCH);

        if (priv_proc_cred_perm(curproc->p_cred, p, NULL, VWRITE) != 0) {
                sprunlock(p);
                return (EPERM);
        }
```

```
260         if ((t = idtot(p, lwpid)) == NULL || (lwp = ttolwp(t)) == NULL) {
261                 sprunlock(p);
262                 return (ESRCH);
263         }

265         if ((lpdp = p->p_brand_data) == NULL ||
266             (lldp = lwp->lwp_brand) == NULL) {
267                 sprunlock(p);
268                 return (ESRCH);
269         }

271         if (set) {
272                 /*
273                  * Enable the ptrace flag for this LWP and this process. Note
274                  * that we will turn off the LWP's ptrace flag, but we don't
275                  * turn off the process's ptrace flag.
276                  */
277                 lldp->br_ptrace = 1;
278                 lpdp->l_ptrace = 1;

280                 addr = lpdp->l_traceflag;

282                 mutex_exit(&p->p_lock);

284                 /*
285                  * This can fail only in some rare corner cases where the
286                  * process is exiting or we're completely out of memory. In
287                  * these cases, it's sufficient to return an error to the ptrace
288                  * consumer and leave the process-wide flag set.
289                  */
290                 ret = uwrite(p, &flag, sizeof (flag), addr);

292                 mutex_enter(&p->p_lock);

294                 /*
295                  * If we couldn't set the trace flag, unset the LWP's ptrace
296                  * flag as there ptrace consumer won't expect this LWP to stop.
297                  */
298                 if (ret != 0)
299                         lldp->br_ptrace = 0;
300         } else {
301                 lldp->br_ptrace = 0;
302                 ret = 0;
303         }

305         sprunlock(p);

307         if (ret != 0)
308                 ret = EIO;

310         return (ret);
311 }

313 static void
314 lx_ptrace_fire(void)
315 {
316         kthread_t *t = curthread;
317         klwp_t *lwp = ttolwp(t);
318         lx_lwp_data_t *lldp = lwp->lwp_brand;

320         /*
321          * The ptrace flag only applies until the next event is encountered
322          * for the given LWP. If it's set, turn off the flag and poke the
323          * controlling process by raising a signal.
324          */
```



```
325         if (lldp->br_ptrace) {
326                 lldp->br_ptrace = 0;
327                 tsignal(t, SIGTRAP);
328         }
329 }

331 void
332 lx_brand_systrace_enable(void)
333 {
334         extern void lx_brand_int80_enable(void);

336         ASSERT(!lx_systrace_enabled);

338         lx_brand_int80_enable();

340         lx_systrace_enabled = 1;
341 }

343 void
344 lx_brand_systrace_disable(void)
345 {
346         extern void lx_brand_int80_disable(void);

348         ASSERT(lx_systrace_enabled);

350         lx_brand_int80_disable();

352         lx_systrace_enabled = 0;
353 }

355 void
356 lx_init_brand_data(zone_t *zone)
357 {
358         lx_zone_data_t *data;
359         ASSERT(zone->zone_brand == &lx_brand);
360         ASSERT(zone->zone_brand_data == NULL);
361         data = (lx_zone_data_t *)kmem_zalloc(sizeof (lx_zone_data_t), KM_SLEEP);
362         /*
363          * Set the default lxzd_kernel_version to LX_KERN_2_4.
364          * This can be changed by a call to setattr() during zone boot.
365          */
366         data->lxzd_kernel_version = LX_KERN_2_4;
367         data->lxzd_max_syscall = LX_NSYSCALLS_2_4;
368         zone->zone_brand_data = data;
369 }

371 void
372 lx_free_brand_data(zone_t *zone)
373 {
374         kmem_free(zone->zone_brand_data, sizeof (lx_zone_data_t));
375 }

377 /*
378  * Get the addresses of the user-space system call handler and attach it to
379  * the proc structure. Returning 0 indicates success; the value returned
380  * by the system call is the value stored in rval. Returning a non-zero
381  * value indicates a failure; the value returned is used to set errno, -1
382  * is returned from the syscall and the contents of rval are ignored. To
383  * set errno and have the syscall return a value other than -1 we can
384  * manually set errno and rval and return 0.
385  */
386 int
387 lx_brandsys(int cmd, int64_t *rval, uintptr_t arg1, uintptr_t arg2,
388     uintptr_t arg3, uintptr_t arg4, uintptr_t arg5, uintptr_t arg6)
389 {
390         kthread_t *t = curthread;
```

```
391         proc_t *p = ttoproc(t);
392         lx_proc_data_t *pd;
393         int linux_call;
394         struct termios *termios;
395         uint_t termios_len;
396         int error;
397         lx_brand_registration_t reg;
398
399         /*
400          * There is one operation that is suppored for non-branded
401          * process.  B_EXEC_BRAND.  This is the equilivant of an
402          * exec call, but the new process that is created will be
403          * a branded process.
404          */
405         if (cmd == B_EXEC_BRAND) {
406                 ASSERT(p->p_zone != NULL);
407                 ASSERT(p->p_zone->zone_brand == &lx_brand);
408                 return (exec_common(
409                     (char *)arg1, (const char **)arg2, (const char **)arg3,
410                     EBA_BRAND));
411         }
412
413         /* For all other operations this must be a branded process. */
414         if (p->p_brand == NULL)
415                 return (set_errno(ENOSYS));
416
417         ASSERT(p->p_brand == &lx_brand);
418         ASSERT(p->p_brand_data != NULL);
419
420         switch (cmd) {
421         case B_REGISTER:
422                 if (p->p_model == DATAMODEL_NATIVE) {
423                         if (copyin((void *)arg1, &reg, sizeof (reg)) != 0) {
424                                 lx_print("Failed to copyin brand registration "
425                                     "at 0x%p\n", (void *)arg1);
426                                 return (EFAULT);
427                         }
428 #ifdef _LP64
429                 } else {
430                         lx_brand_registration32_t reg32;
431
432                         if (copyin((void *)arg1, &reg32, sizeof (reg32)) != 0) {
433                                 lx_print("Failed to copyin brand registration "
434                                     "at 0x%p\n", (void *)arg1);
435                                 return (EFAULT);
436                         }
437
438                         reg.lxbr_version = (uint_t)reg32.lxbr_version;
439                         reg.lxbr_handler =
440                             (void *)(uintptr_t)reg32.lxbr_handler;
441                         reg.lxbr_tracehandler =
442                             (void *)(uintptr_t)reg32.lxbr_tracehandler;
443                         reg.lxbr_traceflag =
444                             (void *)(uintptr_t)reg32.lxbr_traceflag;
445 #endif
446                 }
447
448                 if (reg.lxbr_version != LX_VERSION_1) {
449                         lx_print("Invalid brand library version (%u)\n",
450                             reg.lxbr_version);
451                         return (EINVAL);
452                 }
453
454                 lx_print("Assigning brand 0x%p and handler 0x%p to proc 0x%p\n",
455                     (void *)&lx_brand, (void *)reg.lxbr_handler, (void *)p);
456                 pd = p->p_brand_data;
```



```
457                 pd->l_handler = (uintptr_t)reg.lxbr_handler;
458                 pd->l_tracehandler = (uintptr_t)reg.lxbr_tracehandler;
459                 pd->l_traceflag = (uintptr_t)reg.lxbr_traceflag;
460                 *rval = 0;
461                 return (0);
462         case B_TTYMODES:
463                 /* This is necessary for emulating TCGETS ioctls. */
464                 if (ddi_prop_lookup_byte_array(DDI_DEV_T_ANY, ddi_root_node(),
465                     DDI_PROP_NOTPROM, "ttymodes", (uchar_t **)&termios,
466                     &termios_len) != DDI_SUCCESS)
467                         return (EIO);
468
469                 ASSERT(termios_len == sizeof (*termios));
470
471                 if (copyout(&termios, (void *)arg1, sizeof (termios)) != 0) {
472                         ddi_prop_free(termios);
473                         return (EFAULT);
474                 }
475
476                 ddi_prop_free(termios);
477                 *rval = 0;
478                 return (0);
479
480         case B_ELFDATA:
481                 pd = curproc->p_brand_data;
482                 if (copyout(&pd->l_elf_data, (void *)arg1,
483                     sizeof (lx_elf_data_t)) != 0) {
484                         (void) set_errno(EFAULT);
485                         return (*rval = -1);
486                 }
487                 *rval = 0;
488                 return (0);
489
490         case B_EXEC_NATIVE:
491                 error = exec_common(
492                     (char *)arg1, (const char **)arg2, (const char **)arg3,
493                     EBA_NATIVE);
494                 if (error) {
495                         (void) set_errno(error);
496                         return (*rval = -1);
497                 }
498                 return (*rval = 0);
499
500         case B_LPID_TO_SPAIR:
501                 /*
502                  * Given a Linux pid as arg1, return the Solaris pid in arg2 and
503                  * the Solaris LWP in arg3.  We also translate pid 1 (which is
504                  * hardcoded in many applications) to the zone's init process.
505                  */
506                 {
507                         pid_t s_pid;
508                         id_t s_tid;
509
510                         if ((pid_t)arg1 == 1) {
511                                 s_pid = p->p_zone->zone_proc_initpid;
512                                 /* handle the dead/missing init(1M) case */
513                                 if (s_pid == -1)
514                                         s_pid = 1;
515                                 s_tid = 1;
516                         } else if (lx_lpid_to_spair((pid_t)arg1, &s_pid,
517                             &s_tid) < 0)
518                                 return (ESRCH);
519
520                         if (copyout(&s_pid, (void *)arg2,
521                             sizeof (s_pid)) != 0 ||
522                             copyout(&s_tid, (void *)arg3, sizeof (s_tid)) != 0)
```

```
523                                 return (EFAULT);

525                         *rval = 0;
526                         return (0);
527                 }

529         case B_PTRACE_SYSCALL:
530                 *rval = lx_ptrace_syscall_set((pid_t)arg1, (id_t)arg2,
531                     (int)arg3);
532                 return (0);

534         case B_SYSENTRY:
535                 if (lx_systrace_enabled) {
536                         uint32_t args[6];

538                         ASSERT(lx_systrace_entry_ptr != NULL);

540                         if (copyin((void *)arg2, args, sizeof (args)) != 0)
541                                 return (EFAULT);

543                         (*lx_systrace_entry_ptr)(arg1, args[0], args[1],
544                             args[2], args[3], args[4], args[5]);
545                 }

547                 lx_ptrace_fire();

549                 pd = p->p_brand_data;

551                 /*
552                  * If neither DTrace not ptrace are interested in tracing
553                  * this process any more, turn off the trace flag.
554                  */
555                 if (!lx_systrace_enabled && !pd->l_ptrace)
556                         (void) suword32((void *)pd->l_traceflag, 0);

558                 *rval = 0;
559                 return (0);

561         case B_SYSRETURN:
562                 if (lx_systrace_enabled) {
563                         ASSERT(lx_systrace_return_ptr != NULL);

565                         (*lx_systrace_return_ptr)(arg1, arg2, arg2, 0, 0, 0, 0);
566                 }

568                 lx_ptrace_fire();

570                 pd = p->p_brand_data;

572                 /*
573                  * If neither DTrace not ptrace are interested in tracing
574                  * this process any more, turn off the trace flag.
575                  */
576                 if (!lx_systrace_enabled && !pd->l_ptrace)
577                         (void) suword32((void *)pd->l_traceflag, 0);

579                 *rval = 0;
580                 return (0);

582         case B_SET_AFFINITY_MASK:
583         case B_GET_AFFINITY_MASK:
584                 /*
585                  * Retrieve or store the CPU affinity mask for the
586                  * requested linux pid.
587                  *
588                  * arg1 is a linux PID (0 means curthread).
```



```
589                  * arg2 is the size of the given mask.
590                  * arg3 is the address of the affinity mask.
591                  */
592                 return (lx_sched_affinity(cmd, arg1, arg2, arg3, rval));

594         default:
595                 linux_call = cmd - B_EMULATE_SYSCALL;
596                 /*
597                  * Only checking against highest syscall number for all kernel
598                  * versions, since check for specific kernel version is done
599                  * in userland prior to this call, and duplicating logic would
600                  * be redundant.
601                  */
602                 if (linux_call >= 0 && linux_call < LX_NSYSCALLS) {
603                         *rval = lx_emulate_syscall(linux_call, arg1, arg2,
604                             arg3, arg4, arg5, arg6);
605                         return (0);
606                 }
607         }

609         return (EINVAL);
610 }

612 int
613 lx_get_zone_kern_version(zone_t *zone)
614 {
615         return (((lx_zone_data_t *)zone->zone_brand_data)->lxzd_kernel_version);
616 }

618 int
619 lx_get_kern_version()
620 {
621         return (lx_get_zone_kern_version(curzone));
622 }

624 void
625 lx_set_kern_version(zone_t *zone, int vers)
626 {
627         lx_zone_data_t *lxzd = (lx_zone_data_t *)zone->zone_brand_data;

629         lxzd->lxzd_kernel_version = vers;
630         if (vers == LX_KERN_2_6)
631                 lxzd->lxzd_max_syscall = LX_NSYSCALLS_2_6;
632 }

634 /*
635  * Copy the per-process brand data from a parent proc to a child.
636  */
637 void
638 lx_copy_procdata(proc_t *child, proc_t *parent)
639 {
640         lx_proc_data_t *cpd, *ppd;

642         ppd = parent->p_brand_data;

644         ASSERT(ppd != NULL);

646         cpd = kmem_alloc(sizeof (lx_proc_data_t), KM_SLEEP);
647         *cpd = *ppd;

649         child->p_brand_data = cpd;
650 }

652 /*
653  * Currently, only 32-bit branded ELF executables are supported.
654  */
```

```
655 #if defined(_LP64)
656 #define elfexec                 elf32exec
657 #define mapexec_brand           mapexec32_brand
658 #endif /* _LP64 */

660 /*
661  * Exec routine called by elfexec() to load 32-bit Linux binaries.
662  */
663 static int
664 lx_elfexec(struct vnode *vp, struct execa *uap, struct uarg *args,
665     struct intpdata *idata, int level, long *execsz, int setid,
666     caddr_t exec_file, struct cred *cred, int brand_action)
667 {
668         int             error;
669         vnode_t         *nvp;
670         auxv32_t        phdr_auxv32[3] = {
671             { AT_SUN_BRAND_LX_PHDR, 0 },
672             { AT_SUN_BRAND_AUX2, 0 },
673             { AT_SUN_BRAND_AUX3, 0 }
674         };
675         Elf32_Ehdr      ehdr;
676         Elf32_Addr      uphdr_vaddr;
677         intptr_t        voffset;
678         int             interp;
679         int             i;
680         struct execenv  env;
681         struct user     *up = PTOU(ttoproc(curthread));
682         lx_elf_data_t   *edp =
683             &((lx_proc_data_t *)ttoproc(curthread)->p_brand_data)->l_elf_data;

685         ASSERT(ttoproc(curthread)->p_brand == &lx_brand);
686         ASSERT(ttoproc(curthread)->p_brand_data != NULL);

688         /*
689          * Set the brandname and library name for the new process so that
690          * elfexec() puts them onto the stack.
691          */
692         args->brandname = LX_BRANDNAME;
693         args->emulator = LX_LIB_PATH;

695         /*
696          * We will exec the brand library, and map in the linux linker and the
697          * linux executable.
698          */
699         if ((error = lookupname(LX_LIB_PATH, UIO_SYSSPACE, FOLLOW, NULLVPP,
700             &nvp))) {
701                 uprintf("%s: not found.", LX_LIB);
702                 return (error);
703         }

705         if ((error = elfexec(nvp, uap, args, idata, level + 1, execsz, setid,
706             exec_file, cred, brand_action))) {
707                 VN_RELE(nvp);
708                 return (error);
709         }
710         VN_RELE(nvp);

712         bzero(&env, sizeof (env));

714         if ((error = mapexec_brand(vp, args, &ehdr, &uphdr_vaddr, &voffset,
715             exec_file, &interp, &env.ex_bssbase, &env.ex_brkbase,
716             &env.ex_brksize, NULL)))
717                 return (error);

719         /*
720          * Save off the important properties of the lx executable. The brand
```



```
721          * library will ask us for this data later, when it is ready to set
722          * things up for the lx executable.
723          */
724         edp->ed_phdr = (uphdr_vaddr == -1) ? voffset + ehdr.e_phoff :
725             voffset + uphdr_vaddr;
726         edp->ed_entry = voffset + ehdr.e_entry;
727         edp->ed_phent = ehdr.e_phentsize;
728         edp->ed_phnum = ehdr.e_phnum;

730         if (interp) {
731                 if (ehdr.e_type == ET_DYN) {
732                         /*
733                          * This is a shared object executable, so we need to
734                          * pick a reasonable place to put the heap. Just don't
735                          * use the first page.
736                          */
737                         env.ex_brkbase = (caddr_t)PAGESIZE;
738                         env.ex_bssbase = (caddr_t)PAGESIZE;
739                 }

741                 /*
742                  * If the program needs an interpreter (most do), map it in and
743                  * store relevant information about it in the aux vector, where
744                  * the brand library can find it.
745                  */
746                 if ((error = lookupname(LX_LINKER, UIO_SYSSPACE, FOLLOW, NULLVPP
747                     &nvp))) {
748                         uprintf("%s: not found.", LX_LINKER);
749                         return (error);
750                 }
751                 if ((error = mapexec_brand(nvp, args, &ehdr, &uphdr_vaddr,
752                     &voffset, exec_file, &interp, NULL, NULL, NULL, NULL))) {
753                         VN_RELE(nvp);
754                         return (error);
755                 }
756                 VN_RELE(nvp);

758                 /*
759                  * Now that we know the base address of the brand's linker,
760                  * place it in the aux vector.
761                  */
762                 edp->ed_base = voffset;
763                 edp->ed_ldentry = voffset + ehdr.e_entry;
764         } else {
765                 /*
766                  * This program has no interpreter. The lx brand library will
767                  * jump to the address in the AT_SUN_BRAND_LDENTRY aux vector,
768                  * so in this case, put the entry point of the main executable
769                  * there.
770                  */
771                 if (ehdr.e_type == ET_EXEC) {
772                         /*
773                          * An executable with no interpreter, this must be a
774                          * statically linked executable, which means we loaded
775                          * it at the address specified in the elf header, in
776                          * which case the e_entry field of the elf header is an
777                          * absolute address.
778                          */
779                         edp->ed_ldentry = ehdr.e_entry;
780                         edp->ed_entry = ehdr.e_entry;
781                 } else {
782                         /*
783                          * A shared object with no interpreter, we use the
784                          * calculated address from above.
785                          */
786                         edp->ed_ldentry = edp->ed_entry;
```

```
788                         /*
789                          * In all situations except an ET_DYN elf object with no
790                          * interpreter, we want to leave the brk and base
791                          * values set by mapexec_brand alone. Normally when
792                          * running ET_DYN objects on Solaris (most likely
793                          * /lib/ld.so.1) the kernel sets brk and base to 0 since
794                          * it doesn't know where to put the heap, and later the
795                          * linker will call brk() to initialize the heap in:
796                          *      usr/src/cmd/sgs/rtld/common/setup.c:setup()
797                          * after it has determined where to put it.  (This
798                          * decision is made after the linker loads and inspects
799                          * elf properties of the target executable being run.)
800                          *
801                          * So for ET_DYN Linux executables, we also don't know
802                          * where the heap should go, so we'll set the brk and
803                          * base to 0.  But in this case the Solaris linker will
804                          * not initialize the heap, so when the Linux linker
805                          * starts running there is no heap allocated.  This
806                          * seems to be ok on Linux 2.4 based systems because the
807                          * Linux linker/libc fall back to using mmap() to
808                          * allocate memory. But on 2.6 systems, running
809                          * applications by specifying them as command line
810                          * arguments to the linker results in segfaults for an
811                          * as yet undetermined reason (which seems to indicatej
812                          * that a more permanent fix for heap initalization in
813                          * these cases may be necessary).
814                          */
815                         if (ehdr.e_type == ET_DYN) {
816                                 env.ex_bssbase = (caddr_t)0;
817                                 env.ex_brkbase = (caddr_t)0;
818                                 env.ex_brksize = 0;
819                         }
820                 }

822         }

824         env.ex_vp = vp;
825         setexecenv(&env);

827         /*
828          * We don't need to copy this stuff out. It is only used by our
829          * tools to locate the lx linker's debug section. But we should at
830          * least try to keep /proc's view of the aux vector consistent with
831          * what's on the process stack.
832          */
833         phdr_auxv32[0].a_un.a_val = edp->ed_phdr;

835         /*
836          * Linux 2.6 programs such as ps will print an error message if the
837          * following aux entry is missing
838          */
839         if (lx_get_kern_version() >= LX_KERN_2_6) {
840                 phdr_auxv32[1].a_type = AT_CLKTCK;
841                 phdr_auxv32[1].a_un.a_val = hz;
842         }

844         if (copyout(&phdr_auxv32, args->auxp_brand,
845             sizeof (phdr_auxv32)) == -1)
846                 return (EFAULT);

848         /*
849          * /proc uses the AT_ENTRY aux vector entry to deduce
850          * the location of the executable in the address space. The user
851          * structure contains a copy of the aux vector that needs to have those
852          * entries patched with the values of the real lx executable (they
```



```
853          * currently contain the values from the lx brand library that was
854          * elfexec'd, above).
855          *
856          * For live processes, AT_BASE is used to locate the linker segment,
857          * which /proc and friends will later use to find Solaris symbols
858          * (such as rtld_db_preinit). However, for core files, /proc uses
859          * AT_ENTRY to find the right segment to label as the executable.
860          * So we set AT_ENTRY to be the entry point of the linux executable,
861          * but leave AT_BASE to be the address of the Solaris linker.
862          */
863         for (i = 0; i < __KERN_NAUXV_IMPL; i++) {
864                 if (up->u_auxv[i].a_type == AT_ENTRY)
865                         up->u_auxv[i].a_un.a_val = edp->ed_entry;
866                 if (up->u_auxv[i].a_type == AT_SUN_BRAND_LX_PHDR)
867                         up->u_auxv[i].a_un.a_val = edp->ed_phdr;
868         }

870         return (0);
871 }

873 int
874 _init(void)
875 {
876         int err = 0;

878         /* pid/tid conversion hash tables */
879         lx_pid_init();

881         /* for lx_futex() */
882         lx_futex_init();

884         err = mod_install(&modlinkage);
885         if (err != 0) {
886                 cmn_err(CE_WARN, "Couldn't install lx brand module");

888                 /*
889                  * This looks drastic, but it should never happen.  These
890                  * two data structures should be completely free-able until
891                  * they are used by Linux processes.  Since the brand
892                  * wasn't loaded there should be no Linux processes, and
893                  * thus no way for these data structures to be modified.
894                  */
895                 lx_pid_fini();
896                 if (lx_futex_fini())
897                         panic("lx brand module cannot be loaded or unloaded.");
898         }
899         return (err);
900 }

902 int
903 _info(struct modinfo *modinfop)
904 {
905         return (mod_info(&modlinkage, modinfop));
906 }

908 int
909 _fini(void)
910 {
911         int err;
912         int futex_done = 0;

914         /*
915          * If there are any zones using this brand, we can't allow it to be
916          * unloaded.
917          */
918         if (brand_zone_count(&lx_brand))
```

```
919                 return (EBUSY);

921         lx_pid_fini();

923         if ((err = lx_futex_fini()) != 0)
924                 goto done;
925         futex_done = 1;

927         err = mod_remove(&modlinkage);

929 done:
930         if (err) {
931                 /*
932                  * If we can't unload the module, then we have to get it
933                  * back into a sane state.
934                  */
935                 lx_pid_init();

937                 if (futex_done)
938                         lx_futex_init();

940         }

942         return (err);
943 }
944 #endif /* ! codereview */
```

```
**********************************************************
    8557 Tue Jan 14 16:17:19 2014
new/usr/src/uts/common/brand/lx/os/lx_misc.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/errno.h>
#include <sys/systm.h>
#include <sys/archsystm.h>
#include <sys/privregs.h>
#include <sys/exec.h>
#include <sys/lwp.h>
#include <sys/sem.h>
#include <sys/brand.h>
#include <sys/lx_brand.h>
#include <sys/lx_pid.h>
#include <sys/lx_futex.h>

/* Linux specific functions and definitions */
void lx_setrval(klwp_t *, int, int);
void lx_exec();
int lx_initlwp(klwp_t *);
void lx_forklwp(klwp_t *, klwp_t *);
void lx_exitlwp(klwp_t *);
void lx_freelwp(klwp_t *);
static void lx_save(klwp_t *);
static void lx_restore(klwp_t *);
extern void lx_ptrace_free(proc_t *);

/*
 * Set the return code for the forked child, always zero
 */
/*ARGSUSED*/
void
lx_setrval(klwp_t *lwp, int v1, int v2)
{
        lwptoregs(lwp)->r_r0 = 0;
}

/*
 * Reset process state on exec(2)
 */
```



```
void
lx_exec()
{
        klwp_t *lwp = ttolwp(curthread);
        struct lx_lwp_data *lwpd = lwptolxlwp(lwp);
        int err;

        /*
         * There are two mutually exclusive special cases we need to
         * address.  First, if this was a native process prior to this
         * exec(), then this lwp won't have its brand-specific data
         * initialized and it won't be assigned a Linux PID yet.  Second,
         * if this was a multi-threaded Linux process and this lwp wasn't
         * the main lwp, then we need to make its Solaris and Linux PIDS
         * match.
         */
        if (lwpd == NULL) {
                err = lx_initlwp(lwp);
                /*
                 * Only possible failure from this routine should be an
                 * inability to allocate a new PID.  Since single-threaded
                 * processes don't need a new PID, we should never hit this
                 * error.
                 */
                ASSERT(err == 0);
                lwpd = lwptolxlwp(lwp);
        } else if (curthread->t_tid != 1) {
                lx_pid_reassign(curthread);
        }

        installctx(lwptot(lwp), lwp, lx_save, lx_restore, NULL, NULL, lx_save,
            NULL);

        /*
         * clear out the tls array
         */
        bzero(lwpd->br_tls, sizeof (lwpd->br_tls));

        /*
         * reset the tls entries in the gdt
         */
        kpreempt_disable();
        lx_restore(lwp);
        kpreempt_enable();
}

void
lx_exitlwp(klwp_t *lwp)
{
        struct lx_lwp_data *lwpd = lwptolxlwp(lwp);
        proc_t *p;
        kthread_t *t;
        sigqueue_t *sqp = NULL;
        pid_t ppid;
        id_t ptid;

        if (lwpd == NULL)
                return;         /* second time thru' */

        if (lwpd->br_clear_ctidp != NULL) {
                (void) suword32(lwpd->br_clear_ctidp, 0);
                (void) lx_futex((uintptr_t)lwpd->br_clear_ctidp, FUTEX_WAKE, 1,
                    NULL, NULL, 0);
        }

        if (lwpd->br_signal != 0) {
```

```
128                 /*
129                  * The first thread in a process doesn't cause a signal to
130                  * be sent when it exits.  It was created by a fork(), not
131                  * a clone(), so the parent should get signalled when the
132                  * process exits.
133                  */
134                 if (lwpd->br_ptid == -1)
135                         goto free;

137                 sqp = kmem_zalloc(sizeof (sigqueue_t), KM_SLEEP);
138                 /*
139                  * If br_ppid is 0, it means this is a CLONE_PARENT thread,
140                  * so the signal goes to the parent process - not to a
141                  * specific thread in this process.
142                  */
143                 p = lwptoproc(lwp);
144                 if (lwpd->br_ppid == 0) {
145                         mutex_enter(&p->p_lock);
146                         ppid = p->p_ppid;
147                         t = NULL;
148                 } else {
149                         /*
150                          * If we have been reparented to init or if our
151                          * parent thread is gone, then nobody gets
152                          * signaled.
153                          */
154                         if ((lx_lwp_ppid(lwp, &ppid, &ptid) == 1) ||
155                             (ptid == -1))
156                                 goto free;

158                         mutex_enter(&pidlock);
159                         if ((p = prfind(ppid)) == NULL || p->p_stat == SIDL) {
160                                 mutex_exit(&pidlock);
161                                 goto free;
162                         }
163                         mutex_enter(&p->p_lock);
164                         mutex_exit(&pidlock);

166                         if ((t = idtot(p, ptid)) == NULL) {
167                                 mutex_exit(&p->p_lock);
168                                 goto free;
169                         }
170                 }

172                 sqp->sq_info.si_signo = lwpd->br_signal;
173                 sqp->sq_info.si_code = lwpd->br_exitwhy;
174                 sqp->sq_info.si_status = lwpd->br_exitwhat;
175                 sqp->sq_info.si_pid = lwpd->br_pid;
176                 sqp->sq_info.si_uid = crgetruid(CRED());
177                 sigaddqa(p, t, sqp);
178                 mutex_exit(&p->p_lock);
179                 sqp = NULL;
180         }

182 free:
183         if (sqp)
184                 kmem_free(sqp, sizeof (sigqueue_t));

186         lx_freelwp(lwp);
187 }

189 void
190 lx_freelwp(klwp_t *lwp)
191 {
192         struct lx_lwp_data *lwpd = lwptolxlwp(lwp);
```



```
194         if (lwpd != NULL) {
195                 (void) removectx(lwptot(lwp), lwp, lx_save, lx_restore,
196                     NULL, NULL, lx_save, NULL);
197                 if (lwpd->br_pid != 0)
198                         lx_pid_rele(lwptoproc(lwp)->p_pid,
199                             lwptot(lwp)->t_tid);

201                 lwp->lwp_brand = NULL;
202                 kmem_free(lwpd, sizeof (struct lx_lwp_data));
203         }
204 }

206 int
207 lx_initlwp(klwp_t *lwp)
208 {
209         struct lx_lwp_data *lwpd;
210         struct lx_lwp_data *plwpd;
211         kthread_t *tp = lwptot(lwp);

213         lwpd = kmem_zalloc(sizeof (struct lx_lwp_data), KM_SLEEP);
214         lwpd->br_exitwhy = CLD_EXITED;
215         lwpd->br_lwp = lwp;
216         lwpd->br_clear_ctidp = NULL;
217         lwpd->br_set_ctidp = NULL;
218         lwpd->br_signal = 0;
219         /*
220          * lwpd->br_affinitymask was zeroed by kmem_zalloc().
221          */

223         /*
224          * The first thread in a process has ppid set to the parent
225          * process's pid, and ptid set to -1.  Subsequent threads in the
226          * process have their ppid set to the pid of the thread that
227          * created them, and their ptid to that thread's tid.
228          */
229         if (tp->t_next == tp) {
230                 lwpd->br_ppid = tp->t_procp->p_ppid;
231                 lwpd->br_ptid = -1;
232         } else if (ttolxlwp(curthread) != NULL) {
233                 plwpd = ttolxlwp(curthread);
234                 bcopy(plwpd->br_tls, lwpd->br_tls, sizeof (lwpd->br_tls));
235                 lwpd->br_ppid = plwpd->br_pid;
236                 lwpd->br_ptid = curthread->t_tid;
237         } else {
238                 /*
239                  * Oddball case: the parent thread isn't a Linux process.
240                  */
241                 lwpd->br_ppid = 0;
242                 lwpd->br_ptid = -1;
243         }
244         lwp->lwp_brand = lwpd;

246         if (lx_pid_assign(tp)) {
247                 kmem_free(lwpd, sizeof (struct lx_lwp_data));
248                 lwp->lwp_brand = NULL;
249                 return (-1);
250         }
251         lwpd->br_tgid = lwpd->br_pid;

253         installctx(lwptot(lwp), lwp, lx_save, lx_restore, NULL, NULL,
254             lx_save, NULL);

256         return (0);
257 }

259 /*
```

```
 * There is no need to have any locking for either the source or
 * destination struct lx_lwp_data structs.  This is always run in the
 * thread context of the source thread, and the destination thread is
 * always newly created and not referred to from anywhere else.
 */
void
lx_forklwp(klwp_t *srclwp, klwp_t *dstlwp)
{
        struct lx_lwp_data *src = srclwp->lwp_brand;
        struct lx_lwp_data *dst = dstlwp->lwp_brand;

        dst->br_ppid = src->br_pid;
        dst->br_ptid = lwptot(srclwp)->t_tid;
        bcopy(src->br_tls, dst->br_tls, sizeof (dst->br_tls));

        /*
         * copy only these flags
         */
        dst->br_lwp_flags = src->br_lwp_flags & BR_CPU_BOUND;
        dst->br_clone_args = NULL;
}

/*
 * When switching a Linux process off the CPU, clear its GDT entries.
 */
/* ARGSUSED */
static void
lx_save(klwp_t *t)
{
        int i;

#if defined(__amd64)
        reset_sregs();
#endif
        for (i = 0; i < LX_TLSNUM; i++)
                gdt_update_usegd(GDT_TLSMIN + i, &null_udesc);
}

/*
 * When switching a Linux process on the CPU, set its GDT entries.
 */
static void
lx_restore(klwp_t *t)
{
        struct lx_lwp_data *lwpd = lwptolxlwp(t);
        user_desc_t *tls;
        int i;

        ASSERT(lwpd);

        tls = lwpd->br_tls;
        for (i = 0; i < LX_TLSNUM; i++)
                gdt_update_usegd(GDT_TLSMIN + i, &tls[i]);
}

void
lx_set_gdt(int entry, user_desc_t *descrp)
{

        gdt_update_usegd(entry, descrp);
}

void
lx_clear_gdt(int entry)
{
        gdt_update_usegd(entry, &null_udesc);
```



```
}

longlong_t
lx_nosys()
{
        return (set_errno(ENOSYS));
}

longlong_t
lx_opnotsupp()
{
        return (set_errno(EOPNOTSUPP));
}

/*
 * Brand-specific routine to check if given non-Solaris standard segment
 * register values should be modified to other values.
 */
/*ARGSUSED*/
greg_t
lx_fixsegreg(greg_t sr, model_t datamodel)
{
        ASSERT(sr == (sr & 0xffff));

        /*
         * Force the SR into the LDT in ring 3 for 32-bit processes.
         *
         * 64-bit processes get the null GDT selector since they are not
         * allowed to have a private LDT.
         */
#if defined(__amd64)
        return (datamodel == DATAMODEL_ILP32 ? (sr | SEL_TI_LDT | SEL_UPL) : 0);
#elif defined(__i386)
        datamodel = datamodel;  /* datamodel currently unused for 32-bit */
        return (sr | SEL_TI_LDT | SEL_UPL);
#endif  /* __amd64 */
}
#endif /* ! codereview */
```

```
**********************************************************
    8381 Tue Jan 14 16:17:19 2014
new/usr/src/uts/common/brand/lx/os/lx_pid.c
LX zone support should now build and packages of relevance produced.
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/types.h>
#include <sys/param.h>
#include <sys/sysmacros.h>
#include <sys/bitmap.h>
#include <sys/var.h>
#include <sys/thread.h>
#include <sys/proc.h>
#include <sys/brand.h>
#include <sys/zone.h>
#include <sys/lx_brand.h>
#include <sys/lx_pid.h>

#define LINUX_PROC_FACTOR       8       /* factor down the hash table by this */
static int hash_len = 4;                /* desired average hash chain length */
static int hash_size;                   /* no of buckets in the hash table */

static struct lx_pid **stol_pid_hash;
static struct lx_pid **ltos_pid_hash;

#define LTOS_HASH(pid)          ((pid) & (hash_size - 1))
#define STOL_HASH(pid, tid)     (((pid) + (tid)) & (hash_size - 1))

static kmutex_t hash_lock;

static void
lx_pid_insert_hash(struct lx_pid *lpidp)
{
        int shash = STOL_HASH(lpidp->s_pid, lpidp->s_tid);
        int lhash = LTOS_HASH(lpidp->l_pid);

        ASSERT(MUTEX_HELD(&hash_lock));

        lpidp->stol_next = stol_pid_hash[shash];
        stol_pid_hash[shash] = lpidp;
```



```
        lpidp->ltos_next = ltos_pid_hash[lhash];
        ltos_pid_hash[lhash] = lpidp;
}

static struct lx_pid *
lx_pid_remove_hash(pid_t pid, id_t tid)
{
        struct lx_pid **hpp;
        struct lx_pid *lpidp = NULL;

        ASSERT(MUTEX_HELD(&hash_lock));

        hpp = &stol_pid_hash[STOL_HASH(pid, tid)];
        while (*hpp) {
                if ((*hpp)->s_pid == pid && (*hpp)->s_tid == tid) {
                        lpidp = *hpp;
                        *hpp = (*hpp)->stol_next;
                        break;
                }
                hpp = &(*hpp)->stol_next;
        }

        /*
         * when called during error recovery the pid may already
         * be released
         */
        if (lpidp == NULL)
                return (NULL);

        hpp = &ltos_pid_hash[LTOS_HASH(lpidp->l_pid)];
        while (*hpp) {
                if (*hpp == lpidp) {
                        *hpp = lpidp->ltos_next;
                        break;
                }
                hpp = &(*hpp)->ltos_next;
        }

        return (lpidp);
}

struct pid * pid_find(pid_t pid);

/*
 * given a solaris pid/tid pair, create a linux pid
 */
int
lx_pid_assign(kthread_t *t)
{
        proc_t *p = ttoproc(t);
        pid_t s_pid = p->p_pid;
        id_t s_tid = t->t_tid;
        struct pid *pidp;
        struct lx_pid *lpidp;
        lx_lwp_data_t *lwpd = ttolxlwp(t);
        pid_t newpid;

        if (p->p_lwpcnt > 0) {
                /*
                 * Allocate a pid for any thread other than the first
                 */
                if ((newpid = pid_allocate(p, 0, 0)) < 0)
                        return (-1);

                pidp = pid_find(newpid);
        } else {
```

```
127                 pidp = NULL;
128                 newpid = s_pid;
129         }

131         lpidp = kmem_alloc(sizeof (struct lx_pid), KM_SLEEP);
132         lpidp->l_pid = newpid;
133         lpidp->s_pid = s_pid;
134         lpidp->s_tid = s_tid;
135         lpidp->l_pidp = pidp;
136         lpidp->l_start = t->t_start;

138         /*
139          * now put the pid into the linux-solaris and solaris-linux
140          * conversion hash tables
141          */
142         mutex_enter(&hash_lock);
143         lx_pid_insert_hash(lpidp);
144         mutex_exit(&hash_lock);

146         lwpd->br_pid = newpid;

148         return (0);
149 }

151 /*
152  * If we are exec()ing the process, this thread's tid is about to be reset
153  * to 1.  Make sure the Linux PID bookkeeping reflects that change.
154  */
155 void
156 lx_pid_reassign(kthread_t *t)
157 {
158         proc_t *p = ttoproc(t);
159         struct pid *old_pidp;
160         struct lx_pid *lpidp;

162         ASSERT(p->p_lwpcnt == 1);

164         mutex_enter(&hash_lock);

166         /*
167          * Clean up all the traces of this thread's 'fake' Linux PID.
168          */
169         lpidp = lx_pid_remove_hash(p->p_pid, t->t_tid);
170         ASSERT(lpidp != NULL);
171         old_pidp = lpidp->l_pidp;
172         lpidp->l_pidp = NULL;

174         /*
175          * Now register this thread as (pid, 1).
176          */
177         lpidp->l_pid = p->p_pid;
178         lpidp->s_pid = p->p_pid;
179         lpidp->s_tid = 1;
180         lx_pid_insert_hash(lpidp);

182         mutex_exit(&hash_lock);

184         if (old_pidp)
185                 (void) pid_rele(old_pidp);
186 }

188 /*
189  * release a solaris pid/tid pair
190  */
191 void
192 lx_pid_rele(pid_t pid, id_t tid)
```



```
193 {
194         struct lx_pid *lpidp;

196         mutex_enter(&hash_lock);
197         lpidp = lx_pid_remove_hash(pid, tid);
198         mutex_exit(&hash_lock);

200         if (lpidp) {
201                 if (lpidp->l_pidp)
202                         (void) pid_rele(lpidp->l_pidp);

204                 kmem_free(lpidp, sizeof (*lpidp));
205         }
206 }

208 /*
209  * given a linux pid, return the solaris pid/tid pair
210  */
211 int
212 lx_lpid_to_spair(pid_t l_pid, pid_t *s_pid, id_t *s_tid)
213 {
214         struct lx_pid *hp;

216         mutex_enter(&hash_lock);
217         for (hp = ltos_pid_hash[LTOS_HASH(l_pid)]; hp; hp = hp->ltos_next) {
218                 if (l_pid == hp->l_pid) {
219                         if (s_pid)
220                                 *s_pid = hp->s_pid;
221                         if (s_tid)
222                                 *s_tid = hp->s_tid;
223                         break;
224                 }
225         }
226         mutex_exit(&hash_lock);
227         if (hp != NULL)
228                 return (0);

230         /*
231          * We didn't find this pid in our translation table.
232          * But this still could be the pid of a native process
233          * running in the current zone so check for that here.
234          *
235          * Note that prfind() only searches for processes in the current zone.
236          */
237         mutex_enter(&pidlock);
238         if (prfind(l_pid) != NULL) {
239                 mutex_exit(&pidlock);
240                 if (s_pid)
241                         *s_pid = l_pid;
242                 if (s_tid)
243                         *s_tid = 0;
244                 return (0);
245         }
246         mutex_exit(&pidlock);

248         return (-1);
249 }

251 /*
252  * Given an lwp, return the Linux pid of its parent.  If the caller
253  * wants them, we return the Solaris (pid, tid) as well.
254  */
255 pid_t
256 lx_lwp_ppid(klwp_t *lwp, pid_t *ppidp, id_t *ptidp)
257 {
258         lx_lwp_data_t *lwpd = lwptolxlwp(lwp);
```

```
259         proc_t *p = lwptoproc(lwp);
260         struct lx_pid *hp;
261         pid_t zoneinit = curproc->p_zone->zone_proc_initpid;
262         pid_t lppid, ppid;
263
264         /*
265          * Be sure not to return a parent pid that should be invisible
266          * within this zone.
267          */
268         ppid = ((p->p_flag & SZONETOP)
269             ? curproc->p_zone->zone_zsched->p_pid : p->p_ppid);
270
271         /*
272          * If the parent process's pid is the zone's init process, force it
273          * to the Linux init pid value of 1.
274          */
275         if (ppid == zoneinit)
276                 ppid = 1;
277
278         /*
279          * There are two cases in which the Linux definition of a 'parent'
280          * matches that of Solaris:
281          *
282          * - if our tgid is the same as our PID, then we are either the
283          *   first thread in the process or a CLONE_THREAD thread.
284          *
285          * - if the brand lwp value for ppid is 0, then we are either the
286          *   child of a differently-branded process or a CLONE_PARENT thread.
287          */
288         if (p->p_pid == lwpd->br_tgid || lwpd->br_ppid == 0) {
289                 if (ppidp != NULL)
290                         *ppidp = ppid;
291                 if (ptidp != NULL)
292                         *ptidp = -1;
293                 return (ppid);
294         }
295
296         /*
297          * Set the default Linux parent pid to be the pid of the zone's init
298          * process; this will get converted back to the Linux default of 1
299          * later.
300          */
301         lppid = zoneinit;
302
303         /*
304          * If the process's parent isn't init, try and look up the Linux "pid"
305          * corresponding to the process's parent.
306          */
307         if (ppid != 1) {
308                 /*
309                  * In all other cases, we are looking for the parent of this
310                  * specific thread, which in Linux refers to the thread that
311                  * clone()d it.   We stashed that thread's PID away when this
312                  * thread was created.
313                  */
314                 mutex_enter(&hash_lock);
315                 for (hp = ltos_pid_hash[LTOS_HASH(lwpd->br_ppid)]; hp;
316                     hp = hp->ltos_next) {
317                         if (lwpd->br_ppid == hp->l_pid) {
318                                 /*
319                                  * We found the PID we were looking for, but
320                                  * since we cached its value in this LWP's brand
321                                  * structure, it has exited and been reused by
322                                  * another process.
323                                  */
324                                 if (hp->l_start > lwptot(lwp)->t_start)
```



```
325                                         break;
326
327                                 lppid = lwpd->br_ppid;
328                                 if (ppidp != NULL)
329                                         *ppidp = hp->s_pid;
330                                 if (ptidp != NULL)
331                                         *ptidp = hp->s_tid;
332
333                                 break;
334                         }
335                 }
336                 mutex_exit(&hash_lock);
337         }
338
339         if (lppid == zoneinit) {
340                 lppid = 1;
341
342                 if (ppidp != NULL)
343                         *ppidp = lppid;
344                 if (ptidp != NULL)
345                         *ptidp = -1;
346         }
347
348         return (lppid);
349 }
350
351 void
352 lx_pid_init(void)
353 {
354         hash_size = 1 << highbit(v.v_proc / (hash_len * LINUX_PROC_FACTOR));
355
356         stol_pid_hash = kmem_zalloc(sizeof (struct lx_pid *) * hash_size,
357             KM_SLEEP);
358         ltos_pid_hash = kmem_zalloc(sizeof (struct lx_pid *) * hash_size,
359             KM_SLEEP);
360
361         mutex_init(&hash_lock, NULL, MUTEX_DEFAULT, NULL);
362 }
363
364 void
365 lx_pid_fini(void)
366 {
367         kmem_free(stol_pid_hash, sizeof (struct lx_pid *) * hash_size);
368         kmem_free(ltos_pid_hash, sizeof (struct lx_pid *) * hash_size);
369 }
370 #endif /* ! codereview */
```

```
**********************************************************
   11890 Tue Jan 14 16:17:19 2014
new/usr/src/uts/common/brand/lx/os/lx_syscall.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/kmem.h>
#include <sys/errno.h>
#include <sys/thread.h>
#include <sys/systm.h>
#include <sys/syscall.h>
#include <sys/proc.h>
#include <sys/modctl.h>
#include <sys/cmn_err.h>
#include <sys/model.h>
#include <sys/brand.h>
#include <sys/machbrand.h>
#include <sys/lx_syscalls.h>
#include <sys/lx_brand.h>
#include <sys/lx_impl.h>

/*
 * Some system calls return either a 32-bit or a 64-bit value, depending
 * on the datamodel.
 */
#ifdef  _LP64
#define V_RVAL  SE_64RVAL
#else
#define V_RVAL  SE_32RVAL1
#endif

/*
 * Define system calls that return a native 'long' quantity i.e. a 32-bit
 * or 64-bit integer - depending on how the kernel is itself compiled
 * e.g. read(2) returns 'ssize_t' in the kernel and in userland.
 */
#define LX_CL(name, call, narg)      \
        { V_RVAL, (name), (llfcn_t)(call), (narg) }
```



```
/*
 * Returns a 32 bit quantity regardless of datamodel
 */
#define LX_CI(name, call, narg)      \
        { SE_32RVAL1, (name), (llfcn_t)(call), (narg) }

extern longlong_t lx_nosys(void);
#define LX_NOSYS(name)                  \
        {SE_64RVAL, (name), (llfcn_t)lx_nosys, 0}

lx_sysent_t lx_sysent[] =
{
        LX_NOSYS("lx_nosys"),                                   /* 0 */
        LX_NOSYS("exit"),                                       /* 0 */
        LX_NOSYS("lx_fork"),
        LX_NOSYS("read"),
        LX_NOSYS("write"),
        LX_NOSYS("open"),
        LX_NOSYS("close"),
        LX_NOSYS("waitpid"),
        LX_NOSYS("creat"),
        LX_NOSYS("link"),
        LX_NOSYS("unlink"),                                     /* 10 */
        LX_NOSYS("exec"),
        LX_NOSYS("chdir"),
        LX_NOSYS("gtime"),
        LX_NOSYS("mknod"),
        LX_NOSYS("chmod"),
        LX_NOSYS("lchown16"),
        LX_NOSYS("break"),
        LX_NOSYS("stat"),
        LX_NOSYS("lseek"),
        LX_CL("getpid", lx_getpid,      0),                     /* 20 */
        LX_NOSYS("mount"),
        LX_NOSYS("umount"),
        LX_NOSYS("setuid16"),
        LX_NOSYS("getuid16"),
        LX_NOSYS("stime"),
        LX_NOSYS("ptrace"),
        LX_NOSYS("alarm"),
        LX_NOSYS("fstat"),
        LX_NOSYS("pause"),
        LX_NOSYS("utime"),                                      /* 30 */
        LX_NOSYS("stty"),
        LX_NOSYS("gtty"),
        LX_NOSYS("access"),
        LX_NOSYS("nice"),
        LX_NOSYS("ftime"),
        LX_NOSYS("sync"),
        LX_CL("kill",           lx_kill,                2),
        LX_NOSYS("rename"),
        LX_NOSYS("mkdir"),
        LX_NOSYS("rmdir"),                                      /* 40 */
        LX_NOSYS("dup"),
        LX_NOSYS("pipe"),
        LX_NOSYS("times"),
        LX_NOSYS("prof"),
        LX_CL("brk",    lx_brk,         1),
        LX_NOSYS("setgid16"),
        LX_NOSYS("getgid16"),
        LX_NOSYS("signal"),
        LX_NOSYS("geteuid16"),
        LX_NOSYS("getegid16"),                                  /* 50 */
        LX_NOSYS("sysacct"),
        LX_NOSYS("umount2"),
        LX_NOSYS("lock"),
```

```
128         LX_NOSYS("ioctl"),
129         LX_NOSYS("fcntl"),
130         LX_NOSYS("mpx"),
131         LX_NOSYS("setpgid"),
132         LX_NOSYS("ulimit"),
133         LX_NOSYS("olduname"),
134         LX_NOSYS("umask"),                                  /* 60 */
135         LX_NOSYS("chroot"),
136         LX_NOSYS("ustat"),
137         LX_NOSYS("dup2"),
138         LX_CL("getppid",        lx_getppid,     0),
139         LX_NOSYS("pgrp"),
140         LX_NOSYS("setsid"),
141         LX_NOSYS("sigaction"),
142         LX_NOSYS("sgetmask"),
143         LX_NOSYS("ssetmask"),
144         LX_NOSYS("setreuid16"),                             /* 70 */
145         LX_NOSYS("setregid16"),
146         LX_NOSYS("sigsuspend"),
147         LX_NOSYS("sigpending"),
148         LX_NOSYS("sethostname"),
149         LX_NOSYS("setrlimit"),
150         LX_NOSYS("old_getrlimit"),
151         LX_NOSYS("getrusage"),
152         LX_NOSYS("gettimeofday"),
153         LX_NOSYS("settimeofday"),
154         LX_NOSYS("getgroups16"),                            /* 80 */
155         LX_NOSYS("setgroups16"),
156         LX_NOSYS("old_select"),
157         LX_NOSYS("symlink"),
158         LX_NOSYS("oldlstat"),
159         LX_NOSYS("readlink"),
160         LX_NOSYS("uselib"),
161         LX_NOSYS("swapon"),
162         LX_NOSYS("reboot"),
163         LX_NOSYS("old_readdir"),
164         LX_NOSYS("old_mmap"),                               /* 90 */
165         LX_NOSYS("munmap"),
166         LX_NOSYS("truncate"),
167         LX_NOSYS("ftruncate"),
168         LX_NOSYS("fchmod"),
169         LX_NOSYS("fchown16"),
170         LX_NOSYS("getpriority"),
171         LX_NOSYS("setpriority"),
172         LX_NOSYS("profil"),
173         LX_NOSYS("statfs"),
174         LX_NOSYS("fstatfs"),                                /* 100 */
175         LX_NOSYS("ioperm"),
176         LX_NOSYS("socketcall"),
177         LX_NOSYS("syslog"),
178         LX_NOSYS("setitimer"),
179         LX_NOSYS("getitimer"),
180         LX_NOSYS("newstat"),
181         LX_NOSYS("newslstat"),
182         LX_NOSYS("newsfstat"),
183         LX_NOSYS("uname"),
184         LX_NOSYS("oldiopl"),                                /* 110 */
185         LX_NOSYS("oldvhangup"),
186         LX_NOSYS("idle"),
187         LX_NOSYS("vm86old"),
188         LX_NOSYS("wait4"),
189         LX_NOSYS("swapoff"),
190         LX_CL("sysinfo", lx_sysinfo,    1),
191         LX_NOSYS("ipc"),
192         LX_NOSYS("fsync"),
193         LX_NOSYS("sigreturn"),
```



```
194         LX_CL("clone",  lx_clone,       5),                 /* 120 */
195         LX_NOSYS("setdomainname"),
196         LX_NOSYS("newuname"),
197         LX_CL("modify_ldt",     lx_modify_ldt,  3),
198         LX_NOSYS("adjtimex"),
199         LX_NOSYS("mprotect"),
200         LX_NOSYS("sigprocmask"),
201         LX_NOSYS("create_module"),
202         LX_NOSYS("init_module"),
203         LX_NOSYS("delete_module"),
204         LX_NOSYS("get_kernel_syms"),                        /* 130 */
205         LX_NOSYS("quotactl"),
206         LX_NOSYS("getpgid"),
207         LX_NOSYS("fchdir"),
208         LX_NOSYS("bdflush"),
209         LX_NOSYS("sysfs"),
210         LX_NOSYS("personality"),
211         LX_NOSYS("afs_syscall"),
212         LX_NOSYS("setfsuid16"),
213         LX_NOSYS("setfsgid16"),
214         LX_NOSYS("llseek"),                                 /* 140 */
215         LX_NOSYS("getdents"),
216         LX_NOSYS("select"),
217         LX_NOSYS("flock"),
218         LX_NOSYS("msync"),
219         LX_NOSYS("readv"),
220         LX_NOSYS("writev"),
221         LX_NOSYS("getsid"),
222         LX_NOSYS("fdatasync"),
223         LX_NOSYS("sysctl"),
224         LX_NOSYS("mlock"),                                  /* 150 */
225         LX_NOSYS("munlock"),
226         LX_NOSYS("mlockall"),
227         LX_NOSYS("munlockall"),
228         LX_CL("sched_setparam", lx_sched_setparam, 2),
229         LX_CL("sched_getparam", lx_sched_getparam, 2),
230         LX_NOSYS("sched_setscheduler"),
231         LX_NOSYS("sched_getscheduler"),
232         LX_NOSYS("yield"),
233         LX_NOSYS("sched_get_priority_max"),
234         LX_NOSYS("sched_get_priority_min"),                 /* 160 */
235         LX_CL("sched_rr_get_interval", lx_sched_rr_get_interval, 2),
236         LX_NOSYS("nanosleep"),
237         LX_NOSYS("mremap"),
238         LX_CL("setresuid16",            lx_setresuid16, 3),
239         LX_NOSYS("getresuid16"),
240         LX_NOSYS("vm86"),
241         LX_NOSYS("query_module"),
242         LX_NOSYS("poll"),
243         LX_NOSYS("nfsserctl"),
244         LX_CL("setresgid16",            lx_setresgid16, 3),     /* 170 */
245         LX_NOSYS("getresgid16"),
246         LX_NOSYS("prctl"),
247         LX_NOSYS("rt_sigreturn"),
248         LX_NOSYS("rt_sigaction"),
249         LX_NOSYS("rt_sigprocmask"),
250         LX_NOSYS("rt_sigpending"),
251         LX_NOSYS("rt_sigtimedwait"),
252         LX_NOSYS("rt_sigqueueinfo"),
253         LX_NOSYS("rt_sigsuspend"),
254         LX_NOSYS("pread64"),                                /* 180 */
255         LX_NOSYS("pwrite64"),
256         LX_NOSYS("chown16"),
257         LX_NOSYS("getcwd"),
258         LX_NOSYS("capget"),
259         LX_NOSYS("capset"),
```

```
260         LX_NOSYS("sigaltstack"),
261         LX_NOSYS("sendfile"),
262         LX_NOSYS("getpmsg"),
263         LX_NOSYS("putpmsg"),
264         LX_NOSYS("vfork"),                                  /* 190 */
265         LX_NOSYS("getrlimit"),
266         LX_NOSYS("mmap2"),
267         LX_NOSYS("truncate64"),
268         LX_NOSYS("ftruncate64"),
269         LX_NOSYS("stat64"),
270         LX_NOSYS("lstat64"),
271         LX_NOSYS("fstat64"),
272         LX_NOSYS("lchown"),
273         LX_NOSYS("getuid"),
274         LX_NOSYS("getgid"),                                 /* 200 */
275         LX_NOSYS("geteuid"),
276         LX_NOSYS("getegid"),
277         LX_NOSYS("setreuid"),
278         LX_NOSYS("setregid"),
279         LX_NOSYS("getgroups"),
280         LX_CL("setgroups",      lx_setgroups,   2),
281         LX_NOSYS("fchown"),
282         LX_CL("setresuid",      lx_setresuid,   3),
283         LX_NOSYS("getresuid"),
284         LX_CL("setresgid",      lx_setresgid,   3),         /* 210 */
285         LX_NOSYS("getresgid"),
286         LX_NOSYS("chown"),
287         LX_NOSYS("setuid"),
288         LX_NOSYS("setgid"),
289         LX_NOSYS("setfsuid"),
290         LX_NOSYS("setfsgid"),
291         LX_NOSYS("pivot_root"),
292         LX_NOSYS("mincore"),
293         LX_NOSYS("madvise"),
294         LX_NOSYS("getdents64"),                             /* 220 */
295         LX_NOSYS("fcntl64"),
296         LX_NOSYS("lx_nosys"),
297         LX_NOSYS("security"),
298         LX_CL("gettid", lx_gettid,      0),
299         LX_NOSYS("readahead"),
300         LX_NOSYS("setxattr"),
301         LX_NOSYS("lsetxattr"),
302         LX_NOSYS("fsetxattr"),
303         LX_NOSYS("getxattr"),
304         LX_NOSYS("lgetxattr"),                              /* 230 */
305         LX_NOSYS("fgetxattr"),
306         LX_NOSYS("listxattr"),
307         LX_NOSYS("llistxattr"),
308         LX_NOSYS("flistxattr"),
309         LX_NOSYS("removexattr"),
310         LX_NOSYS("lremovexattr"),
311         LX_NOSYS("fremovexattr"),
312         LX_CL("tkill",          lx_tkill,               2),
313         LX_NOSYS("sendfile64"),
314         LX_CL("futex",          lx_futex,               6),     /* 240 */
315         LX_NOSYS("sched_setaffinity"),
316         LX_NOSYS("sched_getaffinity"),
317         LX_CL("set_thread_area",        lx_set_thread_area,     1),
318         LX_CL("get_thread_area",        lx_get_thread_area,     1),
319         LX_NOSYS("io_setup"),
320         LX_NOSYS("io_destroy"),
321         LX_NOSYS("io_getevents"),
322         LX_NOSYS("io_submit"),
323         LX_NOSYS("io_cancel"),
324         LX_NOSYS("fadvise64"),                              /* 250 */
325         LX_NOSYS("lx_nosys"),
```



```
326         LX_NOSYS("exit_group"),
327         LX_NOSYS("lookup_dcookie"),
328         LX_NOSYS("epoll_create"),
329         LX_NOSYS("epoll_ctl"),
330         LX_NOSYS("epoll_wait"),
331         LX_NOSYS("remap_file_pages"),
332         LX_CL("set_tid_address",        lx_set_tid_address,     1),
333         LX_NOSYS("timer_create"),
334         LX_NOSYS("timer_settime"),                          /* 260 */
335         LX_NOSYS("timer_gettime"),
336         LX_NOSYS("timer_getoverrun"),
337         LX_NOSYS("timer_delete"),
338         LX_NOSYS("clock_settime"),
339         LX_NOSYS("clock_gettime"),
340         LX_NOSYS("clock_getres"),
341         LX_NOSYS("clock_nanosleep"),
342         LX_NOSYS("statfs64"),
343         LX_NOSYS("fstatfs64"),
344         LX_NOSYS("tgkill"),                                 /* 270 */
345         /* The following are Linux 2.6 system calls */
346         LX_NOSYS("utimes"),
347         LX_NOSYS("fadvise64_64"),
348         LX_NOSYS("vserver"),
349         LX_NOSYS("mbind"),
350         LX_NOSYS("get_mempolicy"),
351         LX_NOSYS("set_mempolicy"),
352         LX_NOSYS("mq_open"),
353         LX_NOSYS("mq_unlink"),
354         LX_NOSYS("mq_timedsend"),
355         LX_NOSYS("mq_timedreceive"),                        /* 280 */
356         LX_NOSYS("mq_notify"),
357         LX_NOSYS("mq_getsetattr"),
358         LX_NOSYS("kexec_load"),
359         LX_NOSYS("waitid"),
360         LX_NOSYS("sys_setaltroot"),
361         LX_NOSYS("add_key"),
362         LX_NOSYS("request_key"),
363         LX_NOSYS("keyctl"),
364         LX_NOSYS("ioprio_set"),
365         LX_NOSYS("ioprio_get"),                             /* 290 */
366         LX_NOSYS("inotify_init"),
367         LX_NOSYS("inotify_add_watch"),
368         LX_NOSYS("inotify_rm_watch"),
369         LX_NOSYS("migrate_pages"),
370         LX_NOSYS("openat"),
371         LX_NOSYS("mkdirat"),
372         LX_NOSYS("mknodat"),
373         LX_NOSYS("fchownat"),
374         LX_NOSYS("futimesat"),
375         LX_NOSYS("fstatat64"),                              /* 300 */
376         LX_NOSYS("unlinkat"),
377         LX_NOSYS("renameat"),
378         LX_NOSYS("linkat"),
379         LX_NOSYS("syslinkat"),
380         LX_NOSYS("readlinkat"),
381         LX_NOSYS("fchmodat"),
382         LX_NOSYS("faccessat"),
383         LX_NOSYS("pselect6"),
384         LX_NOSYS("ppoll"),
385         LX_NOSYS("unshare"),                                /* 310 */
386         LX_NOSYS("set_robust_list"),
387         LX_NOSYS("get_robust_list"),
388         LX_NOSYS("splice"),
389         LX_NOSYS("sync_file_range"),
390         LX_NOSYS("tee"),
391         LX_NOSYS("vmsplice"),
```

```
392         LX_NOSYS("move_pages"),
393         NULL    /* NULL-termination is required for lx_systrace */
394 };

396 int64_t
397 lx_emulate_syscall(int num, uintptr_t arg1, uintptr_t arg2,
398     uintptr_t arg3, uintptr_t arg4, uintptr_t arg5, uintptr_t arg6)
399 {
400         struct lx_sysent *jsp;
401         int64_t rval;

403         rval = (int64_t)0;

405         jsp = &(lx_sysent[num]);

407         switch (jsp->sy_narg) {
408         case 0: {
409                 lx_print("--> %s()\n", jsp->sy_name);
410                 rval = (int64_t)jsp->sy_callc();
411                 break;
412         }
413         case 1: {
414                 lx_print("--> %s(0x%lx)\n", jsp->sy_name, arg1);
415                 rval = (int64_t)jsp->sy_callc(arg1);
416                 break;
417         }
418         case 2: {
419                 lx_print("--> %s(0x%lx, 0x%lx)\n", jsp->sy_name, arg1, arg2);
420                 rval = (int64_t)jsp->sy_callc(arg1, arg2);
421                 break;
422         }
423         case 3: {
424                 lx_print("--> %s(0x%lx, 0x%lx, 0x%lx)\n",
425                     jsp->sy_name, arg1, arg2, arg3);
426                 rval = (int64_t)jsp->sy_callc(arg1, arg2, arg3);
427                 break;
428         }
429         case 4: {
430                 lx_print("--> %s(0x%lx, 0x%lx, 0x%lx, 0x%lx)\n",
431                     jsp->sy_name, arg1, arg2, arg3, arg4);
432                 rval = (int64_t)jsp->sy_callc(arg1, arg2, arg3, arg4);
433                 break;
434         }
435         case 5: {
436                 lx_print("--> %s(0x%lx, 0x%lx, 0x%lx, 0x%lx, 0x%lx)\n",
437                     jsp->sy_name, arg1, arg2, arg3, arg4, arg5);
438                 rval = (int64_t)jsp->sy_callc(arg1, arg2, arg3, arg4, arg5);
439                 break;
440         }
441         case 6: {
442                 lx_print("--> %s(0x%lx, 0x%lx, 0x%lx, 0x%lx,"
443                     " 0x%lx, 0x%lx)\n",
444                     jsp->sy_name, arg1, arg2, arg3, arg4, arg5, arg6);
445                 rval = (int64_t)jsp->sy_callc(arg1, arg2, arg3, arg4, arg5,
446                     arg6);
447                 break;
448         }
449         default:
450                 panic("Invalid syscall entry: #%d at 0x%p\n", num, (void *)jsp);
451         }
452         lx_print("----------> return  (0x%llx)\n", (long long)rval);
453         return (rval);
454 }
455 #endif /* ! codereview */
```

```
**********************************************************
    6862 Tue Jan 14 16:17:19 2014
new/usr/src/uts/common/brand/lx/procfs/lx_proc.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _LXPROC_H
#define _LXPROC_H

#ifdef __cplusplus
extern "C" {
#endif

/*
 * lxproc.h: declarations, data structures and macros for lxprocfs
 */


#include <sys/types.h>
#include <sys/param.h>
#include <sys/policy.h>
#include <sys/debug.h>
#include <sys/dirent.h>
#include <sys/errno.h>
#include <sys/file.h>
#include <sys/kmem.h>
#include <sys/pathname.h>
#include <sys/proc.h>
#include <sys/systm.h>
#include <sys/var.h>
#include <sys/user.h>
#include <sys/t_lock.h>
#include <sys/sysmacros.h>
#include <sys/cred.h>
#include <sys/priv.h>
#include <sys/vnode.h>
#include <sys/vfs.h>
#include <sys/statvfs.h>
#include <sys/cmn_err.h>
#include <sys/zone.h>
#include <sys/uio.h>
#include <sys/utsname.h>
```



```
#include <sys/dnlc.h>
#include <sys/atomic.h>
#include <sys/sunddi.h>
#include <sys/sunldi.h>
#include <vm/as.h>
#include <vm/anon.h>

/*
 * Convert a vnode into an lxpr_mnt_t
 */
#define VTOLXPM(vp)     ((lxpr_mnt_t *)(vp)->v_vfsp->vfs_data)

/*
 * convert a vnode into an lxpr_node
 */
#define VTOLXP(vp)      ((lxpr_node_t *)(vp)->v_data)

/*
 * convert a lxprnode into a vnode
 */
#define LXPTOV(lxpnp)   ((lxpnp)->lxpr_vnode)

/*
 * convert a lxpr_node into zone for fs
 */
#define LXPTOZ(lxpnp) \
        (((lxpr_mnt_t *)(lxpnp)->lxpr_vnode->v_vfsp->vfs_data)->lxprm_zone)

#define LXPNSIZ         256     /* max size of lx /proc file name entries */

/*
 * Pretend that a directory entry takes 16 bytes
 */
#define LXPR_SDSIZE     16

/*
 * Node/file types for lx /proc files
 * (directories and files contained therein).
 */
typedef enum lxpr_nodetype {
        LXPR_PROCDIR,           /* /proc                */
        LXPR_PIDDIR,            /* /proc/<pid>          */
        LXPR_PID_CMDLINE,       /* /proc/<pid>/cmdline  */
        LXPR_PID_CPU,           /* /proc/<pid>/cpu      */
        LXPR_PID_CURDIR,        /* /proc/<pid>/cwd      */
        LXPR_PID_ENV,           /* /proc/<pid>/environ  */
        LXPR_PID_EXE,           /* /proc/<pid>/exe      */
        LXPR_PID_MAPS,          /* /proc/<pid>/maps     */
        LXPR_PID_MEM,           /* /proc/<pid>/mem      */
        LXPR_PID_ROOTDIR,       /* /proc/<pid>/root     */
        LXPR_PID_STAT,          /* /proc/<pid>/stat     */
        LXPR_PID_STATM,         /* /proc/<pid>/statm    */
        LXPR_PID_STATUS,        /* /proc/<pid>/status   */
        LXPR_PID_FDDIR,         /* /proc/<pid>/fd       */
        LXPR_PID_FD_FD,         /* /proc/<pid>/fd/nn    */
        LXPR_CMDLINE,           /* /proc/cmdline        */
        LXPR_CPUINFO,           /* /proc/cpuinfo        */
        LXPR_DEVICES,           /* /proc/devices        */
        LXPR_DMA,               /* /proc/dma            */
        LXPR_FILESYSTEMS,       /* /proc/filesystems    */
        LXPR_INTERRUPTS,        /* /proc/interrupts     */
        LXPR_IOPORTS,           /* /proc/ioports        */
        LXPR_KCORE,             /* /proc/kcore          */
        LXPR_KMSG,              /* /proc/kmsg           */
        LXPR_LOADAVG,           /* /proc/loadavg        */
        LXPR_MEMINFO,           /* /proc/meminfo        */
```

```
128         LXPR_MOUNTS,            /* /proc/mounts         */
129         LXPR_NETDIR,            /* /proc/net            */
130         LXPR_NET_ARP,           /* /proc/net/arp        */
131         LXPR_NET_DEV,           /* /proc/net/dev        */
132         LXPR_NET_DEV_MCAST,     /* /proc/net/dev_mcast  */
133         LXPR_NET_IGMP,          /* /proc/net/igmp       */
134         LXPR_NET_IP_MR_CACHE,   /* /proc/net/ip_mr_cache */
135         LXPR_NET_IP_MR_VIF,     /* /proc/net/ip_mr_vif  */
136         LXPR_NET_MCFILTER,      /* /proc/net/mcfilter   */
137         LXPR_NET_NETSTAT,       /* /proc/net/netstat    */
138         LXPR_NET_RAW,           /* /proc/net/raw        */
139         LXPR_NET_ROUTE,         /* /proc/net/route      */
140         LXPR_NET_RPC,           /* /proc/net/rpc        */
141         LXPR_NET_RT_CACHE,      /* /proc/net/rt_cache   */
142         LXPR_NET_SOCKSTAT,      /* /proc/net/sockstat   */
143         LXPR_NET_SNMP,          /* /proc/net/snmp       */
144         LXPR_NET_STAT,          /* /proc/net/stat       */
145         LXPR_NET_TCP,           /* /proc/net/tcp        */
146         LXPR_NET_UDP,           /* /proc/net/udp        */
147         LXPR_NET_UNIX,          /* /proc/net/unix       */
148         LXPR_PARTITIONS,        /* /proc/partitions     */
149         LXPR_SELF,              /* /proc/self           */
150         LXPR_STAT,              /* /proc/stat           */
151         LXPR_UPTIME,            /* /proc/uptime         */
152         LXPR_VERSION,           /* /proc/version        */
153         LXPR_NFILES             /* number of lx /proc file types */
154 } lxpr_nodetype_t;


157 /*
158  * Number of fds allowed for in the inode number calculation
159  * per process (if a process has more fds then inode numbers
160  * may be duplicated)
161  */
162 #define LXPR_FD_PERPROC 2000

164 /*
165  * external dirent characteristics
166  */
167 #define LXPRMAXNAMELEN  14
168 typedef struct {
169         lxpr_nodetype_t d_type;
170         char            d_name[LXPRMAXNAMELEN];
171 } lxpr_dirent_t;

173 /*
174  * This is the lxprocfs private data object
175  * which is attached to v_data in the vnode structure
176  */
177 typedef struct lxpr_node {
178         lxpr_nodetype_t lxpr_type;      /* type of this node            */
179         vnode_t         *lxpr_vnode;    /* vnode for the node           */
180         vnode_t         *lxpr_parent;   /* parent directory             */
181         vnode_t         *lxpr_realvp;   /* real vnode, file in dirs     */
182         timestruc_t     lxpr_time;      /* creation etc time for file   */
183         mode_t          lxpr_mode;      /* file mode bits               */
184         uid_t           lxpr_uid;       /* file owner                   */
185         gid_t           lxpr_gid;       /* file group owner             */
186         pid_t           lxpr_pid;       /* pid of proc referred to      */
187         ino_t           lxpr_ino;       /* node id                      */
188         ldi_handle_t    lxpr_cons_ldih; /* ldi handle for console device */
189 } lxpr_node_t;

191 struct zone;    /* forward declaration */

193 /*
```



```
194  * This is the lxprocfs private data object
195  * which is attached to vfs_data in the vfs structure
196  */
197 typedef struct lxpr_mnt {
198         lxpr_node_t     *lxprm_node;    /* node at root of proc mount */
199         struct zone     *lxprm_zone;    /* zone for this mount */
200         ldi_ident_t     lxprm_li;       /* ident for ldi */
201 } lxpr_mnt_t;

203 extern vnodeops_t       *lxpr_vnodeops;
204 extern int              nproc_highbit;  /* highbit(v.v_nproc)           */

206 typedef struct mounta   mounta_t;

208 extern void lxpr_initnodecache();
209 extern void lxpr_fininodecache();
210 extern void lxpr_initrootnode(lxpr_node_t **, vfs_t *);
211 extern ino_t lxpr_inode(lxpr_nodetype_t, pid_t, int);
212 extern ino_t lxpr_parentinode(lxpr_node_t *);
213 extern lxpr_node_t *lxpr_getnode(vnode_t *, lxpr_nodetype_t, proc_t *, int);
214 extern void lxpr_freenode(lxpr_node_t *);

216 typedef struct lxpr_uiobuf lxpr_uiobuf_t;
217 extern lxpr_uiobuf_t *lxpr_uiobuf_new(uio_t *);
218 extern void lxpr_uiobuf_free(lxpr_uiobuf_t *);
219 extern int lxpr_uiobuf_flush(lxpr_uiobuf_t *);
220 extern void lxpr_uiobuf_seek(lxpr_uiobuf_t *, offset_t);
221 extern void lxpr_uiobuf_write(lxpr_uiobuf_t *, const char *, size_t);
222 extern void lxpr_uiobuf_printf(lxpr_uiobuf_t *, const char *, ...);
223 extern void lxpr_uiobuf_seterr(lxpr_uiobuf_t *, int);

225 proc_t *lxpr_lock(pid_t);
226 void lxpr_unlock(proc_t *);

228 #ifdef  __cplusplus
229 }
230 #endif

232 #endif /* _LXPROC_H */
233 #endif /* ! codereview */
```

```
**********************************************************
   11131 Tue Jan 14 16:17:20 2014
new/usr/src/uts/common/brand/lx/procfs/lx_prsubr.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

/*
 * lxprsubr.c: Various functions for the /lxproc vnodeops.
 */

#include <sys/varargs.h>

#include <sys/cpuvar.h>
#include <sys/mman.h>
#include <sys/vmsystm.h>
#include <sys/prsystm.h>

#include "lx_proc.h"

#define LXPRCACHE_NAME "lxpr_cache"

static int lxpr_node_constructor(void*, void*, int);
static void lxpr_node_destructor(void*, void*);

static kmem_cache_t *lxpr_node_cache;

struct lxpr_uiobuf {
        uio_t *uiop;
        char *buffer;
        uint32_t buffsize;
        char *pos;
        size_t beg;
        int error;
};

#define BUFSIZE 4000

struct lxpr_uiobuf *
lxpr_uiobuf_new(uio_t *uiop)
{
```



```
        /* Allocate memory for both lxpr_uiobuf and output buffer */
        struct lxpr_uiobuf *uiobuf =
            kmem_alloc(sizeof (struct lxpr_uiobuf) + BUFSIZE, KM_SLEEP);

        uiobuf->uiop = uiop;
        uiobuf->buffer = (char *)&uiobuf[1];
        uiobuf->buffsize = BUFSIZE;
        uiobuf->pos = uiobuf->buffer;
        uiobuf->beg = 0;
        uiobuf->error = 0;

        return (uiobuf);
}

void
lxpr_uiobuf_free(struct lxpr_uiobuf *uiobuf)
{
        ASSERT(uiobuf != NULL);
        ASSERT(uiobuf->pos == uiobuf->buffer);

        kmem_free(uiobuf, sizeof (struct lxpr_uiobuf) + uiobuf->buffsize);
}

void
lxpr_uiobuf_seek(struct lxpr_uiobuf *uiobuf, offset_t offset)
{
        uiobuf->uiop->uio_offset = offset;
}

void
lxpr_uiobuf_seterr(struct lxpr_uiobuf *uiobuf, int err)
{
        ASSERT(uiobuf->error == 0);

        uiobuf->error = err;
}

int
lxpr_uiobuf_flush(struct lxpr_uiobuf *uiobuf)
{
        off_t off = uiobuf->uiop->uio_offset;
        caddr_t uaddr = uiobuf->buffer;
        size_t beg = uiobuf->beg;

        size_t size = uiobuf->pos - uaddr;

        if (uiobuf->error == 0 && uiobuf->uiop->uio_resid != 0) {
                ASSERT(off >= beg);

                if (beg+size > off && off >= 0)
                        uiobuf->error =
                            uiomove(uaddr+(off-beg), size-(off-beg),
                            UIO_READ, uiobuf->uiop);

                uiobuf->beg += size;
        }

        uiobuf->pos = uaddr;

        return (uiobuf->error);
}

void
lxpr_uiobuf_write(struct lxpr_uiobuf *uiobuf, const char *buf, size_t size)
{
        /* While we can still carry on */
```

```
128         while (uiobuf->error == 0 && uiobuf->uiop->uio_resid != 0) {
129                 uint_t remain
130                     = uiobuf->buffsize-(uiobuf->pos-uiobuf->buffer);
131
132                 /* Enough space in buffer? */
133                 if (remain >= size) {
134                         bcopy(buf, uiobuf->pos, size);
135                         uiobuf->pos += size;
136                         return;
137                 }
138
139                 /* Not enough space, so copy all we can and try again */
140                 bcopy(buf, uiobuf->pos, remain);
141                 uiobuf->pos += remain;
142                 (void) lxpr_uiobuf_flush(uiobuf);
143                 buf += remain;
144                 size -= remain;
145         }
146 }
147
148 #define TYPBUFFSIZE 256
149 void
150 lxpr_uiobuf_printf(struct lxpr_uiobuf *uiobuf, const char *fmt, ...)
151 {
152         va_list args;
153         char buff[TYPBUFFSIZE];
154         int len;
155         char *buffer;
156
157         /* Can we still do any output */
158         if (uiobuf->error != 0 || uiobuf->uiop->uio_resid == 0)
159                 return;
160
161         va_start(args, fmt);
162
163         /* Try using stack allocated buffer */
164         len = vsnprintf(buff, TYPBUFFSIZE, fmt, args);
165         if (len < TYPBUFFSIZE) {
166                 va_end(args);
167                 lxpr_uiobuf_write(uiobuf, buff, len);
168                 return;
169         }
170
171         /* Not enough space in pre-allocated buffer */
172         buffer = kmem_alloc(len+1, KM_SLEEP);
173
174         /*
175          * We know we allocated the correct amount of space
176          * so no check on the return value
177          */
178         (void) vsnprintf(buffer, len+1, fmt, args);
179         lxpr_uiobuf_write(uiobuf, buffer, len);
180         va_end(args);
181         kmem_free(buffer, len+1);
182 }
183
184 /*
185  * lxpr_lock():
186  *
187  * Lookup process from pid and return with p_plock and P_PR_LOCK held.
188  */
189 proc_t *
190 lxpr_lock(pid_t pid)
191 {
192         proc_t *p;
193         kmutex_t *mp;
```



```
194
195         ASSERT(!MUTEX_HELD(&pidlock));
196
197         for (;;) {
198                 mutex_enter(&pidlock);
199
200                 /*
201                  * If the pid is 1, we really want the zone's init process
202                  */
203                 p = prfind((pid == 1) ?
204                     curproc->p_zone->zone_proc_initpid : pid);
205
206                 if (p == NULL || p->p_stat == SIDL) {
207                         mutex_exit(&pidlock);
208                         return (NULL);
209                 }
210                 /*
211                  * p_lock is persistent, but p itself is not -- it could
212                  * vanish during cv_wait().  Load p->p_lock now so we can
213                  * drop it after cv_wait() without referencing p.
214                  */
215                 mp = &p->p_lock;
216                 mutex_enter(mp);
217
218                 mutex_exit(&pidlock);
219
220                 if (!(p->p_proc_flag & P_PR_LOCK))
221                         break;
222
223                 cv_wait(&pr_pid_cv[p->p_slot], mp);
224                 mutex_exit(mp);
225         }
226         p->p_proc_flag |= P_PR_LOCK;
227         THREAD_KPRI_REQUEST();
228         return (p);
229 }
230
231 /*
232  * lxpr_unlock()
233  *
234  * Unlock locked process
235  */
236 void
237 lxpr_unlock(proc_t *p)
238 {
239         ASSERT(p->p_proc_flag & P_PR_LOCK);
240         ASSERT(MUTEX_HELD(&p->p_lock));
241         ASSERT(!MUTEX_HELD(&pidlock));
242
243         cv_signal(&pr_pid_cv[p->p_slot]);
244         p->p_proc_flag &= ~P_PR_LOCK;
245         mutex_exit(&p->p_lock);
246         THREAD_KPRI_RELEASE();
247 }
248
249 void
250 lxpr_initnodecache()
251 {
252         lxpr_node_cache = kmem_cache_create(LXPRCACHE_NAME,
253             sizeof (lxpr_node_t), 0,
254             lxpr_node_constructor, lxpr_node_destructor, NULL, NULL, NULL, 0);
255 }
256
257 void
258 lxpr_fininodecache()
259 {
```

```
260         kmem_cache_destroy(lxpr_node_cache);
261 }

263 /* ARGSUSED */
264 static int
265 lxpr_node_constructor(void *buf, void *un, int kmflags)
266 {
267         lxpr_node_t     *lxpnp = buf;
268         vnode_t         *vp;

270         vp = lxpnp->lxpr_vnode = vn_alloc(kmflags);
271         if (vp == NULL)
272                 return (-1);

274         (void) vn_setops(vp, lxpr_vnodeops);
275         vp->v_data = lxpnp;

277         return (0);
278 }

280 /* ARGSUSED */
281 static void
282 lxpr_node_destructor(void *buf, void *un)
283 {
284         lxpr_node_t     *lxpnp = buf;

286         vn_free(LXPTOV(lxpnp));
287 }

289 /*
290  * Calculate an inode number
291  *
292  * This takes various bits of info and munges them
293  * to give the inode number for an lxproc node
294  */
295 ino_t
296 lxpr_inode(lxpr_nodetype_t type, pid_t pid, int fd)
297 {
298         if (pid == 1)
299                 pid = curproc->p_zone->zone_proc_initpid;

301         switch (type) {
302         case LXPR_PIDDIR:
303                 return (pid + 1);
304         case LXPR_PROCDIR:
305                 return (maxpid + 2);
306         case LXPR_PID_FD_FD:
307                 return (maxpid + 2 +
308                     (pid * (LXPR_FD_PERPROC + LXPR_NFILES)) +
309                     LXPR_NFILES + fd);
310         default:
311                 return (maxpid + 2 +
312                     (pid * (LXPR_FD_PERPROC + LXPR_NFILES)) +
313                     type);
314         }
315 }

317 /*
318  * Return inode number of parent (directory)
319  */
320 ino_t
321 lxpr_parentinode(lxpr_node_t *lxpnp)
322 {
323         /*
324          * If the input node is the root then the parent inode
325          * is the mounted on inode so just return our inode number
```



```
326          */
327         if (lxpnp->lxpr_type != LXPR_PROCDIR)
328                 return (VTOLXP(lxpnp->lxpr_parent)->lxpr_ino);
329         else
330                 return (lxpnp->lxpr_ino);
331 }

333 /*
334  * Allocate a new lxproc node
335  *
336  * This also allocates the vnode associated with it
337  */
338 lxpr_node_t *
339 lxpr_getnode(vnode_t *dp, lxpr_nodetype_t type, proc_t *p, int fd)
340 {
341         lxpr_node_t *lxpnp;
342         vnode_t *vp;
343         user_t *up;
344         timestruc_t now;

346         /*
347          * Allocate a new node. It is deallocated in vop_innactive
348          */
349         lxpnp = kmem_cache_alloc(lxpr_node_cache, KM_SLEEP);

351         /*
352          * Set defaults (may be overridden below)
353          */
354         gethrestime(&now);
355         lxpnp->lxpr_type = type;
356         lxpnp->lxpr_realvp = NULL;
357         lxpnp->lxpr_parent = dp;
358         VN_HOLD(dp);
359         if (p != NULL) {
360                 lxpnp->lxpr_pid = ((p->p_pid ==
361                     curproc->p_zone->zone_proc_initpid) ? 1 : p->p_pid);

363                 lxpnp->lxpr_time = PTOU(p)->u_start;
364                 lxpnp->lxpr_uid = crgetruid(p->p_cred);
365                 lxpnp->lxpr_gid = crgetrgid(p->p_cred);
366                 lxpnp->lxpr_ino = lxpr_inode(type, p->p_pid, fd);
367         } else {
368                 /* Pretend files without a proc belong to sched */
369                 lxpnp->lxpr_pid = 0;
370                 lxpnp->lxpr_time = now;
371                 lxpnp->lxpr_uid = lxpnp->lxpr_gid = 0;
372                 lxpnp->lxpr_ino = lxpr_inode(type, 0, 0);
373         }

375         /* initialize the vnode data */
376         vp = lxpnp->lxpr_vnode;
377         vn_reinit(vp);
378         vp->v_flag = VNOCACHE|VNOMAP|VNOSWAP|VNOMOUNT;
379         vp->v_vfsp = dp->v_vfsp;

381         /*
382          * Do node specific stuff
383          */
384         switch (type) {
385         case LXPR_PROCDIR:
386                 vp->v_flag |= VROOT;
387                 vp->v_type = VDIR;
388                 lxpnp->lxpr_mode = 0555;        /* read-search by everyone */
389                 break;

391         case LXPR_PID_CURDIR:
```

```
392                 ASSERT(p != NULL);
393
394                 /*
395                  * Zombie check.  p_stat is officially protected by pidlock,
396                  * but we can't grab pidlock here because we already hold
397                  * p_lock.  Luckily if we look at the process exit code
398                  * we see that p_stat only transisions from SRUN to SZOMB
399                  * while p_lock is held.  Aside from this, the only other
400                  * p_stat transition that we need to be aware about is
401                  * SIDL to SRUN, but that's not a problem since lxpr_lock()
402                  * ignores nodes in the SIDL state so we'll never get a node
403                  * that isn't already in the SRUN state.
404                  */
405                 if (p->p_stat == SZOMB) {
406                         lxpnp->lxpr_realvp = NULL;
407                 } else {
408                         up = PTOU(p);
409                         lxpnp->lxpr_realvp = up->u_cdir;
410                         ASSERT(lxpnp->lxpr_realvp != NULL);
411                         VN_HOLD(lxpnp->lxpr_realvp);
412                 }
413                 vp->v_type = VLNK;
414                 lxpnp->lxpr_mode = 0777;        /* anyone does anything ! */
415                 break;
416
417         case LXPR_PID_ROOTDIR:
418                 ASSERT(p != NULL);
419                 /* Zombie check.  see locking comment above */
420                 if (p->p_stat == SZOMB) {
421                         lxpnp->lxpr_realvp = NULL;
422                 } else {
423                         up = PTOU(p);
424                         lxpnp->lxpr_realvp =
425                             up->u_rdir != NULL ? up->u_rdir : rootdir;
426                         ASSERT(lxpnp->lxpr_realvp != NULL);
427                         VN_HOLD(lxpnp->lxpr_realvp);
428                 }
429                 vp->v_type = VLNK;
430                 lxpnp->lxpr_mode = 0777;        /* anyone does anything ! */
431                 break;
432
433         case LXPR_PID_EXE:
434                 ASSERT(p != NULL);
435                 lxpnp->lxpr_realvp = p->p_exec;
436                 if (lxpnp->lxpr_realvp != NULL) {
437                         VN_HOLD(lxpnp->lxpr_realvp);
438                 }
439                 vp->v_type = VLNK;
440                 lxpnp->lxpr_mode = 0777;
441                 break;
442
443         case LXPR_SELF:
444                 vp->v_type = VLNK;
445                 lxpnp->lxpr_mode = 0777;        /* anyone does anything ! */
446                 break;
447
448         case LXPR_PID_FD_FD:
449                 ASSERT(p != NULL);
450                 /* lxpr_realvp is set after we return */
451                 vp->v_type = VLNK;
452                 lxpnp->lxpr_mode = 0700;        /* read-write-exe owner only */
453                 break;
454
455         case LXPR_PID_FDDIR:
456                 ASSERT(p != NULL);
457                 vp->v_type = VDIR;
```



```
458                 lxpnp->lxpr_mode = 0500;        /* read-search by owner only */
459                 break;
460
461         case LXPR_PIDDIR:
462                 ASSERT(p != NULL);
463                 vp->v_type = VDIR;
464                 lxpnp->lxpr_mode = 0511;
465                 break;
466
467         case LXPR_NETDIR:
468                 vp->v_type = VDIR;
469                 lxpnp->lxpr_mode = 0555;        /* read-search by all */
470                 break;
471
472         case LXPR_PID_ENV:
473         case LXPR_PID_MEM:
474                 ASSERT(p != NULL);
475                 /*FALLTHRU*/
476         case LXPR_KCORE:
477                 vp->v_type = VREG;
478                 lxpnp->lxpr_mode = 0400;        /* read-only by owner only */
479                 break;
480
481         default:
482                 vp->v_type = VREG;
483                 lxpnp->lxpr_mode = 0444;        /* read-only by all */
484                 break;
485         }
486
487         return (lxpnp);
488 }
489
490
491 /*
492  * Free the storage obtained from lxpr_getnode().
493  */
494 void
495 lxpr_freenode(lxpr_node_t *lxpnp)
496 {
497         ASSERT(lxpnp != NULL);
498         ASSERT(LXPTOV(lxpnp) != NULL);
499
500         /*
501          * delete any association with realvp
502          */
503         if (lxpnp->lxpr_realvp != NULL)
504                 VN_RELE(lxpnp->lxpr_realvp);
505
506         /*
507          * delete any association with parent vp
508          */
509         if (lxpnp->lxpr_parent != NULL)
510                 VN_RELE(lxpnp->lxpr_parent);
511
512         /*
513          * Release the lxprnode.
514          */
515         kmem_cache_free(lxpr_node_cache, lxpnp);
516 }
517 #endif /* ! codereview */
```

```
**********************************************************
    8115 Tue Jan 14 16:17:20 2014
new/usr/src/uts/common/brand/lx/procfs/lx_prvfsops.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

/*
 * lxprvfsops.c: vfs operations for /lxprocfs.
 */

#include <sys/types.h>
#include <sys/param.h>
#include <sys/cmn_err.h>
#include <sys/cred.h>
#include <sys/debug.h>
#include <sys/errno.h>
#include <sys/proc.h>
#include <sys/stat.h>
#include <sys/statvfs.h>
#include <sys/sysmacros.h>
#include <sys/systm.h>
#include <sys/var.h>
#include <sys/vfs.h>
#include <sys/vfs_opreg.h>
#include <sys/vnode.h>
#include <sys/mode.h>
#include <sys/signal.h>
#include <sys/user.h>
#include <sys/mount.h>
#include <sys/bitmap.h>
#include <sys/kmem.h>
#include <sys/policy.h>
#include <sys/modctl.h>
#include <sys/sunddi.h>
#include <sys/sunldi.h>
#include <sys/lx_impl.h>

#include "lx_proc.h"

/* Module level parameters */
```



```
static int      lxprocfstype;
static dev_t    lxprocdev;
static kmutex_t lxpr_mount_lock;

int nproc_highbit;      /* highbit(v.v_nproc) */

static int lxpr_mount(vfs_t *, vnode_t *, mounta_t *, cred_t *);
static int lxpr_unmount(vfs_t *, int, cred_t *);
static int lxpr_root(vfs_t *, vnode_t **);
static int lxpr_statvfs(vfs_t *, statvfs64_t *);
static int lxpr_init(int, char *);

static vfsdef_t vfw = {
        VFSDEF_VERSION,
        "lx_proc",
        lxpr_init,
        VSW_ZMOUNT,
        NULL
};

/*
 * Module linkage information for the kernel.
 */
extern struct mod_ops mod_fsops;

static struct modlfs modlfs = {
        &mod_fsops, "generic linux procfs", &vfw
};

static struct modlinkage modlinkage = {
        MODREV_1, (void *)&modlfs, NULL
};

int
_init(void)
{
        return (mod_install(&modlinkage));
}

int
_info(struct modinfo *modinfop)
{
        return (mod_info(&modlinkage, modinfop));
}

int
_fini(void)
{
        int retval;

        /*
         * attempt to unload the module
         */
        if ((retval = mod_remove(&modlinkage)) != 0)
                goto done;

        /*
         * destroy lxpr_node cache
         */
        lxpr_fininodecache();

        /*
         * clean out the vfsops and vnodeops
         */
        (void) vfs_freevfsops_by_type(lxprocfstype);
        vn_freevnodeops(lxpr_vnodeops);
```

```
129         mutex_destroy(&lxpr_mount_lock);
130 done:
131         return (retval);
132 }

134 static int
135 lxpr_init(int fstype, char *name)
136 {
137         static const fs_operation_def_t lxpr_vfsops_template[] = {
138                 VFSNAME_MOUNT,          { .vfs_mount = lxpr_mount },
139                 VFSNAME_UNMOUNT,        { .vfs_unmount = lxpr_unmount },
140                 VFSNAME_ROOT,           { .vfs_root = lxpr_root },
141                 VFSNAME_STATVFS,        { .vfs_statvfs = lxpr_statvfs },
142                 NULL,                   NULL
143         };
144         extern const fs_operation_def_t lxpr_vnodeops_template[];
145         int error;
146         major_t dev;

148         nproc_highbit = highbit(v.v_proc);
149         lxprocfstype = fstype;
150         ASSERT(lxprocfstype != 0);

152         mutex_init(&lxpr_mount_lock, NULL, MUTEX_DEFAULT, NULL);

154         /*
155          * Associate VFS ops vector with this fstype.
156          */
157         error = vfs_setfsops(fstype, lxpr_vfsops_template, NULL);
158         if (error != 0) {
159                 cmn_err(CE_WARN, "lxpr_init: bad vfs ops template");
160                 return (error);
161         }

163         /*
164          * Set up vnode ops vector too.
165          */
166         error = vn_make_ops(name, lxpr_vnodeops_template, &lxpr_vnodeops);
167         if (error != 0) {
168                 (void) vfs_freevfsops_by_type(fstype);
169                 cmn_err(CE_WARN, "lxpr_init: bad vnode ops template");
170                 return (error);
171         }

173         /*
174          * Assign a unique "device" number (used by stat(2)).
175          */
176         if ((dev = getudev()) == (major_t)-1) {
177                 cmn_err(CE_WARN, "lxpr_init: can't get unique device number");
178                 dev = 0;
179         }

181         /*
182          * Make the pseudo device
183          */
184         lxprocdev = makedevice(dev, 0);

186         /*
187          * Initialise cache for lxpr_nodes
188          */
189         lxpr_initnodecache();

191         return (0);
192 }
```



```
194 static int
195 lxpr_mount(vfs_t *vfsp, vnode_t *mvp, mounta_t *uap, cred_t *cr)
196 {
197         lxpr_mnt_t *lxpr_mnt;
198         zone_t *zone = curproc->p_zone;
199         ldi_ident_t li;
200         int err;

202         /*
203          * must be root to mount
204          */
205         if (secpolicy_fs_mount(cr, mvp, vfsp) != 0)
206                 return (EPERM);

208         /*
209          * mount point must be a directory
210          */
211         if (mvp->v_type != VDIR)
212                 return (ENOTDIR);

214         if (zone == global_zone) {
215                 zone_t *mntzone;

217                 mntzone = zone_find_by_path(refstr_value(vfsp->vfs_mntpt));
218                 zone_rele(mntzone);
219                 if (zone != mntzone)
220                         return (EBUSY);
221         }

223         /*
224          * Having the resource be anything but "lxproc" doesn't make sense
225          */
226         vfs_setresource(vfsp, "lxproc",0);

228         lxpr_mnt = kmem_alloc(sizeof (*lxpr_mnt), KM_SLEEP);

230         if ((err = ldi_ident_from_mod(&modlinkage, &li)) != 0) {
231                 kmem_free(lxpr_mnt, sizeof (*lxpr_mnt));
232                 return (err);
233         }

235         lxpr_mnt->lxprm_li = li;

237         mutex_enter(&lxpr_mount_lock);

239         /*
240          * Ensure we don't allow overlaying mounts
241          */
242         mutex_enter(&mvp->v_lock);
243         if ((uap->flags & MS_OVERLAY) == 0 &&
244             (mvp->v_count > 1 || (mvp->v_flag & VROOT))) {
245                 mutex_exit(&mvp->v_lock);
246                 mutex_exit(&lxpr_mount_lock);
247                 kmem_free(lxpr_mnt, sizeof ((*lxpr_mnt)));
248                 return (EBUSY);
249         }
250         mutex_exit(&mvp->v_lock);

252         /*
253          * allocate the first vnode
254          */
255         zone_hold(lxpr_mnt->lxprm_zone = zone);

257         /* Arbitrarily set the parent vnode to the mounted over directory */
258         lxpr_mnt->lxprm_node = lxpr_getnode(mvp, LXPR_PROCDIR, NULL, 0);
```

```
260         /* Correctly set the fs for the root node */
261         lxpr_mnt->lxprm_node->lxpr_vnode->v_vfsp = vfsp;

263         vfs_make_fsid(&vfsp->vfs_fsid, lxprocdev, lxprocfstype);
264         vfsp->vfs_bsize = DEV_BSIZE;
265         vfsp->vfs_fstype = lxprocfstype;
266         vfsp->vfs_data = (caddr_t)lxpr_mnt;
267         vfsp->vfs_dev = lxprocdev;

269         mutex_exit(&lxpr_mount_lock);

271         return (0);
272 }

274 static int
275 lxpr_unmount(vfs_t *vfsp, int flag, cred_t *cr)
276 {
277         lxpr_mnt_t *lxpr_mnt = (lxpr_mnt_t *)vfsp->vfs_data;
278         vnode_t *vp;
279         int count;

281         ASSERT(lxpr_mnt != NULL);
282         vp = LXPTOV(lxpr_mnt->lxprm_node);

284         mutex_enter(&lxpr_mount_lock);

286         /*
287          * must be root to unmount
288          */
289         if (secpolicy_fs_unmount(cr, vfsp) != 0) {
290                 mutex_exit(&lxpr_mount_lock);
291                 return (EPERM);
292         }

294         /*
295          * forced unmount is not supported by this file system
296          */
297         if (flag & MS_FORCE) {
298                 mutex_exit(&lxpr_mount_lock);
299                 return (ENOTSUP);
300         }

302         /*
303          * Ensure that no vnodes are in use on this mount point.
304          */
305         mutex_enter(&vp->v_lock);
306         count = vp->v_count;
307         mutex_exit(&vp->v_lock);
308         if (count > 1) {
309                 mutex_exit(&lxpr_mount_lock);
310                 return (EBUSY);
311         }


314         /*
315          * purge the dnlc cache for vnode entries
316          * associated with this file system
317          */
318         count = dnlc_purge_vfsp(vfsp, 0);

320         /*
321          * free up the lxprnode
322          */
323         lxpr_freenode(lxpr_mnt->lxprm_node);
324         zone_rele(lxpr_mnt->lxprm_zone);
325         kmem_free(lxpr_mnt, sizeof (*lxpr_mnt));
```



```
327         mutex_exit(&lxpr_mount_lock);

329         return (0);
330 }

332 static int
333 lxpr_root(vfs_t *vfsp, vnode_t **vpp)
334 {
335         lxpr_node_t *lxpnp = ((lxpr_mnt_t *)vfsp->vfs_data)->lxprm_node;
336         vnode_t *vp = LXPTOV(lxpnp);

338         VN_HOLD(vp);
339         *vpp = vp;
340         return (0);
341 }

343 static int
344 lxpr_statvfs(vfs_t *vfsp, statvfs64_t *sp)
345 {
346         int n;
347         dev32_t d32;
348         extern uint_t nproc;

350         n = v.v_proc - nproc;

352         bzero((caddr_t)sp, sizeof (*sp));
353         sp->f_bsize     = DEV_BSIZE;
354         sp->f_frsize    = DEV_BSIZE;
355         sp->f_blocks    = (fsblkcnt64_t)0;
356         sp->f_bfree     = (fsblkcnt64_t)0;
357         sp->f_bavail    = (fsblkcnt64_t)0;
358         sp->f_files     = (fsfilcnt64_t)v.v_proc + 2;
359         sp->f_ffree     = (fsfilcnt64_t)n;
360         sp->f_favail    = (fsfilcnt64_t)n;
361         (void) cmpldev(&d32, vfsp->vfs_dev);
362         sp->f_fsid      = d32;
363         /* It is guaranteed that vsw_name will fit in f_basetype */
364         (void) strcpy(sp->f_basetype, vfssw[lxprocfstype].vsw_name);
365         sp->f_flag = vf_to_stf(vfsp->vfs_flag);
366         sp->f_namemax = 64;             /* quite arbitrary */
367         bzero(sp->f_fstr, sizeof (sp->f_fstr));

369         /* We know f_fstr is 32 chars */
370         (void) strcpy(sp->f_fstr, "/proc");
371         (void) strcpy(&sp->f_fstr[6], "/proc");

373         return (0);
374 }
375 #endif /* ! codereview */
```

```
**********************************************************
   74808 Tue Jan 14 16:17:20 2014
new/usr/src/uts/common/brand/lx/procfs/lx_prvnops.c
LX zone support should now build and packages of relevance produced.
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * lxpr_vnops.c:  Vnode operations for the lx /proc file system
 *
 * Assumptions and Gotchas:
 *
 * In order to preserve Solaris' security policy. This file system's
 * functionality does not override Solaris' security policies even if
 * that means breaking Linux compatibility.
 *
 * Linux has no concept of lwps so we only implement procs here as in the
 * old /proc interface.
 */

#include <sys/cpupart.h>
#include <sys/cpuvar.h>
#include <sys/session.h>
#include <sys/vmparam.h>
#include <sys/mman.h>
#include <vm/rm.h>
#include <vm/seg_vn.h>
#include <sys/sdt.h>
#include <lx_signum.h>
#include <sys/strlog.h>
#include <sys/stropts.h>
#include <sys/cmn_err.h>
#include <sys/lx_brand.h>
#include <sys/x86_archext.h>
#include <sys/archsystm.h>
#include <sys/fp.h>
#include <sys/pool_pset.h>
#include <sys/pset.h>
#include <sys/zone.h>
#include <sys/pghw.h>
#include <sys/vfs_opreg.h>
```



```
/* Dependent on the Solaris procfs */
extern kthread_t *prchoose(proc_t *);

#include "lx_proc.h"

extern pgcnt_t swapfs_minfree;
extern time_t boot_time;

/*
 * Pointer to the vnode ops vector for this fs.
 * This is instantiated in lxprinit() in lxpr_vfsops.c
 */
vnodeops_t *lxpr_vnodeops;

static int lxpr_open(vnode_t **, int, cred_t *, caller_context_t *);
static int lxpr_close(vnode_t *, int, int, offset_t, cred_t *,
    caller_context_t *);
static int lxpr_read(vnode_t *, uio_t *, int, cred_t *, caller_context_t *);
static int lxpr_getattr(vnode_t *, vattr_t *, int, cred_t *,
    caller_context_t *);
static int lxpr_access(vnode_t *, int, int, cred_t *, caller_context_t *);
static int lxpr_lookup(vnode_t *, char *, vnode_t **,
    pathname_t *, int, vnode_t *, cred_t *, caller_context_t *, int *,
    pathname_t *);
static int lxpr_readdir(vnode_t *, uio_t *, cred_t *, int *,
    caller_context_t *, int);
static int lxpr_readlink(vnode_t *, uio_t *, cred_t *, caller_context_t *);
static int lxpr_cmp(vnode_t *, vnode_t *, caller_context_t *);
static int lxpr_realvp(vnode_t *, vnode_t **, caller_context_t *);
static int lxpr_sync(void);
static void lxpr_inactive(vnode_t *, cred_t *, caller_context_t *);

static vnode_t *lxpr_lookup_procdir(vnode_t *, char *);
static vnode_t *lxpr_lookup_piddir(vnode_t *, char *);
static vnode_t *lxpr_lookup_not_a_dir(vnode_t *, char *);
static vnode_t *lxpr_lookup_fddir(vnode_t *, char *);
static vnode_t *lxpr_lookup_netdir(vnode_t *, char *);

static int lxpr_readdir_procdir(lxpr_node_t *, uio_t *, int *);
static int lxpr_readdir_piddir(lxpr_node_t *, uio_t *, int *);
static int lxpr_readdir_not_a_dir(lxpr_node_t *, uio_t *, int *);
static int lxpr_readdir_fddir(lxpr_node_t *, uio_t *, int *);
static int lxpr_readdir_netdir(lxpr_node_t *, uio_t *, int *);

static void lxpr_read_invalid(lxpr_node_t *, lxpr_uiobuf_t *);
static void lxpr_read_empty(lxpr_node_t *, lxpr_uiobuf_t *);
static void lxpr_read_cpuinfo(lxpr_node_t *, lxpr_uiobuf_t *);
static void lxpr_read_isdir(lxpr_node_t *, lxpr_uiobuf_t *);
static void lxpr_read_fd(lxpr_node_t *, lxpr_uiobuf_t *);
static void lxpr_read_kmsg(lxpr_node_t *, lxpr_uiobuf_t *);
static void lxpr_read_loadavg(lxpr_node_t *, lxpr_uiobuf_t *);
static void lxpr_read_meminfo(lxpr_node_t *, lxpr_uiobuf_t *);
static void lxpr_read_mounts(lxpr_node_t *, lxpr_uiobuf_t *);
static void lxpr_read_partitions(lxpr_node_t *, lxpr_uiobuf_t *);
static void lxpr_read_stat(lxpr_node_t *, lxpr_uiobuf_t *);
static void lxpr_read_uptime(lxpr_node_t *, lxpr_uiobuf_t *);
static void lxpr_read_version(lxpr_node_t *, lxpr_uiobuf_t *);

static void lxpr_read_pid_cmdline(lxpr_node_t *, lxpr_uiobuf_t *);
static void lxpr_read_pid_maps(lxpr_node_t *, lxpr_uiobuf_t *);
static void lxpr_read_pid_stat(lxpr_node_t *, lxpr_uiobuf_t *);
static void lxpr_read_pid_statm(lxpr_node_t *, lxpr_uiobuf_t *);
static void lxpr_read_pid_status(lxpr_node_t *, lxpr_uiobuf_t *);

static void lxpr_read_net_arp(lxpr_node_t *, lxpr_uiobuf_t *);
static void lxpr_read_net_dev(lxpr_node_t *, lxpr_uiobuf_t *);
```

```
127 static void lxpr_read_net_dev_mcast(lxpr_node_t *, lxpr_uiobuf_t *);
128 static void lxpr_read_net_igmp(lxpr_node_t *, lxpr_uiobuf_t *);
129 static void lxpr_read_net_ip_mr_cache(lxpr_node_t *, lxpr_uiobuf_t *);
130 static void lxpr_read_net_ip_mr_vif(lxpr_node_t *, lxpr_uiobuf_t *);
131 static void lxpr_read_net_mcfilter(lxpr_node_t *, lxpr_uiobuf_t *);
132 static void lxpr_read_net_netstat(lxpr_node_t *, lxpr_uiobuf_t *);
133 static void lxpr_read_net_raw(lxpr_node_t *, lxpr_uiobuf_t *);
134 static void lxpr_read_net_route(lxpr_node_t *, lxpr_uiobuf_t *);
135 static void lxpr_read_net_rpc(lxpr_node_t *, lxpr_uiobuf_t *);
136 static void lxpr_read_net_rt_cache(lxpr_node_t *, lxpr_uiobuf_t *);
137 static void lxpr_read_net_sockstat(lxpr_node_t *, lxpr_uiobuf_t *);
138 static void lxpr_read_net_snmp(lxpr_node_t *, lxpr_uiobuf_t *);
139 static void lxpr_read_net_stat(lxpr_node_t *, lxpr_uiobuf_t *);
140 static void lxpr_read_net_tcp(lxpr_node_t *, lxpr_uiobuf_t *);
141 static void lxpr_read_net_udp(lxpr_node_t *, lxpr_uiobuf_t *);
142 static void lxpr_read_net_unix(lxpr_node_t *, lxpr_uiobuf_t *);

144 /*
145  * Simple conversion
146  */
147 #define btok(x) ((x) >> 10)                     /* bytes to kbytes */
148 #define ptok(x) ((x) << (PAGESHIFT - 10))       /* pages to kbytes */

150 /*
151  * The lx /proc vnode operations vector
152  */
153 const fs_operation_def_t lxpr_vnodeops_template[] = {
154         VOPNAME_OPEN,           { .vop_open = lxpr_open },
155         VOPNAME_CLOSE,          { .vop_close = lxpr_close },
156         VOPNAME_READ,           { .vop_read = lxpr_read },
157         VOPNAME_GETATTR,        { .vop_getattr = lxpr_getattr },
158         VOPNAME_ACCESS,         { .vop_access = lxpr_access },
159         VOPNAME_LOOKUP,         { .vop_lookup = lxpr_lookup },
160         VOPNAME_READDIR,        { .vop_readdir = lxpr_readdir },
161         VOPNAME_READLINK,       { .vop_readlink = lxpr_readlink },
162         VOPNAME_FSYNC,          { .error = lxpr_sync },
163         VOPNAME_SEEK,           { .error = lxpr_sync },
164         VOPNAME_INACTIVE,       { .vop_inactive = lxpr_inactive },
165         VOPNAME_CMP,            { .vop_cmp = lxpr_cmp },
166         VOPNAME_REALVP,         { .vop_realvp = lxpr_realvp },
167         NULL,                   NULL
168 };


171 /*
172  * file contents of an lx /proc directory.
173  */
174 static lxpr_dirent_t lx_procdir[] = {
175         { LXPR_CMDLINE,         "cmdline" },
176         { LXPR_CPUINFO,         "cpuinfo" },
177         { LXPR_DEVICES,         "devices" },
178         { LXPR_DMA,             "dma" },
179         { LXPR_FILESYSTEMS,     "filesystems" },
180         { LXPR_INTERRUPTS,      "interrupts" },
181         { LXPR_IOPORTS,         "ioports" },
182         { LXPR_KCORE,           "kcore" },
183         { LXPR_KMSG,            "kmsg" },
184         { LXPR_LOADAVG,         "loadavg" },
185         { LXPR_MEMINFO,         "meminfo" },
186         { LXPR_MOUNTS,          "mounts" },
187         { LXPR_NETDIR,          "net" },
188         { LXPR_PARTITIONS,      "partitions" },
189         { LXPR_SELF,            "self" },
190         { LXPR_STAT,            "stat" },
191         { LXPR_UPTIME,          "uptime" },
192         { LXPR_VERSION,         "version" }
```



```
193 };

195 #define PROCDIRFILES    (sizeof (lx_procdir) / sizeof (lx_procdir[0]))

197 /*
198  * Contents of an lx /proc/<pid> directory.
199  */
200 static lxpr_dirent_t piddir[] = {
201         { LXPR_PID_CMDLINE,     "cmdline" },
202         { LXPR_PID_CPU,         "cpu" },
203         { LXPR_PID_CURDIR,      "cwd" },
204         { LXPR_PID_ENV,         "environ" },
205         { LXPR_PID_EXE,         "exe" },
206         { LXPR_PID_MAPS,        "maps" },
207         { LXPR_PID_MEM,         "mem" },
208         { LXPR_PID_ROOTDIR,     "root" },
209         { LXPR_PID_STAT,        "stat" },
210         { LXPR_PID_STATM,       "statm" },
211         { LXPR_PID_STATUS,      "status" },
212         { LXPR_PID_FDDIR,       "fd" }
213 };

215 #define PIDDIRFILES     (sizeof (piddir) / sizeof (piddir[0]))

217 /*
218  * contents of lx /proc/net directory
219  */
220 static lxpr_dirent_t netdir[] = {
221         { LXPR_NET_ARP,         "arp" },
222         { LXPR_NET_DEV,         "dev" },
223         { LXPR_NET_DEV_MCAST,   "dev_mcast" },
224         { LXPR_NET_IGMP,        "igmp" },
225         { LXPR_NET_IP_MR_CACHE, "ip_mr_cache" },
226         { LXPR_NET_IP_MR_VIF,   "ip_mr_vif" },
227         { LXPR_NET_MCFILTER,    "mcfilter" },
228         { LXPR_NET_NETSTAT,     "netstat" },
229         { LXPR_NET_RAW,         "raw" },
230         { LXPR_NET_ROUTE,       "route" },
231         { LXPR_NET_RPC,         "rpc" },
232         { LXPR_NET_RT_CACHE,    "rt_cache" },
233         { LXPR_NET_SOCKSTAT,    "sockstat" },
234         { LXPR_NET_SNMP,        "snmp" },
235         { LXPR_NET_STAT,        "stat" },
236         { LXPR_NET_TCP,         "tcp" },
237         { LXPR_NET_UDP,         "udp" },
238         { LXPR_NET_UNIX,        "unix" }
239 };

241 #define NETDIRFILES     (sizeof (netdir) / sizeof (netdir[0]))

243 /*
244  * lxpr_open(): Vnode operation for VOP_OPEN()
245  */
246 static int
247 lxpr_open(vnode_t **vpp, int flag, cred_t *cr, caller_context_t *ct)
248 {
249         vnode_t         *vp = *vpp;
250         lxpr_node_t     *lxpnp = VTOLXP(vp);
251         lxpr_nodetype_t type = lxpnp->lxpr_type;
252         vnode_t         *rvp;
253         int             error = 0;

255         /*
256          * We only allow reading in this file systrem
257          */
258         if (flag & FWRITE)
```

```
259                 return (EROFS);

261         /*
262          * If we are opening an underlying file only allow regular files
263          * reject the open for anything but a regular file.
264          * Just do it if we are opening the current or root directory.
265          */
266         if (lxpnp->lxpr_realvp != NULL) {
267                 rvp = lxpnp->lxpr_realvp;

269                 if (type == LXPR_PID_FD_FD && rvp->v_type != VREG)
270                         error = EACCES;
271                 else {
272                         /*
273                          * Need to hold rvp since VOP_OPEN() may release it.
274                          */
275                         VN_HOLD(rvp);
276                         error = VOP_OPEN(&rvp, flag, cr, ct);
277                         if (error) {
278                                 VN_RELE(rvp);
279                         } else {
280                                 *vpp = rvp;
281                                 VN_RELE(vp);
282                         }
283                 }
284         }

286         if (type == LXPR_KMSG) {
287                 ldi_ident_t     li = VTOLXPM(vp)->lxprm_li;
288                 struct strioctl str;
289                 int             rv;

291                 /*
292                  * Open the zone's console device using the layered driver
293                  * interface.
294                  */
295                 if ((error = ldi_open_by_name("/dev/log", FREAD, cr,
296                     &lxpnp->lxpr_cons_ldih, li)) != 0)
297                         return (error);

299                 /*
300                  * Send an ioctl to the underlying console device, letting it
301                  * know we're interested in getting console messages.
302                  */
303                 str.ic_cmd = I_CONSLOG;
304                 str.ic_timout = 0;
305                 str.ic_len = 0;
306                 str.ic_dp = NULL;
307                 if ((error = ldi_ioctl(lxpnp->lxpr_cons_ldih, I_STR,
308                     (intptr_t)&str, FKIOCTL, cr, &rv)) != 0)
309                         return (error);
310         }

312         return (error);
313 }


316 /*
317  * lxpr_close(): Vnode operation for VOP_CLOSE()
318  */
319 /* ARGSUSED */
320 static int
321 lxpr_close(vnode_t *vp, int flag, int count, offset_t offset, cred_t *cr,
322     caller_context_t *ct)
323 {
324         lxpr_node_t     *lxpr = VTOLXP(vp);
```



```
325         lxpr_nodetype_t type = lxpr->lxpr_type;
326         int             err;

328         /*
329          * we should never get here because the close is done on the realvp
330          * for these nodes
331          */
332         ASSERT(type != LXPR_PID_FD_FD &&
333             type != LXPR_PID_CURDIR &&
334             type != LXPR_PID_ROOTDIR &&
335             type != LXPR_PID_EXE);

337         if (type == LXPR_KMSG) {
338                 if ((err = ldi_close(lxpr->lxpr_cons_ldih, 0, cr)) != 0)
339                         return (err);
340         }

342         return (0);
343 }

345 static void (*lxpr_read_function[LXPR_NFILES])() = {
346         lxpr_read_isdir,                /* /proc                */
347         lxpr_read_isdir,                /* /proc/<pid>          */
348         lxpr_read_pid_cmdline,          /* /proc/<pid>/cmdline  */
349         lxpr_read_empty,                /* /proc/<pid>/cpu      */
350         lxpr_read_invalid,              /* /proc/<pid>/cwd      */
351         lxpr_read_empty,                /* /proc/<pid>/environ  */
352         lxpr_read_invalid,              /* /proc/<pid>/exe      */
353         lxpr_read_pid_maps,             /* /proc/<pid>/maps     */
354         lxpr_read_empty,                /* /proc/<pid>/mem      */
355         lxpr_read_invalid,              /* /proc/<pid>/root     */
356         lxpr_read_pid_stat,             /* /proc/<pid>/stat     */
357         lxpr_read_pid_statm,            /* /proc/<pid>/statm    */
358         lxpr_read_pid_status,           /* /proc/<pid>/status   */
359         lxpr_read_isdir,                /* /proc/<pid>/fd       */
360         lxpr_read_fd,                   /* /proc/<pid>/fd/nn    */
361         lxpr_read_empty,                /* /proc/cmdline        */
362         lxpr_read_cpuinfo,              /* /proc/cpuinfo        */
363         lxpr_read_empty,                /* /proc/devices        */
364         lxpr_read_empty,                /* /proc/dma            */
365         lxpr_read_empty,                /* /proc/filesystems    */
366         lxpr_read_empty,                /* /proc/interrupts     */
367         lxpr_read_empty,                /* /proc/ioports        */
368         lxpr_read_empty,                /* /proc/kcore          */
369         lxpr_read_kmsg,                 /* /proc/kmsg           */
370         lxpr_read_loadavg,              /* /proc/loadavg        */
371         lxpr_read_meminfo,              /* /proc/meminfo        */
372         lxpr_read_mounts,               /* /proc/mounts         */
373         lxpr_read_isdir,                /* /proc/net            */
374         lxpr_read_net_arp,              /* /proc/net/arp        */
375         lxpr_read_net_dev,              /* /proc/net/dev        */
376         lxpr_read_net_dev_mcast,        /* /proc/net/dev_mcast  */
377         lxpr_read_net_igmp,             /* /proc/net/igmp       */
378         lxpr_read_net_ip_mr_cache,      /* /proc/net/ip_mr_cache */
379         lxpr_read_net_ip_mr_vif,        /* /proc/net/ip_mr_vif  */
380         lxpr_read_net_mcfilter,         /* /proc/net/mcfilter   */
381         lxpr_read_net_netstat,          /* /proc/net/netstat    */
382         lxpr_read_net_raw,              /* /proc/net/raw        */
383         lxpr_read_net_route,            /* /proc/net/route      */
384         lxpr_read_net_rpc,              /* /proc/net/rpc        */
385         lxpr_read_net_rt_cache,         /* /proc/net/rt_cache   */
386         lxpr_read_net_sockstat,         /* /proc/net/sockstat   */
387         lxpr_read_net_snmp,             /* /proc/net/snmp       */
388         lxpr_read_net_stat,             /* /proc/net/stat       */
389         lxpr_read_net_tcp,              /* /proc/net/tcp        */
390         lxpr_read_net_udp,              /* /proc/net/udp        */
```

```
391         lxpr_read_net_unix,             /* /proc/net/unix       */
392         lxpr_read_partitions,           /* /proc/partitions     */
393         lxpr_read_invalid,              /* /proc/self           */
394         lxpr_read_stat,                 /* /proc/stat           */
395         lxpr_read_uptime,               /* /proc/uptime         */
396         lxpr_read_version,              /* /proc/version        */
397 };

399 /*
400  * Array of lookup functions, indexed by lx /proc file type.
401  */
402 static vnode_t *(*lxpr_lookup_function[LXPR_NFILES])() = {
403         lxpr_lookup_procdir,            /* /proc                */
404         lxpr_lookup_piddir,             /* /proc/<pid>          */
405         lxpr_lookup_not_a_dir,          /* /proc/<pid>/cmdline  */
406         lxpr_lookup_not_a_dir,          /* /proc/<pid>/cpu      */
407         lxpr_lookup_not_a_dir,          /* /proc/<pid>/cwd      */
408         lxpr_lookup_not_a_dir,          /* /proc/<pid>/environ  */
409         lxpr_lookup_not_a_dir,          /* /proc/<pid>/exe      */
410         lxpr_lookup_not_a_dir,          /* /proc/<pid>/maps     */
411         lxpr_lookup_not_a_dir,          /* /proc/<pid>/mem      */
412         lxpr_lookup_not_a_dir,          /* /proc/<pid>/root     */
413         lxpr_lookup_not_a_dir,          /* /proc/<pid>/stat     */
414         lxpr_lookup_not_a_dir,          /* /proc/<pid>/statm    */
415         lxpr_lookup_not_a_dir,          /* /proc/<pid>/status   */
416         lxpr_lookup_fddir,              /* /proc/<pid>/fd       */
417         lxpr_lookup_not_a_dir,          /* /proc/<pid>/fd/nn    */
418         lxpr_lookup_not_a_dir,          /* /proc/cmdline        */
419         lxpr_lookup_not_a_dir,          /* /proc/cpuinfo        */
420         lxpr_lookup_not_a_dir,          /* /proc/devices        */
421         lxpr_lookup_not_a_dir,          /* /proc/dma            */
422         lxpr_lookup_not_a_dir,          /* /proc/filesystems    */
423         lxpr_lookup_not_a_dir,          /* /proc/interrupts     */
424         lxpr_lookup_not_a_dir,          /* /proc/ioports        */
425         lxpr_lookup_not_a_dir,          /* /proc/kcore          */
426         lxpr_lookup_not_a_dir,          /* /proc/kmsg           */
427         lxpr_lookup_not_a_dir,          /* /proc/loadavg        */
428         lxpr_lookup_not_a_dir,          /* /proc/meminfo        */
429         lxpr_lookup_not_a_dir,          /* /proc/mounts         */
430         lxpr_lookup_netdir,             /* /proc/net            */
431         lxpr_lookup_not_a_dir,          /* /proc/net/arp        */
432         lxpr_lookup_not_a_dir,          /* /proc/net/dev        */
433         lxpr_lookup_not_a_dir,          /* /proc/net/dev_mcast  */
434         lxpr_lookup_not_a_dir,          /* /proc/net/igmp       */
435         lxpr_lookup_not_a_dir,          /* /proc/net/ip_mr_cache */
436         lxpr_lookup_not_a_dir,          /* /proc/net/ip_mr_vif  */
437         lxpr_lookup_not_a_dir,          /* /proc/net/mcfilter   */
438         lxpr_lookup_not_a_dir,          /* /proc/net/netstat    */
439         lxpr_lookup_not_a_dir,          /* /proc/net/raw        */
440         lxpr_lookup_not_a_dir,          /* /proc/net/route      */
441         lxpr_lookup_not_a_dir,          /* /proc/net/rpc        */
442         lxpr_lookup_not_a_dir,          /* /proc/net/rt_cache   */
443         lxpr_lookup_not_a_dir,          /* /proc/net/sockstat   */
444         lxpr_lookup_not_a_dir,          /* /proc/net/snmp       */
445         lxpr_lookup_not_a_dir,          /* /proc/net/stat       */
446         lxpr_lookup_not_a_dir,          /* /proc/net/tcp        */
447         lxpr_lookup_not_a_dir,          /* /proc/net/udp        */
448         lxpr_lookup_not_a_dir,          /* /proc/net/unix       */
449         lxpr_lookup_not_a_dir,          /* /proc/partitions     */
450         lxpr_lookup_not_a_dir,          /* /proc/self           */
451         lxpr_lookup_not_a_dir,          /* /proc/stat           */
452         lxpr_lookup_not_a_dir,          /* /proc/uptime         */
453         lxpr_lookup_not_a_dir,          /* /proc/version        */
454 };

456 /*
```



```
457  * Array of readdir functions, indexed by /proc file type.
458  */
459 static int (*lxpr_readdir_function[LXPR_NFILES])() = {
460         lxpr_readdir_procdir,           /* /proc                */
461         lxpr_readdir_piddir,            /* /proc/<pid>          */
462         lxpr_readdir_not_a_dir,         /* /proc/<pid>/cmdline  */
463         lxpr_readdir_not_a_dir,         /* /proc/<pid>/cpu      */
464         lxpr_readdir_not_a_dir,         /* /proc/<pid>/cwd      */
465         lxpr_readdir_not_a_dir,         /* /proc/<pid>/environ  */
466         lxpr_readdir_not_a_dir,         /* /proc/<pid>/exe      */
467         lxpr_readdir_not_a_dir,         /* /proc/<pid>/maps     */
468         lxpr_readdir_not_a_dir,         /* /proc/<pid>/mem      */
469         lxpr_readdir_not_a_dir,         /* /proc/<pid>/root     */
470         lxpr_readdir_not_a_dir,         /* /proc/<pid>/stat     */
471         lxpr_readdir_not_a_dir,         /* /proc/<pid>/statm    */
472         lxpr_readdir_not_a_dir,         /* /proc/<pid>/status   */
473         lxpr_readdir_fddir,             /* /proc/<pid>/fd       */
474         lxpr_readdir_not_a_dir,         /* /proc/<pid>/fd/nn    */
475         lxpr_readdir_not_a_dir,         /* /proc/cmdline        */
476         lxpr_readdir_not_a_dir,         /* /proc/cpuinfo        */
477         lxpr_readdir_not_a_dir,         /* /proc/devices        */
478         lxpr_readdir_not_a_dir,         /* /proc/dma            */
479         lxpr_readdir_not_a_dir,         /* /proc/filesystems    */
480         lxpr_readdir_not_a_dir,         /* /proc/interrupts     */
481         lxpr_readdir_not_a_dir,         /* /proc/ioports        */
482         lxpr_readdir_not_a_dir,         /* /proc/kcore          */
483         lxpr_readdir_not_a_dir,         /* /proc/kmsg           */
484         lxpr_readdir_not_a_dir,         /* /proc/loadavg        */
485         lxpr_readdir_not_a_dir,         /* /proc/meminfo        */
486         lxpr_readdir_not_a_dir,         /* /proc/mounts         */
487         lxpr_readdir_netdir,            /* /proc/net            */
488         lxpr_readdir_not_a_dir,         /* /proc/net/arp        */
489         lxpr_readdir_not_a_dir,         /* /proc/net/dev        */
490         lxpr_readdir_not_a_dir,         /* /proc/net/dev_mcast  */
491         lxpr_readdir_not_a_dir,         /* /proc/net/igmp       */
492         lxpr_readdir_not_a_dir,         /* /proc/net/ip_mr_cache */
493         lxpr_readdir_not_a_dir,         /* /proc/net/ip_mr_vif  */
494         lxpr_readdir_not_a_dir,         /* /proc/net/mcfilter   */
495         lxpr_readdir_not_a_dir,         /* /proc/net/netstat    */
496         lxpr_readdir_not_a_dir,         /* /proc/net/raw        */
497         lxpr_readdir_not_a_dir,         /* /proc/net/route      */
498         lxpr_readdir_not_a_dir,         /* /proc/net/rpc        */
499         lxpr_readdir_not_a_dir,         /* /proc/net/rt_cache   */
500         lxpr_readdir_not_a_dir,         /* /proc/net/sockstat   */
501         lxpr_readdir_not_a_dir,         /* /proc/net/snmp       */
502         lxpr_readdir_not_a_dir,         /* /proc/net/stat       */
503         lxpr_readdir_not_a_dir,         /* /proc/net/tcp        */
504         lxpr_readdir_not_a_dir,         /* /proc/net/udp        */
505         lxpr_readdir_not_a_dir,         /* /proc/net/unix       */
506         lxpr_readdir_not_a_dir,         /* /proc/partitions     */
507         lxpr_readdir_not_a_dir,         /* /proc/self           */
508         lxpr_readdir_not_a_dir,         /* /proc/stat           */
509         lxpr_readdir_not_a_dir,         /* /proc/uptime         */
510         lxpr_readdir_not_a_dir,         /* /proc/version        */
511 };


514 /*
515  * lxpr_read(): Vnode operation for VOP_READ()
516  *
517  * As the format of all the files that can be read in the lx procfs is human
518  * readable and not binary structures there do not have to be different
519  * read variants depending on whether the reading process model is 32 or 64 bits
520  * (at least in general, and certainly the difference is unlikely to be enough
521  * to justify have different routines for 32 and 64 bit reads
522  */
```

```
523 /* ARGSUSED */
524 static int
525 lxpr_read(vnode_t *vp, uio_t *uiop, int ioflag, cred_t *cr,
526     caller_context_t *ct)
527 {
528         lxpr_node_t *lxpnp = VTOLXP(vp);
529         lxpr_nodetype_t type = lxpnp->lxpr_type;
530         lxpr_uiobuf_t *uiobuf = lxpr_uiobuf_new(uiop);
531         int error;
533         ASSERT(type < LXPR_NFILES);
535         lxpr_read_function[type](lxpnp, uiobuf);
537         error = lxpr_uiobuf_flush(uiobuf);
538         lxpr_uiobuf_free(uiobuf);
540         return (error);
541 }


544 /*
545  * lxpr_read_invalid(), lxpr_read_isdir(), lxpr_read_empty()
546  *
547  * Various special case reads:
548  * - trying to read a directory
549  * - invalid file (used to mean a file that should be implemented,
550  *   but isn't yet)
551  * - empty file
552  * - wait to be able to read a file that will never have anything to read
553  */
554 /* ARGSUSED */
555 static void
556 lxpr_read_isdir(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
557 {
558         lxpr_uiobuf_seterr(uiobuf, EISDIR);
559 }
561 /* ARGSUSED */
562 static void
563 lxpr_read_invalid(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
564 {
565         lxpr_uiobuf_seterr(uiobuf, EINVAL);
566 }
568 /* ARGSUSED */
569 static void
570 lxpr_read_empty(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
571 {
572 }
574 /*
575  * lxpr_read_pid_cmdline():
576  *
577  * This is not precisely compatible with linux:
578  *
579  * The linux cmdline returns argv with the correct separation
580  * using \0 between the arguments, we cannot do that without
581  * copying the real argv from the correct process context.
582  * This is too difficult to attempt so we pretend that the
583  * entire cmdline is just argv[0]. This is good enough for
584  * ps to display correctly, but might cause some other things
585  * not to work correctly.
586  */
587 static void
588 lxpr_read_pid_cmdline(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
```



```
589 {
590         proc_t *p;
592         ASSERT(lxpnp->lxpr_type == LXPR_PID_CMDLINE);
594         p = lxpr_lock(lxpnp->lxpr_pid);
595         if (p == NULL) {
596                 lxpr_uiobuf_seterr(uiobuf, EINVAL);
597                 return;
598         }
600         if (PTOU(p)->u_argv != 0) {
601                 char *buff = PTOU(p)->u_psargs;
602                 int len = strlen(buff);
603                 lxpr_unlock(p);
604                 lxpr_uiobuf_write(uiobuf, buff, len+1);
605         } else {
606                 lxpr_unlock(p);
607         }
608 }


611 /*
612  * lxpr_read_pid_maps(): memory map file
613  */
614 static void
615 lxpr_read_pid_maps(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
616 {
617         proc_t *p;
618         struct as *as;
619         struct seg *seg;
620         char *buf;
621         int buflen = MAXPATHLEN;
622         struct print_data {
623                 caddr_t saddr;
624                 caddr_t eaddr;
625                 int type;
626                 char prot[5];
627                 uint32_t offset;
628                 vnode_t *vp;
629                 struct print_data *next;
630         } *print_head = NULL;
631         struct print_data **print_tail = &print_head;
632         struct print_data *pbuf;
634         ASSERT(lxpnp->lxpr_type == LXPR_PID_MAPS);
636         p = lxpr_lock(lxpnp->lxpr_pid);
637         if (p == NULL) {
638                 lxpr_uiobuf_seterr(uiobuf, EINVAL);
639                 return;
640         }
642         as = p->p_as;
644         if (as == &kas) {
645                 lxpr_unlock(p);
646                 return;
647         }
649         mutex_exit(&p->p_lock);
651         /* Iterate over all segments in the address space */
652         AS_LOCK_ENTER(as, &as->a_lock, RW_READER);
653         for (seg = AS_SEGFIRST(as); seg != NULL; seg = AS_SEGNEXT(as, seg)) {
654                 vnode_t *vp;
```

```
655                 uint_t protbits;

657                 pbuf = kmem_alloc(sizeof (*pbuf), KM_SLEEP);

659                 pbuf->saddr = seg->s_base;
660                 pbuf->eaddr = seg->s_base+seg->s_size;
661                 pbuf->type = SEGOP_GETTYPE(seg, seg->s_base);

663                 /*
664                  * Cheat and only use the protection bits of the first page
665                  * in the segment
666                  */
667                 (void) strncpy(pbuf->prot, "----", sizeof (pbuf->prot));
668                 (void) SEGOP_GETPROT(seg, seg->s_base, 0, &protbits);

670                 if (protbits & PROT_READ)          pbuf->prot[0] = 'r';
671                 if (protbits & PROT_WRITE)         pbuf->prot[1] = 'w';
672                 if (protbits & PROT_EXEC)          pbuf->prot[2] = 'x';
673                 if (pbuf->type & MAP_SHARED)       pbuf->prot[3] = 's';
674                 else if (pbuf->type & MAP_PRIVATE) pbuf->prot[3] = 'p';

676                 if (seg->s_ops == &segvn_ops &&
677                     SEGOP_GETVP(seg, seg->s_base, &vp) == 0 &&
678                     vp != NULL && vp->v_type == VREG) {
679                         VN_HOLD(vp);
680                         pbuf->vp = vp;
681                 } else {
682                         pbuf->vp = NULL;
683                 }

685                 pbuf->offset = (uint32_t)SEGOP_GETOFFSET(seg, pbuf->saddr);

687                 pbuf->next = NULL;
688                 *print_tail = pbuf;
689                 print_tail = &pbuf->next;
690         }
691         AS_LOCK_EXIT(as, &as->a_lock);
692         mutex_enter(&p->p_lock);
693         lxpr_unlock(p);

695         buf = kmem_alloc(buflen, KM_SLEEP);

697         /* print the data we've extracted */
698         pbuf = print_head;
699         while (pbuf != NULL) {
700                 struct print_data *pbuf_next;
701                 vattr_t vattr;

703                 int maj = 0;
704                 int min = 0;
705                 int inode = 0;

707                 *buf = '\0';
708                 if (pbuf->vp != NULL) {
709                         vattr.va_mask = AT_FSID | AT_NODEID;
710                         if (VOP_GETATTR(pbuf->vp, &vattr, 0, CRED(),
711                             NULL) == 0) {
712                                 maj = getmajor(vattr.va_fsid);
713                                 min = getminor(vattr.va_fsid);
714                                 inode = vattr.va_nodeid;
715                         }
716                         (void) vnodetopath(NULL, pbuf->vp, buf, buflen, CRED());
717                         VN_RELE(pbuf->vp);
718                 }

720                 if (*buf != '\0') {
```



```
721                         lxpr_uiobuf_printf(uiobuf,
722                             "%08x-%08x %s %08x %02d:%03d %d %s\n",
723                             pbuf->saddr, pbuf->eaddr, pbuf->prot, pbuf->offset,
724                             maj, min, inode, buf);
725                 } else {
726                         lxpr_uiobuf_printf(uiobuf,
727                             "%08x-%08x %s %08x %02d:%03d %d\n",
728                             pbuf->saddr, pbuf->eaddr, pbuf->prot, pbuf->offset,
729                             maj, min, inode);
730                 }

732                 pbuf_next = pbuf->next;
733                 kmem_free(pbuf, sizeof (*pbuf));
734                 pbuf = pbuf_next;
735         }

737         kmem_free(buf, buflen);
738 }

740 /*
741  * lxpr_read_pid_statm(): memory status file
742  */
743 static void
744 lxpr_read_pid_statm(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
745 {
746         proc_t *p;
747         struct as *as;
748         size_t vsize;
749         size_t rss;

751         ASSERT(lxpnp->lxpr_type == LXPR_PID_STATM);

753         p = lxpr_lock(lxpnp->lxpr_pid);
754         if (p == NULL) {
755                 lxpr_uiobuf_seterr(uiobuf, EINVAL);
756                 return;
757         }

759         as = p->p_as;

761         mutex_exit(&p->p_lock);

763         AS_LOCK_ENTER(as, &as->a_lock, RW_READER);
764         vsize = btopr(as->a_resvsize);
765         rss = rm_asrss(as);
766         AS_LOCK_EXIT(as, &as->a_lock);

768         mutex_enter(&p->p_lock);
769         lxpr_unlock(p);

771         lxpr_uiobuf_printf(uiobuf,
772             "%lu %lu %lu %lu %lu %lu %lu\n",
773             vsize, rss, 0l, rss, 0l, 0l, 0l);
774 }

776 /*
777  * lxpr_read_pid_status(): status file
778  */
779 static void
780 lxpr_read_pid_status(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
781 {
782         proc_t *p;
783         kthread_t *t;
784         user_t *up;
785         cred_t *cr;
786         const gid_t *groups;
```

```
787         int    ngroups;
788         struct as *as;
789         char *status;
790         pid_t pid, ppid;
791         size_t vsize;
792         size_t rss;
793         k_sigset_t current, ignore, handle;
794         int    i, lx_sig;

796         ASSERT(lxpnp->lxpr_type == LXPR_PID_STATUS);

798         p = lxpr_lock(lxpnp->lxpr_pid);
799         if (p == NULL) {
800                 lxpr_uiobuf_seterr(uiobuf, EINVAL);
801                 return;
802         }

804         pid = p->p_pid;

806         /*
807          * Convert pid to the Linux default of 1 if we're the zone's init
808          * process
809          */
810         if (pid == curproc->p_zone->zone_proc_initpid) {
811                 pid = 1;
812                 ppid = 0;       /* parent pid for init is 0 */
813         } else {
814                 /*
815                  * Make sure not to reference parent PIDs that reside outside
816                  * the zone
817                  */
818                 ppid = ((p->p_flag & SZONETOP)
819                     ? curproc->p_zone->zone_zsched->p_pid : p->p_ppid);

821                 /*
822                  * Convert ppid to the Linux default of 1 if our parent is the
823                  * zone's init process
824                  */
825                 if (ppid == curproc->p_zone->zone_proc_initpid)
826                         ppid = 1;
827         }

829         t = prchoose(p);
830         if (t != NULL) {
831                 switch (t->t_state) {
832                 case TS_SLEEP:
833                         status = "S (sleeping)";
834                         break;
835                 case TS_RUN:
836                 case TS_ONPROC:
837                         status = "R (running)";
838                         break;
839                 case TS_ZOMB:
840                         status = "Z (zombie)";
841                         break;
842                 case TS_STOPPED:
843                         status = "T (stopped)";
844                         break;
845                 default:
846                         status = "! (unknown)";
847                         break;
848                 }
849                 thread_unlock(t);
850         } else {
851                 /*
852                  * there is a hole in the exit code, where a proc can have
```



```
853                  * no threads but it is yet to be flagged SZOMB. We will
854                  * assume we are about to become a zombie
855                  */
856                 status = "Z (zombie)";
857         }

859         up = PTOU(p);
860         mutex_enter(&p->p_crlock);
861         crhold(cr = p->p_cred);
862         mutex_exit(&p->p_crlock);

864         lxpr_uiobuf_printf(uiobuf,
865             "Name:\t%s\n"
866             "State:\t%s\n"
867             "Tgid:\t%d\n"
868             "Pid:\t%d\n"
869             "PPid:\t%d\n"
870             "TracerPid:\t%d\n"
871             "Uid:\t%u\t%u\t%u\t%u\n"
872             "Gid:\t%u\t%u\t%u\t%u\n"
873             "FDSize:\t%d\n"
874             "Groups:\t",
875             up->u_comm,
876             status,
877             pid, /* thread group id - same as pid until we map lwps to procs */
878             pid,
879             ppid,
880             0,
881             crgetruid(cr), crgetuid(cr), crgetsuid(cr), crgetuid(cr),
882             crgetrgid(cr), crgetgid(cr), crgetsgid(cr), crgetgid(cr),
883             p->p_fno_ctl);

885         ngroups = crgetngroups(cr);
886         groups  = crgetgroups(cr);
887         for (i = 0; i < ngroups; i++) {
888                 lxpr_uiobuf_printf(uiobuf,
889                     "%u ",
890                     groups[i]);
891         }
892         crfree(cr);

894         as = p->p_as;
895         if ((p->p_stat != SZOMB) && !(p->p_flag & SSYS) && (as != &kas)) {
896                 mutex_exit(&p->p_lock);
897                 AS_LOCK_ENTER(as, &as->a_lock, RW_READER);
898                 vsize = as->a_resvsize;
899                 rss = rm_asrss(as);
900                 AS_LOCK_EXIT(as, &as->a_lock);
901                 mutex_enter(&p->p_lock);

903                 lxpr_uiobuf_printf(uiobuf,
904                     "\n"
905                     "VmSize:\t%8lu kB\n"
906                     "VmLck:\t%8lu kB\n"
907                     "VmRSS:\t%8lu kB\n"
908                     "VmData:\t%8lu kB\n"
909                     "VmStk:\t%8lu kB\n"
910                     "VmExe:\t%8lu kB\n"
911                     "VmLib:\t%8lu kB",
912                     btok(vsize),
913                     0l,
914                     ptok(rss),
915                     0l,
916                     btok(p->p_stksize),
917                     ptok(rss),
918                     0l);
```

```
919         }

921         sigemptyset(&current);
922         sigemptyset(&ignore);
923         sigemptyset(&handle);

925         for (i = 1; i < NSIG; i++) {
926                 lx_sig = stol_signo[i];

928                 if ((lx_sig > 0) && (lx_sig < LX_NSIG)) {
929                         if (sigismember(&p->p_sig, i))
930                                 sigaddset(&current, lx_sig);

932                         if (up->u_signal[i - 1] == SIG_IGN)
933                                 sigaddset(&ignore, lx_sig);
934                         else if (up->u_signal[i - 1] != SIG_DFL)
935                                 sigaddset(&handle, lx_sig);
936                 }
937         }

939         lxpr_uiobuf_printf(uiobuf,
940             "\n"
941             "SigPnd:\t%08x%08x\n"
942             "SigBlk:\t%08x%08x\n"
943             "SigIgn:\t%08x%08x\n"
944             "SigCgt:\t%08x%08x\n"
945             "CapInh:\t%016x\n"
946             "CapPrm:\t%016x\n"
947             "CapEff:\t%016x\n",
948             current.__sigbits[1], current.__sigbits[0],
949             0, 0, /* signals blocked on per thread basis */
950             ignore.__sigbits[1], ignore.__sigbits[0],
951             handle.__sigbits[1], handle.__sigbits[0],
952             /* Can't do anything with linux capabilities */
953             0,
954             0,
955             0);

957         lxpr_unlock(p);
958 }


961 /*
962  * lxpr_read_pid_stat(): pid stat file
963  */
964 static void
965 lxpr_read_pid_stat(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
966 {
967         proc_t *p;
968         kthread_t *t;
969         struct as *as;
970         char stat;
971         pid_t pid, ppid, pgpid, spid;
972         gid_t psgid;
973         dev_t psdev;
974         size_t rss, vsize;
975         int nice, pri;
976         caddr_t wchan;
977         processorid_t cpu;

979         ASSERT(lxpnp->lxpr_type == LXPR_PID_STAT);

981         p = lxpr_lock(lxpnp->lxpr_pid);
982         if (p == NULL) {
983                 lxpr_uiobuf_seterr(uiobuf, EINVAL);
984                 return;
```



```
985         }

987         pid = p->p_pid;

989         /*
990          * Set Linux defaults if we're the zone's init process
991          */
992         if (pid == curproc->p_zone->zone_proc_initpid) {
993                 pid = 1;                /* PID for init */
994                 ppid = 0;               /* parent PID for init is 0 */
995                 pgpid = 0;              /* process group for init is 0 */
996                 psgid = (gid_t)-1;      /* credential GID for init is -1 */
997                 spid = 0;               /* session id for init is 0 */
998                 psdev = 0;              /* session device for init is 0 */
999         } else {
1000                /*
1001                 * Make sure not to reference parent PIDs that reside outside
1002                 * the zone
1003                 */
1004                ppid = ((p->p_flag & SZONETOP)
1005                    ? curproc->p_zone->zone_zsched->p_pid : p->p_ppid);

1007                /*
1008                 * Convert ppid to the Linux default of 1 if our parent is the
1009                 * zone's init process
1010                 */
1011                if (ppid == curproc->p_zone->zone_proc_initpid)
1012                        ppid = 1;

1014                pgpid = p->p_pgrp;

1016                mutex_enter(&p->p_splock);
1017                mutex_enter(&p->p_sessp->s_lock);
1018                spid = p->p_sessp->s_sid;
1019                /* XXBRAND psdev = DEV_TO_LXDEV(p->p_sessp->s_dev, VCHR); */
1020                psdev = p->p_sessp->s_dev;
1021                if (p->p_sessp->s_cred)
1022                        psgid = crgetgid(p->p_sessp->s_cred);
1023                else
1024                        psgid = crgetgid(p->p_cred);

1026                mutex_exit(&p->p_sessp->s_lock);
1027                mutex_exit(&p->p_splock);
1028        }

1030        t = prchoose(p);
1031        if (t != NULL) {
1032                switch (t->t_state) {
1033                case TS_SLEEP:
1034                        stat = 'S'; break;
1035                case TS_RUN:
1036                case TS_ONPROC:
1037                        stat = 'R'; break;
1038                case TS_ZOMB:
1039                        stat = 'Z'; break;
1040                case TS_STOPPED:
1041                        stat = 'T'; break;
1042                default:
1043                        stat = '!'; break;
1044                }

1046                if (CL_DONICE(t, NULL, 0, &nice) != 0)
1047                        nice = 0;

1049                pri = v.v_maxsyspri - t->t_pri;
1050                wchan = t->t_wchan;
```

```
1051                 cpu = t->t_cpu->cpu_seqid;
1052                 thread_unlock(t);
1053         } else {
1054                 /* Only zombies have no threads */
1055                 stat = 'Z';
1056                 nice = 0;
1057                 pri = 0;
1058                 wchan = 0;
1059                 cpu = 0;
1060         }
1061         as = p->p_as;
1062         mutex_exit(&p->p_lock);
1063         AS_LOCK_ENTER(as, &as->a_lock, RW_READER);
1064         vsize = as->a_resvsize;
1065         rss = rm_asrss(as);
1066         AS_LOCK_EXIT(as, &as->a_lock);
1067         mutex_enter(&p->p_lock);

1069         lxpr_uiobuf_printf(uiobuf,
1070             "%d (%s) %c %d %d %d %d %d "
1071             "%lu %lu %lu %lu %lu "
1072             "%lu %lu %ld %ld "
1073             "%d %d "
1074             "0 "
1075             "%ld %lu "
1076             "%lu %ld %llu "
1077             "%lu %lu %u "
1078             "%lu %lu "
1079             "%lu %lu %lu %lu "
1080             "%lu "
1081             "%lu %lu "
1082             "%d "
1083             "%d"
1084             "\n",
1085             pid,
1086             PTOU(p)->u_comm,
1087             stat,
1088             ppid, pgpid,
1089             spid, psdev, psgid,
1090             0l, 0l, 0l, 0l, 0l, /* flags, minflt, cminflt, majflt, cmajflt */
1091             p->p_utime, p->p_stime, p->p_cutime, p->p_cstime,
1092             pri, nice,
1093             0l, PTOU(p)->u_ticks, /* ticks till next SIGALARM, start time */
1094             vsize, rss, p->p_vmem_ctl,
1095             0l, 0l, USRSTACK, /* startcode, endcode, startstack */
1096             0l, 0l, /* kstkesp, kstkeip */
1097             0l, 0l, 0l, 0l, /* signal, blocked, sigignore, sigcatch */
1098             wchan,
1099             0l, 0l, /* nswap, cnswap */
1100             0, /* exit_signal */
1101             cpu);

1103         lxpr_unlock(p);
1104 }

1106 /* ARGSUSED */
1107 static void
1108 lxpr_read_net_arp(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
1109 {
1110 }

1112 /* ARGSUSED */
1113 static void
1114 lxpr_read_net_dev(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
1115 {
1116         lxpr_uiobuf_printf(uiobuf, "Inter-|   Receive                 "
```



```
1117             "                             |  Transmit\n");
1118         lxpr_uiobuf_printf(uiobuf, " face |bytes    packets errs drop fifo"
1119             " frame compressed multicast|bytes    packets errs drop fifo"
1120             " colls carrier compressed\n");

1122         /*
1123          * XXX: data about each interface should go here, but we'll wait to
1124          * see if anybody wants to use it.
1125          */
1126 }

1128 /* ARGSUSED */
1129 static void
1130 lxpr_read_net_dev_mcast(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
1131 {
1132 }

1134 /* ARGSUSED */
1135 static void
1136 lxpr_read_net_igmp(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
1137 {
1138 }

1140 /* ARGSUSED */
1141 static void
1142 lxpr_read_net_ip_mr_cache(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
1143 {
1144 }

1146 /* ARGSUSED */
1147 static void
1148 lxpr_read_net_ip_mr_vif(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
1149 {
1150 }

1152 /* ARGSUSED */
1153 static void
1154 lxpr_read_net_mcfilter(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
1155 {
1156 }

1158 /* ARGSUSED */
1159 static void
1160 lxpr_read_net_netstat(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
1161 {
1162 }

1164 /* ARGSUSED */
1165 static void
1166 lxpr_read_net_raw(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
1167 {
1168 }

1170 /* ARGSUSED */
1171 static void
1172 lxpr_read_net_route(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
1173 {
1174 }

1176 /* ARGSUSED */
1177 static void
1178 lxpr_read_net_rpc(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
1179 {
1180 }

1182 /* ARGSUSED */
```

```
1183 static void
1184 lxpr_read_net_rt_cache(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
1185 {
1186 }

1188 /* ARGSUSED */
1189 static void
1190 lxpr_read_net_sockstat(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
1191 {
1192 }

1194 /* ARGSUSED */
1195 static void
1196 lxpr_read_net_snmp(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
1197 {
1198 }

1200 /* ARGSUSED */
1201 static void
1202 lxpr_read_net_stat(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
1203 {
1204 }

1206 /* ARGSUSED */
1207 static void
1208 lxpr_read_net_tcp(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
1209 {
1210 }

1212 /* ARGSUSED */
1213 static void
1214 lxpr_read_net_udp(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
1215 {
1216 }

1218 /* ARGSUSED */
1219 static void
1220 lxpr_read_net_unix(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
1221 {
1222 }

1224 /*
1225  * lxpr_read_kmsg(): read the contents of the kernel message queue. We
1226  * translate this into the reception of console messages for this lx zone; each
1227  * read copies out a single zone console message, or blocks until the next one
1228  * is produced.
1229  */

1231 #define LX_KMSG_PRI     "<0>"

1233 static void
1234 lxpr_read_kmsg(lxpr_node_t *lxpnp, struct lxpr_uiobuf *uiobuf)
1235 {
1236         ldi_handle_t    lh = lxpnp->lxpr_cons_ldih;
1237         mblk_t          *mp;

1239         if (ldi_getmsg(lh, &mp, NULL) == 0) {
1240                 /*
1241                  * lx procfs doesn't like successive reads to the same file
1242                  * descriptor unless we do an explicit rewind each time.
1243                  */
1244                 lxpr_uiobuf_seek(uiobuf, 0);

1246                 lxpr_uiobuf_printf(uiobuf, "%s%s", LX_KMSG_PRI,
1247                     mp->b_cont->b_rptr);
```



```
1249                 freemsg(mp);
1250         }
1251 }

1253 /*
1254  * lxpr_read_loadavg(): read the contents of the "loadavg" file.
1255  *
1256  * Just enough for uptime to work
1257  */
1258 extern int nthread;

1260 static void
1261 lxpr_read_loadavg(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
1262 {
1263         ulong_t avenrun1;
1264         ulong_t avenrun5;
1265         ulong_t avenrun15;
1266         ulong_t avenrun1_cs;
1267         ulong_t avenrun5_cs;
1268         ulong_t avenrun15_cs;
1269         int loadavg[3];
1270         int *loadbuf;
1271         cpupart_t *cp;

1273         uint_t nrunnable = 0;
1274         rctl_qty_t nlwps;

1276         ASSERT(lxpnp->lxpr_type == LXPR_LOADAVG);

1278         mutex_enter(&cpu_lock);

1280         /*
1281          * Need to add up values over all CPU partitions. If pools are active,
1282          * only report the values of the zone's partition, which by definition
1283          * includes the current CPU.
1284          */
1285         if (pool_pset_enabled()) {
1286                 psetid_t psetid = zone_pset_get(curproc->p_zone);

1288                 ASSERT(curproc->p_zone != &zone0);
1289                 cp = CPU->cpu_part;

1291                 nrunnable = cp->cp_nrunning + cp->cp_nrunnable;
1292                 (void) cpupart_get_loadavg(psetid, &loadavg[0], 3);
1293                 loadbuf = &loadavg[0];

1295                 /*
1296                  * We'll report the total number of lwps in the zone for the
1297                  * "nproc" parameter of /proc/loadavg; good enough for lx.
1298                  */
1299                 nlwps = curproc->p_zone->zone_nlwps;
1300         } else {
1301                 cp = cp_list_head;
1302                 do {
1303                         nrunnable += cp->cp_nrunning + cp->cp_nrunnable;
1304                 } while ((cp = cp->cp_next) != cp_list_head);

1306                 loadbuf = &avenrun[0];

1308                 /*
1309                  * This will report kernel threads as well as user lwps, but it
1310                  * should be good enough for lx consumers.
1311                  */
1312                 nlwps = nthread;
1313         }
```

```
1315         mutex_exit(&cpu_lock);

1317         avenrun1 = loadbuf[0] >> FSHIFT;
1318         avenrun1_cs = ((loadbuf[0] & (FSCALE-1)) * 100) >> FSHIFT;
1319         avenrun5 = loadbuf[1] >> FSHIFT;
1320         avenrun5_cs = ((loadbuf[1] & (FSCALE-1)) * 100) >> FSHIFT;
1321         avenrun15 = loadbuf[2] >> FSHIFT;
1322         avenrun15_cs = ((loadbuf[2] & (FSCALE-1)) * 100) >> FSHIFT;

1324         lxpr_uiobuf_printf(uiobuf,
1325             "%ld.%02d %ld.%02d %ld.%02d %d/%d %d\n",
1326             avenrun1, avenrun1_cs,
1327             avenrun5, avenrun5_cs,
1328             avenrun15, avenrun15_cs,
1329             nrunnable, nlwps, 0);
1330 }

1332 /*
1333  * lxpr_read_meminfo(): read the contents of the "meminfo" file.
1334  */
1335 static void
1336 lxpr_read_meminfo(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
1337 {
1338         long total_mem = physmem * PAGESIZE;
1339         long free_mem = freemem * PAGESIZE;
1340         long total_swap = k_anoninfo.ani_max * PAGESIZE;
1341         long used_swap = k_anoninfo.ani_phys_resv * PAGESIZE;

1343         ASSERT(lxpnp->lxpr_type == LXPR_MEMINFO);

1345         lxpr_uiobuf_printf(uiobuf,
1346             "        total:    used:    free:  shared: buffers:  cached:\n"
1347             "Mem:  %8lu %8lu %8lu %8u %8u %8u\n"
1348             "Swap: %8lu %8lu %8lu\n"
1349             "MemTotal:  %8lu kB\n"
1350             "MemFree:   %8lu kB\n"
1351             "MemShared: %8u kB\n"
1352             "Buffers:   %8u kB\n"
1353             "Cached:    %8u kB\n"
1354             "SwapCached:%8u kB\n"
1355             "Active:    %8u kB\n"
1356             "Inactive:  %8u kB\n"
1357             "HighTotal: %8u kB\n"
1358             "HighFree:  %8u kB\n"
1359             "LowTotal:  %8u kB\n"
1360             "LowFree:   %8u kB\n"
1361             "SwapTotal: %8lu kB\n"
1362             "SwapFree:  %8lu kB\n",
1363             total_mem, total_mem - free_mem, free_mem, 0, 0, 0,
1364             total_swap, used_swap, total_swap - used_swap,
1365             btok(total_mem),                            /* MemTotal */
1366             btok(free_mem),                             /* MemFree */
1367             0,                                          /* MemShared */
1368             0,                                          /* Buffers */
1369             0,                                          /* Cached */
1370             0,                                          /* SwapCached */
1371             0,                                          /* Active */
1372             0,                                          /* Inactive */
1373             0,                                          /* HighTotal */
1374             0,                                          /* HighFree */
1375             btok(total_mem),                            /* LowTotal */
1376             btok(free_mem),                             /* LowFree */
1377             btok(total_swap),                           /* SwapTotal */
1378             btok(total_swap - used_swap));              /* SwapFree */
1379 }
```



```
1381 /*
1382  * lxpr_read_mounts():
1383  */
1384 /* ARGSUSED */
1385 static void
1386 lxpr_read_mounts(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
1387 {
1388         struct vfs *vfsp;
1389         struct vfs *vfslist;
1390         zone_t *zone = LXPTOZ(lxpnp);
1391         struct print_data {
1392                 refstr_t *vfs_mntpt;
1393                 refstr_t *vfs_resource;
1394                 uint_t vfs_flag;
1395                 int vfs_fstype;
1396                 struct print_data *next;
1397         } *print_head = NULL;
1398         struct print_data **print_tail = &print_head;
1399         struct print_data *printp;

1401         vfs_list_read_lock();

1403         if (zone == global_zone) {
1404                 vfsp = vfslist = rootvfs;
1405         } else {
1406                 vfsp = vfslist = zone->zone_vfslist;
1407                 /*
1408                  * If the zone has a root entry, it will be the first in
1409                  * the list.  If it doesn't, we conjure one up.
1410                  */
1411                 if (vfslist == NULL ||
1412                     strcmp(refstr_value(vfsp->vfs_mntpt),
1413                     zone->zone_rootpath) != 0) {
1414                         struct vfs *tvfsp;
1415                         /*
1416                          * The root of the zone is not a mount point.  The vfs
1417                          * we want to report is that of the zone's root vnode.
1418                          */
1419                         tvfsp = zone->zone_rootvp->v_vfsp;

1421                         lxpr_uiobuf_printf(uiobuf,
1422                             "/ / %s %s 0 0\n",
1423                             vfssw[tvfsp->vfs_fstype].vsw_name,
1424                             tvfsp->vfs_flag & VFS_RDONLY ? "ro" : "rw");

1426                 }
1427                 if (vfslist == NULL) {
1428                         vfs_list_unlock();
1429                         return;
1430                 }
1431         }

1433         /*
1434          * Later on we have to do a lookupname, which can end up causing
1435          * another vfs_list_read_lock() to be called. Which can lead to a
1436          * deadlock. To avoid this, we extract the data we need into a local
1437          * list, then we can run this list without holding vfs_list_read_lock()
1438          * We keep the list in the same order as the vfs_list
1439          */
1440         do {
1441                 /* Skip mounts we shouldn't show */
1442                 if (vfsp->vfs_flag & VFS_NOMNTTAB) {
1443                         goto nextfs;
1444                 }

1446                 printp = kmem_alloc(sizeof (*printp), KM_SLEEP);
```

```
1447                refstr_hold(vfsp->vfs_mntpt);
1448                printp->vfs_mntpt = vfsp->vfs_mntpt;
1449                refstr_hold(vfsp->vfs_resource);
1450                printp->vfs_resource = vfsp->vfs_resource;
1451                printp->vfs_flag = vfsp->vfs_flag;
1452                printp->vfs_fstype = vfsp->vfs_fstype;
1453                printp->next = NULL;

1455                *print_tail = printp;
1456                print_tail = &printp->next;

1458 nextfs:
1459                vfsp = (zone == global_zone) ?
1460                    vfsp->vfs_next : vfsp->vfs_zone_next;

1462        } while (vfsp != vfslist);

1464        vfs_list_unlock();

1466        /*
1467         * now we can run through what we've extracted without holding
1468         * vfs_list_read_lock()
1469         */
1470        printp = print_head;
1471        while (printp != NULL) {
1472                struct print_data *printp_next;
1473                const char *resource;
1474                char *mntpt;
1475                struct vnode *vp;
1476                int error;

1478                mntpt = (char *)refstr_value(printp->vfs_mntpt);
1479                resource = refstr_value(printp->vfs_resource);

1481                if (mntpt != NULL && mntpt[0] != '\0')
1482                        mntpt = ZONE_PATH_TRANSLATE(mntpt, zone);
1483                else
1484                        mntpt = "-";

1486                error = lookupname(mntpt, UIO_SYSSPACE, FOLLOW, NULLVPP, &vp);

1488                if (error != 0)
1489                        goto nextp;

1491                if (!(vp->v_flag & VROOT)) {
1492                        VN_RELE(vp);
1493                        goto nextp;
1494                }
1495                VN_RELE(vp);

1497                if (resource != NULL && resource[0] != '\0') {
1498                        if (resource[0] == '/') {
1499                                resource = ZONE_PATH_VISIBLE(resource, zone) ?
1500                                    ZONE_PATH_TRANSLATE(resource, zone) :
1501                                    mntpt;
1502                        }
1503                } else {
1504                        resource = "-";
1505                }

1507                lxpr_uiobuf_printf(uiobuf,
1508                    "%s %s %s %s 0 0\n",
1509                    resource, mntpt, vfssw[printp->vfs_fstype].vsw_name,
1510                    printp->vfs_flag & VFS_RDONLY ? "ro" : "rw");

1512 nextp:
```



```
1513                printp_next = printp->next;
1514                refstr_rele(printp->vfs_mntpt);
1515                refstr_rele(printp->vfs_resource);
1516                kmem_free(printp, sizeof (*printp));
1517                printp = printp_next;

1519        }
1520 }

1522 /*
1523  * lxpr_read_partitions():
1524  *
1525  * We don't support partitions in a local zone because it requires access to
1526  * physical devices.  But we need to fake up enough of the file to show that we
1527  * have no partitions.
1528  */
1529 /* ARGSUSED */
1530 static void
1531 lxpr_read_partitions(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
1532 {
1533         lxpr_uiobuf_printf(uiobuf,
1534             "major minor  #blocks  name     rio rmerge rsect ruse "
1535             "wio wmerge wsect wuse running use aveq\n\n");
1536 }

1538 /*
1539  * lxpr_read_version(): read the contents of the "version" file.
1540  */
1541 /* ARGSUSED */
1542 static void
1543 lxpr_read_version(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
1544 {
1545         char *vers;
1546         if (lx_get_zone_kern_version(LXPTOZ(lxpnp)) <= LX_KERN_2_4)
1547                 vers = LX_UNAME_RELEASE_2_4;
1548         else
1549                 vers = LX_UNAME_RELEASE_2_6;

1551         lxpr_uiobuf_printf(uiobuf,
1552             "%s version %s (%s version %d.%d.%d) "
1553             "#%s SMP %s\n",
1554             LX_UNAME_SYSNAME, vers,
1555 #if defined(__GNUC__)
1556             "gcc",
1557             __GNUC__,
1558             __GNUC_MINOR__,
1559             __GNUC_PATCHLEVEL__,
1560 #else
1561             "Sun C",
1562             __SUNPRO_C / 0x100,
1563             (__SUNPRO_C & 0xff) / 0x10,
1564             __SUNPRO_C & 0xf,
1565 #endif
1566             LX_UNAME_VERSION,
1567             "00:00:00 00/00/00");
1568 }


1571 /*
1572  * lxpr_read_stat(): read the contents of the "stat" file.
1573  *
1574  */
1575 /* ARGSUSED */

1577 static void
1578 lxpr_read_stat(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
```

```
1579 {
1580         cpu_t *cp, *cpstart;
1581         int pools_enabled;
1582         ulong_t idle_cum = 0;
1583         ulong_t sys_cum  = 0;
1584         ulong_t user_cum = 0;
1585         ulong_t irq_cum = 0;
1586         uint_t cpu_nrunnable_cum = 0;
1587         uint_t w_io_cum = 0;

1589         ulong_t pgpgin_cum    = 0;
1590         ulong_t pgpgout_cum   = 0;
1591         ulong_t pgswapout_cum = 0;
1592         ulong_t pgswapin_cum  = 0;
1593         ulong_t intr_cum = 0;
1594         ulong_t pswitch_cum = 0;
1595         ulong_t forks_cum = 0;
1596         hrtime_t msnsecs[NCMSTATES];
1597         int lx_kern_version = lx_get_zone_kern_version(LXPTOZ(lxpnp));
1598         /* temporary variable since scalehrtime modifies data in place */
1599         hrtime_t tmptime;

1601         ASSERT(lxpnp->lxpr_type == LXPR_STAT);

1603         mutex_enter(&cpu_lock);
1604         pools_enabled = pool_pset_enabled();

1606         /* Calculate cumulative stats */
1607         cp = cpstart = CPU;
1608         do {
1609                 int i;

1611                 /*
1612                  * Don't count CPUs that aren't even in the system
1613                  * or aren't up yet.
1614                  */
1615                 if ((cp->cpu_flags & CPU_EXISTS) == 0) {
1616                         continue;
1617                 }

1619                 get_cpu_mstate(cp, msnsecs);

1621                 idle_cum += NSEC_TO_TICK(msnsecs[CMS_IDLE]);
1622                 sys_cum  += NSEC_TO_TICK(msnsecs[CMS_SYSTEM]);
1623                 user_cum += NSEC_TO_TICK(msnsecs[CMS_USER]);

1625                 pgpgin_cum += CPU_STATS(cp, vm.pgpgin);
1626                 pgpgout_cum += CPU_STATS(cp, vm.pgpgout);
1627                 pgswapin_cum += CPU_STATS(cp, vm.pgswapin);
1628                 pgswapout_cum += CPU_STATS(cp, vm.pgswapout);

1630                 if (lx_kern_version >= LX_KERN_2_6) {
1631                         cpu_nrunnable_cum += cp->cpu_disp->disp_nrunnable;
1632                         w_io_cum += CPU_STATS(cp, sys.iowait);
1633                         for (i = 0; i < NCMSTATES; i++) {
1634                                 tmptime = cp->cpu_intracct[i];
1635                                 scalehrtime(&tmptime);
1636                                 irq_cum += NSEC_TO_TICK(tmptime);
1637                         }
1638                 }

1640                 for (i = 0; i < PIL_MAX; i++)
1641                         intr_cum += CPU_STATS(cp, sys.intr[i]);

1643                 pswitch_cum += CPU_STATS(cp, sys.pswitch);
1644                 forks_cum += CPU_STATS(cp, sys.sysfork);
```



```
1645                 forks_cum += CPU_STATS(cp, sys.sysvfork);

1647                 if (pools_enabled)
1648                         cp = cp->cpu_next_part;
1649                 else
1650                         cp = cp->cpu_next;
1651         } while (cp != cpstart);

1653         if (lx_kern_version >= LX_KERN_2_6) {
1654                 lxpr_uiobuf_printf(uiobuf,
1655                     "cpu %ld %ld %ld %ld %ld %ld %ld\n",
1656                     user_cum, 0, sys_cum, idle_cum, 0, irq_cum, 0);
1657         } else {
1658                 lxpr_uiobuf_printf(uiobuf,
1659                     "cpu %ld %ld %ld %ld\n",
1660                     user_cum, 0, sys_cum, idle_cum);
1661         }

1663         /* Do per processor stats */
1664         do {
1665                 int i;

1667                 ulong_t idle_ticks;
1668                 ulong_t sys_ticks;
1669                 ulong_t user_ticks;
1670                 ulong_t irq_ticks = 0;

1672                 /*
1673                  * Don't count CPUs that aren't even in the system
1674                  * or aren't up yet.
1675                  */
1676                 if ((cp->cpu_flags & CPU_EXISTS) == 0) {
1677                         continue;
1678                 }

1680                 get_cpu_mstate(cp, msnsecs);

1682                 idle_ticks = NSEC_TO_TICK(msnsecs[CMS_IDLE]);
1683                 sys_ticks  = NSEC_TO_TICK(msnsecs[CMS_SYSTEM]);
1684                 user_ticks = NSEC_TO_TICK(msnsecs[CMS_USER]);

1686                 if (lx_kern_version >= LX_KERN_2_6) {
1687                         for (i = 0; i < NCMSTATES; i++) {
1688                                 tmptime = cp->cpu_intracct[i];
1689                                 scalehrtime(&tmptime);
1690                                 irq_ticks += NSEC_TO_TICK(tmptime);
1691                         }

1693                         lxpr_uiobuf_printf(uiobuf,
1694                             "cpu%d %ld %ld %ld %ld %ld %ld %ld\n",
1695                             cp->cpu_id, user_ticks, 0, sys_ticks, idle_ticks,
1696                             0, irq_ticks, 0);
1697                 } else {
1698                         lxpr_uiobuf_printf(uiobuf,
1699                             "cpu%d %ld %ld %ld %ld\n",
1700                             cp->cpu_id,
1701                             user_ticks, 0, sys_ticks, idle_ticks);
1702                 }

1704                 if (pools_enabled)
1705                         cp = cp->cpu_next_part;
1706                 else
1707                         cp = cp->cpu_next;
1708         } while (cp != cpstart);

1710         mutex_exit(&cpu_lock);
```

```
1712         if (lx_kern_version >= LX_KERN_2_6) {
1713                 lxpr_uiobuf_printf(uiobuf,
1714                     "page %lu %lu\n"
1715                     "swap %lu %lu\n"
1716                     "intr %lu\n"
1717                     "ctxt %lu\n"
1718                     "btime %lu\n"
1719                     "processes %lu\n"
1720                     "procs_running %lu\n"
1721                     "procs_blocked %lu\n",
1722                     pgpgin_cum, pgpgout_cum,
1723                     pgswapin_cum, pgswapout_cum,
1724                     intr_cum,
1725                     pswitch_cum,
1726                     boot_time,
1727                     forks_cum,
1728                     cpu_nrunnable_cum,
1729                     w_io_cum);
1730         } else {
1731                 lxpr_uiobuf_printf(uiobuf,
1732                     "page %lu %lu\n"
1733                     "swap %lu %lu\n"
1734                     "intr %lu\n"
1735                     "ctxt %lu\n"
1736                     "btime %lu\n"
1737                     "processes %lu\n",
1738                     pgpgin_cum, pgpgout_cum,
1739                     pgswapin_cum, pgswapout_cum,
1740                     intr_cum,
1741                     pswitch_cum,
1742                     boot_time,
1743                     forks_cum);
1744         }
1745 }


1748 /*
1749  * lxpr_read_uptime(): read the contents of the "uptime" file.
1750  *
1751  * format is: "%.2lf, %.2lf",uptime_secs, idle_secs
1752  * Use fixed point arithmetic to get 2 decimal places
1753  */
1754 /* ARGSUSED */
1755 static void
1756 lxpr_read_uptime(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
1757 {
1758         cpu_t *cp, *cpstart;
1759         int pools_enabled;
1760         ulong_t idle_cum = 0;
1761         ulong_t cpu_count = 0;
1762         ulong_t idle_s;
1763         ulong_t idle_cs;
1764         ulong_t up_s;
1765         ulong_t up_cs;
1766         hrtime_t birthtime;
1767         hrtime_t centi_sec = 10000000;  /* 10^7 */

1769         ASSERT(lxpnp->lxpr_type == LXPR_UPTIME);

1771         /* Calculate cumulative stats */
1772         mutex_enter(&cpu_lock);
1773         pools_enabled = pool_pset_enabled();

1775         cp = cpstart = CPU;
1776         do {
```



```
1777                 /*
1778                  * Don't count CPUs that aren't even in the system
1779                  * or aren't up yet.
1780                  */
1781                 if ((cp->cpu_flags & CPU_EXISTS) == 0) {
1782                         continue;
1783                 }

1785                 idle_cum += CPU_STATS(cp, sys.cpu_ticks_idle);
1786                 idle_cum += CPU_STATS(cp, sys.cpu_ticks_wait);
1787                 cpu_count += 1;

1789                 if (pools_enabled)
1790                         cp = cp->cpu_next_part;
1791                 else
1792                         cp = cp->cpu_next;
1793         } while (cp != cpstart);
1794         mutex_exit(&cpu_lock);

1796         /* Getting the Zone zsched process startup time */
1797         birthtime = LXPTOZ(lxpnp)->zone_zsched->p_mstart;
1798         up_cs = (gethrtime() - birthtime) / centi_sec;
1799         up_s = up_cs / 100;
1800         up_cs %= 100;

1802         ASSERT(cpu_count > 0);
1803         idle_cum /= cpu_count;
1804         idle_s = idle_cum / hz;
1805         idle_cs = idle_cum % hz;
1806         idle_cs *= 100;
1807         idle_cs /= hz;

1809         lxpr_uiobuf_printf(uiobuf,
1810             "%ld.%02d %ld.%02d\n", up_s, up_cs, idle_s, idle_cs);
1811 }

1813 static const char *amd_x_edx[] = {
1814         NULL,   NULL,   NULL,   NULL,
1815         NULL,   NULL,   NULL,   NULL,
1816         NULL,   NULL,   NULL,   "syscall",
1817         NULL,   NULL,   NULL,   NULL,
1818         NULL,   NULL,   NULL,   "mp",
1819         "nx",   NULL,   "mmxext", NULL,
1820         NULL,   NULL,   NULL,   NULL,
1821         NULL,   "lm",   "3dnowext", "3dnow"
1822 };

1824 static const char *amd_x_ecx[] = {
1825         "lahf_lm", NULL, "svm", NULL,
1826         "altmovcr8"
1827 };

1829 static const char *tm_x_edx[] = {
1830         "recovery", "longrun", NULL, "lrti"
1831 };

1833 /*
1834  * Intel calls no-execute "xd" in its docs, but Linux still reports it as "nx."
1835  */
1836 static const char *intc_x_edx[] = {
1837         NULL,   NULL,   NULL,   NULL,
1838         NULL,   NULL,   NULL,   NULL,
1839         NULL,   NULL,   NULL,   "syscall",
1840         NULL,   NULL,   NULL,   NULL,
1841         NULL,   NULL,   NULL,   NULL,
1842         "nx",   NULL,   NULL,   NULL,
```

```
1843         NULL,   NULL,   NULL,   NULL,
1844         NULL,   "lm",   NULL,   NULL
1845 };

1847 static const char *intc_edx[] = {
1848         "fpu",  "vme",  "de",   "pse",
1849         "tsc",  "msr",  "pae",  "mce",
1850         "cx8",  "apic",  NULL,  "sep",
1851         "mtrr", "pge",  "mca",  "cmov",
1852         "pat",  "pse36", "pn",  "clflush",
1853         NULL,   "dts",  "acpi", "mmx",
1854         "fxsr", "sse",  "sse2", "ss",
1855         "ht",   "tm",   "ia64", "pbe"
1856 };

1858 /*
1859  * "sse3" on linux is called "pni" (Prescott New Instructions).
1860  */
1861 static const char *intc_ecx[] = {
1862         "pni",  NULL,   NULL, "monitor",
1863         "ds_cpl", NULL, NULL, "est",
1864         "tm2",  NULL,   "cid", NULL,
1865         NULL,   "cx16", "xtpr"
1866 };

1868 static void
1869 lxpr_read_cpuinfo(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
1870 {
1871         int i;
1872         uint32_t bits;
1873         cpu_t *cp, *cpstart;
1874         int pools_enabled;
1875         const char **fp;
1876         char brandstr[CPU_IDSTRLEN];
1877         struct cpuid_regs cpr;
1878         int maxeax;
1879         int std_ecx, std_edx, ext_ecx, ext_edx;

1881         ASSERT(lxpnp->lxpr_type == LXPR_CPUINFO);

1883         mutex_enter(&cpu_lock);
1884         pools_enabled = pool_pset_enabled();

1886         cp = cpstart = CPU;
1887         do {
1888                 /*
1889                  * This returns the maximum eax value for standard cpuid
1890                  * functions in eax.
1891                  */
1892                 cpr.cp_eax = 0;
1893                 (void) cpuid_insn(cp, &cpr);
1894                 maxeax = cpr.cp_eax;

1896                 /*
1897                  * Get standard x86 feature flags.
1898                  */
1899                 cpr.cp_eax = 1;
1900                 (void) cpuid_insn(cp, &cpr);
1901                 std_ecx = cpr.cp_ecx;
1902                 std_edx = cpr.cp_edx;

1904                 /*
1905                  * Now get extended feature flags.
1906                  */
1907                 cpr.cp_eax = 0x80000001;
1908                 (void) cpuid_insn(cp, &cpr);
```



```
1909                 ext_ecx = cpr.cp_ecx;
1910                 ext_edx = cpr.cp_edx;

1912                 (void) cpuid_getbrandstr(cp, brandstr, CPU_IDSTRLEN);

1914                 lxpr_uiobuf_printf(uiobuf,
1915                     "processor\t: %d\n"
1916                     "vendor_id\t: %s\n"
1917                     "cpu family\t: %d\n"
1918                     "model\t\t: %d\n"
1919                     "model name\t: %s\n"
1920                     "stepping\t: %d\n"
1921                     "cpu MHz\t\t: %u.%03u\n",
1922                     cp->cpu_id, cpuid_getvendorstr(cp), cpuid_getfamily(cp),
1923                     cpuid_getmodel(cp), brandstr, cpuid_getstep(cp),
1924                     (uint32_t)(cpu_freq_hz / 1000000),
1925                     ((uint32_t)(cpu_freq_hz / 1000)) % 1000);

1927                 lxpr_uiobuf_printf(uiobuf, "cache size\t: %u KB\n",
1928                     getl2cacheinfo(cp, NULL, NULL, NULL) / 1024);

1930 /*              if (x86_feature & X86_HTT) { */
1931                         /*
1932                          * 'siblings' is used for HT-style threads
1933                          */
1934 /*                      lxpr_uiobuf_printf(uiobuf,
1935                             "physical id\t: %lu\n"
1936                             "siblings\t: %u\n",
1937                             pg_plat_hw_instance_id(cp, PGHW_CHIP),
1938                             cpuid_get_ncpu_per_chip(cp));
1939                 }
1940 */
1941                 /*
1942                  * Since we're relatively picky about running on older hardware,
1943                  * we can be somewhat cavalier about the answers to these ones.
1944                  *
1945                  * In fact, given the hardware we support, we just say:
1946                  *
1947                  *      fdiv_bug        : no    (if we're on a 64-bit kernel)
1948                  *      hlt_bug         : no
1949                  *      f00f_bug        : no
1950                  *      coma_bug        : no
1951                  *      wp              : yes   (write protect in supervsr mode)
1952                  */
1953                 lxpr_uiobuf_printf(uiobuf,
1954                     "fdiv_bug\t: %s\n"
1955                     "hlt_bug \t: no\n"
1956                     "f00f_bug\t: no\n"
1957                     "coma_bug\t: no\n"
1958                     "fpu\t\t: %s\n"
1959                     "fpu_exception\t: %s\n"
1960                     "cpuid level\t: %d\n"
1961                     "flags\t\t:",
1962 #if defined(__i386)
1963                     fpu_pentium_fdivbug ? "yes" : "no",
1964 #else
1965                     "no",
1966 #endif /* __i386 */
1967                     fpu_exists ? "yes" : "no", fpu_exists ? "yes" : "no",
1968                     maxeax);

1970                 for (bits = std_edx, fp = intc_edx, i = 0;
1971                     i < sizeof (intc_edx) / sizeof (intc_edx[0]); fp++, i++)
1972                         if ((bits & (1 << i)) != 0 && *fp)
1973                                 lxpr_uiobuf_printf(uiobuf, " %s", *fp);
```

```
1975                 /*
1976                  * name additional features where appropriate
1977                  */
1978                 switch (x86_vendor) {
1979                 case X86_VENDOR_Intel:
1980                         for (bits = ext_edx, fp = intc_x_edx, i = 0;
1981                             i < sizeof (intc_x_edx) / sizeof (intc_x_edx[0]);
1982                             fp++, i++)
1983                                 if ((bits & (1 << i)) != 0 && *fp)
1984                                         lxpr_uiobuf_printf(uiobuf, " %s", *fp);
1985                         break;

1987                 case X86_VENDOR_AMD:
1988                         for (bits = ext_edx, fp = amd_x_edx, i = 0;
1989                             i < sizeof (amd_x_edx) / sizeof (amd_x_edx[0]);
1990                             fp++, i++)
1991                                 if ((bits & (1 << i)) != 0 && *fp)
1992                                         lxpr_uiobuf_printf(uiobuf, " %s", *fp);

1994                         for (bits = ext_ecx, fp = amd_x_ecx, i = 0;
1995                             i < sizeof (amd_x_ecx) / sizeof (amd_x_ecx[0]);
1996                             fp++, i++)
1997                                 if ((bits & (1 << i)) != 0 && *fp)
1998                                         lxpr_uiobuf_printf(uiobuf, " %s", *fp);
1999                         break;

2001                 case X86_VENDOR_TM:
2002                         for (bits = ext_edx, fp = tm_x_edx, i = 0;
2003                             i < sizeof (tm_x_edx) / sizeof (tm_x_edx[0]);
2004                             fp++, i++)
2005                                 if ((bits & (1 << i)) != 0 && *fp)
2006                                         lxpr_uiobuf_printf(uiobuf, " %s", *fp);
2007                         break;
2008                 default:
2009                         break;
2010                 }

2012                 for (bits = std_ecx, fp = intc_ecx, i = 0;
2013                     i < sizeof (intc_ecx) / sizeof (intc_ecx[0]); fp++, i++)
2014                         if ((bits & (1 << i)) != 0 && *fp)
2015                                 lxpr_uiobuf_printf(uiobuf, " %s", *fp);

2017                 lxpr_uiobuf_printf(uiobuf, "\n\n");

2019                 if (pools_enabled)
2020                         cp = cp->cpu_next_part;
2021                 else
2022                         cp = cp->cpu_next;
2023         } while (cp != cpstart);

2025         mutex_exit(&cpu_lock);
2026 }

2028 /* ARGSUSED */
2029 static void
2030 lxpr_read_fd(lxpr_node_t *lxpnp, lxpr_uiobuf_t *uiobuf)
2031 {
2032         ASSERT(lxpnp->lxpr_type == LXPR_PID_FD_FD);
2033         lxpr_uiobuf_seterr(uiobuf, EFAULT);
2034 }



2038 /*
2039  * lxpr_getattr(): Vnode operation for VOP_GETATTR()
2040  */
```



```
2041 static int
2042 lxpr_getattr(vnode_t *vp, vattr_t *vap, int flags, cred_t *cr,
2043     caller_context_t *ct)
2044 {
2045         register lxpr_node_t *lxpnp = VTOLXP(vp);
2046         lxpr_nodetype_t type = lxpnp->lxpr_type;
2047         extern uint_t nproc;
2048         int error;

2050         /*
2051          * Return attributes of underlying vnode if ATTR_REAL
2052          *
2053          * but keep fd files with the symlink permissions
2054          */
2055         if (lxpnp->lxpr_realvp != NULL && (flags & ATTR_REAL)) {
2056                 vnode_t *rvp = lxpnp->lxpr_realvp;

2058                 /*
2059                  * withold attribute information to owner or root
2060                  */
2061                 if ((error = VOP_ACCESS(rvp, 0, 0, cr, ct)) != 0) {
2062                         return (error);
2063                 }

2065                 /*
2066                  * now its attributes
2067                  */
2068                 if ((error = VOP_GETATTR(rvp, vap, flags, cr, ct)) != 0) {
2069                         return (error);
2070                 }

2072                 /*
2073                  * if it's a file in lx /proc/pid/fd/xx then set its
2074                  * mode and keep it looking like a symlink
2075                  */
2076                 if (type == LXPR_PID_FD_FD) {
2077                         vap->va_mode = lxpnp->lxpr_mode;
2078                         vap->va_type = vp->v_type;
2079                         vap->va_size = 0;
2080                         vap->va_nlink = 1;
2081                 }
2082                 return (0);
2083         }

2085         /* Default attributes, that may be overridden below */
2086         bzero(vap, sizeof (*vap));
2087         vap->va_atime = vap->va_mtime = vap->va_ctime = lxpnp->lxpr_time;
2088         vap->va_nlink = 1;
2089         vap->va_type = vp->v_type;
2090         vap->va_mode = lxpnp->lxpr_mode;
2091         vap->va_fsid = vp->v_vfsp->vfs_dev;
2092         vap->va_blksize = DEV_BSIZE;
2093         vap->va_uid = lxpnp->lxpr_uid;
2094         vap->va_gid = lxpnp->lxpr_gid;
2095         vap->va_nodeid = lxpnp->lxpr_ino;

2097         switch (type) {
2098         case LXPR_PROCDIR:
2099                 vap->va_nlink = nproc + 2 + PROCDIRFILES;
2100                 vap->va_size = (nproc + 2 + PROCDIRFILES) * LXPR_SDSIZE;
2101                 break;
2102         case LXPR_PIDDIR:
2103                 vap->va_nlink = PIDDIRFILES;
2104                 vap->va_size = PIDDIRFILES * LXPR_SDSIZE;
2105                 break;
2106         case LXPR_SELF:
```

```
2107                 vap->va_uid = crgetruid(curproc->p_cred);
2108                 vap->va_gid = crgetrgid(curproc->p_cred);
2109                 break;
2110         default:
2111                 break;
2112         }

2114         vap->va_nblocks = (fsblkcnt64_t)btod(vap->va_size);
2115         return (0);
2116 }


2119 /*
2120  * lxpr_access(): Vnode operation for VOP_ACCESS()
2121  */
2122 static int
2123 lxpr_access(vnode_t *vp, int mode, int flags, cred_t *cr, caller_context_t *ct)
2124 {
2125         lxpr_node_t *lxpnp = VTOLXP(vp);
2126         int shift = 0;
2127         proc_t *tp;

2129         /* lx /proc is a read only file system */
2130         if (mode & VWRITE)
2131                 return (EROFS);

2133         /*
2134          * If this is a restricted file, check access permissions.
2135          */
2136         switch (lxpnp->lxpr_type) {
2137         case LXPR_PIDDIR:
2138                 return (0);
2139         case LXPR_PID_CURDIR:
2140         case LXPR_PID_ENV:
2141         case LXPR_PID_EXE:
2142         case LXPR_PID_MAPS:
2143         case LXPR_PID_MEM:
2144         case LXPR_PID_ROOTDIR:
2145         case LXPR_PID_FDDIR:
2146         case LXPR_PID_FD_FD:
2147                 if ((tp = lxpr_lock(lxpnp->lxpr_pid)) == NULL)
2148                         return (ENOENT);
2149                 if (tp != curproc && secpolicy_proc_access(cr) != 0 &&
2150                     priv_proc_cred_perm(cr, tp, NULL, mode) != 0) {
2151                         lxpr_unlock(tp);
2152                         return (EACCES);
2153                 }
2154                 lxpr_unlock(tp);
2155         default:
2156                 break;
2157         }

2159         if (lxpnp->lxpr_realvp != NULL) {
2160                 /*
2161                  * For these we use the underlying vnode's accessibility.
2162                  */
2163                 return (VOP_ACCESS(lxpnp->lxpr_realvp, mode, flags, cr, ct));
2164         }

2166         /* If user is root allow access regardless of permission bits */
2167         if (secpolicy_proc_access(cr) == 0)
2168                 return (0);

2170         /*
2171          * Access check is based on only
2172          * one of owner, group, public.
```



```
2173          * If not owner, then check group.
2174          * If not a member of the group, then
2175          * check public access.
2176          */
2177         if (crgetuid(cr) != lxpnp->lxpr_uid) {
2178                 shift += 3;
2179                 if (!groupmember((uid_t)lxpnp->lxpr_gid, cr))
2180                         shift += 3;
2181         }

2183         mode &= ~(lxpnp->lxpr_mode << shift);

2185         if (mode == 0)
2186                 return (0);

2188         return (EACCES);
2189 }




2194 /* ARGSUSED */
2195 static vnode_t *
2196 lxpr_lookup_not_a_dir(vnode_t *dp, char *comp)
2197 {
2198         return (NULL);
2199 }


2202 /*
2203  * lxpr_lookup(): Vnode operation for VOP_LOOKUP()
2204  */
2205 /* ARGSUSED */
2206 static int
2207 lxpr_lookup(vnode_t *dp, char *comp, vnode_t **vpp, pathname_t *pathp,
2208         int flags, vnode_t *rdir, cred_t *cr, caller_context_t *ct,
2209         int *direntflags, pathname_t *realpnp)
2210 {
2211         lxpr_node_t *lxpnp = VTOLXP(dp);
2212         lxpr_nodetype_t type = lxpnp->lxpr_type;
2213         int error;

2215         ASSERT(dp->v_type == VDIR);
2216         ASSERT(type < LXPR_NFILES);

2218         /*
2219          * we should never get here because the lookup
2220          * is done on the realvp for these nodes
2221          */
2222         ASSERT(type != LXPR_PID_FD_FD &&
2223             type != LXPR_PID_CURDIR &&
2224             type != LXPR_PID_ROOTDIR);

2226         /*
2227          * restrict lookup permission to owner or root
2228          */
2229         if ((error = lxpr_access(dp, VEXEC, 0, cr, ct)) != 0) {
2230                 return (error);
2231         }

2233         /*
2234          * Just return the parent vnode
2235          * if thats where we are trying to go
2236          */
2237         if (strcmp(comp, "..") == 0) {
2238                 VN_HOLD(lxpnp->lxpr_parent);
```

```
2239                *vpp = lxpnp->lxpr_parent;
2240                return (0);
2241        }

2243        /*
2244         * Special handling for directory searches
2245         * Note: null component name is synonym for
2246         * current directory being searched.
2247         */
2248        if ((dp->v_type == VDIR) && (*comp == '\0' || strcmp(comp, ".") == 0)) {
2249                VN_HOLD(dp);
2250                *vpp = dp;
2251                return (0);
2252        }

2254        *vpp = (lxpr_lookup_function[type](dp, comp));
2255        return ((*vpp == NULL) ? ENOENT : 0);
2256 }

2258 /*
2259  * Do a sequential search on the given directory table
2260  */
2261 static vnode_t *
2262 lxpr_lookup_common(vnode_t *dp, char *comp, proc_t *p,
2263     lxpr_dirent_t *dirtab, int dirtablen)
2264 {
2265        lxpr_node_t *lxpnp;
2266        int count;

2268        for (count = 0; count < dirtablen; count++) {
2269                if (strcmp(dirtab[count].d_name, comp) == 0) {
2270                        lxpnp = lxpr_getnode(dp, dirtab[count].d_type, p, 0);
2271                        dp = LXPTOV(lxpnp);
2272                        ASSERT(dp != NULL);
2273                        return (dp);
2274                }
2275        }
2276        return (NULL);
2277 }


2280 static vnode_t *
2281 lxpr_lookup_piddir(vnode_t *dp, char *comp)
2282 {
2283        proc_t *p;

2285        ASSERT(VTOLXP(dp)->lxpr_type == LXPR_PIDDIR);

2287        p = lxpr_lock(VTOLXP(dp)->lxpr_pid);
2288        if (p == NULL)
2289                return (NULL);

2291        dp = lxpr_lookup_common(dp, comp, p, piddir, PIDDIRFILES);

2293        lxpr_unlock(p);

2295        return (dp);
2296 }


2299 /*
2300  * Lookup one of the process's open files.
2301  */
2302 static vnode_t *
2303 lxpr_lookup_fddir(vnode_t *dp, char *comp)
2304 {
```



```
2305        lxpr_node_t *dlxpnp = VTOLXP(dp);
2306        lxpr_node_t *lxpnp;
2307        vnode_t *vp = NULL;
2308        proc_t *p;
2309        file_t *fp;
2310        uint_t fd;
2311        int c;
2312        uf_entry_t *ufp;
2313        uf_info_t *fip;

2315        ASSERT(dlxpnp->lxpr_type == LXPR_PID_FDDIR);

2317        /*
2318         * convert the string rendition of the filename
2319         * to a file descriptor
2320         */
2321        fd = 0;
2322        while ((c = *comp++) != '\0') {
2323                int ofd;
2324                if (c < '0' || c > '9')
2325                        return (NULL);

2327                ofd = fd;
2328                fd = 10*fd + c - '0';
2329                /* integer overflow */
2330                if (fd / 10 != ofd)
2331                        return (NULL);
2332        }

2334        /*
2335         * get the proc to work with and lock it
2336         */
2337        p = lxpr_lock(dlxpnp->lxpr_pid);
2338        if ((p == NULL))
2339                return (NULL);

2341        /*
2342         * If the process is a zombie or system process
2343         * it can't have any open files.
2344         */
2345        if ((p->p_stat == SZOMB) || (p->p_flag & SSYS) || (p->p_as == &kas)) {
2346                lxpr_unlock(p);
2347                return (NULL);
2348        }

2350        /*
2351         * get us a fresh node/vnode
2352         */
2353        lxpnp = lxpr_getnode(dp, LXPR_PID_FD_FD, p, fd);

2355        /*
2356         * get open file info
2357         */
2358        fip = (&(p)->p_user.u_finfo);
2359        mutex_enter(&fip->fi_lock);

2361        /*
2362         * got the fd data so now done with this proc
2363         */
2364        lxpr_unlock(p);

2366        if (fd < fip->fi_nfiles) {
2367                UF_ENTER(ufp, fip, fd);
2368                /*
2369                 * ensure the fd is still kosher.
2370                 * it may have gone between the readdir and
```

```
2371                  * the lookup
2372                  */
2373                 if (fip->fi_list[fd].uf_file == NULL) {
2374                         mutex_exit(&fip->fi_lock);
2375                         UF_EXIT(ufp);
2376                         lxpr_freenode(lxpnp);
2377                         return (NULL);
2378                 }

2380                 if ((fp = ufp->uf_file) != NULL)
2381                         vp = fp->f_vnode;
2382                 UF_EXIT(ufp);
2383         }
2384         mutex_exit(&fip->fi_lock);

2386         if (vp == NULL) {
2387                 lxpr_freenode(lxpnp);
2388                 return (NULL);
2389         } else {
2390                 /*
2391                  * Fill in the lxpr_node so future references will
2392                  * be able to find the underlying vnode.
2393                  * The vnode is held on the realvp.
2394                  */
2395                 lxpnp->lxpr_realvp = vp;
2396                 VN_HOLD(lxpnp->lxpr_realvp);
2397         }

2399         dp = LXPTOV(lxpnp);
2400         ASSERT(dp != NULL);

2402         return (dp);
2403 }


2406 static vnode_t *
2407 lxpr_lookup_netdir(vnode_t *dp, char *comp)
2408 {
2409         ASSERT(VTOLXP(dp)->lxpr_type == LXPR_NETDIR);

2411         dp = lxpr_lookup_common(dp, comp, NULL, netdir, NETDIRFILES);

2413         return (dp);
2414 }


2417 static vnode_t *
2418 lxpr_lookup_procdir(vnode_t *dp, char *comp)
2419 {
2420         ASSERT(VTOLXP(dp)->lxpr_type == LXPR_PROCDIR);

2422         /*
2423          * We know all the names of files & dirs in our
2424          * file system structure except those that are pid names.
2425          * These change as pids are created/deleted etc.
2426          * So just look for a number as the first char to see if we
2427          * are we doing pid lookups?
2428          *
2429          * Don't need to check for "self" as it is implemented as a symlink
2430          */
2431         if (*comp >= '0' && *comp <= '9') {
2432                 pid_t pid = 0;
2433                 lxpr_node_t *lxpnp = NULL;
2434                 proc_t *p;
2435                 int c;
```



```
2437                 while ((c = *comp++) != '\0')
2438                         pid = 10*pid + c - '0';

2440                 /*
2441                  * Can't continue if the process is still loading
2442                  * or it doesn't really exist yet (or maybe it just died!)
2443                  */
2444                 p = lxpr_lock(pid);
2445                 if (p == NULL)
2446                         return (NULL);

2448                 if (secpolicy_basic_procinfo(CRED(), p, curproc) != 0) {
2449                         lxpr_unlock(p);
2450                         return (NULL);
2451                 }

2453                 /*
2454                  * allocate and fill in a new lx /proc node
2455                  */
2456                 lxpnp = lxpr_getnode(dp, LXPR_PIDDIR, p, 0);

2458                 lxpr_unlock(p);

2460                 dp = LXPTOV(lxpnp);
2461                 ASSERT(dp != NULL);

2463                 return (dp);

2465         }

2467         /* Lookup fixed names */
2468         return (lxpr_lookup_common(dp, comp, NULL, lx_procdir, PROCDIRFILES));
2469 }




2474 /*
2475  * lxpr_readdir(): Vnode operation for VOP_READDIR()
2476  */
2477 /* ARGSUSED */
2478 static int
2479 lxpr_readdir(vnode_t *dp, uio_t *uiop, cred_t *cr, int *eofp,
2480         caller_context_t *ct, int flags)
2481 {
2482         lxpr_node_t *lxpnp = VTOLXP(dp);
2483         lxpr_nodetype_t type = lxpnp->lxpr_type;
2484         ssize_t uresid;
2485         off_t uoffset;
2486         int error;

2488         ASSERT(dp->v_type == VDIR);
2489         ASSERT(type < LXPR_NFILES);

2491         /*
2492          * we should never get here because the readdir
2493          * is done on the realvp for these nodes
2494          */
2495         ASSERT(type != LXPR_PID_FD_FD &&
2496             type != LXPR_PID_CURDIR &&
2497             type != LXPR_PID_ROOTDIR);

2499         /*
2500          * restrict readdir permission to owner or root
2501          */
2502         if ((error = lxpr_access(dp, VREAD, 0, cr, ct)) != 0)
```

```
2503                 return (error);

2505         uoffset = uiop->uio_offset;
2506         uresid = uiop->uio_resid;

2508         /* can't do negative reads */
2509         if (uoffset < 0 || uresid <= 0)
2510                 return (EINVAL);

2512         /* can't read directory entries that don't exist! */
2513         if (uoffset % LXPR_SDSIZE)
2514                 return (ENOENT);

2516         return (lxpr_readdir_function[lxpnp->lxpr_type](lxpnp, uiop, eofp));
2517 }


2520 /* ARGSUSED */
2521 static int
2522 lxpr_readdir_not_a_dir(lxpr_node_t *lxpnp, uio_t *uiop, int *eofp)
2523 {
2524         return (ENOTDIR);
2525 }

2527 /*
2528  * This has the common logic for returning directory entries
2529  */
2530 static int
2531 lxpr_readdir_common(lxpr_node_t *lxpnp, uio_t *uiop, int *eofp,
2532     lxpr_dirent_t *dirtab, int dirtablen)
2533 {
2534         /* bp holds one dirent64 structure */
2535         longlong_t bp[DIRENT64_RECLEN(LXPNSIZ) / sizeof (longlong_t)];
2536         dirent64_t *dirent = (dirent64_t *)bp;
2537         ssize_t oresid; /* save a copy for testing later */
2538         ssize_t uresid;

2540         oresid = uiop->uio_resid;

2542         /* clear out the dirent buffer */
2543         bzero(bp, sizeof (bp));

2545         /*
2546          * Satisfy user request
2547          */
2548         while ((uresid = uiop->uio_resid) > 0) {
2549                 int dirindex;
2550                 off_t uoffset;
2551                 int reclen;
2552                 int error;

2554                 uoffset = uiop->uio_offset;
2555                 dirindex  = (uoffset / LXPR_SDSIZE) - 2;

2557                 if (uoffset == 0) {

2559                         dirent->d_ino = lxpnp->lxpr_ino;
2560                         dirent->d_name[0] = '.';
2561                         dirent->d_name[1] = '\0';
2562                         reclen = DIRENT64_RECLEN(1);

2564                 } else if (uoffset == LXPR_SDSIZE) {

2566                         dirent->d_ino = lxpr_parentinode(lxpnp);
2567                         dirent->d_name[0] = '.';
2568                         dirent->d_name[1] = '.';
```



```
2569                         dirent->d_name[2] = '\0';
2570                         reclen = DIRENT64_RECLEN(2);

2572                 } else if (dirindex < dirtablen) {
2573                         int slen = strlen(dirtab[dirindex].d_name);

2575                         dirent->d_ino = lxpr_inode(dirtab[dirindex].d_type,
2576                             lxpnp->lxpr_pid, 0);

2578                         ASSERT(slen < LXPNSIZ);
2579                         (void) strcpy(dirent->d_name, dirtab[dirindex].d_name);
2580                         reclen = DIRENT64_RECLEN(slen);

2582                 } else {
2583                         /* Run out of table entries */
2584                         if (eofp) {
2585                                 *eofp = 1;
2586                         }
2587                         return (0);
2588                 }

2590                 dirent->d_off = (off64_t)(uoffset + LXPR_SDSIZE);
2591                 dirent->d_reclen = (ushort_t)reclen;

2593                 /*
2594                  * if the size of the data to transfer is greater
2595                  * that that requested then we can't do it this transfer.
2596                  */
2597                 if (reclen > uresid) {
2598                         /*
2599                          * Error if no entries have been returned yet.
2600                          */
2601                         if (uresid == oresid) {
2602                                 return (EINVAL);
2603                         }
2604                         break;
2605                 }

2607                 /*
2608                  * uiomove() updates both uiop->uio_resid and
2609                  * uiop->uio_offset by the same amount.  But we want
2610                  * uiop->uio_offset to change in increments
2611                  * of LXPR_SDSIZE, which is different from the number of bytes
2612                  * being returned to the user.
2613                  * So we set uiop->uio_offset separately, ignoring what
2614                  * uiomove() does.
2615                  */
2616                 if ((error = uiomove((caddr_t)dirent, reclen, UIO_READ, uiop)))
2617                         return (error);
2618                 }

2620                 uiop->uio_offset = uoffset + LXPR_SDSIZE;
2621         }

2623         /* Have run out of space, but could have just done last table entry */
2624         if (eofp) {
2625                 *eofp =
2626                     (uiop->uio_offset >= ((dirtablen+2) * LXPR_SDSIZE)) ? 1 : 0;
2627         }
2628         return (0);
2629 }


2632 static int
2633 lxpr_readdir_procdir(lxpr_node_t *lxpnp, uio_t *uiop, int *eofp)
2634 {
```

```
2635         /* bp holds one dirent64 structure */
2636         longlong_t bp[DIRENT64_RECLEN(LXPNSIZ) / sizeof (longlong_t)];
2637         dirent64_t *dirent = (dirent64_t *)bp;
2638         ssize_t oresid; /* save a copy for testing later */
2639         ssize_t uresid;
2640         off_t uoffset;
2641         zoneid_t zoneid;
2642         pid_t pid;
2643         int error;
2644         int ceof;

2646         ASSERT(lxpnp->lxpr_type == LXPR_PROCDIR);

2648         oresid = uiop->uio_resid;
2649         zoneid = LXPTOZ(lxpnp)->zone_id;

2651         /*
2652          * We return directory entries in the order:
2653          * "." and ".." then the unique lx procfs files, then the
2654          * directories corresponding to the running processes.
2655          *
2656          * This is a good order because it allows us to more easily
2657          * keep track of where we are betwen calls to getdents().
2658          * If the number of processes changes between calls then we
2659          * can't lose track of where we are in the lx procfs files.
2660          */

2662         /* Do the fixed entries */
2663         error = lxpr_readdir_common(lxpnp, uiop, &ceof, lx_procdir,
2664             PROCDIRFILES);

2666         /* Finished if we got an error or if we couldn't do all the table */
2667         if (error != 0 || ceof == 0)
2668                 return (error);

2670         /* clear out the dirent buffer */
2671         bzero(bp, sizeof (bp));

2673         /* Do the process entries */
2674         while ((uresid = uiop->uio_resid) > 0) {
2675                 proc_t *p;
2676                 int len;
2677                 int reclen;
2678                 int i;

2680                 uoffset = uiop->uio_offset;

2682                 /*
2683                  * Stop when entire proc table has been examined.
2684                  */
2685                 i = (uoffset / LXPR_SDSIZE) - 2 - PROCDIRFILES;
2686                 if (i >= v.v_proc) {
2687                         /* Run out of table entries */
2688                         if (eofp) {
2689                                 *eofp = 1;
2690                         }
2691                         return (0);
2692                 }
2693                 mutex_enter(&pidlock);

2695                 /*
2696                  * Skip indices for which there is no pid_entry, PIDs for
2697                  * which there is no corresponding process, a PID of 0,
2698                  * and anything the security policy doesn't allow
2699                  * us to look at.
2700                  */
```



```
2701                 if ((p = pid_entry(i)) == NULL || p->p_stat == SIDL ||
2702                     p->p_pid == 0 ||
2703                     secpolicy_basic_procinfo(CRED(), p, curproc) != 0) {
2704                         mutex_exit(&pidlock);
2705                         goto next;
2706                 }
2707                 mutex_exit(&pidlock);

2709                 /*
2710                  * Convert pid to the Linux default of 1 if we're the zone's
2711                  * init process, otherwise use the value from the proc
2712                  * structure
2713                  */
2714                 pid = ((p->p_pid != curproc->p_zone->zone_proc_initpid) ?
2715                     p->p_pid : 1);

2717                 /*
2718                  * If this /proc was mounted in the global zone, view
2719                  * all procs; otherwise, only view zone member procs.
2720                  */
2721                 if (zoneid != GLOBAL_ZONEID && p->p_zone->zone_id != zoneid) {
2722                         goto next;
2723                 }

2725                 ASSERT(p->p_stat != 0);

2727                 dirent->d_ino = lxpr_inode(LXPR_PIDDIR, pid, 0);
2728                 len = snprintf(dirent->d_name, LXPNSIZ, "%d", pid);
2729                 ASSERT(len < LXPNSIZ);
2730                 reclen = DIRENT64_RECLEN(len);

2732                 dirent->d_off = (off64_t)(uoffset + LXPR_SDSIZE);
2733                 dirent->d_reclen = (ushort_t)reclen;

2735                 /*
2736                  * if the size of the data to transfer is greater
2737                  * that that requested then we can't do it this transfer.
2738                  */
2739                 if (reclen > uresid) {
2740                         /*
2741                          * Error if no entries have been returned yet.
2742                          */
2743                         if (uresid == oresid)
2744                                 return (EINVAL);
2745                         break;
2746                 }

2748                 /*
2749                  * uiomove() updates both uiop->uio_resid and
2750                  * uiop->uio_offset by the same amount.  But we want
2751                  * uiop->uio_offset to change in increments
2752                  * of LXPR_SDSIZE, which is different from the number of bytes
2753                  * being returned to the user.
2754                  * So we set uiop->uio_offset separately, in the
2755                  * increment of this for loop, ignoring what uiomove() does.
2756                  */
2757                 if ((error = uiomove((caddr_t)dirent, reclen, UIO_READ, uiop)))
2758                         return (error);

2760 next:
2761                 uiop->uio_offset = uoffset + LXPR_SDSIZE;
2762         }

2764         if (eofp)
2765                 *eofp =
2766                     (uiop->uio_offset >=
```

```
2767                     ((v.v_proc + PROCDIRFILES + 2) * LXPR_SDSIZE)) ? 1 : 0;

2769         return (0);
2770 }


2773 static int
2774 lxpr_readdir_piddir(lxpr_node_t *lxpnp, uio_t *uiop, int *eofp)
2775 {
2776         proc_t *p;

2778         ASSERT(lxpnp->lxpr_type == LXPR_PIDDIR);

2780         /* can't read its contents if it died */
2781         mutex_enter(&pidlock);

2783         p = prfind((lxpnp->lxpr_pid == 1) ?
2784             curproc->p_zone->zone_proc_initpid : lxpnp->lxpr_pid);

2786         if (p == NULL || p->p_stat == SIDL) {
2787                 mutex_exit(&pidlock);
2788                 return (ENOENT);
2789         }
2790         mutex_exit(&pidlock);

2792         return (lxpr_readdir_common(lxpnp, uiop, eofp, piddir, PIDDIRFILES));
2793 }


2796 static int
2797 lxpr_readdir_netdir(lxpr_node_t *lxpnp, uio_t *uiop, int *eofp)
2798 {
2799         ASSERT(lxpnp->lxpr_type == LXPR_NETDIR);
2800         return (lxpr_readdir_common(lxpnp, uiop, eofp, netdir, NETDIRFILES));
2801 }


2804 static int
2805 lxpr_readdir_fddir(lxpr_node_t *lxpnp, uio_t *uiop, int *eofp)
2806 {
2807         /* bp holds one dirent64 structure */
2808         longlong_t bp[DIRENT64_RECLEN(LXPNSIZ) / sizeof (longlong_t)];
2809         dirent64_t *dirent = (dirent64_t *)bp;
2810         ssize_t oresid; /* save a copy for testing later */
2811         ssize_t uresid;
2812         off_t uoffset;
2813         int error;
2814         int ceof;
2815         proc_t *p;
2816         int fddirsize;
2817         uf_info_t *fip;


2820         ASSERT(lxpnp->lxpr_type == LXPR_PID_FDDIR);

2822         oresid = uiop->uio_resid;

2824         /* can't read its contents if it died */
2825         p = lxpr_lock(lxpnp->lxpr_pid);
2826         if (p == NULL)
2827                 return (ENOENT);

2829         /* Get open file info */
2830         fip = (&(p)->p_user.u_finfo);

2832         if ((p->p_stat == SZOMB) || (p->p_flag & SSYS) || (p->p_as == &kas))
```



```
2833                 fddirsize = 0;
2834         else
2835                 fddirsize = fip->fi_nfiles;

2837         mutex_enter(&fip->fi_lock);
2838         lxpr_unlock(p);

2840         /* Do the fixed entries (in this case just "." & "..") */
2841         error = lxpr_readdir_common(lxpnp, uiop, &ceof, 0, 0);

2843         /* Finished if we got an error or if we couldn't do all the table */
2844         if (error != 0 || ceof == 0)
2845                 return (error);

2847         /* clear out the dirent buffer */
2848         bzero(bp, sizeof (bp));

2850         /*
2851          * Loop until user's request is satisfied or until
2852          * all file descriptors have been examined.
2853          */
2854         for (; (uresid = uiop->uio_resid) > 0;
2855             uiop->uio_offset = uoffset + LXPR_SDSIZE) {
2856                 int reclen;
2857                 int fd;
2858                 int len;

2860                 uoffset = uiop->uio_offset;

2862                 /*
2863                  * Stop at the end of the fd list
2864                  */
2865                 fd = (uoffset / LXPR_SDSIZE) - 2;
2866                 if (fd >= fddirsize) {
2867                         if (eofp) {
2868                                 *eofp = 1;
2869                         }
2870                         goto out;
2871                 }

2873                 if (fip->fi_list[fd].uf_file == NULL)
2874                         continue;

2876                 dirent->d_ino = lxpr_inode(LXPR_PID_FD_FD, lxpnp->lxpr_pid, fd);
2877                 len = snprintf(dirent->d_name, LXPNSIZ, "%d", fd);
2878                 ASSERT(len < LXPNSIZ);
2879                 reclen = DIRENT64_RECLEN(len);

2881                 dirent->d_off = (off64_t)(uoffset + LXPR_SDSIZE);
2882                 dirent->d_reclen = (ushort_t)reclen;

2884                 if (reclen > uresid) {
2885                         /*
2886                          * Error if no entries have been returned yet.
2887                          */
2888                         if (uresid == oresid)
2889                                 error = EINVAL;
2890                         goto out;
2891                 }

2893                 if ((error = uiomove((caddr_t)dirent, reclen, UIO_READ, uiop)))
2894                         goto out;
2895         }

2897         if (eofp)
2898                 *eofp =
```

```
2899                     (uiop->uio_offset >= ((fddirsize+2) * LXPR_SDSIZE)) ? 1 : 0;

2901 out:
2902         mutex_exit(&fip->fi_lock);
2903         return (error);
2904 }


2907 /*
2908  * lxpr_readlink(): Vnode operation for VOP_READLINK()
2909  */
2910 /* ARGSUSED */
2911 static int
2912 lxpr_readlink(vnode_t *vp, uio_t *uiop, cred_t *cr, caller_context_t *ct)
2913 {
2914         char bp[MAXPATHLEN + 1];
2915         size_t buflen = sizeof (bp);
2916         lxpr_node_t *lxpnp = VTOLXP(vp);
2917         vnode_t *rvp = lxpnp->lxpr_realvp;
2918         pid_t pid;
2919         int error = 0;

2921         /* must be a symbolic link file */
2922         if (vp->v_type != VLNK)
2923                 return (EINVAL);

2925         /* Try to produce a symlink name for anything that has a realvp */
2926         if (rvp != NULL) {
2927                 if ((error = lxpr_access(vp, VREAD, 0, CRED(), ct)) != 0)
2928                         return (error);
2929                 if ((error = vnodetopath(NULL, rvp, bp, buflen, CRED())) != 0)
2930                         return (error);
2931         } else {
2932                 switch (lxpnp->lxpr_type) {
2933                 case LXPR_SELF:
2934                         /*
2935                          * Don't need to check result as every possible int
2936                          * will fit within MAXPATHLEN bytes
2937                          */

2939                         /*
2940                          * Convert pid to the Linux default of 1 if we're the
2941                          * zone's init process
2942                          */
2943                         pid = ((curproc->p_pid !=
2944                             curproc->p_zone->zone_proc_initpid)
2945                             ? curproc->p_pid : 1);

2947                         (void) snprintf(bp, buflen, "%d", pid);
2948                         break;
2949                 case LXPR_PID_CURDIR:
2950                 case LXPR_PID_ROOTDIR:
2951                 case LXPR_PID_EXE:
2952                         return (EACCES);
2953                 default:
2954                         /*
2955                          * Need to return error so that nothing thinks
2956                          * that the symlink is empty and hence "."
2957                          */
2958                         return (EINVAL);
2959                 }
2960         }

2962         /* copy the link data to user space */
2963         return (uiomove(bp, strlen(bp), UIO_READ, uiop));
2964 }
```



```
2967 /*
2968  * lxpr_inactive(): Vnode operation for VOP_INACTIVE()
2969  * Vnode is no longer referenced, deallocate the file
2970  * and all its resources.
2971  */
2972 /* ARGSUSED */
2973 static void
2974 lxpr_inactive(vnode_t *vp, cred_t *cr, caller_context_t *ct)
2975 {
2976         lxpr_freenode(VTOLXP(vp));
2977 }


2980 /*
2981  * lxpr_sync(): Vnode operation for VOP_SYNC()
2982  */
2983 static int
2984 lxpr_sync()
2985 {
2986         /*
2987          * nothing to sync but this
2988          * function must never fail
2989          */
2990         return (0);
2991 }


2994 /*
2995  * lxpr_cmp(): Vnode operation for VOP_CMP()
2996  */
2997 static int
2998 lxpr_cmp(vnode_t *vp1, vnode_t *vp2, caller_context_t *ct)
2999 {
3000         vnode_t *rvp;

3002         while (vn_matchops(vp1, lxpr_vnodeops) &&
3003             (rvp = VTOLXP(vp1)->lxpr_realvp) != NULL)
3004                 vp1 = rvp;
3005         while (vn_matchops(vp2, lxpr_vnodeops) &&
3006             (rvp = VTOLXP(vp2)->lxpr_realvp) != NULL)
3007                 vp2 = rvp;
3008         if (vn_matchops(vp1, lxpr_vnodeops) || vn_matchops(vp2, lxpr_vnodeops))
3009                 return (vp1 == vp2);
3010         return (VOP_CMP(vp1, vp2, ct));
3011 }


3014 /*
3015  * lxpr_realvp(): Vnode operation for VOP_REALVP()
3016  */
3017 static int
3018 lxpr_realvp(vnode_t *vp, vnode_t **vpp, caller_context_t *ct)
3019 {
3020         vnode_t *rvp;

3022         if ((rvp = VTOLXP(vp)->lxpr_realvp) != NULL) {
3023                 vp = rvp;
3024                 if (VOP_REALVP(vp, &rvp, ct) == 0)
3025                         vp = rvp;
3026         }

3028         *vpp = vp;
3029         return (0);
3030 }
```

```
3031 #endif /* ! codereview */
```

```
**********************************************************
    4008 Tue Jan 14 16:17:20 2014
new/usr/src/uts/common/brand/lx/sys/ldlinux.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _SYS_LDLINUX_H
#define _SYS_LDLINUX_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

/*
 * The ldlinux streams module is only intended for use in lx branded zones.
 * This streams module implements the following ioctls:
 *      TIOCSETLD and TIOCGETLD
 *
 * These ioctls are special ioctls supported only by the ldlinux streams
 * module and invoked only by the lx brand emulation library.  These ioctls
 * do not exist on native Linux systems.
 *
 * The TIOCSETLD ioctl is used when emulating the following Linux ioctls:
 *      TCSETS/TCSETSW/TCSETSF
 *      TCSETA/TCSETAW/TCSETAF
 *
 * The TIOCGETLD ioctl is used when emulating the following Linux ioctls:
 *      TCGETS/TCGETA
 *
 * This module is needed to emulate these ioctls because the following arrays:
 *      termio.c_cc
 *      termios.c_cc
 * which are parameters for the following ioctls:
 *      TCSETS/TCSETSW/TCSETSF
 *      TCSETA/TCSETAW/TCSETAF
 *      TCGETS/TCGETA
 *
 * are defined differently on Solaris and Linux.
 *
 * According to the termio(7I) man page on Solaris the following is true of
 * the members of the c_cc array:
 *      The VMIN element is the same element as the VEOF element.
 *      The VTIME element is the same element as the VEOL element.
 *
```



```
 * But on Linux the termios(3) man page states:
 *      These symbolic subscript values are all different, except that
 *      VTIME, VMIN may have the same value as VEOL, VEOF, respectively.
 *
 * While the man page indicates that these values may be the same empirical
 * tests shows them to be different.  Since these values are different on
 * Linux systems it's possible that applications could set the members of
 * the c_cc array to different values and then later expect to be able to
 * read back those same separate values.  The ldlinux module exists to provide
 * a per-stream storage area where the lx_brand emulation library can save
 * these values.  The values are set and retrieved via the TIOCSETLD and
 * TIOCGETLD ioctls respectively.
 */

#include <sys/termios.h>

#ifdef  __cplusplus
extern "C" {
#endif

#define LDLINUX_MOD     "ldlinux"

#ifdef _KERNEL

/*
 * LDLINUX_MODID - This should be a unique number associated with
 * this particular module.  Unfortunatly there is no authority responsible
 * for administering this name space, hence there's no real guarantee that
 * whatever number we choose will be unique.  Luckily, this constant
 * is not really used anywhere by the system.  It is used by some
 * kernel subsystems to check for the presence of certain streams
 * modules with known id vaules.  Since no other kernel subsystem
 * checks for the presence of this module we'll just set the id to 0.
 */
#define LDLINUX_MODID   0

struct ldlinux {
        int     state;          /* state information */
                                /* Linux expects the next four c_cc values */
                                /* to be distinct, whereas solaris (legally) */
                                /* overlaps their storage */
        unsigned char veof;     /* veof value */
        unsigned char veol;     /* veol value */
        unsigned char vmin;     /* vmin value */
        unsigned char vtime;    /* vtime value */
};

#define ISPTSTTY        0x01

#endif /* _KERNEL */

#ifdef  __cplusplus
}
#endif

#endif /* _SYS_LDLINUX_H */
#endif /* ! codereview */
```

```
**********************************************************
    3864 Tue Jan 14 16:17:20 2014
new/usr/src/uts/common/brand/lx/sys/lx_audio.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _LX_AUDIO_H
#define _LX_AUDIO_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/zone.h>

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * name for this driver
 */
#define LX_AUDIO_DRV                    "lx_audio"

/*
 * names for the minor nodes this driver exports
 */
#define LXA_MINORNAME_DEVCTL            "lx_devctl"
#define LXA_MINORNAME_DSP               "lx_dsp"
#define LXA_MINORNAME_MIXER             "lx_mixer"

/*
 * minor numbers for the minor nodes this driver exporrts
 */
#define LXA_MINORNUM_DEVCTL             0
#define LXA_MINORNUM_DSP                1
#define LXA_MINORNUM_MIXER              2
#define LXA_MINORNUM_COUNT              3

/*
 * driver ioctls
 *
 * note that we're layering on top of solaris audio devices so we want
 * to make sure that our ioctls namespace doesn't conflict with theirs.
```



```
 * looking in sys/audioio.h and sys/mixer.h we see that they seem to
 * use an _IO key of 'A' and 'M', so we'll choose an _IO key of 'a.'
 */

/*
 * administrative ioctls.
 * these ioctls are only supported on the DEVCTL minor node
 */
#define LXA_IOC_ZONE_REG                (_IOR('a', 0, lxa_zone_reg_t))
#define LXA_IOC_ZONE_UNREG              (_IOR('a', 1, lxa_zone_reg_t))


/*
 * audio and mixer device ioctls
 * these ioctls are supported on DSP and MIXER minor nodes.
 */
#define LXA_IOC_GETMINORNUM             (_IOR('a', 20, int))

/*
 * audio device ioctls.
 * these ioctls are supports on DSP minor nodes.
 */
#define LXA_IOC_MMAP_OUTPUT             (_IOR('a', 41, int))
#define LXA_IOC_MMAP_PTR                (_IOR('a', 42, int))
#define LXA_IOC_GET_FRAG_INFO           (_IOR('a', 43, lxa_frag_info_t))
#define LXA_IOC_SET_FRAG_INFO           (_IOR('a', 44, lxa_frag_info_t))

/*
 * mixer device ioctls.
 * these ioctls are supports on MIXER minor nodes.
 */
#define LXA_IOC_MIXER_GET_VOL           (_IOR('a', 60, lxa_mixer_levels_t))
#define LXA_IOC_MIXER_SET_VOL           (_IOR('a', 61, lxa_mixer_levels_t))
#define LXA_IOC_MIXER_GET_MIC           (_IOR('a', 62, lxa_mixer_levels_t))
#define LXA_IOC_MIXER_SET_MIC           (_IOR('a', 63, lxa_mixer_levels_t))
#define LXA_IOC_MIXER_GET_PCM           (_IOR('a', 64, lxa_mixer_levels_t))
#define LXA_IOC_MIXER_SET_PCM           (_IOR('a', 65, lxa_mixer_levels_t))

/* command structure for LXA_IOC_ZONE_REG */
#define LXA_INTSTRLEN 32
typedef struct lxa_zone_reg {
        char    lxa_zr_zone_name[ZONENAME_MAX];
        char    lxa_zr_inputdev[LXA_INTSTRLEN];
        char    lxa_zr_outputdev[LXA_INTSTRLEN];
} lxa_zone_reg_t;

/* command structure for LXA_IOC_GET_FRAG_INFO and LXA_IOC_SET_FRAG_INFO */
typedef struct lxa_frag_info {
        int     lxa_fi_size;
        int     lxa_fi_cnt;
} lxa_frag_info_t;

/* command structure for LXA_IOC_MIXER_GET_* and LXA_IOC_MIXER_SET_* */
typedef struct lxa_mixer_levels {
        int     lxa_ml_gain;
        int     lxa_ml_balance;
} lxa_mixer_levels_t;

/* verify that a solaris mixer level structure has valid values */
#define LXA_MIXER_LEVELS_OK(x) (((x)->lxa_ml_gain >= AUDIO_MIN_GAIN) && \
                                ((x)->lxa_ml_gain <= AUDIO_MAX_GAIN) && \
                                ((x)->lxa_ml_balance >= AUDIO_LEFT_BALANCE) && \
                                ((x)->lxa_ml_balance <= AUDIO_RIGHT_BALANCE))

#ifdef  __cplusplus
}
```

```
128 #endif

130 #endif  /* _LX_AUDIO_H */
131 #endif /* ! codereview */
```

```
**********************************************************
   15944 Tue Jan 14 16:17:20 2014
new/usr/src/uts/common/brand/lx/sys/lx_autofs.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _LX_AUTOFS_H
#define _LX_AUTOFS_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

/*
 * The lx_autofs filesystem exists to emulate the Linux autofs filesystem
 * and provide support for the Linux "automount" automounter.
 *
 *
 *
 * +++ Linux automounter background.
 *
 * Linux has two automounters: "amd" and "automount"
 *
 * 1) "amd" is a userland NFS server.  It basically mounts an NFS filesystem
 * at an automount point, and it acts as the NFS server for the mount.  When
 * an access is done to that NFS filesystem, the access is redirected by the
 * kernel to the "amd" process via rpc.  "amd" then looks up any information
 * required to resolve the requests, mounts real NFS filesystems if
 * necessary, and returns.  "amd" has it's own strange configuration
 * mechanism that doesn't seem to be very compatabile with Solaris's network
 * based automounter map support.
 *
 * 2) "automount" is the other Linux automounter.  It utilizes a kernel
 * filesystem (autofs) to provide it's functionality.  Basically, it mounts
 * the autofs filesystem at any automounter controlled mount point.  This
 * filesystem then intercepts and redirects lookup operations (and only
 * lookup ops) to the userland automounter process via a pipe.  (The
 * pipe to the automounter is establised via mount options when the autofs
 * filesystem is mounted.)  When the automounter recieves a request via this
 * pipe, it does lookups to whatever backing store it's configured to use,
 * does mkdir operations on the autofs filesystem, mounts remote NFS
 * filesystems on any leaf directories it just created, and signals the
 * autofs filesystem via an ioctl to let it know that the lookup can
```



```
 * continue.
 *
 *
 *
 * +++ Linux autofs (and automount daemon) notes
 *
 * Since we're mimicking the behavior of the Linux autofs filesystem it's
 * important to document some of it's observed behavior here since there's
 * no doubt that in the future this behavior will change.  These comments
 * apply to the behavior of the automounter as observed on a system
 * running Linux v2.4.21 (autofs is bundled with the Linux kernel).
 *
 * A) Autofs allows root owned, non-automounter processes to create
 * directories in the autofs filesystem.  The autofs filesystem treats the
 * automounter's process group as special, but it doesn't prevent root
 * processes outside of the automounter's process group from creating new
 * directories in the autofs filesystem.
 *
 * B) Autofs doesn't allow creation of any non-directory entries in the
 * autofs filesystem.  No entity can create files (e.g. /bin/touch or
 * VOP_CREATE/VOP_SYMLINK/etc.)  The only entries that can exist within
 * the autofs filesystem are directories.
 *
 * C) Autofs only intercepts vop lookup operations.  Notably, it does _not_
 * intercept and re-direct vop readdir operations.  This means that the
 * observed behavior of the Linux automounter can be considerably different
 * from that of the Solaris automounter.  Specifically, on Solaris if autofs
 * mount point is mounted _without_ the -nobrowse option then if a user does
 * an ls operation (which translates into a vop readdir operation) then the
 * automounter will intercept that operation and list all the possible
 * directories and mount points without actually mounting any filesystems.
 * Essentially, all automounter managed mount points on Linux will behave
 * like "-nobrowse" mount points on Solaris.  Here's an example to
 * illustrate this.  If /ws was mounted on Solaris without the -nobrowse
 * option and an auto_ws yp map was setup as the backing store for this
 * mount point, then an "ls /ws" would list all the keys in the map as
 * valid directories, but an "ls /ws" on Linux would list an emptry
 * directory.
 *
 * D) NFS mounts are performed by the automount process.  When the automount
 * process gets a redirected lookup request, it determines _all_ the
 * possible remote mount points for that request, creates directory paths
 * via mkdir, and mounts the remote filesystems on the newly created paths.
 * So for example, if a machine called mcescher exported /var/crash and
 * /var/core, an "ls /net/mcescher" would result in the following actions
 * being done by the automounter:
 *      mkdir /net/mcescher
 *      mkdir /net/mcescher/var
 *      mkdir /net/mcescher/var/crash
 *      mkdir /net/mcescher/var/core
 *      mount mcescher:/var/crash /var/crash
 *      mount mcescher:/var/crash /var/core
 * once the automounter compleated the work above it would signal the autofs
 * filesystem (via an ioctl) that the lookup could continue.
 *
 * E.1) Autofs only redirects vop lookup operations for path entries that
 * don't already exist in the autofs filesystem.  So for the example above,
 * an initial (after the start of the automounter) "ls /net/mcescher" would
 * result in a request to the automounter.  A subsequest "ls /net/mcescher"
 * would not result in a request to the automounter.  Even if
 * /net/mcescher/var/crash and /net/mcescher/var/core were manually unmounted
 * after the initial "ls /net/mcescher", a subsequest "ls /net/mcescher"
 * would not result in a new request to the automounter.
 *
 * E.2) Autofs lookup requests that are sent to the automounter only include
 * the root directory path component.  So for example, after starting up
```

```
 * the automounter if a user were to do a "ls /net/mcescher/var/crash", the
 * lookup request actually sent to the automounter would just be for
 * "mcescher".  (The same request as if the user had done "ls /net/mcescher".)
 *
 * E.3) The two statements above aren't entirely entirely true.  The Linux
 * autofs filesystem will also redirect lookup operations for leaf
 * directories that don't have a filesystem mounted on them.  Using the
 * example above, if a user did a "ls /net/mcescher", then manually
 * unmounted /net/mcescher/var/crash, and then did an "ls
 * /net/mcescher/var/crash", this would result in a request for
 * "mcescher/var/crash" being sent to the automounter.  The strange thing
 * (a Linux bug perhaps) is that the automounter won't do anything with this
 * request and the lookup will fail.
 *
 * F) The autofs filesystem communication protocol (what ioctls it supports
 * and what data it passes to the automount process) are versioned.  The
 * source for the userland automount daemon (i looked at version v3.1.7)
 * seemed to support two versions of the Linux kernel autofs implementation.
 * Both versions supported communiciation with a pipe and the format of the
 * structure passed via this pipe was the same.  The difference between the
 * two versions was in the functionality supported.  (The v3 version has
 * additional ioctls to support automount timeouts.)
 *
 *
 *
 * +++ lx_autofs notes
 *
 * 1) In general, the lx_autofs filesystem tries to mimic the behavior of the
 * Linux autofs filesystem with the following exceptions:
 *
 *      1.1) We don't bother to implement the E.3 functionality listed above
 *      since it doesn't appear to be of any use.
 *
 *      1.2) We only implement v2 of the automounter protocol since
 *      implementing v3 would take a _lot_ more work.  If this proves to be a
 *      problem we can re-visit this decision later.  (More details about v3
 *      support are included in comments below.)
 *
 * 2) In general, the approach taken for lx_autofs is to keep it as simple
 * as possible and to minimize it's memory usage.  To do this all information
 * about the contents of the lx_autofs filesystem are mirrored in the
 * underlying filesystem that lx_autofs is mounted on and most vop operations
 * are simply passed onto this underlying filesystem.  This means we don't
 * have to implement most the complex operations that a full filesystem
 * normally has to implement.  It also means that most of our filesystem state
 * (wrt the contents of the filesystem) doesn't actually have to be stored
 * in memory, we can simply go to the underlying filesystem to get it when
 * it's requested.  For the purposes of discussion, we'll call the underlying
 * filesystem the "backing store."
 *
 * The backing store is actually directory called ".lx_afs" which is created in
 * the directory where the lx_autofs filesystem is mounted.  When the lx_autofs
 * filesystem is unmounted this backing store directory is deleted.  If this
 * directory exists at mount time (perhaps the system crashed while a previous
 * lx_autofs instance was mounted at the same location) it will be deleted.
 * There are a few implications of using a backing store worth mentioning.
 *
 *      2.1) lx_autofs can't be mounted on a read only filesystem.  If this
 *      proves to be a problem we can probably move the location of the
 *      backing store.
 *
 *      2.2) If the backing store filesystem runs out of space then the
 *      automounter process won't be able to create more directories and mount
 *      new filesystems.  Of course, strange failures usually happen when
 *      filesystems run out of space.
 *
```



```
 * 3) Why aren't we using gfs?  gfs has two different usage models.
 *
 *      3.1) I'm my own filesystem but i'm using gfs to help with managing
 *      readdir operations.
 *
 *      3.2) I'm a gfs filesystem and gfs is managing all my vnodes
 *
 * We're not using the 3.1 interfaces because we don't implement readdir
 * ourselves.  We pass all readdir operations onto the backing store
 * filesystem and utilize its readdir implementation.
 *
 * We're not using the 3.2 interfaces because they are really designed for
 * in memory filesystems where all of the filesystem state is stored in
 * memory.  They don't lend themselves to filesystems where part of the
 * state is in memory and part of the state is on disk.
 *
 * For more information on gfs take a look at the block comments in the
 * top of gfs.c
 */

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * Note that the name of the actual Solaris filesystem is lx_afs and not
 * lx_autofs.  This is becase filesystem names are stupidly limited to 8
 * characters.
 */
#define LX_AUTOFS_NAME                  "lx_afs"

/*
 * Mount options supported.
 */
#define LX_MNTOPT_FD                    "fd"
#define LX_MNTOPT_PGRP                  "pgrp"
#define LX_MNTOPT_MINPROTO              "minproto"
#define LX_MNTOPT_MAXPROTO              "maxproto"

/* Version of the Linux kernel automount protocol we support. */
#define LX_AUTOFS_PROTO_VERSION         2

/*
 * Command structure sent to automount process from lx_autofs via a pipe.
 * This structure is the same for v2 and v3 of the automount protocol
 * (the communication pipe is established at mount time).
 */
typedef struct lx_autofs_pkt {
        int     lap_protover;   /* protocol version number */
        int     lap_constant;   /* always set to 0 */
        int     lap_id;         /* every pkt must have a unique id */
        int     lap_name_len;   /* don't include newline or NULL */
        char    lap_name[256];  /* path component to lookup */
} lx_autofs_pkt_t;

/*
 * Ioctls supprted (v2 protocol).
 */
#define LX_AUTOFS_IOC_READY             0x00009360 /* arg: int */
#define LX_AUTOFS_IOC_FAIL              0x00009361 /* arg: int */
#define LX_AUTOFS_IOC_CATATONIC         0x00009362 /* arg: <none> */

/*
 * Ioctls not supported (v3 protocol).
 *
 * Initially we're only going to support v2 of the Linux kernel automount
```

```
 * protocol.  This means that we don't support the following ioctls.
 *
 * 1) The protocol version ioctl (by not supporting it the automounter
 * will assume version 2).
 *
 * 2) Automounter timeout ioctls.  For v3 and later the automounter can
 * be started with a timeout option.  It will notify the filesystem of
 * this timeout and, if any automounter filesystem root directory entry
 * is not in use, it will notify the automounter via the LX_AUTOFS_IOC_EXPIRE
 * ioctl.  For example, if the timeout is 60 seconds, the Linux
 * automounter will use the LX_AUTOFS_IOC_EXPIRE ioctl to query for
 * timeouts more often than that.  (v3.1.7 of the automount daemon would
 * perform this ioctl every <timeout>/4 seconds.)  Then, if the autofs
 * filesystem will
 * report top level directories that aren't in use to the automounter
 * via this ioctl.  If /net was managed by the automounter and
 * there were the following mount points:
 *      /net/jurassic/var/crash
 *      /net/mcescher/var/crash
 * and no one was looking at any crash dumps on mcescher but someone
 * was analyzing a crash dump on jurassic, then after <timeout> seconds
 * had passed the autofs filesystem would let the automounter know that
 * "mcescher" could be unmounted.  (Note the granularity of notification
 * is directories in the root of the autofs filesystem.)  Here's two
 * ideas for how this functionality could be implemented on Solaris:
 *
 * 2.1) The easy incomplete way.  Don't do any in-use detection.  Simply
 * tell the automounter it can try to unmount the filesystem every time
 * the specified timeout passes.  If the filesystem is in use then the
 * unmount will fail.  This would break down for remote hosts with multiple
 * mounts.  For example, if the automounter had mounted the following
 * filesystems:
 *      /net/jurassic/var/crash
 *      /net/jurassic/var/core
 * and the user was looking at a core file, and the timeout expired, the
 * automounter would recieve notification to unmount "jurassic".  Then
 * it would unmount crash (which would succeed) and then to try unmount
 * core (which would fail).  After that (since the automounter only
 * performs mounts for failed lookups in the root autofs directory)
 * future access to /net/jurassic/var/crash would result to access
 * to an empty autofs directory.  We might be able to work around
 * this by caching which root autofs directories we've timed out,
 * then any access to paths that contain those directories could be
 * stalled and we could resend another request to the automounter.
 * This could work if the automounter ignores mount failures.
 *
 * 2.2) The hard correct way.  The real difficulty here is detecting
 * files in use on other filesystems (say NFS) that have been mounted
 * on top of autofs.  (Detecting in use autofs vnodes should be easy.)
 * to do this we would probably have to create a new brand op to intercept
 * mount/umount filesystem operations.  Then using this entry point we
 * could detect mounts of other filesystems on top of lx_autofs.  When
 * a successful mount finishes we would use the FEM (file event
 * monitoring) framework to push a module onto that filesystem and
 * intercept VOP operations that allocate/free vnodes in that filesystem.
 * (We would also then have to track mount operations on top of that
 * filesystem, etc.)  this would allow us to properly detect any
 * usage of subdirectories of an autofs directory.
 */
#define LX_AUTOFS_IOC_PROTOVER          0x80049363 /* arg: int */
#define LX_AUTOFS_IOC_EXPIRE            0x81109365 /* arg: lx_autofs_expire * */
#define LX_AUTOFS_IOC_SETTIMEOUT        0xc0049364 /* arg: ulong_t */

typedef struct lx_autofs_expire {
        int     lap_protover;   /* protol version number */
        int     lap_constant;   /* always set to 1 */
```



```
        int     lap_name_len;   /* don't include newline or NULL */
        char    lap_name[256];  /* path component that has timed out */
} lx_autofs_expire_t;

#ifdef  __cplusplus
}
#endif

#endif  /* _LX_AUTOFS_H */
#endif /* ! codereview */
```

```
**********************************************************
    2943 Tue Jan 14 16:17:21 2014
new/usr/src/uts/common/brand/lx/sys/lx_autofs_impl.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _LX_AUTOFS_IMPL_H
#define _LX_AUTOFS_IMPL_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

#ifdef  __cplusplus
extern "C" {
#endif

#include <sys/file.h>
#include <sys/id_space.h>
#include <sys/modhash.h>
#include <sys/vnode.h>

#include <sys/lx_autofs.h>

/*
 * Space key.
 * Used to persist data across lx_autofs filesystem module unloads.
 */
#define LX_AUTOFS_SPACE_KEY_UDEV        LX_AUTOFS_NAME "_udev"

/*
 * Name of the backing store directory.
 */
#define LX_AUTOFS_BS_DIR                "." LX_AUTOFS_NAME

#define LX_AUTOFS_VFS_ID_HASH_SIZE      15
#define LX_AUTOFS_VFS_PATH_HASH_SIZE    15
#define LX_AUTOFS_VFS_VN_HASH_SIZE      15

/*
 * VFS data object.
 */
typedef struct lx_autofs_vfs {
```



```
        /* Info about the underlying filesystem and backing store. */
        vnode_t         *lav_mvp;
        char            *lav_bs_name;
        vnode_t         *lav_bs_vp;

        /* Info about the automounter process managing this filesystem. */
        int             lav_fd;
        pid_t           lav_pgrp;
        file_t          *lav_fifo_wr;
        file_t          *lav_fifo_rd;

        /* Each automount requests needs a unique id. */
        id_space_t      *lav_ids;

        /* All remaining structure members are protected by lav_lock. */
        kmutex_t        lav_lock;

        /* Hashes to keep track of outstanding automounter requests. */
        mod_hash_t      *lav_path_hash;
        mod_hash_t      *lav_id_hash;

        /* We need to keep track of all our vnodes. */
        vnode_t         *lav_root;
        mod_hash_t      *lav_vn_hash;
} lx_autofs_vfs_t;

/*
 * Structure to keep track of requests sent to the automounter.
 */
typedef struct lx_autofs_lookup_req {
        /* Packet that gets sent to the automounter. */
        lx_autofs_pkt_t lalr_pkt;

        /* Reference count.  Always updated atomically. */
        uint_t          lalr_ref;

        /*
         * Fields to keep track and sync threads waiting on a lookup.
         * Fields are protected by lalr_lock.
         */
        kmutex_t        lalr_lock;
        kcondvar_t      lalr_cv;
        int             lalr_complete;
} lx_autofs_lookup_req_t;

/*
 * Generic stack structure.
 */
typedef struct stack_elem {
        list_node_t     se_list;
        caddr_t         se_ptr1;
        caddr_t         se_ptr2;
        caddr_t         se_ptr3;
} stack_elem_t;

#ifdef  __cplusplus
}
#endif

#endif  /* _LX_AUTOFS_IMPL_H */
#endif /* ! codereview */
```

```
**********************************************************
    6230 Tue Jan 14 16:17:21 2014
new/usr/src/uts/common/brand/lx/sys/lx_brand.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _LX_BRAND_H
#define _LX_BRAND_H

#ifndef _ASM
#include <sys/types.h>
#include <sys/cpuvar.h>
#include <sys/zone.h>
#endif

#ifdef  __cplusplus
extern "C" {
#endif

#define LX_BRANDNAME    "lx"

/*
 * Brand uname info
 */
#define LX_UNAME_SYSNAME        "Linux"
#define LX_UNAME_RELEASE_2_6    "2.6.18"
#define LX_UNAME_RELEASE_2_4    "2.4.21"
#define LX_UNAME_VERSION        "BrandZ fake linux"
#define LX_UNAME_MACHINE        "i686"

#define LX_LINKER_NAME "ld-linux.so.2"
#define LX_LINKER       "/lib/" LX_LINKER_NAME
#define LX_LIBC_NAME    "libc.so.6"
#define LIB_PATH        "/native/usr/lib/"
#define LX_LIB          "lx_brand.so.1"
#define LX_LIB_PATH     LIB_PATH LX_LIB

#define LX_NSYSCALLS_2_4        270
#define LX_NSYSCALLS_2_6        317
#define LX_NSYSCALLS    LX_NSYSCALLS_2_6

#define LX_KERN_2_4     0
```



```
#define LX_KERN_2_6     1

/*
 * brand(2) subcommands
 *
 * Everything >= 128 is a brand-specific subcommand.
 * 192 to 462 are reserved for system calls, although most of that space is
 * unused.
 */
#define B_LPID_TO_SPAIR         128
#define B_SYSENTRY              129
#define B_SYSRETURN             130
#define B_PTRACE_SYSCALL        131
#define B_SET_AFFINITY_MASK     132
#define B_GET_AFFINITY_MASK     133

#define B_EMULATE_SYSCALL       192

#define LX_VERSION_1            1
#define LX_VERSION              LX_VERSION_1

#define LX_ATTR_RESTART_INIT    ZONE_ATTR_BRAND_ATTRS
#define LX_KERN_VERSION_NUM     (ZONE_ATTR_BRAND_ATTRS + 1)

/* Aux vector containing phdr of linux executable, used by lx_librtld_db */
#define AT_SUN_BRAND_LX_PHDR    AT_SUN_BRAND_AUX1

/* Aux vector containing hz value */
#define AT_CLKTCK       17

#ifndef _ASM

typedef struct lx_brand_registration {
        uint_t lxbr_version;            /* version number */
        void *lxbr_handler;             /* base address of handler */
        void *lxbr_tracehandler;        /* base address of trace handler */
        void *lxbr_traceflag;           /* address of trace flag */
} lx_brand_registration_t;

#ifdef  _SYSCALL32
typedef struct lx_brand_registration32 {
        uint32_t lxbr_version;          /* version number */
        caddr32_t lxbr_handler;         /* base address of handler */
        caddr32_t lxbr_tracehandler;    /* base address of trace handler */
        caddr32_t lxbr_traceflag;       /* address of trace flag */
} lx_brand_registration32_t;
#endif

typedef struct lx_regs {
        long lxr_gs;
        long lxr_edi;
        long lxr_esi;
        long lxr_ebp;
        long lxr_esp;
        long lxr_ebx;
        long lxr_edx;
        long lxr_ecx;
        long lxr_eax;
        long lxr_eip;

        long lxr_orig_eax;
} lx_regs_t;

#endif /* _ASM */

/*
```

```
 * GDT usage
 */
#define GDT_TLSMIN      (GDT_BRANDMIN)
#define GDT_TLSMAX      (GDT_TLSMIN + 2)
#define LX_TLSNUM       (GDT_TLSMAX - GDT_TLSMIN)

#ifndef _ASM

/*
 * Stores information needed by the lx linker to launch the main
 * lx executable.
 */
typedef struct lx_elf_data {
        int     ed_phdr;
        int     ed_phent;
        int     ed_phnum;
        int     ed_entry;
        int     ed_base;
        int     ed_ldentry;
} lx_elf_data_t;

#ifdef  _KERNEL

typedef struct lx_proc_data {
        uintptr_t l_handler;    /* address of user-space handler */
        uintptr_t l_tracehandler; /* address of user-space traced handler */
        uintptr_t l_traceflag;  /* address of 32-bit tracing flag */
        void (*l_sigrestorer[MAXSIG])(void); /* array of sigrestorer fns */
        pid_t l_ppid;           /* pid of originating parent proc */
        uint64_t l_ptrace;      /* process being observed with ptrace */
        lx_elf_data_t l_elf_data; /* ELF data for linux executable */
} lx_proc_data_t;

#endif  /* _KERNEL */

/*
 * A data type big enough to bitmap all Linux possible cpus.
 * The bitmap size is defined as 1024 cpus in the Linux 2.4 and 2.6 man pages
 * for sched_getaffinity() and sched_getaffinity().
 */
#define LX_NCPU         (1024)
#define LX_AFF_ULONGS   (LX_NCPU / (8 * sizeof (ulong_t)))
typedef ulong_t lx_affmask_t[LX_AFF_ULONGS];

#ifdef  _KERNEL

/*
 * lx-specific data in the klwp_t
 */
typedef struct lx_lwp_data {
        uint_t  br_lwp_flags;           /* misc. flags */
        klwp_t  *br_lwp;                /* back pointer to container lwp */
        int     br_signal;              /* signal to send to parent when */
                                        /* clone()'ed child terminates */
        int     br_exitwhy;             /* reason for thread (process) exit */
        int     br_exitwhat;            /* exit code / killing signal */
        lx_affmask_t br_affinitymask;   /* bitmask of CPU sched affinities */
        struct user_desc br_tls[LX_TLSNUM];
                        /* descriptors used by libc for TLS */
        pid_t   br_pid;                 /* converted pid for this thread */
        pid_t   br_tgid;                /* thread group ID for this thread */
        pid_t   br_ppid;                /* parent pid for this thread */
        id_t    br_ptid;                /* parent tid for this thread */
        void    *br_clear_ctidp;        /* clone thread id ptr */
        void    *br_set_ctidp;          /* clone thread id ptr */
```



```
        /*
         * The following struct is used by lx_clone()
         * to pass info into fork()
         */
        void     *br_clone_args;

        uint_t  br_ptrace;              /* ptrace is active for this LWP */
} lx_lwp_data_t;

/* brand specific data */
typedef struct lx_zone_data {
        int lxzd_kernel_version;
        int lxzd_max_syscall;
} lx_zone_data_t;

#define BR_CPU_BOUND    0x0001

#define ttolxlwp(t)     ((struct lx_lwp_data *)ttolwpbrand(t))
#define lwptolxlwp(l)   ((struct lx_lwp_data *)lwptolwpbrand(l))
#define ttolxproc(t)    ((struct lx_proc_data *)(t)->t_procp->p_brand_data)

void    lx_brand_int80_callback(void);
int64_t lx_emulate_syscall(int, uintptr_t, uintptr_t, uintptr_t, uintptr_t,
        uintptr_t, uintptr_t);

extern int lx_get_zone_kern_version(zone_t *);
extern int lx_get_kern_version(void);

extern int lx_debug;
#define lx_print        if (lx_debug) printf

#endif  /* _KERNEL */
#endif /* _ASM */

#ifdef  __cplusplus
}
#endif

#endif  /* _LX_BRAND_H */
#endif /* ! codereview */
```

****
1480 Tue Jan 14 16:17:21 2014
new/usr/src/uts/common/brand/lx/sys/lx_futex.h
Bring back LX zones.
****

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _SYS_LX_FUTEX_H
#define _SYS_LX_FUTEX_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

#ifdef  __cplusplus
extern "C" {
#endif

#define FUTEX_WAIT              0
#define FUTEX_WAKE              1
#define FUTEX_FD                2
#define FUTEX_REQUEUE           3
#define FUTEX_CMP_REQUEUE       4
#define FUTEX_MAX_CMD           FUTEX_CMP_REQUEUE

#ifdef _KERNEL
extern long lx_futex(uintptr_t addr, int cmd, int val, uintptr_t lx_timeout,
    uintptr_t addr2, int val2);
extern void lx_futex_init(void);
extern int lx_futex_fini(void);
#endif /* _KERNEL */

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_LX_FUTEX_H */
#endif /* ! codereview */
```

```
**********************************************************
    1572 Tue Jan 14 16:17:21 2014
new/usr/src/uts/common/brand/lx/sys/lx_impl.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _LX_IMPL_H
#define _LX_IMPL_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/types.h>

#ifdef  __cplusplus
extern "C" {
#endif

typedef int64_t (*llfcn_t)();

typedef struct lx_sysent {
        int     sy_flags;
        char    *sy_name;
        llfcn_t sy_callc;
        char    sy_narg;
} lx_sysent_t;

typedef void (lx_systrace_f)(ulong_t, ulong_t, ulong_t, ulong_t, ulong_t,
    ulong_t, ulong_t);


extern lx_sysent_t lx_sysent[];

extern lx_systrace_f *lx_systrace_entry_ptr;
extern lx_systrace_f *lx_systrace_return_ptr;

extern void lx_brand_systrace_enable(void);
extern void lx_brand_systrace_disable(void);

#ifdef  __cplusplus
}
#endif
```



```
#endif  /* _LX_IMPL_H */
#endif /* ! codereview */
```

```
**********************************************************
    2733 Tue Jan 14 16:17:21 2014
new/usr/src/uts/common/brand/lx/sys/lx_ldt.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _SYS_LINUX_LDT_H
#define _SYS_LINUX_LDT_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/segments.h>

#ifdef  __cplusplus
extern "C" {
#endif

struct ldt_info {
        uint_t  entry_number;
        uint_t  base_addr;
        uint_t  limit;
        uint_t  seg_32bit:1,
                contents:2,
                read_exec_only:1,
                limit_in_pages:1,
                seg_not_present:1,
                useable:1;
};

#define LDT_INFO_EMPTY(info)                                            \
        ((info)->base_addr == 0 && (info)->limit == 0 &&                \
        (info)->contents == 0 && (info)->read_exec_only == 1 &&         \
        (info)->seg_32bit == 0 && (info)->limit_in_pages == 0 &&        \
        (info)->seg_not_present == 1 && (info)->useable == 0)

#if defined(__amd64)
#define SETMODE(desc)   (desc)->usd_long = SDP_SHORT;
#else
#define SETMODE(desc)
#endif

#define LDT_INFO_TO_DESC(info, desc)    {                               \
```



```
        USEGD_SETBASE(desc, (info)->base_addr);                         \
        USEGD_SETLIMIT(desc, (info)->limit);                            \
        (desc)->usd_type = ((info)->contents << 2) |                    \
            ((info)->read_exec_only ^ 1) << 1 | 0x10;                   \
        (desc)->usd_dpl = SEL_UPL;                                      \
        (desc)->usd_p = (info)->seg_not_present ^ 1;                    \
        (desc)->usd_def32 = (info)->seg_32bit;                          \
        (desc)->usd_gran = (info)->limit_in_pages;                      \
        (desc)->usd_avl = (info)->useable;                              \
        SETMODE(desc);                                                  \
}

#define DESC_TO_LDT_INFO(desc, info)    {                               \
        bzero((info), sizeof (*(info)));                                \
        (info)->base_addr = USEGD_GETBASE(desc);                        \
        (info)->limit = USEGD_GETLIMIT(desc);                           \
        (info)->seg_not_present = (desc)->usd_p ^ 1;                    \
        (info)->contents = ((desc)->usd_type >> 2) & 3;                 \
        (info)->read_exec_only = (((desc)->usd_type >> 1) & 1) ^ 1;     \
        (info)->seg_32bit = (desc)->usd_def32;                          \
        (info)->limit_in_pages = (desc)->usd_gran;                      \
        (info)->useable = (desc)->usd_avl;                              \
}

extern void lx_set_gdt(int, user_desc_t *);
extern void lx_clear_gdt(int);

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_LINUX_LDT_H */
#endif /* ! codereview */
```

```
**********************************************************
    1769 Tue Jan 14 16:17:22 2014
new/usr/src/uts/common/brand/lx/sys/lx_pid.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _SYS_LX_PID_H
#define _SYS_LX_PID_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/note.h>

#ifdef  __cplusplus
extern "C" {
#endif

#ifdef _KERNEL
struct lx_pid {
        pid_t   s_pid;                  /* the solaris pid and ... */
        id_t    s_tid;                  /* ... tid pair */
        pid_t   l_pid;                  /* the corresponding linux pid */
        time_t  l_start;                /* birthday of this pid */
        struct pid *l_pidp;
        struct lx_pid *stol_next;       /* link in stol hash table */
        struct lx_pid *ltos_next;       /* link in ltos hash table */
};

extern int lx_pid_assign(kthread_t *);
extern void lx_pid_reassign(kthread_t *);
extern void lx_pid_rele(pid_t, id_t);
extern pid_t lx_lpid_to_spair(pid_t, pid_t *, id_t *);
extern pid_t lx_lwp_ppid(klwp_t *, pid_t *, id_t *);
extern void lx_pid_init(void);
extern void lx_pid_fini(void);
#endif

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_LX_PID_H */
```



```
#endif /* ! codereview */
```

****
1234 Tue Jan 14 16:17:22 2014
new/usr/src/uts/common/brand/lx/sys/lx_ptm.h
Bring back LX zones.
****

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _SYS_PTM_LINUX_H
#define _SYS_PTM_LINUX_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

#ifdef  __cplusplus
extern "C" {
#endif

#define LX_PTM_DRV              "lx_ptm"
#define LX_PTM_MINOR_NODE       "lx_ptmajor"

#define LX_PTM_DEV_TO_PTS(dev)  (getminor(dev) - 1)

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_PTM_LINUX_H */
#endif /* ! codereview */
```

```
**********************************************************
    1482 Tue Jan 14 16:17:22 2014
new/usr/src/uts/common/brand/lx/sys/lx_sched.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _SYS_LINUX_SCHED_H
#define _SYS_LINUX_SCHED_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/procset.h>
#include <sys/priocntl.h>

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * Linux scheduler policies.
 */
#define LX_SCHED_OTHER          0
#define LX_SCHED_FIFO           1
#define LX_SCHED_RR             2

#define LX_PRI_MAX              99

typedef int l_pid_t;

struct lx_sched_param {
        int     lx_sched_prio;
};

extern int sched_setprocset(procset_t *, l_pid_t);
extern long do_priocntlsys(int, procset_t *, void *);

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_LINUX_SCHED_H */
#endif /* ! codereview */
```

```
**********************************************************
    1842 Tue Jan 14 16:17:22 2014
new/usr/src/uts/common/brand/lx/sys/lx_syscalls.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _SYS_LINUX_SYSCALLS_H
#define _SYS_LINUX_SYSCALLS_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

#ifdef  __cplusplus
extern "C" {
#endif

#ifdef  _KERNEL

extern long lx_brk();
extern long lx_getpid();
extern long lx_getppid();
extern long lx_clone();
extern long lx_kill();
extern long lx_tkill();
extern long lx_modify_ldt();
extern long lx_gettid();
extern long lx_futex();
extern long lx_get_thread_area();
extern long lx_sched_getparam();
extern long lx_sched_getscheduler();
extern long lx_sched_rr_get_interval();
extern long lx_sched_setparam();
extern long lx_sched_setscheduler();
extern long lx_set_thread_area();
extern long lx_set_tid_address();
extern long lx_setresgid();
extern long lx_setresgid16();
extern long lx_setresuid();
extern long lx_setresuid16();
extern long lx_sysinfo();
extern long lx_setgroups();
```



```
#endif  /* _KERNEL */

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_LINUX_SYSCALLS_H */
#endif /* ! codereview */
```

```
**********************************************************
    1701 Tue Jan 14 16:17:22 2014
new/usr/src/uts/common/brand/lx/syscall/lx_brk.c
LX zone support should now build and packages of relevance produced.
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/types.h>
#include <sys/systm.h>
#include <sys/errno.h>

/*
 * The brk() system call needs to be in-kernel because Linux expects a call to
 * brk(0) to return the current breakpoint.  In Solaris, the process breakpoint
 * is setup and managed by libc.  Due to the way we link our libraries and the
 * need for Linux to manage its own breakpoint, this has to remain in the
 * kernel.
 */
extern int brk(caddr_t);

long
lx_brk(caddr_t nva)
{
        proc_t *p = curproc;
        klwp_t *lwp = ttolwp(curthread);

        if (nva != 0) {
                (void) brk(nva);

                /*
                 * Despite claims to the contrary in the manpage, when Linux
                 * brk() fails, errno is left unchanged.
                 */
                lwp->lwp_errno = 0;
        }
        return ((long)(p->p_brkbase + p->p_brksize));
}
#endif /* ! codereview */
```

```
**********************************************************
    3676 Tue Jan 14 16:17:22 2014
new/usr/src/uts/common/brand/lx/syscall/lx_clone.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/types.h>
#include <sys/systm.h>
#include <sys/errno.h>
#include <sys/brand.h>
#include <sys/lx_brand.h>
#include <sys/lx_ldt.h>

#define LX_CSIGNAL              0x000000ff
#define LX_CLONE_VM             0x00000100
#define LX_CLONE_FS             0x00000200
#define LX_CLONE_FILES          0x00000400
#define LX_CLONE_SIGHAND        0x00000800
#define LX_CLONE_PID            0x00001000
#define LX_CLONE_PTRACE         0x00002000
#define LX_CLONE_PARENT         0x00008000
#define LX_CLONE_THREAD         0x00010000
#define LX_CLONE_SYSVSEM        0x00040000
#define LX_CLONE_SETTLS         0x00080000
#define LX_CLONE_PARENT_SETTID  0x00100000
#define LX_CLONE_CHILD_CLEARTID 0x00200000
#define LX_CLONE_DETACH         0x00400000
#define LX_CLONE_CHILD_SETTID   0x01000000

/*
 * Our lwp has already been created at this point, so this routine is
 * responsible for setting up all the state needed to track this as a
 * linux cloned thread.
 */
/* ARGSUSED */
long
lx_clone(int flags, void *stkp, void *ptidp, void *ldtinfo, void *ctidp)
{
        struct lx_lwp_data *lwpd = ttolxlwp(curthread);
        struct ldt_info info;
```



```
        struct user_desc descr;
        int tls_index;
        int entry = -1;
        int signo;

        signo = flags & LX_CSIGNAL;
        if (signo < 0 || signo > MAXSIG)
                return (set_errno(EINVAL));

        if (flags & LX_CLONE_SETTLS) {
                if (copyin((caddr_t)ldtinfo, &info, sizeof (info)))
                        return (set_errno(EFAULT));

                if (LDT_INFO_EMPTY(&info))
                        return (set_errno(EINVAL));

                entry = info.entry_number;
                if (entry < GDT_TLSMIN || entry > GDT_TLSMAX)
                        return (set_errno(EINVAL));

                tls_index = entry - GDT_TLSMIN;

                /*
                 * Convert the user-space structure into a real x86
                 * descriptor and copy it into this LWP's TLS array.  We
                 * also load it into the GDT.
                 */
                LDT_INFO_TO_DESC(&info, &descr);
                bcopy(&descr, &lwpd->br_tls[tls_index], sizeof (descr));
                lx_set_gdt(entry, &lwpd->br_tls[tls_index]);
        } else {
                tls_index = -1;
                bzero(&descr, sizeof (descr));
        }

        lwpd->br_clear_ctidp =
            (flags & LX_CLONE_CHILD_CLEARTID) ?  ctidp : NULL;

        if (signo && ! (flags & LX_CLONE_DETACH))
                lwpd->br_signal = signo;
        else
                lwpd->br_signal = 0;

        if (flags & LX_CLONE_THREAD)
                lwpd->br_tgid = curthread->t_procp->p_pid;

        if (flags & LX_CLONE_PARENT)
                lwpd->br_ppid = 0;

        if ((flags & LX_CLONE_CHILD_SETTID) && (ctidp != NULL) &&
            (suword32(ctidp, lwpd->br_pid) != 0)) {
                if (entry >= 0)
                        lx_clear_gdt(entry);
                return (set_errno(EFAULT));
        }
        if ((flags & LX_CLONE_PARENT_SETTID) && (ptidp != NULL) &&
            (suword32(ptidp, lwpd->br_pid) != 0)) {
                if (entry >= 0)
                        lx_clear_gdt(entry);
                return (set_errno(EFAULT));
        }

        return (lwpd->br_pid);
}

long
```

```
128 lx_set_tid_address(int *tidp)
129 {
130         struct lx_lwp_data *lwpd = ttolxlwp(curthread);

132         lwpd->br_clear_ctidp = tidp;

134         return (lwpd->br_pid);
135 }
136 #endif /* ! codereview */
```

```
**********************************************************
   12309 Tue Jan 14 16:17:23 2014
new/usr/src/uts/common/brand/lx/syscall/lx_futex.c
LX zone support should now build and packages of relevance produced.
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/types.h>
#include <sys/systm.h>
#include <sys/errno.h>
#include <sys/debug.h>
#include <vm/as.h>
#include <vm/seg.h>
#include <vm/seg_vn.h>
#include <vm/page.h>
#include <sys/mman.h>
#include <sys/timer.h>
#include <sys/condvar.h>
#include <sys/inttypes.h>
#include <sys/lx_futex.h>

/*
 * Futexes are a Linux-specific implementation of inter-process mutexes.
 * They are designed to use shared memory for simple, uncontested
 * operations, and rely on the kernel to resolve any contention issues.
 *
 * Most of the information in this section comes from the paper "Futexes
 * Are Tricky", by Ulrich Drepper.  This paper is currently available at:
 * http://people.redhat.com/~drepper/futex.pdf.
 *
 * A futex itself a 4-byte integer, which must be 4-byte aligned.  The
 * value of this integer is expected to be modified using user-level atomic
 * operations.  The futex(4) design itself does not impose any semantic
 * constraints on the value stored in the futex; it is up to the
 * application to define its own protocol.
 *
 * When the application decides that kernel intervention is required, it
 * will use the futex(2) system call.  There are 5 different operations
 * that can be performed on a futex, using this system call.  Since this
 * interface has evolved over time, there are several different prototypes
```



```
 * available to the user.  Fortunately, there is only a single kernel-level
 * interface:
 *
 * long sys_futex(void *futex1, int cmd, int val1,
 *      struct timespec *timeout, void *futex2, int val2)
 *
 * The kernel-level operations that may be performed on a futex are:
 *
 * FUTEX_WAIT
 *
 *      Atomically verify that futex1 contains the value val1.  If it
 *      doesn't, return EWOULDBLOCK.  If it does contain the expected
 *      value, the thread will sleep until somebody performs a FUTEX_WAKE
 *      on the futex.  The caller may also specify a timeout, indicating
 *      the maximum time the thread should sleep.  If the timer expires,
 *      the call returns ETIMEDOUT.  If the thread is awoken with a signal,
 *      the call returns EINTR.  Otherwise, the call returns 0.
 *
 * FUTEX_WAKE
 *
 *      Wake up val1 processes that are waiting on futex1.  The call
 *      returns the number of blocked threads that were woken up.
 *
 * FUTEX_CMP_REQUEUE
 *
 *      If the value stored in futex1 matches that passed in in val2, wake
 *      up val1 processes that are waiting on futex1.  Otherwise, return
 *      EAGAIN.
 *
 *      If there are more than val1 threads waiting on the futex, remove
 *      the remaining threads from this futex, and requeue them on futex2.
 *      The caller can limit the number of threads being requeued by
 *      encoding an integral numerical value in the position usually used
 *      for the timeout pointer.
 *
 *      The call returns the number of blocked threads that were woken up
 *      or requeued.
 *
 * FUTEX_REQUEUE
 *
 *       Identical to FUTEX_CMP_REQUEUE except that it does not use val2.
 *       This command has been declared broken and obsolete, but we still
 *       need to support it.
 *
 * FUTEX_FD
 *
 *      Return a file descriptor, which can be used to refer to the futex.
 *      We don't support this operation.
 */

/*
 * This structure is used to track all the threads currently waiting on a
 * futex.  There is one fwaiter_t for each blocked thread.  We store all
 * fwaiter_t's in a hash structure, indexed by the memid_t of the integer
 * containing the futex's value.
 *
 * At the moment, all fwaiter_t's for a single futex are simply dumped into
 * the hash bucket.  If futex contention ever becomes a hot path, we can
 * chain a single futex's waiters together.
 */
typedef struct fwaiter {
        memid_t         fw_memid;       /* memid of the user-space futex */
        kcondvar_t      fw_cv;          /* cond var */
        struct fwaiter  *fw_next;       /* hash queue */
        struct fwaiter  *fw_prev;       /* hash queue */
        volatile int    fw_woken;
```

```
} fwaiter_t;

#define MEMID_COPY(s, d) \
        { (d)->val[0] = (s)->val[0]; (d)->val[1] = (s)->val[1]; }
#define MEMID_EQUAL(s, d) \
        ((d)->val[0] == (s)->val[0] && (d)->val[1] == (s)->val[1])

/* Borrowed from the page freelist hash code.  */
#define HASH_SHIFT_SZ   7
#define HASH_SIZE       (1 << HASH_SHIFT_SZ)
#define HASH_FUNC(id)                                           \
        ((((uintptr_t)((id)->val[1]) >> PAGESHIFT) +                \
        ((uintptr_t)((id)->val[1]) >> (PAGESHIFT + HASH_SHIFT_SZ)) +    \
        ((uintptr_t)((id)->val[0]) >> 3) +                          \
        ((uintptr_t)((id)->val[0]) >> (3 + HASH_SHIFT_SZ)) +            \
        ((uintptr_t)((id)->val[0]) >> (3 + 2 * HASH_SHIFT_SZ))) &       \
        (HASH_SIZE - 1))

static fwaiter_t *futex_hash[HASH_SIZE];
static kmutex_t futex_hash_lock[HASH_SIZE];

static void
futex_hashin(fwaiter_t *fwp)
{
        int index;

        index = HASH_FUNC(&fwp->fw_memid);
        ASSERT(MUTEX_HELD(&futex_hash_lock[index]));

        fwp->fw_prev = NULL;
        fwp->fw_next = futex_hash[index];
        if (fwp->fw_next)
                fwp->fw_next->fw_prev = fwp;
        futex_hash[index] = fwp;
}

static void
futex_hashout(fwaiter_t *fwp)
{
        int index;

        index = HASH_FUNC(&fwp->fw_memid);
        ASSERT(MUTEX_HELD(&futex_hash_lock[index]));

        if (fwp->fw_prev)
                fwp->fw_prev->fw_next = fwp->fw_next;
        if (fwp->fw_next)
                fwp->fw_next->fw_prev = fwp->fw_prev;
        if (futex_hash[index] == fwp)
                futex_hash[index] = fwp->fw_next;

        fwp->fw_prev = NULL;
        fwp->fw_next = NULL;
}

/*
 * Go to sleep until somebody does a WAKE operation on this futex, we get a
 * signal, or the timeout expires.
 */
static int
futex_wait(memid_t *memid, caddr_t addr, int val, timespec_t *timeout)
{
        int err, ret;
        int32_t curval;
        fwaiter_t fw;
        int index;
```



```
        fw.fw_woken = 0;
        MEMID_COPY(memid, &fw.fw_memid);
        cv_init(&fw.fw_cv, NULL, CV_DEFAULT, NULL);

        index = HASH_FUNC(&fw.fw_memid);
        mutex_enter(&futex_hash_lock[index]);

        if (fuword32(addr, (uint32_t *)&curval)) {
                err = set_errno(EFAULT);
                goto out;
        }
        if (curval != val) {
                err = set_errno(EWOULDBLOCK);
                goto out;
        }

        futex_hashin(&fw);

        err = 0;
        while ((fw.fw_woken == 0) && (err == 0)) {
                ret = cv_waituntil_sig(&fw.fw_cv, &futex_hash_lock[index],
                        timeout, timechanged);
                if (ret < 0)
                        err = set_errno(ETIMEDOUT);
                else if (ret == 0)
                        err = set_errno(EINTR);
        }

        /*
         * The futex is normally hashed out in wakeup.  If we timed out or
         * got a signal, we need to hash it out here instead.
         */
        if (fw.fw_woken == 0)
                futex_hashout(&fw);

out:
        mutex_exit(&futex_hash_lock[index]);

        return (err);
}

/*
 * Wake up to wake_threads threads that are blocked on the futex at memid.
 */
static int
futex_wake(memid_t *memid, int wake_threads)
{
        fwaiter_t *fwp, *next;
        int index;
        int ret = 0;

        index = HASH_FUNC(memid);

        mutex_enter(&futex_hash_lock[index]);

        for (fwp = futex_hash[index]; fwp && ret < wake_threads; fwp = next) {
                next = fwp->fw_next;
                if (MEMID_EQUAL(&fwp->fw_memid, memid)) {
                        futex_hashout(fwp);
                        fwp->fw_woken = 1;
                        cv_signal(&fwp->fw_cv);
                        ret++;
                }
        }
```

```
259         mutex_exit(&futex_hash_lock[index]);

261         return (ret);
262 }

264 /*
265  * Wake up to wake_threads waiting on the futex at memid.  If there are
266  * more than that many threads waiting, requeue the remaining threads on
267  * the futex at requeue_memid.
268  */
269 static int
270 futex_requeue(memid_t *memid, memid_t *requeue_memid, int wake_threads,
271         ulong_t requeue_threads, caddr_t addr, int *cmpval)
272 {
273         fwaiter_t *fwp, *next;
274         int index1, index2;
275         int ret = 0;
276         int32_t curval;
277         kmutex_t *l1, *l2;

279         /*
280          * To ensure that we don't miss a wakeup if the value of cmpval
281          * changes, we need to grab locks on both the original and new hash
282          * buckets.  To avoid deadlock, we always grab the lower-indexed
283          * lock first.
284          */
285         index1 = HASH_FUNC(memid);
286         index2 = HASH_FUNC(requeue_memid);

288         if (index1 == index2) {
289                 l1 = &futex_hash_lock[index1];
290                 l2 = NULL;
291         } else if (index1 < index2) {
292                 l1 = &futex_hash_lock[index1];
293                 l2 = &futex_hash_lock[index2];
294         } else {
295                 l1 = &futex_hash_lock[index2];
296                 l2 = &futex_hash_lock[index1];
297         }

299         mutex_enter(l1);
300         if (l2 != NULL)
301                 mutex_enter(l2);

303         if (cmpval != NULL) {
304                 if (fuword32(addr, (uint32_t *)&curval)) {
305                         ret = -EFAULT;
306                         goto out;
307                 }
308                 if (curval != *cmpval) {
309                         ret = -EAGAIN;
310                         goto out;
311                 }
312         }

314         for (fwp = futex_hash[index1]; fwp; fwp = next) {
315                 next = fwp->fw_next;
316                 if (!MEMID_EQUAL(&fwp->fw_memid, memid))
317                         continue;

319                 futex_hashout(fwp);
320                 if (ret++ < wake_threads) {
321                         fwp->fw_woken = 1;
322                         cv_signal(&fwp->fw_cv);
323                 } else {
324                         MEMID_COPY(requeue_memid, &fwp->fw_memid);
```



```
325                         futex_hashin(fwp);

327                         if ((ret - wake_threads) >= requeue_threads)
328                                 break;
329                 }
330         }

332 out:
333         if (l2 != NULL)
334                 mutex_exit(l2);
335         mutex_exit(l1);

337         if (ret < 0)
338                 return (set_errno(-ret));
339         return (ret);
340 }

342 /*
343  * Copy in the relative timeout provided by the application and convert it
344  * to an absolute timeout.
345  */
346 static int
347 get_timeout(void *lx_timeout, timestruc_t *timeout)
348 {
349         timestruc_t now;

351         if (get_udatamodel() == DATAMODEL_NATIVE) {
352                 if (copyin(lx_timeout, timeout, sizeof (timestruc_t)))
353                         return (EFAULT);
354         }
355 #ifdef _SYSCALL32_IMPL
356         else {
357                 timestruc32_t timeout32;
358                 if (copyin(lx_timeout, &timeout32, sizeof (timestruc32_t)))
359                         return (EFAULT);
360                 timeout->tv_sec = (time_t)timeout32.tv_sec;
361                 timeout->tv_nsec = timeout32.tv_nsec;
362         }
363 #endif
364         gethrestime(&now);

366         if (itimerspecfix(timeout))
367                 return (EINVAL);

369         timespecadd(timeout, &now);
370         return (0);
371 }

373 long
374 lx_futex(uintptr_t addr, int cmd, int val, uintptr_t lx_timeout,
375         uintptr_t addr2, int val2)
376 {
377         struct as *as = curproc->p_as;
378         memid_t memid, requeue_memid;
379         timestruc_t timeout;
380         timestruc_t *tptr = NULL;
381         int requeue_threads = NULL;
382         int *requeue_cmp = NULL;
383         int rval = 0;

385         /* must be aligned on int boundary */
386         if (addr & 0x3)
387                 return (set_errno(EINVAL));

389         /* Sanity check the futex command */
390         if (cmd < 0 || cmd > FUTEX_MAX_CMD)
```

```
391                 return (set_errno(EINVAL));

393         /* Copy in the timeout structure from userspace. */
394         if (cmd == FUTEX_WAIT && lx_timeout != NULL) {
395                 rval = get_timeout((timespec_t *)lx_timeout, &timeout);
396                 if (rval != 0)
397                         return (set_errno(rval));
398                 tptr = &timeout;
399         }

401         if (cmd == FUTEX_REQUEUE || cmd == FUTEX_CMP_REQUEUE) {
402                 if (cmd == FUTEX_CMP_REQUEUE)
403                         requeue_cmp = &val2;

405                 /*
406                  * lx_timeout is nominally a pointer to a userspace
407                  * address.  For these two commands, it actually contains
408                  * an integer which indicates the maximum number of threads
409                  * to requeue.  This is horrible, and I'm sorry.
410                  */
411                 requeue_threads = (int)lx_timeout;
412         }

414         /*
415          * Translate the process-specific, user-space futex virtual
416          * address(es) to universal memid.
417          */
418         rval = as_getmemid(as, (void *)addr, &memid);
419         if (rval != 0)
420                 return (set_errno(rval));

422         if (cmd == FUTEX_REQUEUE || cmd == FUTEX_CMP_REQUEUE) {
423                 rval = as_getmemid(as, (void *)addr2, &requeue_memid);
424                 if (rval)
425                         return (set_errno(rval));
426         }

428         switch (cmd) {
429         case FUTEX_WAIT:
430                 rval = futex_wait(&memid, (void *)addr, val, tptr);
431                 break;

433         case FUTEX_WAKE:
434                 rval = futex_wake(&memid, val);
435                 break;

437         case FUTEX_CMP_REQUEUE:
438         case FUTEX_REQUEUE:
439                 rval = futex_requeue(&memid, &requeue_memid, val,
440                     requeue_threads, (void *)addr2, requeue_cmp);

442                 break;
443         }

445         return (rval);
446 }

448 void
449 lx_futex_init(void)
450 {
451         int i;

453         for (i = 0; i < HASH_SIZE; i++)
454                 mutex_init(&futex_hash_lock[i], NULL, MUTEX_DEFAULT, NULL);
455         bzero(futex_hash, sizeof (futex_hash));
456 }
```



```
458 int
459 lx_futex_fini(void)
460 {
461         int i, err;

463         err = 0;
464         for (i = 0; (err == 0) && (i < HASH_SIZE); i++) {
465                 mutex_enter(&futex_hash_lock[i]);
466                 if (futex_hash[i] != NULL)
467                         err = EBUSY;
468                 mutex_exit(&futex_hash_lock[i]);
469         }
470         return (err);
471 }
472 #endif /* ! codereview */
```

```
**********************************************************
    1638 Tue Jan 14 16:17:23 2014
new/usr/src/uts/common/brand/lx/syscall/lx_getpid.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/zone.h>
#include <sys/types.h>
#include <sys/systm.h>
#include <sys/thread.h>
#include <sys/cpuvar.h>
#include <sys/brand.h>
#include <sys/lx_brand.h>
#include <sys/lx_pid.h>

/*
 * return the pid
 */
long
lx_getpid()
{
        lx_lwp_data_t *lwpd = ttolxlwp(curthread);
        long rv;

        if (curproc->p_pid == curproc->p_zone->zone_proc_initpid) {
                rv = 1;
        } else {
                ASSERT(lwpd != NULL);
                rv = lwpd->br_tgid;
        }

        return (rv);
}

/*
 * return the parent pid
 */
long
lx_getppid(void)
{
```



```
        return (lx_lwp_ppid(ttolwp(curthread), NULL, NULL));
}

/*
 * return the thread id
 */
long
lx_gettid(void)
{
        lx_lwp_data_t *lwpd = ttolxlwp(curthread);

        return (lwpd->br_pid);
}
#endif /* ! codereview */
```

```
**********************************************************
    7156 Tue Jan 14 16:17:23 2014
new/usr/src/uts/common/brand/lx/syscall/lx_id.c
LX zone support should now build and packages of relevance produced.
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */


#pragma ident   "%Z%%M% %I%     %E% SMI"


#include <sys/types.h>
#include <sys/systm.h>
#include <sys/errno.h>
#include <sys/zone.h>
#include <sys/cred_impl.h>
#include <sys/policy.h>

typedef ushort_t        l_uid16_t;
typedef ushort_t        l_gid16_t;
typedef uint_t          l_uid_t;
typedef uint_t          l_gid_t;

#define LINUX_UID16_TO_UID32(uid16)     \
        (((uid16) == (l_uid16_t)-1) ? ((l_uid_t)-1) : (l_uid_t)(uid16))

#define LINUX_GID16_TO_GID32(gid16)     \
        (((gid16) == (l_gid16_t)-1) ? ((l_gid_t)-1) : (l_gid_t)(gid16))

#define LX_NGROUPS_MAX  32
extern int setgroups(int, gid_t *);

/*
 * This function is based on setreuid in common/syscall/uid.c and exists
 * because Solaris does not have a way to explicitly set the saved uid (suid)
 * from any other system call.
 */
long
lx_setresuid(l_uid_t ruid, l_uid_t euid, l_uid_t suid)
{
        proc_t  *p;
        int     error = 0;
```



```
        int     do_nocd = 0;
        int     uidchge = 0;
        uid_t   oldruid = ruid;
        cred_t  *cr, *newcr;
        zoneid_t zoneid = getzoneid();

        if ((ruid != -1 && (ruid > MAXUID)) ||
            (euid != -1 && (euid > MAXUID)) ||
            (suid != -1 && (suid > MAXUID))) {
                error = EINVAL;
                goto done;
        }

        /*
         * Need to pre-allocate the new cred structure before grabbing
         * the p_crlock mutex.
         */
        newcr = cralloc();

        p = ttoproc(curthread);

retry:
        mutex_enter(&p->p_crlock);
        cr = p->p_cred;

        if (ruid != -1 &&
            ruid != cr->cr_ruid && ruid != cr->cr_uid &&
            ruid != cr->cr_suid && secpolicy_allow_setid(cr, ruid, B_FALSE)) {
                error = EPERM;
        } else if (euid != -1 &&
            euid != cr->cr_ruid && euid != cr->cr_uid &&
            euid != cr->cr_suid && secpolicy_allow_setid(cr, euid, B_FALSE)) {
                error = EPERM;
        } else if (suid != -1 &&
            suid != cr->cr_ruid && suid != cr->cr_uid &&
            suid != cr->cr_suid && secpolicy_allow_setid(cr, suid, B_FALSE)) {
                error = EPERM;
        } else {
                if (!uidchge && ruid != -1 && cr->cr_ruid != ruid) {
                        /*
                         * The ruid of the process is going to change. In order
                         * to avoid a race condition involving the
                         * process count associated with the newly given ruid,
                         * we increment the count before assigning the
                         * credential to the process.
                         * To do that, we'll have to take pidlock, so we first
                         * release p_crlock.
                         */
                        mutex_exit(&p->p_crlock);
                        uidchge = 1;
                        mutex_enter(&pidlock);
                        upcount_inc(ruid, zoneid);
                        mutex_exit(&pidlock);
                        /*
                         * As we released p_crlock we can't rely on the cr
                         * we read. So retry the whole thing.
                         */
                        goto retry;
                }
                crhold(cr);
                crcopy_to(cr, newcr);
                p->p_cred = newcr;

                if (euid != -1)
                        newcr->cr_uid = euid;
                if (suid != -1)
```

```
127                         newcr->cr_suid = suid;
128                 if (ruid != -1) {
129                         oldruid = newcr->cr_ruid;
130                         newcr->cr_ruid = ruid;
131                         ASSERT(ruid != oldruid ? uidchge : 1);
132                 }

134                 /*
135                  * A process that gives up its privilege
136                  * must be marked to produce no core dump.
137                  */
138                 if ((cr->cr_uid != newcr->cr_uid ||
139                     cr->cr_ruid != newcr->cr_ruid ||
140                     cr->cr_suid != newcr->cr_suid))
141                         do_nocd = 1;

143                 crfree(cr);
144         }
145         mutex_exit(&p->p_crlock);

147         /*
148          * We decrement the number of processes associated with the oldruid
149          * to match the increment above, even if the ruid of the process
150          * did not change or an error occurred (oldruid == uid).
151          */
152         if (uidchge) {
153                 ASSERT(oldruid != -1 && ruid != -1);
154                 mutex_enter(&pidlock);
155                 upcount_dec(oldruid, zoneid);
156                 mutex_exit(&pidlock);
157         }

159         if (error == 0) {
160                 if (do_nocd) {
161                         mutex_enter(&p->p_lock);
162                         p->p_flag |= SNOCD;
163                         mutex_exit(&p->p_lock);
164                 }
165                 crset(p, newcr);        /* broadcast to process threads */
166                 goto done;
167         }
168         crfree(newcr);
169 done:
170         if (error)
171                 return (set_errno(error));
172         else
173                 return (0);
174 }

176 long
177 lx_setresuid16(l_uid16_t ruid16, l_uid16_t euid16, l_uid16_t suid16)
178 {
179         long    rval;

181         rval = lx_setresuid(
182                         LINUX_UID16_TO_UID32(ruid16),
183                         LINUX_UID16_TO_UID32(euid16),
184                         LINUX_UID16_TO_UID32(suid16));

186         return (rval);
187 }

189 /*
190  * This function is based on setregid in common/syscall/gid.c
191  */
192 long
```



```
193 lx_setresgid(l_gid_t rgid, l_gid_t egid, l_gid_t sgid)
194 {
195         proc_t  *p;
196         int     error = 0;
197         int     do_nocd = 0;
198         cred_t  *cr, *newcr;

200         if ((rgid != -1 && (rgid > MAXUID)) ||
201             (egid != -1 && (egid > MAXUID)) ||
202             (sgid != -1 && (sgid > MAXUID))) {
203                 error = EINVAL;
204                 goto done;
205         }

207         /*
208          * Need to pre-allocate the new cred structure before grabbing
209          * the p_crlock mutex.
210          */
211         newcr = cralloc();

213         p = ttoproc(curthread);
214         mutex_enter(&p->p_crlock);
215         cr = p->p_cred;

217         if (rgid != -1 &&
218             rgid != cr->cr_rgid && rgid != cr->cr_gid &&
219             rgid != cr->cr_sgid && secpolicy_allow_setid(cr, -1, B_FALSE)) {
220                 error = EPERM;
221         } else if (egid != -1 &&
222             egid != cr->cr_rgid && egid != cr->cr_gid &&
223             egid != cr->cr_sgid && secpolicy_allow_setid(cr, -1, B_FALSE)) {
224                 error = EPERM;
225         } else if (sgid != -1 &&
226             sgid != cr->cr_rgid && sgid != cr->cr_gid &&
227             sgid != cr->cr_sgid && secpolicy_allow_setid(cr, -1, B_FALSE)) {
228                 error = EPERM;
229         } else {
230                 crhold(cr);
231                 crcopy_to(cr, newcr);
232                 p->p_cred = newcr;

234                 if (egid != -1)
235                         newcr->cr_gid = egid;
236                 if (sgid != -1)
237                         newcr->cr_sgid = sgid;
238                 if (rgid != -1)
239                         newcr->cr_rgid = rgid;

241                 /*
242                  * A process that gives up its privilege
243                  * must be marked to produce no core dump.
244                  */
245                 if ((cr->cr_gid != newcr->cr_gid ||
246                     cr->cr_rgid != newcr->cr_rgid ||
247                     cr->cr_sgid != newcr->cr_sgid))
248                         do_nocd = 1;

250                 crfree(cr);
251         }
252         mutex_exit(&p->p_crlock);

254         if (error == 0) {
255                 if (do_nocd) {
256                         mutex_enter(&p->p_lock);
257                         p->p_flag |= SNOCD;
258                         mutex_exit(&p->p_lock);
```

```
259                 }
260                 crset(p, newcr);        /* broadcast to process threads */
261                 goto done;
262         }
263         crfree(newcr);
264 done:
265         if (error)
266                 return (set_errno(error));
267         else
268                 return (0);
269 }

271 long
272 lx_setresgid16(l_gid16_t rgid16, l_gid16_t egid16, l_gid16_t sgid16)
273 {
274         long    rval;

276         rval = lx_setresgid(
277                         LINUX_GID16_TO_GID32(rgid16),
278                         LINUX_GID16_TO_GID32(egid16),
279                         LINUX_GID16_TO_GID32(sgid16));

281         return (rval);
282 }

284 /*
285  * Linux defines NGROUPS_MAX to be 32, but on Solaris it is only 16. We employ
286  * the terrible hack below so that tests may proceed, if only on DEBUG kernels.
287  */
288 long
289 lx_setgroups(int ngroups, gid_t *grouplist)
290 {
291 #ifdef DEBUG
292         if (ngroups > ngroups_max && ngroups <= LX_NGROUPS_MAX)
293                 ngroups = ngroups_max;
294 #endif /* DEBUG */

296         return (setgroups(ngroups, grouplist));
297 }
298 #endif /* ! codereview */
```

```
**********************************************************
    6436 Tue Jan 14 16:17:23 2014
new/usr/src/uts/common/brand/lx/syscall/lx_kill.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */


#pragma ident   "%Z%%M% %I%     %E% SMI"


#include <sys/types.h>
#include <sys/systm.h>
#include <sys/errno.h>
#include <sys/proc.h>
#include <sys/zone.h>
#include <sys/thread.h>
#include <sys/signal.h>
#include <sys/brand.h>
#include <sys/lx_brand.h>
#include <sys/lx_pid.h>
#include <lx_signum.h>

extern int kill(pid_t, int);

/*
 * Check if it is legal to send this signal to the init process.  Linux
 * kill(2) semantics dictate that no _unhandled_ signal may be sent to pid
 * 1.
 */
static int
init_sig_check(int sig, pid_t pid)
{
        proc_t *p;
        int rv = 0;

        mutex_enter(&pidlock);

        if (((p = prfind(pid)) == NULL) || (p->p_stat == SIDL))
                rv = ESRCH;
        else if (sig && (sigismember(&cantmask, sig) ||
            (PTOU(p)->u_signal[sig-1] == SIG_DFL) ||
            (PTOU(p)->u_signal[sig-1] == SIG_IGN)))
```



```
                rv = EPERM;

        mutex_exit(&pidlock);

        return (rv);
}

long
lx_tkill(pid_t pid, int lx_sig)
{
        kthread_t *t;
        proc_t *pp;
        pid_t initpid;
        sigqueue_t *sqp;
        struct lx_lwp_data *br = ttolxlwp(curthread);
        int tid = 1;    /* default tid */
        int sig, rv;

        /*
         * Unlike kill(2), Linux tkill(2) doesn't allow signals to
         * be sent to process IDs <= 0 as it doesn't overlay any special
         * semantics on the pid.
         */
        if ((pid <= 0) || ((lx_sig < 0) || (lx_sig >= LX_NSIG)) ||
            ((sig = ltos_signo[lx_sig]) < 0))
                return (set_errno(EINVAL));

        /*
         * If the Linux pid is 1, translate the pid to the actual init
         * pid for the zone.  Note that Linux dictates that no unhandled
         * signals may be sent to init, so check for that, too.
         *
         * Otherwise, extract the tid and real pid from the Linux pid.
         */
        initpid = curproc->p_zone->zone_proc_initpid;
        if (pid == 1)
                pid = initpid;
        if ((pid == initpid) && ((rv = init_sig_check(sig, pid)) != 0))
                return (set_errno(rv));
        else if (lx_lpid_to_spair(pid, &pid, &tid) < 0)
                return (set_errno(ESRCH));

        sqp = kmem_zalloc(sizeof (sigqueue_t), KM_SLEEP);

        /*
         * Find the process for the passed pid...
         */
        mutex_enter(&pidlock);
        if (((pp = prfind(pid)) == NULL) || (pp->p_stat == SIDL)) {
                mutex_exit(&pidlock);
                rv = set_errno(ESRCH);
                goto free_and_exit;
        }
        mutex_enter(&pp->p_lock);
        mutex_exit(&pidlock);

        /*
         * Deny permission to send the signal if either of the following
         * is true:
         *
         *      + The signal is SIGCONT and the target pid is not in the same
         *        session as the sender
         *
         *      + prochasprocperm() shows the user lacks sufficient permission
         *        to send the signal to the target pid
         */
```

```
128         if (((sig == SIGCONT) && (pp->p_sessp != curproc->p_sessp)) ||
129             (!prochasprocperm(pp, curproc, CRED()))) {
130                 mutex_exit(&pp->p_lock);
131                 rv = set_errno(EPERM);
132                 goto free_and_exit;
133         }

135         /* check for the tid */
136         if ((t = idtot(pp, tid)) == NULL) {
137                 mutex_exit(&pp->p_lock);
138                 rv = set_errno(ESRCH);
139                 goto free_and_exit;
140         }

142         /* a signal of 0 means just check for the existence of the thread */
143         if (lx_sig == 0) {
144                 mutex_exit(&pp->p_lock);
145                 rv = 0;
146                 goto free_and_exit;
147         }

149         sqp->sq_info.si_signo = sig;
150         sqp->sq_info.si_code = SI_LWP;
151         sqp->sq_info.si_pid = br->br_pid;
152         sqp->sq_info.si_uid = crgetruid(CRED());
153         sigaddqa(pp, t, sqp);

155         mutex_exit(&pp->p_lock);

157         return (0);

159 free_and_exit:
160         kmem_free(sqp, sizeof (sigqueue_t));
161         return (rv);
162 }

164 long
165 lx_kill(pid_t lx_pid, int lx_sig)
166 {
167         pid_t s_pid, initpid;
168         sigsend_t v;
169         zone_t *zone = curproc->p_zone;
170         struct proc *p;
171         int err, sig, nfound;

173         if ((lx_sig < 0) || (lx_sig >= LX_NSIG) ||
174             ((sig = ltos_signo[lx_sig]) < 0))
175                 return (set_errno(EINVAL));

177         /*
178          * Since some linux apps rely on init(1M) having PID 1, we
179          * transparently translate 1 to the real init(1M)'s pid.  We then
180          * check to be sure that it is legal for this process to send this
181          * signal to init(1M).
182          */
183         initpid = zone->zone_proc_initpid;
184         if (lx_pid == 1 || lx_pid == -1) {
185                 s_pid = initpid;
186         } else if (lx_pid == 0) {
187                 s_pid = 0;
188         } else if (lx_pid > 0) {
189                 if (lx_lpid_to_spair(lx_pid, &s_pid, NULL) != 0) {
190                         /*
191                          * If we didn't find this pid that means it doesn't
192                          * exist in this zone.
193                          */
```



```
194                         return (set_errno(ESRCH));
195                 }
196         } else {
197                 ASSERT(lx_pid < 0);
198                 if (lx_lpid_to_spair(-lx_pid, &s_pid, NULL) != 0) {
199                         /*
200                          * If we didn't find this pid it means that the
201                          * process group leader doesn't exist in this zone.
202                          * In this case assuming that the Linux pid is
203                          * the same as the Solaris pid will get us the
204                          * correct behavior.
205                          */
206                         s_pid = -lx_pid;
207                 }
208         }

210         if ((s_pid == initpid) && ((err = init_sig_check(sig, s_pid)) != 0))
211                 return (set_errno(err));

213         /*
214          * For individual processes, kill() semantics are the same between
215          * Solaris and Linux.
216          */
217         if (lx_pid >= 0)
218                 return (kill(s_pid, sig));

220         /*
221          * In Solaris, sending a signal to -pid means "send a signal to
222          * everyone in process group pid."  In Linux it means "send a
223          * signal to everyone in the group other than init."  Sending a
224          * signal to -1 means "send a signal to every process except init
225          * and myself."
226          */

228         bzero(&v, sizeof (v));
229         v.sig = sig;
230         v.checkperm = 1;
231         v.sicode = SI_USER;
232         err = 0;

234         mutex_enter(&pidlock);

236         p = (lx_pid == -1) ? practive : pgfind(s_pid);
237         nfound = 0;
238         while (err == 0 && p != NULL) {
239                 if ((p->p_zone == zone) && (p->p_stat != SIDL) &&
240                     (p->p_pid != initpid) && (lx_pid < -1 || p != curproc)) {
241                         nfound++;
242                         err = sigsendproc(p, &v);
243                 }

245                 p = (lx_pid == -1) ? p->p_next : p->p_pglink;
246         }
247         mutex_exit(&pidlock);
248         if (nfound == 0)
249                 err = ESRCH;
250         else if (err == 0 && v.perm == 0)
251                 err = EPERM;
252         return (err ? set_errno(err) : 0);
253 }
254 #endif /* ! codereview */
```

```
**********************************************************
    2644 Tue Jan 14 16:17:23 2014
new/usr/src/uts/common/brand/lx/syscall/lx_modify_ldt.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */


#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/types.h>
#include <sys/systm.h>
#include <sys/errno.h>
#include <sys/segments.h>
#include <sys/archsystm.h>
#include <sys/proc.h>
#include <sys/sysi86.h>
#include <sys/cmn_err.h>
#include <sys/lx_ldt.h>

/*
 * Read the ldt_info structure in from the Linux app, convert it to an ssd
 * structure, and then call setdscr() to do all the heavy lifting.
 */
static int
write_ldt(void *data, ulong_t count)
{
        user_desc_t usd;
        struct ssd ssd;
        struct ldt_info ldt_inf;
        proc_t *pp = curthread->t_procp;
        int err;

        if (count != sizeof (ldt_inf))
                return (set_errno(EINVAL));

        if (copyin(data, &ldt_inf, sizeof (ldt_inf)))
                return (set_errno(EFAULT));

        if (ldt_inf.entry_number >= MAXNLDT)
                return (set_errno(EINVAL));

        LDT_INFO_TO_DESC(&ldt_inf, &usd);
```



```
        usd_to_ssd(&usd, &ssd, SEL_LDT(ldt_inf.entry_number));

        /*
         * Get everyone into a safe state before changing the LDT.
         */
        if (!holdlwps(SHOLDFORK1))
                return (set_errno(EINTR));

        err = setdscr(&ssd);

        /*
         * Release the hounds!
         */
        mutex_enter(&pp->p_lock);
        continuelwps(pp);
        mutex_exit(&pp->p_lock);

        return (err ? set_errno(err) : 0);
}

static int
read_ldt(void *uptr, ulong_t count)
{
        proc_t *pp = curproc;
        int bytes;

        if (pp->p_ldt == NULL)
                return (0);

        bytes = (pp->p_ldtlimit + 1) * sizeof (user_desc_t);
        if (bytes > count)
                bytes = count;

        if (copyout(pp->p_ldt, uptr, bytes))
                return (set_errno(EFAULT));

        return (bytes);
}

long
lx_modify_ldt(int op, void *data, ulong_t count)
{
        int rval;

        switch (op) {
        case 0:
                rval = read_ldt(data, count);
                break;

        case 1:
                rval = write_ldt(data, count);
                break;

        default:
                rval = set_errno(ENOSYS);
                break;
        }

        return (rval);
}
#endif /* ! codereview */
```

```
**********************************************************
   12221 Tue Jan 14 16:17:24 2014
new/usr/src/uts/common/brand/lx/syscall/lx_sched.c
LX zone support should now build and packages of relevance produced.
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/types.h>
#include <sys/systm.h>
#include <sys/errno.h>
#include <sys/proc.h>
#include <sys/cpu.h>
#include <sys/rtpriocntl.h>
#include <sys/tspriocntl.h>
#include <sys/processor.h>
#include <sys/brand.h>
#include <sys/lx_pid.h>
#include <sys/lx_sched.h>
#include <sys/lx_brand.h>

extern long priocntl_common(int, procset_t *, int, caddr_t, caddr_t, uio_seg_t);

int
lx_sched_affinity(int cmd, uintptr_t pid, int len, uintptr_t maskp,
    int64_t *rval)
{
        pid_t           s_pid;
        id_t            s_tid;
        kthread_t       *t = curthread;
        lx_lwp_data_t   *lx_lwp;

        if (cmd != B_GET_AFFINITY_MASK && cmd != B_SET_AFFINITY_MASK)
                return (set_errno(EINVAL));

        /*
         * The caller wants to know how large the mask should be.
         */
        if (cmd == B_GET_AFFINITY_MASK && len == 0) {
                *rval = sizeof (lx_affmask_t);
                return (0);
        }

        /*
         * Otherwise, ensure they have a large enough mask.
         */
        if (cmd == B_GET_AFFINITY_MASK && len < sizeof (lx_affmask_t)) {
                *rval = -1;
                return (set_errno(EINVAL));
        }

        if (pid == 0) {
                s_pid = curproc->p_pid;
                s_tid = curthread->t_tid;
        } else if (lx_lpid_to_spair((pid_t)pid, &s_pid, &s_tid) == -1) {
                return (set_errno(ESRCH));
        }

        /*
         * For now, we only support manipulating threads in the
         * same process.
         */
        if (curproc->p_pid != s_pid)
                return (set_errno(EPERM));

        /*
         * We must hold the process lock so that the thread list
         * doesn't change while we're looking at it. We'll hold
         * the lock until we no longer reference the
         * corresponding lwp.
         */

        mutex_enter(&curproc->p_lock);

        do {
                if (t->t_tid == s_tid)
                        break;
                t = t->t_forw;
        } while (t != curthread);

        /*
         * If the given PID is in the current thread's process,
         * then we _must_ find it in the process's thread list.
         */
        ASSERT(t->t_tid == s_tid);

        lx_lwp = t->t_lwp->lwp_brand;

        if (cmd == B_SET_AFFINITY_MASK) {
                if (copyin_nowatch((void *)maskp, &lx_lwp->br_affinitymask,
                    sizeof (lx_affmask_t)) != 0) {
                        mutex_exit(&curproc->p_lock);
                        return (set_errno(EFAULT));
                }

                *rval = 0;
        } else {
                if (copyout_nowatch(&lx_lwp->br_affinitymask, (void *)maskp,
                    sizeof (lx_affmask_t)) != 0) {
                        mutex_exit(&curproc->p_lock);
                        return (set_errno(EFAULT));
                }

                *rval = sizeof (lx_affmask_t);
        }

        mutex_exit(&curproc->p_lock);
```

```
127         return (0);
128 }

130 long
131 lx_sched_setscheduler(l_pid_t pid, int policy, struct lx_sched_param *param)
132 {
133         klwp_t *lwp = ttolwp(curthread);
134         procset_t procset;
135         procset_t procset_cid;
136         pcparms_t pcparm;
137         pcinfo_t pcinfo;
138         struct lx_sched_param sched_param;
139         tsparms_t *tsp;
140         int prio, maxupri;
141         int rv;

143         if (pid < 0)
144                 return (set_errno(ESRCH));

146         if ((rv = sched_setprocset(&procset, pid)))
147                 return (rv);

149         if (copyin(param, &sched_param, sizeof (sched_param)))
150                 return (set_errno(EFAULT));

152         prio = sched_param.lx_sched_prio;

154         if (policy < 0) {
155                 /*
156                  * get the class id
157                  */
158                 pcparm.pc_cid = PC_CLNULL;
159                 (void) do_priocntlsys(PC_GETPARMS, &procset, &pcparm);
160                 if (lwp->lwp_errno)
161                         return (lwp->lwp_errno);

163                 /*
164                  * get the current policy
165                  */
166                 bzero(&pcinfo, sizeof (pcinfo));
167                 pcinfo.pc_cid = pcparm.pc_cid;
168                 (void) do_priocntlsys(PC_GETCLINFO, &procset, &pcinfo);
169                 if (lwp->lwp_errno)
170                         return (lwp->lwp_errno);

172                 if (strcmp(pcinfo.pc_clname, "TS") == 0)
173                         policy = LX_SCHED_OTHER;
174                 else if (strcmp(pcinfo.pc_clname, "RT") == 0)
175                         policy = ((rtparms_t *)pcparm.pc_clparms)->rt_tqnsecs ==
176                                  RT_TQINF ? LX_SCHED_FIFO : LX_SCHED_RR;
177                 else
178                         return (set_errno(EINVAL));
179         }

181         bzero(&pcinfo, sizeof (pcinfo));
182         bzero(&pcparm, sizeof (pcparm));
183         setprocset(&procset_cid, POP_AND, P_PID, 0, P_ALL, 0);
184         switch (policy) {
185         case LX_SCHED_FIFO:
186         case LX_SCHED_RR:
187                 (void) strcpy(pcinfo.pc_clname, "RT");
188                 (void) do_priocntlsys(PC_GETCID, &procset_cid, &pcinfo);
189                 if (lwp->lwp_errno)
190                         return (lwp->lwp_errno);

192                 if (prio < 0 ||
```



```
193                     prio > ((rtinfo_t *)pcinfo.pc_clinfo)->rt_maxpri)
194                         return (set_errno(EINVAL));
195                 pcparm.pc_cid = pcinfo.pc_cid;
196                 ((rtparms_t *)pcparm.pc_clparms)->rt_pri = prio;
197                 ((rtparms_t *)pcparm.pc_clparms)->rt_tqnsecs =
198                         policy == LX_SCHED_RR ? RT_TQDEF : RT_TQINF;
199                 break;

201         case LX_SCHED_OTHER:
202                 (void) strcpy(pcinfo.pc_clname, "TS");
203                 (void) do_priocntlsys(PC_GETCID, &procset_cid, &pcinfo);
204                 if (lwp->lwp_errno)
205                         return (lwp->lwp_errno);

207                 maxupri = ((tsinfo_t *)pcinfo.pc_clinfo)->ts_maxupri;
208                 if (prio > maxupri || prio < -maxupri)
209                         return (set_errno(EINVAL));

211                 pcparm.pc_cid = pcinfo.pc_cid;
212                 tsp = (tsparms_t *)pcparm.pc_clparms;
213                 tsp->ts_upri = prio;
214                 tsp->ts_uprilim = TS_NOCHANGE;
215                 break;

217         default:
218                 return (set_errno(EINVAL));
219         }

221         /*
222          * finally set scheduling policy and parameters
223          */
224         (void) do_priocntlsys(PC_SETPARMS, &procset, &pcparm);

226         return (0);
227 }

229 long
230 lx_sched_getscheduler(l_pid_t pid)
231 {
232         klwp_t *lwp = ttolwp(curthread);
233         procset_t procset;
234         pcparms_t pcparm;
235         pcinfo_t pcinfo;
236         int policy;
237         int rv;

239         if (pid < 0)
240                 return (set_errno(ESRCH));

242         if ((rv = sched_setprocset(&procset, pid)))
243                 return (rv);

245         /*
246          * get the class id
247          */
248         pcparm.pc_cid = PC_CLNULL;
249         (void) do_priocntlsys(PC_GETPARMS, &procset, &pcparm);
250         if (lwp->lwp_errno)
251                 return (lwp->lwp_errno);

253         /*
254          * get the class info and identify the equivalent linux policy
255          */
256         bzero(&pcinfo, sizeof (pcinfo));
257         pcinfo.pc_cid = pcparm.pc_cid;
258         (void) do_priocntlsys(PC_GETCLINFO, &procset, &pcinfo);
```

```
259         if (lwp->lwp_errno)
260                 return (lwp->lwp_errno);

262         if (strcmp(pcinfo.pc_clname, "TS") == 0)
263                 policy = LX_SCHED_OTHER;
264         else if (strcmp(pcinfo.pc_clname, "RT") == 0)
265                 policy = ((rtparms_t *)pcparm.pc_clparms)->rt_tqnsecs ==
266                         RT_TQINF ? LX_SCHED_FIFO : LX_SCHED_RR;
267         else
268                 policy = set_errno(EINVAL);

270         return (policy);
271 }

273 long
274 lx_sched_setparam(l_pid_t pid, struct lx_sched_param *param)
275 {
276         klwp_t *lwp = ttolwp(curthread);
277         procset_t procset;
278         procset_t procset_cid;
279         pcparms_t pcparm;
280         pcinfo_t pcinfo;
281         struct lx_sched_param sched_param;
282         tsparms_t *tsp;
283         int policy;
284         int prio, maxupri;
285         int rv;

287         if (pid < 0)
288                 return (set_errno(ESRCH));

290         if ((rv = sched_setprocset(&procset, pid)))
291                 return (rv);

293         if (copyin(param, &sched_param, sizeof (sched_param)))
294                 return (set_errno(EFAULT));

296         prio = sched_param.lx_sched_prio;

298         /*
299          * get the class id
300          */
301         pcparm.pc_cid = PC_CLNULL;
302         (void) do_priocntlsys(PC_GETPARMS, &procset, &pcparm);
303         if (lwp->lwp_errno)
304                 return (lwp->lwp_errno);

306         /*
307          * get the current policy
308          */
309         bzero(&pcinfo, sizeof (pcinfo));
310         pcinfo.pc_cid = pcparm.pc_cid;
311         (void) do_priocntlsys(PC_GETCLINFO, &procset, &pcinfo);
312         if (lwp->lwp_errno)
313                 return (lwp->lwp_errno);

315         if (strcmp(pcinfo.pc_clname, "TS") == 0)
316                 policy = LX_SCHED_OTHER;
317         else if (strcmp(pcinfo.pc_clname, "RT") == 0)
318                 policy = ((rtparms_t *)pcparm.pc_clparms)->rt_tqnsecs ==
319                         RT_TQINF ? LX_SCHED_FIFO : LX_SCHED_RR;
320         else
321                 return (set_errno(EINVAL));

323         bzero(&pcinfo, sizeof (pcinfo));
324         bzero(&pcparm, sizeof (pcparm));
```



```
325         setprocset(&procset_cid, POP_AND, P_PID, 0, P_ALL, 0);
326         switch (policy) {
327         case LX_SCHED_FIFO:
328         case LX_SCHED_RR:
329                 (void) strcpy(pcinfo.pc_clname, "RT");
330                 (void) do_priocntlsys(PC_GETCID, &procset_cid, &pcinfo);
331                 if (lwp->lwp_errno)
332                         return (lwp->lwp_errno);

334                 if (prio < 0 ||
335                     prio > ((rtinfo_t *)pcinfo.pc_clinfo)->rt_maxpri)
336                         return (set_errno(EINVAL));
337                 pcparm.pc_cid = pcinfo.pc_cid;
338                 ((rtparms_t *)pcparm.pc_clparms)->rt_pri = prio;
339                 ((rtparms_t *)pcparm.pc_clparms)->rt_tqnsecs =
340                         policy == LX_SCHED_RR ? RT_TQDEF : RT_TQINF;
341                 break;

343         case LX_SCHED_OTHER:
344                 (void) strcpy(pcinfo.pc_clname, "TS");
345                 (void) do_priocntlsys(PC_GETCID, &procset_cid, &pcinfo);
346                 if (lwp->lwp_errno)
347                         return (lwp->lwp_errno);

349                 maxupri = ((tsinfo_t *)pcinfo.pc_clinfo)->ts_maxupri;
350                 if (prio > maxupri || prio < -maxupri)
351                         return (set_errno(EINVAL));

353                 pcparm.pc_cid = pcinfo.pc_cid;
354                 tsp = (tsparms_t *)pcparm.pc_clparms;
355                 tsp->ts_upri = prio;
356                 tsp->ts_uprilim = TS_NOCHANGE;
357                 break;

359         default:
360                 return (set_errno(EINVAL));
361         }

363         /*
364          * finally set scheduling policy and parameters
365          */
366         (void) do_priocntlsys(PC_SETPARMS, &procset, &pcparm);

368         return (0);
369 }

371 long
372 lx_sched_getparam(l_pid_t pid, struct lx_sched_param *param)
373 {
374         klwp_t *lwp = ttolwp(curthread);
375         struct lx_sched_param local_param;
376         procset_t procset;
377         pcparms_t pcparm;
378         pcinfo_t pcinfo;
379         tsinfo_t *tsi;
380         int prio, scale;
381         int rv;

383         if (pid < 0)
384                 return (set_errno(ESRCH));

386         if ((rv = sched_setprocset(&procset, pid)))
387                 return (rv);

389         /*
390          * get the class id
```

```
391          */
392         pcparm.pc_cid = PC_CLNULL;
393         (void) do_priocntlsys(PC_GETPARMS, &procset, &pcparm);
394         if (lwp->lwp_errno)
395                 return (lwp->lwp_errno);

397         /*
398          * get the class info and identify the equivalent linux policy
399          */
400         bzero(&pcinfo, sizeof (pcinfo));
401         pcinfo.pc_cid = pcparm.pc_cid;
402         (void) do_priocntlsys(PC_GETCLINFO, &procset, &pcinfo);
403         if (lwp->lwp_errno)
404                 return (lwp->lwp_errno);

406         bzero(&local_param, sizeof (local_param));
407         if (strcmp(pcinfo.pc_clname, "TS") == 0) {
408                 /*
409                  * I don't know if we need to do this, coz it can't be
410                  * changed from zero anyway.....
411                  */
412                 tsi = (tsinfo_t *)pcinfo.pc_clinfo;
413                 prio = ((tsparms_t *)pcparm.pc_clparms)->ts_upri;
414                 scale = tsi->ts_maxupri;
415                 if (scale == 0)
416                         local_param.lx_sched_prio = 0;
417                 else
418                         local_param.lx_sched_prio = -(prio * 20) / scale;
419         } else if (strcmp(pcinfo.pc_clname, "RT") == 0)
420                 local_param.lx_sched_prio =
421                         ((rtparms_t *)pcparm.pc_clparms)->rt_pri;
422         else
423                 rv = set_errno(EINVAL);

425         if (rv == 0)
426                 if (copyout(&local_param, param, sizeof (local_param)))
427                         return (set_errno(EFAULT));

429         return (rv);
430 }

432 long
433 lx_sched_rr_get_interval(l_pid_t pid, struct timespec *ival)
434 {
435         klwp_t *lwp = ttolwp(curthread);
436         struct timespec interval;
437         procset_t procset;
438         pcparms_t pcparm;
439         pcinfo_t pcinfo;
440         int rv;

442         if (pid < 0)
443                 return (set_errno(ESRCH));

445         if ((rv = sched_setprocset(&procset, pid)))
446                 return (rv);

448         /*
449          * get the class id
450          */
451         pcparm.pc_cid = PC_CLNULL;
452         (void) do_priocntlsys(PC_GETPARMS, &procset, &pcparm);
453         if (lwp->lwp_errno)
454                 return (lwp->lwp_errno);

456         /*
```



```
457          * get the class info and identify the equivalent linux policy
458          */
459         setprocset(&procset, POP_AND, P_PID, 0, P_ALL, 0);
460         bzero(&pcinfo, sizeof (pcinfo));
461         (void) strcpy(pcinfo.pc_clname, "RT");
462         (void) do_priocntlsys(PC_GETCID, &procset, &pcinfo);
463         if (lwp->lwp_errno)
464                 return (lwp->lwp_errno);

466         if (pcparm.pc_cid == pcinfo.pc_cid &&
467             ((rtparms_t *)pcparm.pc_clparms)->rt_tqnsecs != RT_TQINF) {
468                 interval.tv_sec = ((rtparms_t *)pcparm.pc_clparms)->rt_tqsecs;
469                 interval.tv_nsec = ((rtparms_t *)pcparm.pc_clparms)->rt_tqnsecs;

471                 if (copyout(&interval, ival, sizeof (interval)))
472                         return (set_errno(EFAULT));

474                 return (0);
475         }

477         return (set_errno(EINVAL));
478 }

480 int
481 sched_setprocset(procset_t *procset, l_pid_t pid)
482 {
483         id_t lid, rid;
484         idtype_t lidtype, ridtype;

486         /*
487          * define the target lwp
488          */
489         if (pid == 0) {
490                 ridtype = P_ALL;
491                 lidtype = P_PID;
492                 rid = 0;
493                 lid = P_MYID;
494         } else {
495                 if (lx_lpid_to_spair(pid, &pid, &lid) < 0)
496                         return (set_errno(ESRCH));
497                 if (pid != curproc->p_pid)
498                         return (set_errno(ESRCH));
499                 rid = 0;
500                 ridtype = P_ALL;
501                 lidtype = P_LWPID;
502         }
503         setprocset(procset, POP_AND, lidtype, lid, ridtype, rid);

505         return (0);
506 }

508 long
509 do_priocntlsys(int cmd, procset_t *procset, void *arg)
510 {
511         return (priocntl_common(PC_VERSION, procset, cmd, (caddr_t)arg, 0,
512             UIO_SYSSPACE));
513 }
514 #endif /* ! codereview */
```

```
**********************************************************
    3598 Tue Jan 14 16:17:24 2014
new/usr/src/uts/common/brand/lx/syscall/lx_sysinfo.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <vm/anon.h>
#include <sys/systm.h>
#include <sys/sysmacros.h>
#include <sys/zone.h>
#include <sys/time.h>

struct lx_sysinfo {
        int32_t si_uptime;      /* Seconds since boot */
        uint32_t si_loads[3];   /* 1, 5, and 15 minute avg runq length */
        uint32_t si_totalram;   /* Total memory size */
        uint32_t si_freeram;    /* Available memory */
        uint32_t si_sharedram;  /* Shared memory */
        uint32_t si_bufferram;  /* Buffer memory */
        uint32_t si_totalswap;  /* Total swap space */
        uint32_t si_freeswap;   /* Avail swap space */
        uint16_t si_procs;      /* Process count */
        uint32_t si_totalhigh;  /* High memory size */
        uint32_t si_freehigh;   /* Avail high memory */
        uint32_t si_mem_unit;   /* Unit size of memory fields */
};

long
lx_sysinfo(struct lx_sysinfo *sip)
{
        struct lx_sysinfo si;
        hrtime_t birthtime;
        zone_t *zone = curthread->t_procp->p_zone;
        proc_t *init_proc;

        /*
         * We don't record the time a zone was booted, so we use the
         * birthtime of that zone's init process instead.
         */
        mutex_enter(&pidlock);
```



```
        init_proc = prfind(zone->zone_proc_initpid);
        if (init_proc != NULL)
                birthtime = init_proc->p_mstart;
        else
                birthtime = p0.p_mstart;
        mutex_exit(&pidlock);
        si.si_uptime = (gethrtime() - birthtime) / NANOSEC;

        /*
         * We scale down the load in avenrun to allow larger load averages
         * to fit in 32 bits.  Linux doesn't, so we remove the scaling
         * here.
         */
        si.si_loads[0] = avenrun[0] << FSHIFT;
        si.si_loads[1] = avenrun[1] << FSHIFT;
        si.si_loads[2] = avenrun[2] << FSHIFT;

        /*
         * In linux each thread looks like a process, so we conflate the
         * two in this stat as well.
         */
        si.si_procs = (int32_t)zone->zone_nlwps;

        /*
         * If the maximum memory stat is less than 1^20 pages (i.e. 4GB),
         * then we report the result in bytes.  Otherwise we use pages.
         * Once we start supporting >1TB x86 systems, we'll need a third
         * option.
         */
        if (MAX(physmem, k_anoninfo.ani_max) < 1024 * 1024) {
                si.si_totalram = physmem * PAGESIZE;
                si.si_freeram = freemem * PAGESIZE;
                si.si_totalswap = k_anoninfo.ani_max * PAGESIZE;
                si.si_freeswap = k_anoninfo.ani_free * PAGESIZE;
                si.si_mem_unit = 1;
        } else {
                si.si_totalram = physmem;
                si.si_freeram = freemem;
                si.si_totalswap = k_anoninfo.ani_max;
                si.si_freeswap = k_anoninfo.ani_free;
                si.si_mem_unit = PAGESIZE;
        }
        si.si_bufferram = 0;
        si.si_sharedram = 0;

        /*
         * These two stats refer to high physical memory.  If an
         * application running in a Linux zone cares about this, then
         * either it or we are broken.
         */
        si.si_totalhigh = 0;
        si.si_freehigh = 0;

        if (copyout(&si, sip, sizeof (si)) != 0)
                return (set_errno(EFAULT));
        return (0);
}
#endif /* ! codereview */
```

```
**********************************************************
    3083 Tue Jan 14 16:17:24 2014
new/usr/src/uts/common/brand/lx/syscall/lx_thread_area.c
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/types.h>
#include <sys/systm.h>
#include <sys/errno.h>
#include <sys/cpuvar.h>
#include <sys/archsystm.h>
#include <sys/proc.h>
#include <sys/brand.h>
#include <sys/lx_brand.h>
#include <sys/lx_ldt.h>

long
lx_get_thread_area(struct ldt_info *inf)
{
        struct lx_lwp_data *jlwp = ttolxlwp(curthread);
        struct ldt_info ldt_inf;
        user_desc_t *dscrp;
        int entry;

        if (fuword32(&inf->entry_number, (uint32_t *)&entry))
                return (set_errno(EFAULT));

        if (entry < GDT_TLSMIN || entry > GDT_TLSMAX)
                return (set_errno(EINVAL));

        dscrp = jlwp->br_tls + entry - GDT_TLSMIN;

        /*
         * convert the solaris ldt to the linux format expected by the
         * caller
         */
        DESC_TO_LDT_INFO(dscrp, &ldt_inf);
        ldt_inf.entry_number = entry;

        if (copyout(&ldt_inf, inf, sizeof (struct ldt_info)))
```



```
                return (set_errno(EFAULT));

        return (0);
}

long
lx_set_thread_area(struct ldt_info *inf)
{
        struct lx_lwp_data *jlwp = ttolxlwp(curthread);
        struct ldt_info ldt_inf;
        user_desc_t *dscrp;
        int entry;
        int i;

        if (copyin(inf, &ldt_inf, sizeof (ldt_inf)))
                return (set_errno(EFAULT));

        entry = ldt_inf.entry_number;
        if (entry == -1) {
                /*
                 * find an empty entry in the tls for this thread
                 */
                for (i = 0, dscrp = jlwp->br_tls;
                                        i < LX_TLSNUM; i++, dscrp++)
                        if (((unsigned long *)dscrp)[0] == 0 &&
                            ((unsigned long *)dscrp)[1] == 0)
                                break;

                if (i < LX_TLSNUM) {
                        /*
                         * found one
                         */
                        entry = i + GDT_TLSMIN;
                        if (suword32(&inf->entry_number, entry))
                                return (set_errno(EFAULT));
                } else {
                        return (set_errno(ESRCH));
                }
        }

        if (entry < GDT_TLSMIN || entry > GDT_TLSMAX)
                return (set_errno(EINVAL));

        /*
         * convert the linux ldt info to standard intel descriptor
         */
        dscrp = jlwp->br_tls + entry - GDT_TLSMIN;

        if (LDT_INFO_EMPTY(&ldt_inf)) {
                ((unsigned long *)dscrp)[0] = 0;
                ((unsigned long *)dscrp)[1] = 0;
        } else {
                LDT_INFO_TO_DESC(&ldt_inf, dscrp);
        }

        /*
         * update the gdt with the new descriptor
         */
        kpreempt_disable();

        for (i = 0, dscrp = jlwp->br_tls; i < LX_TLSNUM; i++, dscrp++)
                lx_set_gdt(GDT_TLSMIN + i, dscrp);

        kpreempt_enable();

        return (0);
```

```
128 }
129 #endif /* ! codereview */
```

```
**********************************************************
    5904 Tue Jan 14 16:17:24 2014
new/usr/src/uts/common/brand/sn1/sn1_brand.c
Bring back LX zones.
**********************************************************
______unchanged_portion_omitted_

#else   /* sparc */

#ifdef  __amd64

struct brand_mach_ops sn1_mops = {
        sn1_brand_sysenter_callback,
        NULL,
#endif /* ! codereview */
        sn1_brand_int91_callback,
        sn1_brand_syscall_callback,
        sn1_brand_syscall32_callback,
        NULL
        sn1_brand_syscall32_callback
};

#else   /* ! __amd64 */

struct brand_mach_ops sn1_mops = {
        sn1_brand_sysenter_callback,
        NULL,
        NULL,
#endif /* ! codereview */
        sn1_brand_syscall_callback,
        NULL,
#endif /* ! codereview */
        NULL
};
#endif  /* __amd64 */

#endif  /* _sparc */

struct brand    sn1_brand = {
        BRAND_VER_1,
        "sn1",
        &sn1_brops,
        &sn1_mops
};

static struct modlbrand modlbrand = {
        &mod_brandops,          /* type of module */
        "Solaris N-1 Brand",    /* description of module */
        &sn1_brand              /* driver ops */
};

static struct modlinkage modlinkage = {
        MODREV_1, (void *)&modlbrand, NULL
};

void
sn1_setbrand(proc_t *p)
{
        brand_solaris_setbrand(p, &sn1_brand);
}

/* ARGSUSED */
int
sn1_getattr(zone_t *zone, int attr, void *buf, size_t *bufsize)
{
        return (EINVAL);
```



```
}

/* ARGSUSED */
int
sn1_setattr(zone_t *zone, int attr, void *buf, size_t bufsize)
{
        return (EINVAL);
}

/*ARGSUSED*/
int
sn1_brandsys(int cmd, int64_t *rval, uintptr_t arg1, uintptr_t arg2,
    uintptr_t arg3, uintptr_t arg4, uintptr_t arg5, uintptr_t arg6)
{
        int     res;

        *rval = 0;

        res = brand_solaris_cmd(cmd, arg1, arg2, arg3, &sn1_brand, SN1_VERSION);
        if (res >= 0)
                return (res);

        return (EINVAL);
}

void
sn1_copy_procdata(proc_t *child, proc_t *parent)
{
        brand_solaris_copy_procdata(child, parent, &sn1_brand);
}

void
sn1_proc_exit(struct proc *p, klwp_t *l)
{
        brand_solaris_proc_exit(p, l, &sn1_brand);
}

void
sn1_exec()
{
        brand_solaris_exec(&sn1_brand);
}

int
sn1_initlwp(klwp_t *l)
{
        return (brand_solaris_initlwp(l, &sn1_brand));
}

void
sn1_forklwp(klwp_t *p, klwp_t *c)
{
        brand_solaris_forklwp(p, c, &sn1_brand);
}

void
sn1_freelwp(klwp_t *l)
{
        brand_solaris_freelwp(l, &sn1_brand);
}

void
sn1_lwpexit(klwp_t *l)
{
        brand_solaris_lwpexit(l, &sn1_brand);
}
```

```
216 /*ARGSUSED*/
217 void
218 sn1_free_brand_data(zone_t *zone)
219 {
220 }

222 /*ARGSUSED*/
223 void
224 sn1_init_brand_data(zone_t *zone)
225 {
226 }

228 int
229 sn1_elfexec(vnode_t *vp, execa_t *uap, uarg_t *args, intpdata_t *idatap,
230         int level, long *execsz, int setid, caddr_t exec_file, cred_t *cred,
231         int brand_action)
232 {
233         return (brand_solaris_elfexec(vp, uap, args, idatap, level, execsz,
234             setid, exec_file, cred, brand_action, &sn1_brand, SN1_BRANDNAME,
235             SN1_LIB, SN1_LIB32, SN1_LINKER, SN1_LINKER32));
236 }

238 int
239 _init(void)
240 {
241         int err;

243         /*
244          * Set up the table indicating which system calls we want to
245          * interpose on.  We should probably build this automatically from
246          * a list of system calls that is shared with the user-space
247          * library.
248          */
249         sn1_emulation_table = kmem_zalloc(NSYSCALL, KM_SLEEP);
250         sn1_emulation_table[SYS_read] = 1;                       /*   3 */
251         sn1_emulation_table[SYS_write] = 1;                      /*   4 */
252         sn1_emulation_table[SYS_time] = 1;                       /*  13 */
253         sn1_emulation_table[SYS_getpid] = 1;                     /*  20 */
254         sn1_emulation_table[SYS_mount] = 1;                      /*  21 */
255         sn1_emulation_table[SYS_getuid] = 1;                     /*  24 */
256         sn1_emulation_table[SYS_times] = 1;                      /*  43 */
257         sn1_emulation_table[SYS_getgid] = 1;                     /*  47 */
258         sn1_emulation_table[SYS_utssys] = 1;                     /*  57 */
259         sn1_emulation_table[SYS_waitid] = 1;                     /* 107 */
260         sn1_emulation_table[SYS_uname] = 1;                      /* 135 */

262         err = mod_install(&modlinkage);
263         if (err) {
264                 cmn_err(CE_WARN, "Couldn't install brand module");
265                 kmem_free(sn1_emulation_table, NSYSCALL);
266         }

268         return (err);
269 }

271 int
272 _info(struct modinfo *modinfop)
273 {
274         return (mod_info(&modlinkage, modinfop));
275 }

277 int
278 _fini(void)
279 {
280         return (brand_solaris_fini(&sn1_emulation_table, &modlinkage,
```



```
281             &sn1_brand));
282 }
```

```
**********************************************************
   16875 Tue Jan 14 16:17:24 2014
new/usr/src/uts/common/brand/solaris10/s10_brand.c
Bring back LX zones.
**********************************************************
______unchanged_portion_omitted_

  96 #else   /* sparc */

  98 #ifdef  __amd64

 100 struct brand_mach_ops s10_mops = {
 101         s10_brand_sysenter_callback,
 102         NULL,
 103 #endif /* ! codereview */
 104         s10_brand_int91_callback,
 105         s10_brand_syscall_callback,
 106         s10_brand_syscall32_callback,
 107         NULL
 102         s10_brand_syscall32_callback
 108 };

 110 #else   /* ! __amd64 */

 112 struct brand_mach_ops s10_mops = {
 113         s10_brand_sysenter_callback,
 114         NULL,
 115         NULL,
 116 #endif /* ! codereview */
 117         s10_brand_syscall_callback,
 118         NULL,
 119 #endif /* ! codereview */
 120         NULL
 121 };
 122 #endif  /* __amd64 */

 124 #endif  /* _sparc */

 126 struct brand    s10_brand = {
 127         BRAND_VER_1,
 128         "solaris10",
 129         &s10_brops,
 130         &s10_mops
 131 };

 133 static struct modlbrand modlbrand = {
 134         &mod_brandops,          /* type of module */
 135         "Solaris 10 Brand",     /* description of module */
 136         &s10_brand              /* driver ops */
 137 };

 139 static struct modlinkage modlinkage = {
 140         MODREV_1, (void *)&modlbrand, NULL
 141 };

 143 void
 144 s10_setbrand(proc_t *p)
 145 {
 146         brand_solaris_setbrand(p, &s10_brand);
 147 }

 149 /*ARGSUSED*/
 150 int
 151 s10_getattr(zone_t *zone, int attr, void *buf, size_t *bufsize)
 152 {
 153         ASSERT(zone->zone_brand == &s10_brand);
```



```
 154         if (attr == S10_EMUL_BITMAP) {
 155                 if (buf == NULL || *bufsize != sizeof (s10_emul_bitmap_t))
 156                         return (EINVAL);
 157                 if (copyout(((s10_zone_data_t *)zone->zone_brand_data)->
 158                     emul_bitmap, buf, sizeof (s10_emul_bitmap_t)) != 0)
 159                         return (EFAULT);
 160                 return (0);
 161         }

 163         return (EINVAL);
 164 }

 166 int
 167 s10_setattr(zone_t *zone, int attr, void *buf, size_t bufsize)
 168 {
 169         ASSERT(zone->zone_brand == &s10_brand);
 170         if (attr == S10_EMUL_BITMAP) {
 171                 if (buf == NULL || bufsize != sizeof (s10_emul_bitmap_t))
 172                         return (EINVAL);
 173                 if (copyin(buf, ((s10_zone_data_t *)zone->zone_brand_data)->
 174                     emul_bitmap, sizeof (s10_emul_bitmap_t)) != 0)
 175                         return (EFAULT);
 176                 return (0);
 177         }

 179         return (EINVAL);
 180 }

 182 #ifdef  __amd64
 183 /*
 184  * The Nevada kernel clears %fs for threads in 64-bit x86 processes but S10's
 185  * libc expects %fs to be nonzero.  This causes some committed
 186  * libc/libthread interfaces (e.g., thr_main()) to fail, which impacts several
 187  * libraries, including libdoor.  This function sets the specified LWP's %fs
 188  * register to the legacy S10 selector value (LWPFS_SEL).
 189  *
 190  * The best solution to the aforementioned problem is backporting CRs
 191  * 6467491 to Solaris 10 so that 64-bit x86 Solaris 10 processes
 192  * would accept zero for %fs.  Backporting the CRs is a requirement for running
 193  * S10 Containers in PV domUs because 64-bit Xen clears %fsbase when %fs is
 194  * nonzero.  Such behavior breaks 64-bit processes because Xen has to fetch the
 195  * FS segments' base addresses from the LWPs' GDTs, which are only capable of
 196  * 32-bit addressing.
 197  */
 198 /*ARGSUSED*/
 199 static void
 200 s10_amd64_correct_fsreg(klwp_t *l)
 201 {
 202         if (lwp_getdatamodel(l) == DATAMODEL_NATIVE) {
 203                 kpreempt_disable();
 204                 l->lwp_pcb.pcb_fs = LWPFS_SEL;
 205                 l->lwp_pcb.pcb_rupdate = 1;
 206                 lwptot(l)->t_post_sys = 1;      /* Guarantee update_sregs() */
 207                 kpreempt_enable();
 208         }
 209 }
 210 #endif  /* __amd64 */

 212 /*
 213  * Native processes are started with the native ld.so.1 as the command.  This
 214  * brand op is invoked by s10_npreload to fix up the command and arguments
 215  * so that apps like pgrep or ps see the expected command strings.
 216  */
 217 int
 218 s10_native(void *cmd, void *args)
 219 {
```

```
220         struct user     *up = PTOU(curproc);
221         char            cmd_buf[MAXCOMLEN + 1];
222         char            arg_buf[PSARGSZ];

224         if (copyin(cmd, &cmd_buf, sizeof (cmd_buf)) != 0)
225                 return (EFAULT);
226         if (copyin(args, &arg_buf, sizeof (arg_buf)) != 0)
227                 return (EFAULT);

229         /*
230          * Make sure that the process' interpreter is the native dynamic linker.
231          * Convention dictates that native processes executing within solaris10-
232          * branded zones are interpreted by the native dynamic linker (the
233          * process and its arguments are specified as arguments to the dynamic
234          * linker).  If this convention is violated (i.e.,
235          * brandsys(B_S10_NATIVE, ...) is invoked by a process that shouldn't be
236          * native), then do nothing and silently indicate success.
237          */
238         if (strcmp(up->u_comm, S10_LINKER_NAME) != 0)
239                 return (0);

241         /*
242          * The sizeof has an extra value for the trailing '\0' so this covers
243          * the appended " " in the following strcmps.
244          */
245         if (strncmp(up->u_psargs, BRAND_NATIVE_LINKER64 " ",
246             sizeof (BRAND_NATIVE_LINKER64)) != 0 &&
247             strncmp(up->u_psargs, BRAND_NATIVE_LINKER32 " ",
248             sizeof (BRAND_NATIVE_LINKER32)) != 0)
249                 return (0);

251         mutex_enter(&curproc->p_lock);
252         (void) strlcpy(up->u_comm, cmd_buf, sizeof (up->u_comm));
253         (void) strlcpy(up->u_psargs, arg_buf, sizeof (up->u_psargs));
254         mutex_exit(&curproc->p_lock);

256         return (0);
257 }

259 /*ARGSUSED*/
260 int
261 s10_brandsys(int cmd, int64_t *rval, uintptr_t arg1, uintptr_t arg2,
262     uintptr_t arg3, uintptr_t arg4, uintptr_t arg5, uintptr_t arg6)
263 {
264         proc_t  *p = curproc;
265         int     res;

267         *rval = 0;

269         if (cmd == B_S10_NATIVE)
270                 return (s10_native((void *)arg1, (void *)arg2));

272         res = brand_solaris_cmd(cmd, arg1, arg2, arg3, &s10_brand, S10_VERSION);
273         if (res >= 0)
274                 return (res);

276         switch ((cmd)) {
277         case B_S10_PIDINFO:
278                 /*
279                  * The s10 brand needs to be able to get the pid of the
280                  * current process and the pid of the zone's init, and it
281                  * needs to do this on every process startup.  Early in
282                  * brand startup, we can't call getpid() because calls to
283                  * getpid() represent a magical signal to some old-skool
284                  * debuggers.  By merging all of this into one call, we
285                  * make this quite a bit cheaper and easier to handle in
```



```
286                  * the brand module.
287                  */
288                 if (copyout(&p->p_pid, (void *)arg1, sizeof (pid_t)) != 0)
289                         return (EFAULT);
290                 if (copyout(&p->p_zone->zone_proc_initpid, (void *)arg2,
291                     sizeof (pid_t)) != 0)
292                         return (EFAULT);
293                 return (0);

295         case B_S10_ISFDXATTRDIR: {
296                 /*
297                  * This subcommand enables the userland brand emulation library
298                  * to determine whether a file descriptor refers to an extended
299                  * file attributes directory.  There is no standard syscall or
300                  * libc function that can make such a determination.
301                  */
302                 file_t *dir_filep;

304                 dir_filep = getf((int)arg1);
305                 if (dir_filep == NULL)
306                         return (EBADF);
307                 ASSERT(dir_filep->f_vnode != NULL);
308                 *rval = IS_XATTRDIR(dir_filep->f_vnode);
309                 releasef((int)arg1);
310                 return (0);
311         }

313 #ifdef  __amd64
314         case B_S10_FSREGCORRECTION:
315                 /*
316                  * This subcommand exists so that the SYS_lwp_private and
317                  * SYS_lwp_create syscalls can manually set the current thread's
318                  * %fs register to the legacy S10 selector value for 64-bit x86
319                  * processes.
320                  */
321                 s10_amd64_correct_fsreg(ttolwp(curthread));
322                 return (0);
323 #endif  /* __amd64 */
324         }

326         return (EINVAL);
327 }

329 void
330 s10_copy_procdata(proc_t *child, proc_t *parent)
331 {
332         brand_solaris_copy_procdata(child, parent, &s10_brand);
333 }

335 void
336 s10_proc_exit(struct proc *p, klwp_t *l)
337 {
338         brand_solaris_proc_exit(p, l, &s10_brand);
339 }

341 void
342 s10_exec()
343 {
344         brand_solaris_exec(&s10_brand);
345 }

347 int
348 s10_initlwp(klwp_t *l)
349 {
350         return (brand_solaris_initlwp(l, &s10_brand));
351 }
```

```
353 void
354 s10_forklwp(klwp_t *p, klwp_t *c)
355 {
356         brand_solaris_forklwp(p, c, &s10_brand);
357
358 #ifdef  __amd64
359         /*
360          * Only correct the child's %fs register if the parent's %fs register
361          * is LWPFS_SEL.  If the parent's %fs register is zero, then the Solaris
362          * 10 environment that we're emulating uses a version of libc that
363          * works when %fs is zero (i.e., it contains backports of CRs 6467491
364          * and 6501650).
365          */
366         if (p->lwp_pcb.pcb_fs == LWPFS_SEL)
367                 s10_amd64_correct_fsreg(c);
368 #endif  /* __amd64 */
369 }
370
371 void
372 s10_freelwp(klwp_t *l)
373 {
374         brand_solaris_freelwp(l, &s10_brand);
375 }
376
377 void
378 s10_lwpexit(klwp_t *l)
379 {
380         brand_solaris_lwpexit(l, &s10_brand);
381 }
382
383 void
384 s10_free_brand_data(zone_t *zone)
385 {
386         kmem_free(zone->zone_brand_data, sizeof (s10_zone_data_t));
387 }
388
389 void
390 s10_init_brand_data(zone_t *zone)
391 {
392         ASSERT(zone->zone_brand == &s10_brand);
393         ASSERT(zone->zone_brand_data == NULL);
394         zone->zone_brand_data = kmem_zalloc(sizeof (s10_zone_data_t), KM_SLEEP);
395 }
396
397 int
398 s10_elfexec(vnode_t *vp, execa_t *uap, uarg_t *args, intpdata_t *idatap,
399         int level, long *execsz, int setid, caddr_t exec_file, cred_t *cred,
400         int brand_action)
401 {
402         return (brand_solaris_elfexec(vp, uap, args, idatap, level, execsz,
403             setid, exec_file, cred, brand_action, &s10_brand, S10_BRANDNAME,
404             S10_LIB, S10_LIB32, S10_LINKER, S10_LINKER32));
405 }
406
407 void
408 s10_sigset_native_to_s10(sigset_t *set)
409 {
410         int nativesig;
411         int s10sig;
412         sigset_t s10set;
413
414         /*
415          * Shortcut: we know the first 32 signals are the same in both
416          * s10 and native Solaris.  Just assign the first word.
417          */
```



```
418         s10set.__sigbits[0] = set->__sigbits[0];
419         s10set.__sigbits[1] = 0;
420         s10set.__sigbits[2] = 0;
421         s10set.__sigbits[3] = 0;
422
423         /*
424          * Copy the remainder of the initial set of common signals.
425          */
426         for (nativesig = 33; nativesig < S10_SIGRTMIN; nativesig++)
427                 if (sigismember(set, nativesig))
428                         sigaddset(&s10set, nativesig);
429
430         /*
431          * Convert any native RT signals to their S10 values.
432          */
433         for (nativesig = _SIGRTMIN, s10sig = S10_SIGRTMIN;
434             nativesig <= _SIGRTMAX && s10sig <= S10_SIGRTMAX;
435             nativesig++, s10sig++) {
436                 if (sigismember(set, nativesig))
437                         sigaddset(&s10set, s10sig);
438         }
439
440         *set = s10set;
441 }
442
443 void
444 s10_sigset_s10_to_native(sigset_t *set)
445 {
446         int s10sig;
447         int nativesig;
448         sigset_t nativeset;
449
450         /*
451          * Shortcut: we know the first 32 signals are the same in both
452          * s10 and native Solaris.  Just assign the first word.
453          */
454         nativeset.__sigbits[0] = set->__sigbits[0];
455         nativeset.__sigbits[1] = 0;
456         nativeset.__sigbits[2] = 0;
457         nativeset.__sigbits[3] = 0;
458
459         /*
460          * Copy the remainder of the initial set of common signals.
461          */
462         for (s10sig = 33; s10sig < S10_SIGRTMIN; s10sig++)
463                 if (sigismember(set, s10sig))
464                         sigaddset(&nativeset, s10sig);
465
466         /*
467          * Convert any S10 RT signals to their native values.
468          */
469         for (s10sig = S10_SIGRTMIN, nativesig = _SIGRTMIN;
470             s10sig <= S10_SIGRTMAX && nativesig <= _SIGRTMAX;
471             s10sig++, nativesig++) {
472                 if (sigismember(set, s10sig))
473                         sigaddset(&nativeset, nativesig);
474         }
475
476         *set = nativeset;
477 }
478
479 int
480 _init(void)
481 {
482         int err;
```

```
484         /*
485          * Set up the table indicating which system calls we want to
486          * interpose on.  We should probably build this automatically from
487          * a list of system calls that is shared with the user-space
488          * library.
489          */
490         s10_emulation_table = kmem_zalloc(NSYSCALL, KM_SLEEP);
491         s10_emulation_table[S10_SYS_forkall] = 1;                /*   2 */
492         s10_emulation_table[S10_SYS_open] = 1;                   /*   5 */
493         s10_emulation_table[S10_SYS_wait] = 1;                   /*   7 */
494         s10_emulation_table[S10_SYS_creat] = 1;                  /*   8 */
495         s10_emulation_table[S10_SYS_link] = 1;                   /*   9 */
496         s10_emulation_table[S10_SYS_unlink] = 1;                 /*  10 */
497         s10_emulation_table[S10_SYS_exec] = 1;                   /*  11 */
498         s10_emulation_table[S10_SYS_mknod] = 1;                  /*  14 */
499         s10_emulation_table[S10_SYS_chmod] = 1;                  /*  15 */
500         s10_emulation_table[S10_SYS_chown] = 1;                  /*  16 */
501         s10_emulation_table[S10_SYS_stat] = 1;                   /*  18 */
502         s10_emulation_table[S10_SYS_umount] = 1;                 /*  22 */
503         s10_emulation_table[S10_SYS_fstat] = 1;                  /*  28 */
504         s10_emulation_table[S10_SYS_utime] = 1;                  /*  30 */
505         s10_emulation_table[S10_SYS_access] = 1;                 /*  33 */
506         s10_emulation_table[SYS_kill] = 1;                       /*  37 */
507         s10_emulation_table[S10_SYS_dup] = 1;                    /*  41 */
508         s10_emulation_table[S10_SYS_pipe] = 1;                   /*  42 */
509         s10_emulation_table[SYS_ioctl] = 1;                      /*  54 */
510         s10_emulation_table[SYS_execve] = 1;                     /*  59 */
511         s10_emulation_table[SYS_acctctl] = 1;                    /*  71 */
512         s10_emulation_table[S10_SYS_issetugid] = 1;              /*  75 */
513         s10_emulation_table[S10_SYS_fsat] = 1;                   /*  76 */
514         s10_emulation_table[S10_SYS_rmdir] = 1;                  /*  79 */
515         s10_emulation_table[S10_SYS_mkdir] = 1;                  /*  80 */
516         s10_emulation_table[SYS_getdents] = 1;                   /*  81 */
517         s10_emulation_table[S10_SYS_poll] = 1;                   /*  87 */
518         s10_emulation_table[S10_SYS_lstat] = 1;                  /*  88 */
519         s10_emulation_table[S10_SYS_symlink] = 1;                /*  89 */
520         s10_emulation_table[S10_SYS_readlink] = 1;               /*  90 */
521         s10_emulation_table[S10_SYS_fchmod] = 1;                 /*  93 */
522         s10_emulation_table[S10_SYS_fchown] = 1;                 /*  94 */
523         s10_emulation_table[SYS_sigprocmask] = 1;                /*  95 */
524         s10_emulation_table[SYS_sigsuspend] = 1;                 /*  96 */
525         s10_emulation_table[SYS_sigaction] = 1;                  /*  98 */
526         s10_emulation_table[SYS_sigpending] = 1;                 /*  99 */
527         s10_emulation_table[SYS_waitid] = 1;                     /* 107 */
528         s10_emulation_table[SYS_sigsendsys] = 1;                 /* 108 */
529 #if defined(__x86)
530         s10_emulation_table[S10_SYS_xstat] = 1;                  /* 123 */
531         s10_emulation_table[S10_SYS_lxstat] = 1;                 /* 124 */
532         s10_emulation_table[S10_SYS_fxstat] = 1;                 /* 125 */
533         s10_emulation_table[S10_SYS_xmknod] = 1;                 /* 126 */
534 #endif
535         s10_emulation_table[S10_SYS_lchown] = 1;                 /* 130 */
536         s10_emulation_table[S10_SYS_rename] = 1;                 /* 134 */
537         s10_emulation_table[SYS_uname] = 1;                      /* 135 */
538         s10_emulation_table[SYS_sysconfig] = 1;                  /* 137 */
539         s10_emulation_table[SYS_systeminfo] = 1;                 /* 139 */
540         s10_emulation_table[S10_SYS_fork1] = 1;                  /* 143 */
541         s10_emulation_table[SYS_sigtimedwait] = 1;               /* 144 */
542         s10_emulation_table[S10_SYS_lwp_sema_wait] = 1;          /* 147 */
543         s10_emulation_table[S10_SYS_utimes] = 1;                 /* 154 */
544         s10_emulation_table[SYS_lwp_create] = 1;                 /* 159 */
545         s10_emulation_table[SYS_lwp_kill] = 1;                   /* 163 */
546         s10_emulation_table[SYS_lwp_sigmask] = 1;                /* 165 */
547 #if defined(__amd64)
548         s10_emulation_table[SYS_lwp_private] = 1;                /* 166 */
549 #endif  /* __amd64 */
```



```
550         s10_emulation_table[S10_SYS_lwp_mutex_lock] = 1;         /* 169 */
551         s10_emulation_table[SYS_pwrite] = 1;                     /* 174 */
552         s10_emulation_table[SYS_acl] = 1;                        /* 185 */
553         s10_emulation_table[SYS_auditsys] = 1;                   /* 186 */
554         s10_emulation_table[SYS_sigqueue] = 1;                   /* 190 */
555         s10_emulation_table[SYS_facl] = 1;                       /* 200 */
556         s10_emulation_table[SYS_signotify] = 1;                  /* 205 */
557         s10_emulation_table[SYS_lwp_mutex_timedlock] = 1;        /* 210 */
558         s10_emulation_table[SYS_getdents64] = 1;                 /* 213 */
559         s10_emulation_table[S10_SYS_stat64] = 1;                 /* 215 */
560         s10_emulation_table[S10_SYS_lstat64] = 1;                /* 216 */
561         s10_emulation_table[S10_SYS_fstat64] = 1;                /* 217 */
562         s10_emulation_table[SYS_pwrite64] = 1;                   /* 223 */
563         s10_emulation_table[S10_SYS_creat64] = 1;                /* 224 */
564         s10_emulation_table[S10_SYS_open64] = 1;                 /* 225 */
565         s10_emulation_table[SYS_zone] = 1;                       /* 227 */
566         s10_emulation_table[S10_SYS_so_socket] = 1;              /* 230 */
567         s10_emulation_table[S10_SYS_accept] = 1;                 /* 234 */
568         s10_emulation_table[SYS_lwp_mutex_trylock] = 1;          /* 251 */

570         err = mod_install(&modlinkage);
571         if (err) {
572                 cmn_err(CE_WARN, "Couldn't install brand module");
573                 kmem_free(s10_emulation_table, NSYSCALL);
574         }

576         return (err);
577 }

579 int
580 _info(struct modinfo *modinfop)
581 {
582         return (mod_info(&modlinkage, modinfop));
583 }

585 int
586 _fini(void)
587 {
588         return (brand_solaris_fini(&s10_emulation_table, &modlinkage,
589             &s10_brand));
590 }
```

```
**********************************************************
   19477 Tue Jan 14 16:17:25 2014
new/usr/src/uts/common/io/ptm.c
Bring back LX zones.
**********************************************************
______unchanged_portion_omitted_
```

```
450 static boolean_t
451 ptmptsopencb(ptmptsopencb_arg_t arg)
452 {
453         struct pt_ttys  *ptmp = (struct pt_ttys *)arg;
454         boolean_t rval;
455
456         PT_ENTER_READ(ptmp);
457         rval = (ptmp->pt_nullmsg != NULL);
458         PT_EXIT_READ(ptmp);
459         return (rval);
460 }
461
462 #endif /* ! codereview */
463 /*
464  * The wput procedure will only handle ioctl and flush messages.
465  */
466 static void
467 ptmwput(queue_t *qp, mblk_t *mp)
468 {
469         struct pt_ttys  *ptmp;
470         struct iocblk   *iocp;
471
472         DBG(("entering ptmwput\n"));
473         ASSERT(qp->q_ptr);
474
475         ptmp = (struct pt_ttys *)qp->q_ptr;
476         PT_ENTER_READ(ptmp);
477
478         switch (mp->b_datap->db_type) {
479         /*
480          * if write queue request, flush master's write
481          * queue and send FLUSHR up slave side. If read
482          * queue request, convert to FLUSHW and putnext().
483          */
484         case M_FLUSH:
485                 {
486                         unsigned char flush_flg = 0;
487
488                         DBG(("ptm got flush request\n"));
489                         if (*mp->b_rptr & FLUSHW) {
490                                 DBG(("got FLUSHW, flush ptm write Q\n"));
491                                 if (*mp->b_rptr & FLUSHBAND)
492                                         /*
493                                          * if it is a FLUSHBAND, do flushband.
494                                          */
495                                         flushband(qp, *(mp->b_rptr + 1),
496                                             FLUSHDATA);
497                                 else
498                                         flushq(qp, FLUSHDATA);
499                                 flush_flg = (*mp->b_rptr & ~FLUSHW) | FLUSHR;
500                         }
501                         if (*mp->b_rptr & FLUSHR) {
502                                 DBG(("got FLUSHR, set FLUSHW\n"));
503                                 flush_flg |= (*mp->b_rptr & ~FLUSHR) | FLUSHW;
504                         }
505                         if (flush_flg != 0 && ptmp->pts_rdq &&
506                             !(ptmp->pt_state & PTLOCK)) {
507                                 DBG(("putnext to pts\n"));
508                                 *mp->b_rptr = flush_flg;
```



```
509                                 putnext(ptmp->pts_rdq, mp);
510                         } else
511                                 freemsg(mp);
512                         break;
513                 }
514
515         case M_IOCTL:
516                 iocp = (struct iocblk *)mp->b_rptr;
517                 switch (iocp->ioc_cmd) {
518                 default:
519                         if ((ptmp->pt_state & PTLOCK) ||
520                             (ptmp->pts_rdq == NULL)) {
521                                 DBG(("got M_IOCTL but no slave\n"));
522                                 miocnak(qp, mp, 0, EINVAL);
523                                 PT_EXIT_READ(ptmp);
524                                 return;
525                         }
526                         (void) putq(qp, mp);
527                         break;
528                 case UNLKPT:
529                         mutex_enter(&ptmp->pt_lock);
530                         ptmp->pt_state &= ~PTLOCK;
531                         mutex_exit(&ptmp->pt_lock);
532                         /*FALLTHROUGH*/
533                 case ISPTM:
534                         DBG(("ack the UNLKPT/ISPTM\n"));
535                         miocack(qp, mp, 0, 0);
536                         break;
537                 case ZONEPT:
538                 {
539                         zoneid_t z;
540                         int error;
541
542                         if ((error = drv_priv(iocp->ioc_cr)) != 0) {
543                                 miocnak(qp, mp, 0, error);
544                                 break;
545                         }
546                         if ((error = miocpullup(mp, sizeof (zoneid_t))) != 0) {
547                                 miocnak(qp, mp, 0, error);
548                                 break;
549                         }
550                         z = *((zoneid_t *)mp->b_cont->b_rptr);
551                         if (z < MIN_ZONEID || z > MAX_ZONEID) {
552                                 miocnak(qp, mp, 0, EINVAL);
553                                 break;
554                         }
555
556                         mutex_enter(&ptmp->pt_lock);
557                         ptmp->pt_zoneid = z;
558                         mutex_exit(&ptmp->pt_lock);
559                         miocack(qp, mp, 0, 0);
560                         break;
561                 }
562                 case OWNERPT:
563                 {
564                         pt_own_t *ptop;
565                         int error;
566                         zone_t *zone;
567
568                         if ((error = miocpullup(mp, sizeof (pt_own_t))) != 0) {
569                                 miocnak(qp, mp, 0, error);
570                                 break;
571                         }
572
573                         zone = zone_find_by_id(ptmp->pt_zoneid);
574                         ptop = (pt_own_t *)mp->b_cont->b_rptr;
```

```
576                         if (!VALID_UID(ptop->pto_ruid, zone) ||
577                             !VALID_GID(ptop->pto_rgid, zone)) {
578                                 zone_rele(zone);
579                                 miocnak(qp, mp, 0, EINVAL);
580                                 break;
581                         }
582                         zone_rele(zone);
583                         mutex_enter(&ptmp->pt_lock);
584                         ptmp->pt_ruid = ptop->pto_ruid;
585                         ptmp->pt_rgid = ptop->pto_rgid;
586                         mutex_exit(&ptmp->pt_lock);
587                         miocack(qp, mp, 0, 0);
588                         break;
589                 }
590                 case PTMPTSOPENCB:
591                 {
592                         mblk_t          *dp;    /* ioctl reply data */
593                         ptmptsopencb_t  *ppocb;

595                         /* only allow the kernel to invoke this ioctl */
596                         if (iocp->ioc_cr != kcred) {
597                                 miocnak(qp, mp, 0, EINVAL);
598                                 break;
599                         }

601                         /* we don't support transparent ioctls */
602                         ASSERT(iocp->ioc_count != TRANSPARENT);
603                         if (iocp->ioc_count == TRANSPARENT) {
604                                 miocnak(qp, mp, 0, EINVAL);
605                                 break;
606                         }

608                         /* allocate a response message */
609                         dp = allocb(sizeof (ptmptsopencb_t), BPRI_MED);
610                         if (dp == NULL) {
611                                 miocnak(qp, mp, 0, EAGAIN);
612                                 break;
613                         }

615                         /* initialize the ioctl results */
616                         ppocb = (ptmptsopencb_t *)dp->b_rptr;
617                         ppocb->ppocb_func = ptmptsopencb;
618                         ppocb->ppocb_arg = (ptmptsopencb_arg_t)ptmp;

620                         /* send the reply data */
621                         mioc2ack(mp, dp, sizeof (ptmptsopencb_t), 0);
622                         qreply(qp, mp);
623 #endif /* ! codereview */
624                         break;
625                 }
626                 }
627                 break;

629         case M_READ:
630                 /* Caused by ldterm - can not pass to slave */
631                 freemsg(mp);
632                 break;

634         /*
635          * send other messages to slave
636          */
637         default:
638                 if ((ptmp->pt_state  & PTLOCK) || (ptmp->pts_rdq == NULL)) {
639                         DBG(("got msg. but no slave\n"));
640                         mp = mexchange(NULL, mp, 2, M_ERROR, -1);
```



```
641                         if (mp != NULL) {
642                                 mp->b_rptr[0] = NOERROR;
643                                 mp->b_rptr[1] = EINVAL;
644                                 qreply(qp, mp);
645                         }
646                         PT_EXIT_READ(ptmp);
647                         return;
648                 }
649                 DBG(("put msg on master's write queue\n"));
650                 (void) putq(qp, mp);
651                 break;
652         }
653         DBG(("return from ptmwput()\n"));
654         PT_EXIT_READ(ptmp);
655 }


658 /*
659  * enable the write side of the slave. This triggers the
660  * slave to send any messages queued on its write side to
661  * the read side of this master.
662  */
663 static void
664 ptmrsrv(queue_t *qp)
665 {
666         struct pt_ttys  *ptmp;

668         DBG(("entering ptmrsrv\n"));
669         ASSERT(qp->q_ptr);

671         ptmp = (struct pt_ttys *)qp->q_ptr;
672         PT_ENTER_READ(ptmp);
673         if (ptmp->pts_rdq) {
674                 qenable(WR(ptmp->pts_rdq));
675         }
676         PT_EXIT_READ(ptmp);
677         DBG(("leaving ptmrsrv\n"));
678 }


681 /*
682  * If there are messages on this queue that can be sent to
683  * slave, send them via putnext(). Else, if queued messages
684  * cannot be sent, leave them on this queue. If priority
685  * messages on this queue, send them to slave no matter what.
686  */
687 static void
688 ptmwsrv(queue_t *qp)
689 {
690         struct pt_ttys  *ptmp;
691         mblk_t          *mp;

693         DBG(("entering ptmwsrv\n"));
694         ASSERT(qp->q_ptr);

696         ptmp = (struct pt_ttys *)qp->q_ptr;

698         if ((mp = getq(qp)) == NULL) {
699                 /* If there are no messages there's nothing to do. */
700                 DBG(("leaving ptmwsrv (no messages)\n"));
701                 return;
702         }

704         PT_ENTER_READ(ptmp);
705         if ((ptmp->pt_state  & PTLOCK) || (ptmp->pts_rdq == NULL)) {
706                 DBG(("in master write srv proc but no slave\n"));
```

```
707                 /*
708                  * Free messages on the write queue and send
709                  * NAK for any M_IOCTL type messages to wakeup
710                  * the user process waiting for ACK/NAK from
711                  * the ioctl invocation
712                  */
713                 do {
714                         if (mp->b_datap->db_type == M_IOCTL)
715                                 miocnak(qp, mp, 0, EINVAL);
716                         else
717                                 freemsg(mp);
718                 } while ((mp = getq(qp)) != NULL);
719                 flushq(qp, FLUSHALL);

721                 mp = mexchange(NULL, NULL, 2, M_ERROR, -1);
722                 if (mp != NULL) {
723                         mp->b_rptr[0] = NOERROR;
724                         mp->b_rptr[1] = EINVAL;
725                         qreply(qp, mp);
726                 }
727                 PT_EXIT_READ(ptmp);
728                 return;
729         }
730         /*
731          * while there are messages on this write queue...
732          */
733         do {
734                 /*
735                  * if don't have control message and cannot put
736                  * msg. on slave's read queue, put it back on
737                  * this queue.
738                  */
739                 if (mp->b_datap->db_type <= QPCTL &&
740                     !bcanputnext(ptmp->pts_rdq, mp->b_band)) {
741                         DBG(("put msg. back on queue\n"));
742                         (void) putbq(qp, mp);
743                         break;
744                 }
745                 /*
746                  * else send the message up slave's stream
747                  */
748                 DBG(("send message to slave\n"));
749                 putnext(ptmp->pts_rdq, mp);
750         } while ((mp = getq(qp)) != NULL);
751         DBG(("leaving ptmwsrv\n"));
752         PT_EXIT_READ(ptmp);
753 }
```

```
*********************************************************
   30239 Tue Jan 14 16:17:25 2014
new/usr/src/uts/common/os/brand.c
Bring back LX zones.
*********************************************************
______unchanged_portion_omitted_
  46 #else /* !__sparcv9 */
  47 struct brand_mach_ops native_mach_ops  = {
  48                 NULL, NULL, NULL, NULL, NULL, NULL
  48                 NULL, NULL, NULL, NULL
  49 };
______unchanged_portion_omitted_
```

```
**********************************************************
   17182 Tue Jan 14 16:17:26 2014
new/usr/src/uts/common/os/pid.c
Bring back LX zones.
**********************************************************
______unchanged_portion_omitted_
```

```
115 struct pid *
116 pid_find(pid_t pid)
117 {
118         struct pid *pidp;

120         mutex_enter(&pidlinklock);
121         pidp = pid_lookup(pid);
122         mutex_exit(&pidlinklock);

124         return (pidp);
125 }

127 #endif /* ! codereview */
128 void
129 pid_setmin(void)
130 {
131         if (jump_pid && jump_pid > mpid)
132                 minpid = mpid = jump_pid;
133         else
134                 minpid = mpid;
135 }

137 /*
138  * When prslots are simply used as an index to determine a process' p_lock,
139  * adjacent prslots share adjacent p_locks.  On machines where the size
140  * of a mutex is smaller than that of a cache line (which, as of this writing,
141  * is true for all machines on which Solaris runs), this can potentially
142  * induce false sharing.  The standard solution for false sharing is to pad
143  * out one's data structures (in this case, struct plock).  However,
144  * given the size and (generally) sparse use of the proc_lock array, this
145  * is suboptimal.  We therefore stride through the proc_lock array with
146  * a stride of PLOCK_SHIFT.  PLOCK_SHIFT should be defined as:
147  *
148  *   log_2 (coherence_granularity / sizeof (kmutex_t))
149  *
150  * Under this scheme, false sharing is still possible -- but only when
151  * the number of active processes is very large.  Note that the one-to-one
152  * mapping between prslots and lockslots is maintained.
153  */
154 static int
155 pid_getlockslot(int prslot)
156 {
157         int even = (v.v_proc >> PLOCK_SHIFT) << PLOCK_SHIFT;
158         int perlap = even >> PLOCK_SHIFT;

160         if (prslot >= even)
161                 return (prslot);

163         return (((prslot % perlap) << PLOCK_SHIFT) + (prslot / perlap));
164 }

166 /*
167  * This function allocates a pid structure, a free pid, and optionally a
168  * slot in the proc table for it.
169  *
170  * pid_allocate() returns the new pid on success, -1 on failure.
171  */
172 pid_t
173 pid_allocate(proc_t *prp, pid_t pid, int flags)
```



```
174 {
175         struct pid *pidp;
176         union procent *pep;
177         pid_t newpid, startpid;

179         pidp = kmem_zalloc(sizeof (struct pid), KM_SLEEP);

181         mutex_enter(&pidlinklock);
182         if ((flags & PID_ALLOC_PROC) && (pep = procentfree) == NULL) {
183                 /*
184                  * ran out of /proc directory entries
185                  */
186                 goto failed;
187         }

189         if (pid != 0) {
190                 VERIFY(minpid == 0);
191                 VERIFY3P(pid, <, mpid);
192                 VERIFY3P(pid_lookup(pid), ==, NULL);
193                 newpid = pid;
194         } else {
195                 /*
196                  * Allocate a pid
197                  */
198                 ASSERT(minpid <= mpid && mpid < maxpid);

200                 startpid = mpid;
201                 for (;;) {
202                         newpid = mpid;
203                         if (++mpid == maxpid)
204                                 mpid = minpid;

206                         if (pid_lookup(newpid) == NULL)
207                                 break;

209                         if (mpid == startpid)
210                                 goto failed;
211                 }
212         }

214         /*
215          * Put pid into the pid hash table.
216          */
217         pidp->pid_link = HASHPID(newpid);
218         HASHPID(newpid) = pidp;
219         pidp->pid_ref = 1;
220         pidp->pid_id = newpid;

222         if (flags & PID_ALLOC_PROC) {
223                 procentfree = pep->pe_next;
224                 pidp->pid_prslot = pep - procdir;
225                 pep->pe_proc = prp;
226                 prp->p_pidp = pidp;
227                 prp->p_lockp = &proc_lock[pid_getlockslot(pidp->pid_prslot)];
228         } else {
229                 pidp->pid_prslot = 0;
230         }

232         mutex_exit(&pidlinklock);

234         return (newpid);

236 failed:
237         mutex_exit(&pidlinklock);
238         kmem_free(pidp, sizeof (struct pid));
239         return (-1);
```

```
}

/*
 * decrement the reference count for pid
 */
int
pid_rele(struct pid *pidp)
{
        struct pid **pidpp;

        mutex_enter(&pidlinklock);
        ASSERT(pidp != &pid0);

        pidpp = &HASHPID(pidp->pid_id);
        for (;;) {
                ASSERT(*pidpp != NULL);
                if (*pidpp == pidp)
                        break;
                pidpp = &(*pidpp)->pid_link;
        }

        *pidpp = pidp->pid_link;
        mutex_exit(&pidlinklock);

        kmem_free(pidp, sizeof (*pidp));
        return (0);
}

void
proc_entry_free(struct pid *pidp)
{
        mutex_enter(&pidlinklock);
        pidp->pid_prinactive = 1;
        procdir[pidp->pid_prslot].pe_next = procentfree;
        procentfree = &procdir[pidp->pid_prslot];
        mutex_exit(&pidlinklock);
}

/*
 * The original task needs to be passed in since the process has already been
 * detached from the task at this point in time.
 */
void
pid_exit(proc_t *prp, struct task *tk)
{
        struct pid *pidp;
        zone_t  *zone = prp->p_zone;

        ASSERT(MUTEX_HELD(&pidlock));

        /*
         * Exit process group.  If it is NULL, it's because fork failed
         * before calling pgjoin().
         */
        ASSERT(prp->p_pgidp != NULL || prp->p_stat == SIDL);
        if (prp->p_pgidp != NULL)
                pgexit(prp);

        sess_rele(prp->p_sessp, B_TRUE);

        pidp = prp->p_pidp;

        proc_entry_free(pidp);

        if (audit_active)
                audit_pfree(prp);
```



```
        if (practive == prp) {
                practive = prp->p_next;
        }

        if (prp->p_next) {
                prp->p_next->p_prev = prp->p_prev;
        }
        if (prp->p_prev) {
                prp->p_prev->p_next = prp->p_next;
        }

        PID_RELE(pidp);

        mutex_destroy(&prp->p_crlock);
        kmem_cache_free(process_cache, prp);
        nproc--;

        /*
         * Decrement the process counts of the original task, project and zone.
         */
        mutex_enter(&zone->zone_nlwps_lock);
        tk->tk_nprocs--;
        tk->tk_proj->kpj_nprocs--;
        zone->zone_nprocs--;
        mutex_exit(&zone->zone_nlwps_lock);
}

/*
 * Find a process visible from the specified zone given its process ID.
 */
proc_t *
prfind_zone(pid_t pid, zoneid_t zoneid)
{
        struct pid *pidp;
        proc_t *p;

        ASSERT(MUTEX_HELD(&pidlock));

        mutex_enter(&pidlinklock);
        pidp = pid_lookup(pid);
        mutex_exit(&pidlinklock);
        if (pidp != NULL && pidp->pid_prinactive == 0) {
                p = procdir[pidp->pid_prslot].pe_proc;
                if (zoneid == ALL_ZONES || p->p_zone->zone_id == zoneid)
                        return (p);
        }
        return (NULL);
}

/*
 * Find a process given its process ID.  This obeys zone restrictions,
 * so if the caller is in a non-global zone it won't find processes
 * associated with other zones.  Use prfind_zone(pid, ALL_ZONES) to
 * bypass this restriction.
 */
proc_t *
prfind(pid_t pid)
{
        zoneid_t zoneid;

        if (INGLOBALZONE(curproc))
                zoneid = ALL_ZONES;
        else
                zoneid = getzoneid();
        return (prfind_zone(pid, zoneid));
```

```
372 }
374 proc_t *
375 pgfind_zone(pid_t pgid, zoneid_t zoneid)
376 {
377         struct pid *pidp;
379         ASSERT(MUTEX_HELD(&pidlock));
381         mutex_enter(&pidlinklock);
382         pidp = pid_lookup(pgid);
383         mutex_exit(&pidlinklock);
384         if (pidp != NULL) {
385                 proc_t *p = pidp->pid_pglink;
387                 if (zoneid == ALL_ZONES || pgid == 0 || p == NULL ||
388                     p->p_zone->zone_id == zoneid)
389                         return (p);
390         }
391         return (NULL);
392 }
394 /*
395  * return the head of the list of processes whose process group ID is 'pgid',
396  * or NULL, if no such process group
397  */
398 proc_t *
399 pgfind(pid_t pgid)
400 {
401         zoneid_t zoneid;
403         if (INGLOBALZONE(curproc))
404                 zoneid = ALL_ZONES;
405         else
406                 zoneid = getzoneid();
407         return (pgfind_zone(pgid, zoneid));
408 }
410 /*
411  * Sets P_PR_LOCK on a non-system process.  Process must be fully created
412  * and not exiting to succeed.
413  *
414  * Returns 0 on success.
415  * Returns 1 if P_PR_LOCK is set.
416  * Returns -1 if proc is in invalid state.
417  */
418 int
419 sprtrylock_proc(proc_t *p)
420 {
421         ASSERT(MUTEX_HELD(&p->p_lock));
423         /* skip system and incomplete processes */
424         if (p->p_stat == SIDL || p->p_stat == SZOMB ||
425             (p->p_flag & (SSYS | SEXITING | SEXITLWPS))) {
426                 return (-1);
427         }
429         if (p->p_proc_flag & P_PR_LOCK)
430                 return (1);
432         p->p_proc_flag |= P_PR_LOCK;
433         THREAD_KPRI_REQUEST();
435         return (0);
436 }
```



```
438 /*
439  * Wait for P_PR_LOCK to become clear.  Returns with p_lock dropped,
440  * and the proc pointer no longer valid, as the proc may have exited.
441  */
442 void
443 sprwaitlock_proc(proc_t *p)
444 {
445         kmutex_t *mp;
447         ASSERT(MUTEX_HELD(&p->p_lock));
448         ASSERT(p->p_proc_flag & P_PR_LOCK);
450         /*
451          * p_lock is persistent, but p itself is not -- it could
452          * vanish during cv_wait().  Load p->p_lock now so we can
453          * drop it after cv_wait() without referencing p.
454          */
455         mp = &p->p_lock;
456         cv_wait(&pr_pid_cv[p->p_slot], mp);
457         mutex_exit(mp);
458 }
460 /*
461  * If pid exists, find its proc, acquire its p_lock and mark it P_PR_LOCK.
462  * Returns the proc pointer on success, NULL on failure.  sprlock() is
463  * really just a stripped-down version of pr_p_lock() to allow practive
464  * walkers like dofusers() and dumpsys() to synchronize with /proc.
465  */
466 proc_t *
467 sprlock_zone(pid_t pid, zoneid_t zoneid)
468 {
469         proc_t *p;
470         int ret;
472         for (;;) {
473                 mutex_enter(&pidlock);
474                 if ((p = prfind_zone(pid, zoneid)) == NULL) {
475                         mutex_exit(&pidlock);
476                         return (NULL);
477                 }
478                 mutex_enter(&p->p_lock);
479                 mutex_exit(&pidlock);
481                 if (panicstr)
482                         return (p);
484                 ret = sprtrylock_proc(p);
485                 if (ret == -1) {
486                         mutex_exit(&p->p_lock);
487                         return (NULL);
488                 } else if (ret == 0) {
489                         break;
490                 }
491                 sprwaitlock_proc(p);
492         }
493         return (p);
494 }
496 proc_t *
497 sprlock(pid_t pid)
498 {
499         zoneid_t zoneid;
501         if (INGLOBALZONE(curproc))
502                 zoneid = ALL_ZONES;
503         else
```

```
504                 zoneid = getzoneid();
505         return (sprlock_zone(pid, zoneid));
506 }

508 void
509 sprlock_proc(proc_t *p)
510 {
511         ASSERT(MUTEX_HELD(&p->p_lock));

513         while (p->p_proc_flag & P_PR_LOCK) {
514                 cv_wait(&pr_pid_cv[p->p_slot], &p->p_lock);
515         }

517         p->p_proc_flag |= P_PR_LOCK;
518         THREAD_KPRI_REQUEST();
519 }

521 void
522 sprunlock(proc_t *p)
523 {
524         if (panicstr) {
525                 mutex_exit(&p->p_lock);
526                 return;
527         }

529         ASSERT(p->p_proc_flag & P_PR_LOCK);
530         ASSERT(MUTEX_HELD(&p->p_lock));

532         cv_signal(&pr_pid_cv[p->p_slot]);
533         p->p_proc_flag &= ~P_PR_LOCK;
534         mutex_exit(&p->p_lock);
535         THREAD_KPRI_RELEASE();
536 }

538 void
539 pid_init(void)
540 {
541         int i;

543         pid_hashsz = 1 << highbit(v.v_proc / pid_hashlen);

545         pidhash = kmem_zalloc(sizeof (struct pid *) * pid_hashsz, KM_SLEEP);
546         procdir = kmem_alloc(sizeof (union procent) * v.v_proc, KM_SLEEP);
547         pr_pid_cv = kmem_zalloc(sizeof (kcondvar_t) * v.v_proc, KM_SLEEP);
548         proc_lock = kmem_zalloc(sizeof (struct plock) * v.v_proc, KM_SLEEP);

550         nproc = 1;
551         practive = proc_sched;
552         proc_sched->p_next = NULL;
553         procdir[0].pe_proc = proc_sched;

555         procentfree = &procdir[1];
556         for (i = 1; i < v.v_proc - 1; i++)
557                 procdir[i].pe_next = &procdir[i+1];
558         procdir[i].pe_next = NULL;

560         HASHPID(0) = &pid0;

562         upcount_init();
563 }

565 proc_t *
566 pid_entry(int slot)
567 {
568         union procent *pep;
569         proc_t *prp;
```



```
571         ASSERT(MUTEX_HELD(&pidlock));
572         ASSERT(slot >= 0 && slot < v.v_proc);

574         pep = procdir[slot].pe_next;
575         if (pep >= procdir && pep < &procdir[v.v_proc])
576                 return (NULL);
577         prp = procdir[slot].pe_proc;
578         if (prp != 0 && prp->p_stat == SIDL)
579                 return (NULL);
580         return (prp);
581 }

583 /*
584  * Send the specified signal to all processes whose process group ID is
585  * equal to 'pgid'
586  */

588 void
589 signal(pid_t pgid, int sig)
590 {
591         struct pid *pidp;
592         proc_t *prp;

594         mutex_enter(&pidlock);
595         mutex_enter(&pidlinklock);
596         if (pgid == 0 || (pidp = pid_lookup(pgid)) == NULL) {
597                 mutex_exit(&pidlinklock);
598                 mutex_exit(&pidlock);
599                 return;
600         }
601         mutex_exit(&pidlinklock);
602         for (prp = pidp->pid_pglink; prp; prp = prp->p_pglink) {
603                 mutex_enter(&prp->p_lock);
604                 sigtoproc(prp, NULL, sig);
605                 mutex_exit(&prp->p_lock);
606         }
607         mutex_exit(&pidlock);
608 }

610 /*
611  * Send the specified signal to the specified process
612  */

614 void
615 prsignal(struct pid *pidp, int sig)
616 {
617         if (!(pidp->pid_prinactive))
618                 psignal(procdir[pidp->pid_prslot].pe_proc, sig);
619 }

621 #include <sys/sunddi.h>

623 /*
624  * DDI/DKI interfaces for drivers to send signals to processes
625  */

627 /*
628  * obtain an opaque reference to a process for signaling
629  */
630 void *
631 proc_ref(void)
632 {
633         struct pid *pidp;

635         mutex_enter(&pidlock);
```

```
636         pidp = curproc->p_pidp;
637         PID_HOLD(pidp);
638         mutex_exit(&pidlock);

640         return (pidp);
641 }

643 /*
644  * release a reference to a process
645  * - a process can exit even if a driver has a reference to it
646  * - one proc_unref for every proc_ref
647  */
648 void
649 proc_unref(void *pref)
650 {
651         mutex_enter(&pidlock);
652         PID_RELE((struct pid *)pref);
653         mutex_exit(&pidlock);
654 }

656 /*
657  * send a signal to a process
658  *
659  * - send the process the signal
660  * - if the process went away, return a -1
661  * - if the process is still there return 0
662  */
663 int
664 proc_signal(void *pref, int sig)
665 {
666         struct pid *pidp = pref;

668         prsignal(pidp, sig);
669         return (pidp->pid_prinactive ? -1 : 0);
670 }


673 static struct upcount   **upc_hash;     /* a boot time allocated array */
674 static ulong_t          upc_hashmask;
675 #define UPC_HASH(x, y)  ((ulong_t)(x ^ y) & upc_hashmask)

677 /*
678  * Get us off the ground.  Called once at boot.
679  */
680 void
681 upcount_init(void)
682 {
683         ulong_t upc_hashsize;

685         /*
686          * An entry per MB of memory is our current guess
687          */
688         /*
689          * 2^20 is a meg, so shifting right by 20 - PAGESHIFT
690          * converts pages to megs (without overflowing a u_int
691          * if you have more than 4G of memory, like ptob(physmem)/1M
692          * would).
693          */
694         upc_hashsize = (1 << highbit(physmem >> (20 - PAGESHIFT)));
695         upc_hashmask = upc_hashsize - 1;
696         upc_hash = kmem_zalloc(upc_hashsize * sizeof (struct upcount *),
697             KM_SLEEP);
698 }

700 /*
701  * Increment the number of processes associated with a given uid and zoneid.
```



```
702  */
703 void
704 upcount_inc(uid_t uid, zoneid_t zoneid)
705 {
706         struct upcount  **upc, **hupc;
707         struct upcount  *new;

709         ASSERT(MUTEX_HELD(&pidlock));
710         new = NULL;
711         hupc = &upc_hash[UPC_HASH(uid, zoneid)];
712 top:
713         upc = hupc;
714         while ((*upc) != NULL) {
715                 if ((*upc)->up_uid == uid && (*upc)->up_zoneid == zoneid) {
716                         (*upc)->up_count++;
717                         if (new) {
718                                 /*
719                                  * did not need 'new' afterall.
720                                  */
721                                 kmem_free(new, sizeof (*new));
722                         }
723                         return;
724                 }
725                 upc = &(*upc)->up_next;
726         }

728         /*
729          * There is no entry for this <uid,zoneid> pair.
730          * Allocate one.  If we have to drop pidlock, check
731          * again.
732          */
733         if (new == NULL) {
734                 new = (struct upcount *)kmem_alloc(sizeof (*new), KM_NOSLEEP);
735                 if (new == NULL) {
736                         mutex_exit(&pidlock);
737                         new = (struct upcount *)kmem_alloc(sizeof (*new),
738                             KM_SLEEP);
739                         mutex_enter(&pidlock);
740                         goto top;
741                 }
742         }


745         /*
746          * On the assumption that a new user is going to do some
747          * more forks, put the new upcount structure on the front.
748          */
749         upc = hupc;

751         new->up_uid = uid;
752         new->up_zoneid = zoneid;
753         new->up_count = 1;
754         new->up_next = *upc;

756         *upc = new;
757 }

759 /*
760  * Decrement the number of processes a given uid and zoneid has.
761  */
762 void
763 upcount_dec(uid_t uid, zoneid_t zoneid)
764 {
765         struct  upcount **upc;
766         struct  upcount *done;
```

```
768         ASSERT(MUTEX_HELD(&pidlock));

770         upc = &upc_hash[UPC_HASH(uid, zoneid)];
771         while ((*upc) != NULL) {
772                 if ((*upc)->up_uid == uid && (*upc)->up_zoneid == zoneid) {
773                         (*upc)->up_count--;
774                         if ((*upc)->up_count == 0) {
775                                 done = *upc;
776                                 *upc = (*upc)->up_next;
777                                 kmem_free(done, sizeof (*done));
778                         }
779                         return;
780                 }
781                 upc = &(*upc)->up_next;
782         }
783         cmn_err(CE_PANIC, "decr_upcount-off the end");
784 }

786 /*
787  * Returns the number of processes a uid has.
788  * Non-existent uid's are assumed to have no processes.
789  */
790 int
791 upcount_get(uid_t uid, zoneid_t zoneid)
792 {
793         struct  upcount *upc;

795         ASSERT(MUTEX_HELD(&pidlock));

797         upc = upc_hash[UPC_HASH(uid, zoneid)];
798         while (upc != NULL) {
799                 if (upc->up_uid == uid && upc->up_zoneid == zoneid) {
800                         return (upc->up_count);
801                 }
802                 upc = upc->up_next;
803         }
804         return (0);
805 }
```

```
**********************************************************
  217247 Tue Jan 14 16:17:26 2014
new/usr/src/uts/common/os/streamio.c
Bring back LX zones.
**********************************************************
______unchanged_portion_omitted_

3093 #endif  /* _LP64 */

3095 /*
3096  * Determine type of job control semantics expected by user.  The
3097  * possibilities are:
3098  *      JCREAD  - Behaves like read() on fd; send SIGTTIN
3099  *      JCWRITE - Behaves like write() on fd; send SIGTTOU if TOSTOP set
3100  *      JCSETP  - Sets a value in the stream; send SIGTTOU, ignore TOSTOP
3101  *      JCGETP  - Gets a value in the stream; no signals.
3102  * See straccess in strsubr.c for usage of these values.
3103  *
3104  * This routine also returns -1 for I_STR as a special case; the
3105  * caller must call again with the real ioctl number for
3106  * classification.
3107  */
3108 static int
3109 job_control_type(int cmd)
3110 {
3111         switch (cmd) {
3112         case I_STR:
3113                 return (-1);

3115         case I_RECVFD:
3116         case I_E_RECVFD:
3117                 return (JCREAD);

3119         case I_FDINSERT:
3120         case I_SENDFD:
3121                 return (JCWRITE);

3123         case TCSETA:
3124         case TCSETAW:
3125         case TCSETAF:
3126         case TCSBRK:
3127         case TCXONC:
3128         case TCFLSH:
3129         case TCDSET:    /* Obsolete */
3130         case TIOCSWINSZ:
3131         case TCSETS:
3132         case TCSETSW:
3133         case TCSETSF:
3134         case TIOCSETD:
3135         case TIOCHPCL:
3136         case TIOCSETP:
3137         case TIOCSETN:
3138         case TIOCEXCL:
3139         case TIOCNXCL:
3140         case TIOCFLUSH:
3141         case TIOCSETC:
3142         case TIOCLBIS:
3143         case TIOCLBIC:
3144         case TIOCLSET:
3145         case TIOCSBRK:
3146         case TIOCCBRK:
3147         case TIOCSDTR:
3148         case TIOCCDTR:
3149         case TIOCSLTC:
3150         case TIOCSTOP:
3151         case TIOCSTART:
```



```
3152         case TIOCSTI:
3153         case TIOCSPGRP:
3154         case TIOCMSET:
3155         case TIOCMBIS:
3156         case TIOCMBIC:
3157         case TIOCREMOTE:
3158         case TIOCSIGNAL:
3159         case LDSETT:
3160         case LDSMAP:    /* Obsolete */
3161         case DIOCSETP:
3162         case I_FLUSH:
3163         case I_SRDOPT:
3164         case I_SETSIG:
3165         case I_SWROPT:
3166         case I_FLUSHBAND:
3167         case I_SETCLTIME:
3168         case I_SERROPT:
3169         case I_ESETSIG:
3170         case FIONBIO:
3171         case FIOASYNC:
3172         case FIOSETOWN:
3173         case JBOOT:     /* Obsolete */
3174         case JTERM:     /* Obsolete */
3175         case JTIMOM:    /* Obsolete */
3176         case JZOMBOOT:  /* Obsolete */
3177         case JAGENT:    /* Obsolete */
3178         case JTRUN:     /* Obsolete */
3179         case JXTPROTO:  /* Obsolete */
3180         case TIOCSETLD:
3181 #endif /* ! codereview */
3182                 return (JCSETP);
3183         }

3185         return (JCGETP);
3186 }

3188 /*
3189  * ioctl for streams
3190  */
3191 int
3192 strioctl(struct vnode *vp, int cmd, intptr_t arg, int flag, int copyflag,
3193     cred_t *crp, int *rvalp)
3194 {
3195         struct stdata *stp;
3196         struct strcmd *scp;
3197         struct strioctl strioc;
3198         struct uio uio;
3199         struct iovec iov;
3200         int access;
3201         mblk_t *mp;
3202         int error = 0;
3203         int done = 0;
3204         ssize_t rmin, rmax;
3205         queue_t *wrq;
3206         queue_t *rdq;
3207         boolean_t kioctl = B_FALSE;
3208         uint32_t auditing = AU_AUDITING();

3210         if (flag & FKIOCTL) {
3211                 copyflag = K_TO_K;
3212                 kioctl = B_TRUE;
3213         }
3214         ASSERT(vp->v_stream);
3215         ASSERT(copyflag == U_TO_K || copyflag == K_TO_K);
3216         stp = vp->v_stream;
```

```
3218         TRACE_3(TR_FAC_STREAMS_FR, TR_IOCTL_ENTER,
3219             "strioctl:stp %p cmd %X arg %lX", stp, cmd, arg);

3221         /*
3222          * If the copy is kernel to kernel, make sure that the FNATIVE
3223          * flag is set.  After this it would be a serious error to have
3224          * no model flag.
3225          */
3226         if (copyflag == K_TO_K)
3227                 flag = (flag & ~FMODELS) | FNATIVE;

3229         ASSERT((flag & FMODELS) != 0);

3231         wrq = stp->sd_wrq;
3232         rdq = _RD(wrq);

3234         access = job_control_type(cmd);

3236         /* We should never see these here, should be handled by iwscn */
3237         if (cmd == SRIOCSREDIR || cmd == SRIOCISREDIR)
3238                 return (EINVAL);

3240         mutex_enter(&stp->sd_lock);
3241         if ((access != -1) && ((error = i_straccess(stp, access)) != 0)) {
3242                 mutex_exit(&stp->sd_lock);
3243                 return (error);
3244         }
3245         mutex_exit(&stp->sd_lock);

3247         /*
3248          * Check for sgttyb-related ioctls first, and complain as
3249          * necessary.
3250          */
3251         switch (cmd) {
3252         case TIOCGETP:
3253         case TIOCSETP:
3254         case TIOCSETN:
3255                 if (sgttyb_handling >= 2 && !sgttyb_complaint) {
3256                         sgttyb_complaint = B_TRUE;
3257                         cmn_err(CE_NOTE,
3258                             "application used obsolete TIOC[GS]ET");
3259                 }
3260                 if (sgttyb_handling >= 3) {
3261                         tsignal(curthread, SIGSYS);
3262                         return (EIO);
3263                 }
3264                 break;
3265         }

3267         mutex_enter(&stp->sd_lock);

3269         switch (cmd) {
3270         case I_RECVFD:
3271         case I_E_RECVFD:
3272         case I_PEEK:
3273         case I_NREAD:
3274         case FIONREAD:
3275         case FIORDCHK:
3276         case I_ATMARK:
3277         case FIONBIO:
3278         case FIOASYNC:
3279                 if (stp->sd_flag & (STRDERR|STPLEX)) {
3280                         error = strgeterr(stp, STRDERR|STPLEX, 0);
3281                         if (error != 0) {
3282                                 mutex_exit(&stp->sd_lock);
3283                                 return (error);
```



```
3284                         }
3285                 }
3286                 break;

3288         default:
3289                 if (stp->sd_flag & (STRDERR|STWRERR|STPLEX)) {
3290                         error = strgeterr(stp, STRDERR|STWRERR|STPLEX, 0);
3291                         if (error != 0) {
3292                                 mutex_exit(&stp->sd_lock);
3293                                 return (error);
3294                         }
3295                 }
3296         }

3298         mutex_exit(&stp->sd_lock);

3300         switch (cmd) {
3301         default:
3302                 /*
3303                  * The stream head has hardcoded knowledge of a
3304                  * miscellaneous collection of terminal-, keyboard- and
3305                  * mouse-related ioctls, enumerated below.  This hardcoded
3306                  * knowledge allows the stream head to automatically
3307                  * convert transparent ioctl requests made by userland
3308                  * programs into I_STR ioctls which many old STREAMS
3309                  * modules and drivers require.
3310                  *
3311                  * No new ioctls should ever be added to this list.
3312                  * Instead, the STREAMS module or driver should be written
3313                  * to either handle transparent ioctls or require any
3314                  * userland programs to use I_STR ioctls (by returning
3315                  * EINVAL to any transparent ioctl requests).
3316                  *
3317                  * More importantly, removing ioctls from this list should
3318                  * be done with the utmost care, since our STREAMS modules
3319                  * and drivers *count* on the stream head performing this
3320                  * conversion, and thus may panic while processing
3321                  * transparent ioctl request for one of these ioctls (keep
3322                  * in mind that third party modules and drivers may have
3323                  * similar problems).
3324                  */
3325                 if (((cmd & IOCTYPE) == LDIOC) ||
3326                     ((cmd & IOCTYPE) == tIOC) ||
3327                     ((cmd & IOCTYPE) == TIOC) ||
3328                     ((cmd & IOCTYPE) == KIOC) ||
3329                     ((cmd & IOCTYPE) == MSIOC) ||
3330                     ((cmd & IOCTYPE) == VUIOC)) {
3331                         /*
3332                          * The ioctl is a tty ioctl - set up strioc buffer
3333                          * and call strdoioctl() to do the work.
3334                          */
3335                         if (stp->sd_flag & STRHUP)
3336                                 return (ENXIO);
3337                         strioc.ic_cmd = cmd;
3338                         strioc.ic_timout = INFTIM;

3340                         switch (cmd) {

3342                         case TCXONC:
3343                         case TCSBRK:
3344                         case TCFLSH:
3345                         case TCDSET:
3346                                 {
3347                                 int native_arg = (int)arg;
3348                                 strioc.ic_len = sizeof (int);
3349                                 strioc.ic_dp = (char *)&native_arg;
```

```
3350                            return (strdoioctl(stp, &strioc, flag,
3351                                K_TO_K, crp, rvalp));
3352                            }

3354                    case TCSETA:
3355                    case TCSETAW:
3356                    case TCSETAF:
3357                            strioc.ic_len = sizeof (struct termio);
3358                            strioc.ic_dp = (char *)arg;
3359                            return (strdoioctl(stp, &strioc, flag,
3360                                copyflag, crp, rvalp));

3362                    case TCSETS:
3363                    case TCSETSW:
3364                    case TCSETSF:
3365                            strioc.ic_len = sizeof (struct termios);
3366                            strioc.ic_dp = (char *)arg;
3367                            return (strdoioctl(stp, &strioc, flag,
3368                                copyflag, crp, rvalp));

3370                    case LDSETT:
3371                            strioc.ic_len = sizeof (struct termcb);
3372                            strioc.ic_dp = (char *)arg;
3373                            return (strdoioctl(stp, &strioc, flag,
3374                                copyflag, crp, rvalp));

3376                    case TIOCSETP:
3377                            strioc.ic_len = sizeof (struct sgttyb);
3378                            strioc.ic_dp = (char *)arg;
3379                            return (strdoioctl(stp, &strioc, flag,
3380                                copyflag, crp, rvalp));

3382                    case TIOCSTI:
3383                            if ((flag & FREAD) == 0 &&
3384                                secpolicy_sti(crp) != 0) {
3385                                    return (EPERM);
3386                            }
3387                            mutex_enter(&stp->sd_lock);
3388                            mutex_enter(&curproc->p_splock);
3389                            if (stp->sd_sidp != curproc->p_sessp->s_sidp &&
3390                                secpolicy_sti(crp) != 0) {
3391                                    mutex_exit(&curproc->p_splock);
3392                                    mutex_exit(&stp->sd_lock);
3393                                    return (EACCES);
3394                            }
3395                            mutex_exit(&curproc->p_splock);
3396                            mutex_exit(&stp->sd_lock);

3398                            strioc.ic_len = sizeof (char);
3399                            strioc.ic_dp = (char *)arg;
3400                            return (strdoioctl(stp, &strioc, flag,
3401                                copyflag, crp, rvalp));

3403                    case TIOCSWINSZ:
3404                            strioc.ic_len = sizeof (struct winsize);
3405                            strioc.ic_dp = (char *)arg;
3406                            return (strdoioctl(stp, &strioc, flag,
3407                                copyflag, crp, rvalp));

3409                    case TIOCSSIZE:
3410                            strioc.ic_len = sizeof (struct ttysize);
3411                            strioc.ic_dp = (char *)arg;
3412                            return (strdoioctl(stp, &strioc, flag,
3413                                copyflag, crp, rvalp));

3415                    case TIOCSSOFTCAR:
```



```
3416                    case KIOCTRANS:
3417                    case KIOCTRANSABLE:
3418                    case KIOCCMD:
3419                    case KIOCSDIRECT:
3420                    case KIOCSCOMPAT:
3421                    case KIOCSKABORTEN:
3422                    case KIOCSRPTDELAY:
3423                    case KIOCSRPTRATE:
3424                    case VUIDSFORMAT:
3425                    case TIOCSPPS:
3426                            strioc.ic_len = sizeof (int);
3427                            strioc.ic_dp = (char *)arg;
3428                            return (strdoioctl(stp, &strioc, flag,
3429                                copyflag, crp, rvalp));

3431                    case KIOCSETKEY:
3432                    case KIOCGETKEY:
3433                            strioc.ic_len = sizeof (struct kiockey);
3434                            strioc.ic_dp = (char *)arg;
3435                            return (strdoioctl(stp, &strioc, flag,
3436                                copyflag, crp, rvalp));

3438                    case KIOCSKEY:
3439                    case KIOCGKEY:
3440                            strioc.ic_len = sizeof (struct kiockeymap);
3441                            strioc.ic_dp = (char *)arg;
3442                            return (strdoioctl(stp, &strioc, flag,
3443                                copyflag, crp, rvalp));

3445                    case KIOCSLED:
3446                            /* arg is a pointer to char */
3447                            strioc.ic_len = sizeof (char);
3448                            strioc.ic_dp = (char *)arg;
3449                            return (strdoioctl(stp, &strioc, flag,
3450                                copyflag, crp, rvalp));

3452                    case MSIOSETPARMS:
3453                            strioc.ic_len = sizeof (Ms_parms);
3454                            strioc.ic_dp = (char *)arg;
3455                            return (strdoioctl(stp, &strioc, flag,
3456                                copyflag, crp, rvalp));

3458                    case VUIDSADDR:
3459                    case VUIDGADDR:
3460                            strioc.ic_len = sizeof (struct vuid_addr_probe);
3461                            strioc.ic_dp = (char *)arg;
3462                            return (strdoioctl(stp, &strioc, flag,
3463                                copyflag, crp, rvalp));

3465                    /*
3466                     * These M_IOCTL's don't require any data to be sent
3467                     * downstream, and the driver will allocate and link
3468                     * on its own mblk_t upon M_IOCACK -- thus we set
3469                     * ic_len to zero and set ic_dp to arg so we know
3470                     * where to copyout to later.
3471                     */
3472                    case TIOCGSOFTCAR:
3473                    case TIOCGWINSZ:
3474                    case TIOCGSIZE:
3475                    case KIOCGTRANS:
3476                    case KIOCGTRANSABLE:
3477                    case KIOCTYPE:
3478                    case KIOCGDIRECT:
3479                    case KIOCGCOMPAT:
3480                    case KIOCLAYOUT:
3481                    case KIOCGLED:
```

```
3482                        case MSIOGETPARMS:
3483                        case MSIOBUTTONS:
3484                        case VUIDGFORMAT:
3485                        case TIOCGPPS:
3486                        case TIOCGPPSEV:
3487                        case TCGETA:
3488                        case TCGETS:
3489                        case LDGETT:
3490                        case TIOCGETP:
3491                        case KIOCGRPTDELAY:
3492                        case KIOCGRPTRATE:
3493                                strioc.ic_len = 0;
3494                                strioc.ic_dp = (char *)arg;
3495                                return (strdoioctl(stp, &strioc, flag,
3496                                    copyflag, crp, rvalp));
3497                        }
3498                }

3500                /*
3501                 * Unknown cmd - send it down as a transparent ioctl.
3502                 */
3503                strioc.ic_cmd = cmd;
3504                strioc.ic_timout = INFTIM;
3505                strioc.ic_len = TRANSPARENT;
3506                strioc.ic_dp = (char *)&arg;

3508                return (strdoioctl(stp, &strioc, flag, copyflag, crp, rvalp));

3510        case I_STR:
3511                /*
3512                 * Stream ioctl.  Read in an strioctl buffer from the user
3513                 * along with any data specified and send it downstream.
3514                 * Strdoioctl will wait allow only one ioctl message at
3515                 * a time, and waits for the acknowledgement.
3516                 */

3518                if (stp->sd_flag & STRHUP)
3519                        return (ENXIO);

3521                error = strcopyin_strioctl((void *)arg, &strioc, flag,
3522                    copyflag);
3523                if (error != 0)
3524                        return (error);

3526                if ((strioc.ic_len < 0) || (strioc.ic_timout < -1))
3527                        return (EINVAL);

3529                access = job_control_type(strioc.ic_cmd);
3530                mutex_enter(&stp->sd_lock);
3531                if ((access != -1) &&
3532                    ((error = i_straccess(stp, access)) != 0)) {
3533                        mutex_exit(&stp->sd_lock);
3534                        return (error);
3535                }
3536                mutex_exit(&stp->sd_lock);

3538                /*
3539                 * The I_STR facility provides a trap door for malicious
3540                 * code to send down bogus streamio(7I) ioctl commands to
3541                 * unsuspecting STREAMS modules and drivers which expect to
3542                 * only get these messages from the stream head.
3543                 * Explicitly prohibit any streamio ioctls which can be
3544                 * passed downstream by the stream head.  Note that we do
3545                 * not block all streamio ioctls because the ioctl
3546                 * numberspace is not well managed and thus it's possible
3547                 * that a module or driver's ioctl numbers may accidentally
```



```
3548                 * collide with them.
3549                 */
3550                switch (strioc.ic_cmd) {
3551                case I_LINK:
3552                case I_PLINK:
3553                case I_UNLINK:
3554                case I_PUNLINK:
3555                case _I_GETPEERCRED:
3556                case _I_PLINK_LH:
3557                        return (EINVAL);
3558                }

3560                error = strdoioctl(stp, &strioc, flag, copyflag, crp, rvalp);
3561                if (error == 0) {
3562                        error = strcopyout_strioctl(&strioc, (void *)arg,
3563                            flag, copyflag);
3564                }
3565                return (error);

3567        case _I_CMD:
3568                /*
3569                 * Like I_STR, but without using M_IOC* messages and without
3570                 * copyins/copyouts beyond the passed-in argument.
3571                 */
3572                if (stp->sd_flag & STRHUP)
3573                        return (ENXIO);

3575                if ((scp = kmem_alloc(sizeof (strcmd_t), KM_NOSLEEP)) == NULL)
3576                        return (ENOMEM);

3578                if (copyin((void *)arg, scp, sizeof (strcmd_t))) {
3579                        kmem_free(scp, sizeof (strcmd_t));
3580                        return (EFAULT);
3581                }

3583                access = job_control_type(scp->sc_cmd);
3584                mutex_enter(&stp->sd_lock);
3585                if (access != -1 && (error = i_straccess(stp, access)) != 0) {
3586                        mutex_exit(&stp->sd_lock);
3587                        kmem_free(scp, sizeof (strcmd_t));
3588                        return (error);
3589                }
3590                mutex_exit(&stp->sd_lock);

3592                *rvalp = 0;
3593                if ((error = strdocmd(stp, scp, crp)) == 0) {
3594                        if (copyout(scp, (void *)arg, sizeof (strcmd_t)))
3595                                error = EFAULT;
3596                }
3597                kmem_free(scp, sizeof (strcmd_t));
3598                return (error);

3600        case I_NREAD:
3601                /*
3602                 * Return number of bytes of data in first message
3603                 * in queue in "arg" and return the number of messages
3604                 * in queue in return value.
3605                 */
3606        {
3607                size_t  size;
3608                int     retval;
3609                int     count = 0;

3611                mutex_enter(QLOCK(rdq));

3613                size = msgdsize(rdq->q_first);
```

```
3614                for (mp = rdq->q_first; mp != NULL; mp = mp->b_next)
3615                        count++;
3616
3617                mutex_exit(QLOCK(rdq));
3618                if (stp->sd_struiordq) {
3619                        infod_t infod;
3620
3621                        infod.d_cmd = INFOD_COUNT;
3622                        infod.d_count = 0;
3623                        if (count == 0) {
3624                                infod.d_cmd |= INFOD_FIRSTBYTES;
3625                                infod.d_bytes = 0;
3626                        }
3627                        infod.d_res = 0;
3628                        (void) infonext(rdq, &infod);
3629                        count += infod.d_count;
3630                        if (infod.d_res & INFOD_FIRSTBYTES)
3631                                size = infod.d_bytes;
3632                }
3633
3634                /*
3635                 * Drop down from size_t to the "int" required by the
3636                 * interface.  Cap at INT_MAX.
3637                 */
3638                retval = MIN(size, INT_MAX);
3639                error = strcopyout(&retval, (void *)arg, sizeof (retval),
3640                    copyflag);
3641                if (!error)
3642                        *rvalp = count;
3643                return (error);
3644        }
3645
3646        case FIONREAD:
3647                /*
3648                 * Return number of bytes of data in all data messages
3649                 * in queue in "arg".
3650                 */
3651        {
3652                size_t  size = 0;
3653                int     retval;
3654
3655                mutex_enter(QLOCK(rdq));
3656                for (mp = rdq->q_first; mp != NULL; mp = mp->b_next)
3657                        size += msgdsize(mp);
3658                mutex_exit(QLOCK(rdq));
3659
3660                if (stp->sd_struiordq) {
3661                        infod_t infod;
3662
3663                        infod.d_cmd = INFOD_BYTES;
3664                        infod.d_res = 0;
3665                        infod.d_bytes = 0;
3666                        (void) infonext(rdq, &infod);
3667                        size += infod.d_bytes;
3668                }
3669
3670                /*
3671                 * Drop down from size_t to the "int" required by the
3672                 * interface.  Cap at INT_MAX.
3673                 */
3674                retval = MIN(size, INT_MAX);
3675                error = strcopyout(&retval, (void *)arg, sizeof (retval),
3676                    copyflag);
3677
3678                *rvalp = 0;
3679                return (error);
```



```
3680        }
3681        case FIORDCHK:
3682                /*
3683                 * FIORDCHK does not use arg value (like FIONREAD),
3684                 * instead a count is returned. I_NREAD value may
3685                 * not be accurate but safe. The real thing to do is
3686                 * to add the msgdsizes of all data  messages until
3687                 * a non-data message.
3688                 */
3689        {
3690                size_t size = 0;
3691
3692                mutex_enter(QLOCK(rdq));
3693                for (mp = rdq->q_first; mp != NULL; mp = mp->b_next)
3694                        size += msgdsize(mp);
3695                mutex_exit(QLOCK(rdq));
3696
3697                if (stp->sd_struiordq) {
3698                        infod_t infod;
3699
3700                        infod.d_cmd = INFOD_BYTES;
3701                        infod.d_res = 0;
3702                        infod.d_bytes = 0;
3703                        (void) infonext(rdq, &infod);
3704                        size += infod.d_bytes;
3705                }
3706
3707                /*
3708                 * Since ioctl returns an int, and memory sizes under
3709                 * LP64 may not fit, we return INT_MAX if the count was
3710                 * actually greater.
3711                 */
3712                *rvalp = MIN(size, INT_MAX);
3713                return (0);
3714        }
3715
3716        case I_FIND:
3717                /*
3718                 * Get module name.
3719                 */
3720        {
3721                char mname[FMNAMESZ + 1];
3722                queue_t *q;
3723
3724                error = (copyflag & U_TO_K ? copyinstr : copystr)((void *)arg,
3725                    mname, FMNAMESZ + 1, NULL);
3726                if (error)
3727                        return ((error == ENAMETOOLONG) ? EINVAL : EFAULT);
3728
3729                /*
3730                 * Return EINVAL if we're handed a bogus module name.
3731                 */
3732                if (fmodsw_find(mname, FMODSW_LOAD) == NULL) {
3733                        TRACE_0(TR_FAC_STREAMS_FR,
3734                            TR_I_CANT_FIND, "couldn't I_FIND");
3735                        return (EINVAL);
3736                }
3737
3738                *rvalp = 0;
3739
3740                /* Look downstream to see if module is there. */
3741                claimstr(stp->sd_wrq);
3742                for (q = stp->sd_wrq->q_next; q; q = q->q_next) {
3743                        if (q->q_flag & QREADR) {
3744                                q = NULL;
3745                                break;
```

```
3746                        }
3747                        if (strcmp(mname, Q2NAME(q)) == 0)
3748                                break;
3749                }
3750                releasestr(stp->sd_wrq);

3752                *rvalp = (q ? 1 : 0);
3753                return (error);
3754        }

3756        case I_PUSH:
3757        case __I_PUSH_NOCTTY:
3758                /*
3759                 * Push a module.
3760                 * For the case __I_PUSH_NOCTTY push a module but
3761                 * do not allocate controlling tty. See bugid 4025044
3762                 */

3764        {
3765                char mname[FMNAMESZ + 1];
3766                fmodsw_impl_t *fp;
3767                dev_t dummydev;

3769                if (stp->sd_flag & STRHUP)
3770                        return (ENXIO);

3772                /*
3773                 * Get module name and look up in fmodsw.
3774                 */
3775                error = (copyflag & U_TO_K ? copyinstr : copystr)((void *)arg,
3776                    mname, FMNAMESZ + 1, NULL);
3777                if (error)
3778                        return ((error == ENAMETOOLONG) ? EINVAL : EFAULT);

3780                if ((fp = fmodsw_find(mname, FMODSW_HOLD | FMODSW_LOAD)) ==
3781                    NULL)
3782                        return (EINVAL);

3784                TRACE_2(TR_FAC_STREAMS_FR, TR_I_PUSH,
3785                    "I_PUSH:fp %p stp %p", fp, stp);

3787                if (error = strstartplumb(stp, flag, cmd)) {
3788                        fmodsw_rele(fp);
3789                        return (error);
3790                }

3792                /*
3793                 * See if any more modules can be pushed on this stream.
3794                 * Note that this check must be done after strstartplumb()
3795                 * since otherwise multiple threads issuing I_PUSHes on
3796                 * the same stream will be able to exceed nstrpush.
3797                 */
3798                mutex_enter(&stp->sd_lock);
3799                if (stp->sd_pushcnt >= nstrpush) {
3800                        fmodsw_rele(fp);
3801                        strendplumb(stp);
3802                        mutex_exit(&stp->sd_lock);
3803                        return (EINVAL);
3804                }
3805                mutex_exit(&stp->sd_lock);

3807                /*
3808                 * Push new module and call its open routine
3809                 * via qattach().  Modules don't change device
3810                 * numbers, so just ignore dummydev here.
3811                 */
```



```
3812                dummydev = vp->v_rdev;
3813                if ((error = qattach(rdq, &dummydev, 0, crp, fp,
3814                    B_FALSE)) == 0) {
3815                        if (vp->v_type == VCHR && /* sorry, no pipes allowed */
3816                            (cmd == I_PUSH) && (stp->sd_flag & STRISTTY)) {
3817                                /*
3818                                 * try to allocate it as a controlling terminal
3819                                 */
3820                                (void) strctty(stp);
3821                        }
3822                }

3824                mutex_enter(&stp->sd_lock);

3826                /*
3827                 * As a performance concern we are caching the values of
3828                 * q_minpsz and q_maxpsz of the module below the stream
3829                 * head in the stream head.
3830                 */
3831                mutex_enter(QLOCK(stp->sd_wrq->q_next));
3832                rmin = stp->sd_wrq->q_next->q_minpsz;
3833                rmax = stp->sd_wrq->q_next->q_maxpsz;
3834                mutex_exit(QLOCK(stp->sd_wrq->q_next));

3836                /* Do this processing here as a performance concern */
3837                if (strmsgsz != 0) {
3838                        if (rmax == INFPSZ)
3839                                rmax = strmsgsz;
3840                        else  {
3841                                if (vp->v_type == VFIFO)
3842                                        rmax = MIN(PIPE_BUF, rmax);
3843                                else    rmax = MIN(strmsgsz, rmax);
3844                        }
3845                }

3847                mutex_enter(QLOCK(wrq));
3848                stp->sd_qn_minpsz = rmin;
3849                stp->sd_qn_maxpsz = rmax;
3850                mutex_exit(QLOCK(wrq));

3852                strendplumb(stp);
3853                mutex_exit(&stp->sd_lock);
3854                return (error);
3855        }

3857        case I_POP:
3858        {
3859                queue_t *q;

3861                if (stp->sd_flag & STRHUP)
3862                        return (ENXIO);
3863                if (!wrq->q_next)       /* for broken pipes */
3864                        return (EINVAL);

3866                if (error = strstartplumb(stp, flag, cmd))
3867                        return (error);

3869                /*
3870                 * If there is an anchor on this stream and popping
3871                 * the current module would attempt to pop through the
3872                 * anchor, then disallow the pop unless we have sufficient
3873                 * privileges; take the cheapest (non-locking) check
3874                 * first.
3875                 */
3876                if (secpolicy_ip_config(crp, B_TRUE) != 0 ||
3877                    (stp->sd_anchorzone != crgetzoneid(crp))) {
```

```
3878                        mutex_enter(&stp->sd_lock);
3879                        /*
3880                         * Anchors only apply if there's at least one
3881                         * module on the stream (sd_pushcnt > 0).
3882                         */
3883                        if (stp->sd_pushcnt > 0 &&
3884                            stp->sd_pushcnt == stp->sd_anchor &&
3885                            stp->sd_vnode->v_type != VFIFO) {
3886                                strendplumb(stp);
3887                                mutex_exit(&stp->sd_lock);
3888                                if (stp->sd_anchorzone != crgetzoneid(crp))
3889                                        return (EINVAL);
3890                                /* Audit and report error */
3891                                return (secpolicy_ip_config(crp, B_FALSE));
3892                        }
3893                        mutex_exit(&stp->sd_lock);
3894                }

3896                q = wrq->q_next;
3897                TRACE_2(TR_FAC_STREAMS_FR, TR_I_POP,
3898                    "I_POP:%p from %p", q, stp);
3899                if (q->q_next == NULL || (q->q_flag & (QREADR|QISDRV))) {
3900                        error = EINVAL;
3901                } else {
3902                        qdetach(_RD(q), 1, flag, crp, B_FALSE);
3903                        error = 0;
3904                }
3905                mutex_enter(&stp->sd_lock);

3907                /*
3908                 * As a performance concern we are caching the values of
3909                 * q_minpsz and q_maxpsz of the module below the stream
3910                 * head in the stream head.
3911                 */
3912                mutex_enter(QLOCK(wrq->q_next));
3913                rmin = wrq->q_next->q_minpsz;
3914                rmax = wrq->q_next->q_maxpsz;
3915                mutex_exit(QLOCK(wrq->q_next));

3917                /* Do this processing here as a performance concern */
3918                if (strmsgsz != 0) {
3919                        if (rmax == INFPSZ)
3920                                rmax = strmsgsz;
3921                        else  {
3922                                if (vp->v_type == VFIFO)
3923                                        rmax = MIN(PIPE_BUF, rmax);
3924                                else    rmax = MIN(strmsgsz, rmax);
3925                        }
3926                }

3928                mutex_enter(QLOCK(wrq));
3929                stp->sd_qn_minpsz = rmin;
3930                stp->sd_qn_maxpsz = rmax;
3931                mutex_exit(QLOCK(wrq));

3933                /* If we popped through the anchor, then reset the anchor. */
3934                if (stp->sd_pushcnt < stp->sd_anchor) {
3935                        stp->sd_anchor = 0;
3936                        stp->sd_anchorzone = 0;
3937                }
3938                strendplumb(stp);
3939                mutex_exit(&stp->sd_lock);
3940                return (error);
3941        }

3943        case _I_MUXID2FD:
```



```
3944        {
3945                /*
3946                 * Create a fd for a I_PLINK'ed lower stream with a given
3947                 * muxid.  With the fd, application can send down ioctls,
3948                 * like I_LIST, to the previously I_PLINK'ed stream.  Note
3949                 * that after getting the fd, the application has to do an
3950                 * I_PUNLINK on the muxid before it can do any operation
3951                 * on the lower stream.  This is required by spec1170.
3952                 *
3953                 * The fd used to do this ioctl should point to the same
3954                 * controlling device used to do the I_PLINK.  If it uses
3955                 * a different stream or an invalid muxid, I_MUXID2FD will
3956                 * fail.  The error code is set to EINVAL.
3957                 *
3958                 * The intended use of this interface is the following.
3959                 * An application I_PLINK'ed a stream and exits.  The fd
3960                 * to the lower stream is gone.  Another application
3961                 * wants to get a fd to the lower stream, it uses I_MUXID2FD.
3962                 */
3963                int muxid = (int)arg;
3964                int fd;
3965                linkinfo_t *linkp;
3966                struct file *fp;
3967                netstack_t *ns;
3968                str_stack_t *ss;

3970                /*
3971                 * Do not allow the wildcard muxid.  This ioctl is not
3972                 * intended to find arbitrary link.
3973                 */
3974                if (muxid == 0) {
3975                        return (EINVAL);
3976                }

3978                ns = netstack_find_by_cred(crp);
3979                ASSERT(ns != NULL);
3980                ss = ns->netstack_str;
3981                ASSERT(ss != NULL);

3983                mutex_enter(&muxifier);
3984                linkp = findlinks(vp->v_stream, muxid, LINKPERSIST, ss);
3985                if (linkp == NULL) {
3986                        mutex_exit(&muxifier);
3987                        netstack_rele(ss->ss_netstack);
3988                        return (EINVAL);
3989                }

3991                if ((fd = ufalloc(0)) == -1) {
3992                        mutex_exit(&muxifier);
3993                        netstack_rele(ss->ss_netstack);
3994                        return (EMFILE);
3995                }
3996                fp = linkp->li_fpdown;
3997                mutex_enter(&fp->f_tlock);
3998                fp->f_count++;
3999                mutex_exit(&fp->f_tlock);
4000                mutex_exit(&muxifier);
4001                setf(fd, fp);
4002                *rvalp = fd;
4003                netstack_rele(ss->ss_netstack);
4004                return (0);
4005        }

4007        case _I_INSERT:
4008        {
4009                /*
```

```
4010                 * To insert a module to a given position in a stream.
4011                 * In the first release, only allow privileged user
4012                 * to use this ioctl. Furthermore, the insert is only allowed
4013                 * below an anchor if the zoneid is the same as the zoneid
4014                 * which created the anchor.
4015                 *
4016                 * Note that we do not plan to support this ioctl
4017                 * on pipes in the first release.  We want to learn more
4018                 * about the implications of these ioctls before extending
4019                 * their support.  And we do not think these features are
4020                 * valuable for pipes.
4021                 */
4022                STRUCT_DECL(strmodconf, strmodinsert);
4023                char mod_name[FMNAMESZ + 1];
4024                fmodsw_impl_t *fp;
4025                dev_t dummydev;
4026                queue_t *tmp_wrq;
4027                int pos;
4028                boolean_t is_insert;

4030                STRUCT_INIT(strmodinsert, flag);
4031                if (stp->sd_flag & STRHUP)
4032                        return (ENXIO);
4033                if (STRMATED(stp))
4034                        return (EINVAL);
4035                if ((error = secpolicy_net_config(crp, B_FALSE)) != 0)
4036                        return (error);
4037                if (stp->sd_anchor != 0 &&
4038                    stp->sd_anchorzone != crgetzoneid(crp))
4039                        return (EINVAL);

4041                error = strcopyin((void *)arg, STRUCT_BUF(strmodinsert),
4042                    STRUCT_SIZE(strmodinsert), copyflag);
4043                if (error)
4044                        return (error);

4046                /*
4047                 * Get module name and look up in fmodsw.
4048                 */
4049                error = (copyflag & U_TO_K ? copyinstr :
4050                    copystr)(STRUCT_FGETP(strmodinsert, mod_name),
4051                    mod_name, FMNAMESZ + 1, NULL);
4052                if (error)
4053                        return ((error == ENAMETOOLONG) ? EINVAL : EFAULT);

4055                if ((fp = fmodsw_find(mod_name, FMODSW_HOLD | FMODSW_LOAD)) ==
4056                    NULL)
4057                        return (EINVAL);

4059                if (error = strstartplumb(stp, flag, cmd)) {
4060                        fmodsw_rele(fp);
4061                        return (error);
4062                }

4064                /*
4065                 * Is this _I_INSERT just like an I_PUSH?  We need to know
4066                 * this because we do some optimizations if this is a
4067                 * module being pushed.
4068                 */
4069                pos = STRUCT_FGET(strmodinsert, pos);
4070                is_insert = (pos != 0);

4072                /*
4073                 * Make sure pos is valid.  Even though it is not an I_PUSH,
4074                 * we impose the same limit on the number of modules in a
4075                 * stream.
```



```
4076                 */
4077                mutex_enter(&stp->sd_lock);
4078                if (stp->sd_pushcnt >= nstrpush || pos < 0 ||
4079                    pos > stp->sd_pushcnt) {
4080                        fmodsw_rele(fp);
4081                        strendplumb(stp);
4082                        mutex_exit(&stp->sd_lock);
4083                        return (EINVAL);
4084                }
4085                if (stp->sd_anchor != 0) {
4086                        /*
4087                         * Is this insert below the anchor?
4088                         * Pushcnt hasn't been increased yet hence
4089                         * we test for greater than here, and greater or
4090                         * equal after qattach.
4091                         */
4092                        if (pos > (stp->sd_pushcnt - stp->sd_anchor) &&
4093                            stp->sd_anchorzone != crgetzoneid(crp)) {
4094                                fmodsw_rele(fp);
4095                                strendplumb(stp);
4096                                mutex_exit(&stp->sd_lock);
4097                                return (EPERM);
4098                        }
4099                }

4101                mutex_exit(&stp->sd_lock);

4103                /*
4104                 * First find the correct position this module to
4105                 * be inserted.  We don't need to call claimstr()
4106                 * as the stream should not be changing at this point.
4107                 *
4108                 * Insert new module and call its open routine
4109                 * via qattach().  Modules don't change device
4110                 * numbers, so just ignore dummydev here.
4111                 */
4112                for (tmp_wrq = stp->sd_wrq; pos > 0;
4113                    tmp_wrq = tmp_wrq->q_next, pos--) {
4114                        ASSERT(SAMESTR(tmp_wrq));
4115                }
4116                dummydev = vp->v_rdev;
4117                if ((error = qattach(_RD(tmp_wrq), &dummydev, 0, crp,
4118                    fp, is_insert)) != 0) {
4119                        mutex_enter(&stp->sd_lock);
4120                        strendplumb(stp);
4121                        mutex_exit(&stp->sd_lock);
4122                        return (error);
4123                }

4125                mutex_enter(&stp->sd_lock);

4127                /*
4128                 * As a performance concern we are caching the values of
4129                 * q_minpsz and q_maxpsz of the module below the stream
4130                 * head in the stream head.
4131                 */
4132                if (!is_insert) {
4133                        mutex_enter(QLOCK(stp->sd_wrq->q_next));
4134                        rmin = stp->sd_wrq->q_next->q_minpsz;
4135                        rmax = stp->sd_wrq->q_next->q_maxpsz;
4136                        mutex_exit(QLOCK(stp->sd_wrq->q_next));

4138                        /* Do this processing here as a performance concern */
4139                        if (strmsgsz != 0) {
4140                                if (rmax == INFPSZ) {
4141                                        rmax = strmsgsz;
```

```
4142                                } else  {
4143                                        rmax = MIN(strmsgsz, rmax);
4144                                }
4145                        }

4147                        mutex_enter(QLOCK(wrq));
4148                        stp->sd_qn_minpsz = rmin;
4149                        stp->sd_qn_maxpsz = rmax;
4150                        mutex_exit(QLOCK(wrq));
4151                }

4153                /*
4154                 * Need to update the anchor value if this module is
4155                 * inserted below the anchor point.
4156                 */
4157                if (stp->sd_anchor != 0) {
4158                        pos = STRUCT_FGET(strmodinsert, pos);
4159                        if (pos >= (stp->sd_pushcnt - stp->sd_anchor))
4160                                stp->sd_anchor++;
4161                }

4163                strendplumb(stp);
4164                mutex_exit(&stp->sd_lock);
4165                return (0);
4166        }

4168        case _I_REMOVE:
4169        {
4170                /*
4171                 * To remove a module with a given name in a stream.  The
4172                 * caller of this ioctl needs to provide both the name and
4173                 * the position of the module to be removed.  This eliminates
4174                 * the ambiguity of removal if a module is inserted/pushed
4175                 * multiple times in a stream.  In the first release, only
4176                 * allow privileged user to use this ioctl.
4177                 * Furthermore, the remove is only allowed
4178                 * below an anchor if the zoneid is the same as the zoneid
4179                 * which created the anchor.
4180                 *
4181                 * Note that we do not plan to support this ioctl
4182                 * on pipes in the first release.  We want to learn more
4183                 * about the implications of these ioctls before extending
4184                 * their support.  And we do not think these features are
4185                 * valuable for pipes.
4186                 *
4187                 * Also note that _I_REMOVE cannot be used to remove a
4188                 * driver or the stream head.
4189                 */
4190                STRUCT_DECL(strmodconf, strmodremove);
4191                queue_t *q;
4192                int pos;
4193                char mod_name[FMNAMESZ + 1];
4194                boolean_t is_remove;

4196                STRUCT_INIT(strmodremove, flag);
4197                if (stp->sd_flag & STRHUP)
4198                        return (ENXIO);
4199                if (STRMATED(stp))
4200                        return (EINVAL);
4201                if ((error = secpolicy_net_config(crp, B_FALSE)) != 0)
4202                        return (error);
4203                if (stp->sd_anchor != 0 &&
4204                    stp->sd_anchorzone != crgetzoneid(crp))
4205                        return (EINVAL);

4207                error = strcopyin((void *)arg, STRUCT_BUF(strmodremove),
```



```
4208                    STRUCT_SIZE(strmodremove), copyflag);
4209                if (error)
4210                        return (error);

4212                error = (copyflag & U_TO_K ? copyinstr :
4213                    copystr)(STRUCT_FGETP(strmodremove, mod_name),
4214                    mod_name, FMNAMESZ + 1, NULL);
4215                if (error)
4216                        return ((error == ENAMETOOLONG) ? EINVAL : EFAULT);

4218                if ((error = strstartplumb(stp, flag, cmd)) != 0)
4219                        return (error);

4221                /*
4222                 * Match the name of given module to the name of module at
4223                 * the given position.
4224                 */
4225                pos = STRUCT_FGET(strmodremove, pos);

4227                is_remove = (pos != 0);
4228                for (q = stp->sd_wrq->q_next; SAMESTR(q) && pos > 0;
4229                    q = q->q_next, pos--)
4230                        ;
4231                if (pos > 0 || !SAMESTR(q) ||
4232                    strcmp(Q2NAME(q), mod_name) != 0) {
4233                        mutex_enter(&stp->sd_lock);
4234                        strendplumb(stp);
4235                        mutex_exit(&stp->sd_lock);
4236                        return (EINVAL);
4237                }

4239                /*
4240                 * If the position is at or below an anchor, then the zoneid
4241                 * must match the zoneid that created the anchor.
4242                 */
4243                if (stp->sd_anchor != 0) {
4244                        pos = STRUCT_FGET(strmodremove, pos);
4245                        if (pos >= (stp->sd_pushcnt - stp->sd_anchor) &&
4246                            stp->sd_anchorzone != crgetzoneid(crp)) {
4247                                mutex_enter(&stp->sd_lock);
4248                                strendplumb(stp);
4249                                mutex_exit(&stp->sd_lock);
4250                                return (EPERM);
4251                        }
4252                }


4255                ASSERT(!(q->q_flag & QREADR));
4256                qdetach(_RD(q), 1, flag, crp, is_remove);

4258                mutex_enter(&stp->sd_lock);

4260                /*
4261                 * As a performance concern we are caching the values of
4262                 * q_minpsz and q_maxpsz of the module below the stream
4263                 * head in the stream head.
4264                 */
4265                if (!is_remove) {
4266                        mutex_enter(QLOCK(wrq->q_next));
4267                        rmin = wrq->q_next->q_minpsz;
4268                        rmax = wrq->q_next->q_maxpsz;
4269                        mutex_exit(QLOCK(wrq->q_next));

4271                        /* Do this processing here as a performance concern */
4272                        if (strmsgsz != 0) {
4273                                if (rmax == INFPSZ)
```

```
4274                                    rmax = strmsgsz;
4275                            else  {
4276                                    if (vp->v_type == VFIFO)
4277                                            rmax = MIN(PIPE_BUF, rmax);
4278                                    else    rmax = MIN(strmsgsz, rmax);
4279                            }
4280                    }

4282                    mutex_enter(QLOCK(wrq));
4283                    stp->sd_qn_minpsz = rmin;
4284                    stp->sd_qn_maxpsz = rmax;
4285                    mutex_exit(QLOCK(wrq));
4286            }

4288            /*
4289             * Need to update the anchor value if this module is removed
4290             * at or below the anchor point.  If the removed module is at
4291             * the anchor point, remove the anchor for this stream if
4292             * there is no module above the anchor point.  Otherwise, if
4293             * the removed module is below the anchor point, decrement the
4294             * anchor point by 1.
4295             */
4296            if (stp->sd_anchor != 0) {
4297                    pos = STRUCT_FGET(strmodremove, pos);
4298                    if (pos == stp->sd_pushcnt - stp->sd_anchor + 1)
4299                            stp->sd_anchor = 0;
4300                    else if (pos > (stp->sd_pushcnt - stp->sd_anchor + 1))
4301                            stp->sd_anchor--;
4302            }

4304            strendplumb(stp);
4305            mutex_exit(&stp->sd_lock);
4306            return (0);
4307    }

4309    case I_ANCHOR:
4310            /*
4311             * Set the anchor position on the stream to reside at
4312             * the top module (in other words, the top module
4313             * cannot be popped).  Anchors with a FIFO make no
4314             * obvious sense, so they're not allowed.
4315             */
4316            mutex_enter(&stp->sd_lock);

4318            if (stp->sd_vnode->v_type == VFIFO) {
4319                    mutex_exit(&stp->sd_lock);
4320                    return (EINVAL);
4321            }
4322            /* Only allow the same zoneid to update the anchor */
4323            if (stp->sd_anchor != 0 &&
4324                stp->sd_anchorzone != crgetzoneid(crp)) {
4325                    mutex_exit(&stp->sd_lock);
4326                    return (EINVAL);
4327            }
4328            stp->sd_anchor = stp->sd_pushcnt;
4329            stp->sd_anchorzone = crgetzoneid(crp);
4330            mutex_exit(&stp->sd_lock);
4331            return (0);

4333    case I_LOOK:
4334            /*
4335             * Get name of first module downstream.
4336             * If no module, return an error.
4337             */
4338            claimstr(wrq);
4339            if (_SAMESTR(wrq) && wrq->q_next->q_next != NULL) {
```



```
4340                    char *name = Q2NAME(wrq->q_next);

4342                    error = strcopyout(name, (void *)arg, strlen(name) + 1,
4343                        copyflag);
4344                    releasestr(wrq);
4345                    return (error);
4346            }
4347            releasestr(wrq);
4348            return (EINVAL);

4350    case I_LINK:
4351    case I_PLINK:
4352            /*
4353             * Link a multiplexor.
4354             */
4355            return (mlink(vp, cmd, (int)arg, crp, rvalp, 0));

4357    case _I_PLINK_LH:
4358            /*
4359             * Link a multiplexor: Call must originate from kernel.
4360             */
4361            if (kioctl)
4362                    return (ldi_mlink_lh(vp, cmd, arg, crp, rvalp));

4364            return (EINVAL);
4365    case I_UNLINK:
4366    case I_PUNLINK:
4367            /*
4368             * Unlink a multiplexor.
4369             * If arg is -1, unlink all links for which this is the
4370             * controlling stream.  Otherwise, arg is an index number
4371             * for a link to be removed.
4372             */
4373    {
4374            struct linkinfo *linkp;
4375            int native_arg = (int)arg;
4376            int type;
4377            netstack_t *ns;
4378            str_stack_t *ss;

4380            TRACE_1(TR_FAC_STREAMS_FR,
4381                TR_I_UNLINK, "I_UNLINK/I_PUNLINK:%p", stp);
4382            if (vp->v_type == VFIFO) {
4383                    return (EINVAL);
4384            }
4385            if (cmd == I_UNLINK)
4386                    type = LINKNORMAL;
4387            else    /* I_PUNLINK */
4388                    type = LINKPERSIST;
4389            if (native_arg == 0) {
4390                    return (EINVAL);
4391            }
4392            ns = netstack_find_by_cred(crp);
4393            ASSERT(ns != NULL);
4394            ss = ns->netstack_str;
4395            ASSERT(ss != NULL);

4397            if (native_arg == MUXID_ALL)
4398                    error = munlinkall(stp, type, crp, rvalp, ss);
4399            else {
4400                    mutex_enter(&muxifier);
4401                    if (!(linkp = findlinks(stp, (int)arg, type, ss))) {
4402                            /* invalid user supplied index number */
4403                            mutex_exit(&muxifier);
4404                            netstack_rele(ss->ss_netstack);
4405                            return (EINVAL);
```

```
4406                        }
4407                        /* munlink drops the muxifier lock */
4408                        error = munlink(stp, linkp, type, crp, rvalp, ss);
4409                }
4410                netstack_rele(ss->ss_netstack);
4411                return (error);
4412        }

4414        case I_FLUSH:
4415                /*
4416                 * send a flush message downstream
4417                 * flush message can indicate
4418                 * FLUSHR - flush read queue
4419                 * FLUSHW - flush write queue
4420                 * FLUSHRW - flush read/write queue
4421                 */
4422                if (stp->sd_flag & STRHUP)
4423                        return (ENXIO);
4424                if (arg & ~FLUSHRW)
4425                        return (EINVAL);

4427                for (;;) {
4428                        if (putnextctl1(stp->sd_wrq, M_FLUSH, (int)arg)) {
4429                                break;
4430                        }
4431                        if (error = strwaitbuf(1, BPRI_HI)) {
4432                                return (error);
4433                        }
4434                }

4436                /*
4437                 * Send down an unsupported ioctl and wait for the nack
4438                 * in order to allow the M_FLUSH to propagate back
4439                 * up to the stream head.
4440                 * Replaces if (qready()) runqueues();
4441                 */
4442                strioc.ic_cmd = -1;     /* The unsupported ioctl */
4443                strioc.ic_timout = 0;
4444                strioc.ic_len = 0;
4445                strioc.ic_dp = NULL;
4446                (void) strdoioctl(stp, &strioc, flag, K_TO_K, crp, rvalp);
4447                *rvalp = 0;
4448                return (0);

4450        case I_FLUSHBAND:
4451        {
4452                struct bandinfo binfo;

4454                error = strcopyin((void *)arg, &binfo, sizeof (binfo),
4455                    copyflag);
4456                if (error)
4457                        return (error);
4458                if (stp->sd_flag & STRHUP)
4459                        return (ENXIO);
4460                if (binfo.bi_flag & ~FLUSHRW)
4461                        return (EINVAL);
4462                while (!(mp = allocb(2, BPRI_HI))) {
4463                        if (error = strwaitbuf(2, BPRI_HI))
4464                                return (error);
4465                }
4466                mp->b_datap->db_type = M_FLUSH;
4467                *mp->b_wptr++ = binfo.bi_flag | FLUSHBAND;
4468                *mp->b_wptr++ = binfo.bi_pri;
4469                putnext(stp->sd_wrq, mp);
4470                /*
4471                 * Send down an unsupported ioctl and wait for the nack
```



```
4472                 * in order to allow the M_FLUSH to propagate back
4473                 * up to the stream head.
4474                 * Replaces if (qready()) runqueues();
4475                 */
4476                strioc.ic_cmd = -1;     /* The unsupported ioctl */
4477                strioc.ic_timout = 0;
4478                strioc.ic_len = 0;
4479                strioc.ic_dp = NULL;
4480                (void) strdoioctl(stp, &strioc, flag, K_TO_K, crp, rvalp);
4481                *rvalp = 0;
4482                return (0);
4483        }

4485        case I_SRDOPT:
4486                /*
4487                 * Set read options
4488                 *
4489                 * RNORM - default stream mode
4490                 * RMSGN - message no discard
4491                 * RMSGD - message discard
4492                 * RPROTNORM - fail read with EBADMSG for M_[PC]PROTOs
4493                 * RPROTDAT - convert M_[PC]PROTOs to M_DATAs
4494                 * RPROTDIS - discard M_[PC]PROTOs and retain M_DATAs
4495                 */
4496                if (arg & ~(RMODEMASK | RPROTMASK))
4497                        return (EINVAL);

4499                if ((arg & (RMSGD|RMSGN)) == (RMSGD|RMSGN))
4500                        return (EINVAL);

4502                mutex_enter(&stp->sd_lock);
4503                switch (arg & RMODEMASK) {
4504                case RNORM:
4505                        stp->sd_read_opt &= ~(RD_MSGDIS | RD_MSGNODIS);
4506                        break;
4507                case RMSGD:
4508                        stp->sd_read_opt = (stp->sd_read_opt & ~RD_MSGNODIS) |
4509                            RD_MSGDIS;
4510                        break;
4511                case RMSGN:
4512                        stp->sd_read_opt = (stp->sd_read_opt & ~RD_MSGDIS) |
4513                            RD_MSGNODIS;
4514                        break;
4515                }

4517                switch (arg & RPROTMASK) {
4518                case RPROTNORM:
4519                        stp->sd_read_opt &= ~(RD_PROTDAT | RD_PROTDIS);
4520                        break;

4522                case RPROTDAT:
4523                        stp->sd_read_opt = ((stp->sd_read_opt & ~RD_PROTDIS) |
4524                            RD_PROTDAT);
4525                        break;

4527                case RPROTDIS:
4528                        stp->sd_read_opt = ((stp->sd_read_opt & ~RD_PROTDAT) |
4529                            RD_PROTDIS);
4530                        break;
4531                }
4532                mutex_exit(&stp->sd_lock);
4533                return (0);

4535        case I_GRDOPT:
4536                /*
4537                 * Get read option and return the value
```

```
4538                 * to spot pointed to by arg
4539                 */
4540        {
4541                int rdopt;

4543                rdopt = ((stp->sd_read_opt & RD_MSGDIS) ? RMSGD :
4544                    ((stp->sd_read_opt & RD_MSGNODIS) ? RMSGN : RNORM));
4545                rdopt |= ((stp->sd_read_opt & RD_PROTDAT) ? RPROTDAT :
4546                    ((stp->sd_read_opt & RD_PROTDIS) ? RPROTDIS : RPROTNORM));

4548                return (strcopyout(&rdopt, (void *)arg, sizeof (int),
4549                    copyflag));
4550        }

4552        case I_SERROPT:
4553                /*
4554                 * Set error options
4555                 *
4556                 * RERRNORM - persistent read errors
4557                 * RERRNONPERSIST - non-persistent read errors
4558                 * WERRNORM - persistent write errors
4559                 * WERRNONPERSIST - non-persistent write errors
4560                 */
4561                if (arg & ~(RERRMASK | WERRMASK))
4562                        return (EINVAL);

4564                mutex_enter(&stp->sd_lock);
4565                switch (arg & RERRMASK) {
4566                case RERRNORM:
4567                        stp->sd_flag &= ~STRDERRNONPERSIST;
4568                        break;
4569                case RERRNONPERSIST:
4570                        stp->sd_flag |= STRDERRNONPERSIST;
4571                        break;
4572                }
4573                switch (arg & WERRMASK) {
4574                case WERRNORM:
4575                        stp->sd_flag &= ~STWRERRNONPERSIST;
4576                        break;
4577                case WERRNONPERSIST:
4578                        stp->sd_flag |= STWRERRNONPERSIST;
4579                        break;
4580                }
4581                mutex_exit(&stp->sd_lock);
4582                return (0);

4584        case I_GERROPT:
4585                /*
4586                 * Get error option and return the value
4587                 * to spot pointed to by arg
4588                 */
4589        {
4590                int erropt = 0;

4592                erropt |= (stp->sd_flag & STRDERRNONPERSIST) ? RERRNONPERSIST :
4593                    RERRNORM;
4594                erropt |= (stp->sd_flag & STWRERRNONPERSIST) ? WERRNONPERSIST :
4595                    WERRNORM;
4596                return (strcopyout(&erropt, (void *)arg, sizeof (int),
4597                    copyflag));
4598        }

4600        case I_SETSIG:
4601                /*
4602                 * Register the calling proc to receive the SIGPOLL
4603                 * signal based on the events given in arg.  If
```



```
4604                 * arg is zero, remove the proc from register list.
4605                 */
4606        {
4607                strsig_t *ssp, *pssp;
4608                struct pid *pidp;

4610                pssp = NULL;
4611                pidp = curproc->p_pidp;
4612                /*
4613                 * Hold sd_lock to prevent traversal of sd_siglist while
4614                 * it is modified.
4615                 */
4616                mutex_enter(&stp->sd_lock);
4617                for (ssp = stp->sd_siglist; ssp && (ssp->ss_pidp != pidp);
4618                    pssp = ssp, ssp = ssp->ss_next)
4619                        ;

4621                if (arg) {
4622                        if (arg & ~(S_INPUT|S_HIPRI|S_MSG|S_HANGUP|S_ERROR|
4623                            S_RDNORM|S_WRNORM|S_RDBAND|S_WRBAND|S_BANDURG)) {
4624                                mutex_exit(&stp->sd_lock);
4625                                return (EINVAL);
4626                        }
4627                        if ((arg & S_BANDURG) && !(arg & S_RDBAND)) {
4628                                mutex_exit(&stp->sd_lock);
4629                                return (EINVAL);
4630                        }

4632                        /*
4633                         * If proc not already registered, add it
4634                         * to list.
4635                         */
4636                        if (!ssp) {
4637                                ssp = kmem_alloc(sizeof (strsig_t), KM_SLEEP);
4638                                ssp->ss_pidp = pidp;
4639                                ssp->ss_pid = pidp->pid_id;
4640                                ssp->ss_next = NULL;
4641                                if (pssp)
4642                                        pssp->ss_next = ssp;
4643                                else
4644                                        stp->sd_siglist = ssp;
4645                                mutex_enter(&pidlock);
4646                                PID_HOLD(pidp);
4647                                mutex_exit(&pidlock);
4648                        }

4650                        /*
4651                         * Set events.
4652                         */
4653                        ssp->ss_events = (int)arg;
4654                } else {
4655                        /*
4656                         * Remove proc from register list.
4657                         */
4658                        if (ssp) {
4659                                mutex_enter(&pidlock);
4660                                PID_RELE(pidp);
4661                                mutex_exit(&pidlock);
4662                                if (pssp)
4663                                        pssp->ss_next = ssp->ss_next;
4664                                else
4665                                        stp->sd_siglist = ssp->ss_next;
4666                                kmem_free(ssp, sizeof (strsig_t));
4667                        } else {
4668                                mutex_exit(&stp->sd_lock);
4669                                return (EINVAL);
```

```
4670                        }
4671                }

4673                /*
4674                 * Recalculate OR of sig events.
4675                 */
4676                stp->sd_sigflags = 0;
4677                for (ssp = stp->sd_siglist; ssp; ssp = ssp->ss_next)
4678                        stp->sd_sigflags |= ssp->ss_events;
4679                mutex_exit(&stp->sd_lock);
4680                return (0);
4681        }

4683        case I_GETSIG:
4684                /*
4685                 * Return (in arg) the current registration of events
4686                 * for which the calling proc is to be signaled.
4687                 */
4688        {
4689                struct strsig *ssp;
4690                struct pid  *pidp;

4692                pidp = curproc->p_pidp;
4693                mutex_enter(&stp->sd_lock);
4694                for (ssp = stp->sd_siglist; ssp; ssp = ssp->ss_next)
4695                        if (ssp->ss_pidp == pidp) {
4696                                error = strcopyout(&ssp->ss_events, (void *)arg,
4697                                    sizeof (int), copyflag);
4698                                mutex_exit(&stp->sd_lock);
4699                                return (error);
4700                        }
4701                mutex_exit(&stp->sd_lock);
4702                return (EINVAL);
4703        }

4705        case I_ESETSIG:
4706                /*
4707                 * Register the ss_pid to receive the SIGPOLL
4708                 * signal based on the events is ss_events arg.  If
4709                 * ss_events is zero, remove the proc from register list.
4710                 */
4711        {
4712                struct strsig *ssp, *pssp;
4713                struct proc *proc;
4714                struct pid  *pidp;
4715                pid_t pid;
4716                struct strsigset ss;

4718                error = strcopyin((void *)arg, &ss, sizeof (ss), copyflag);
4719                if (error)
4720                        return (error);

4722                pid = ss.ss_pid;

4724                if (ss.ss_events != 0) {
4725                        /*
4726                         * Permissions check by sending signal 0.
4727                         * Note that when kill fails it does a set_errno
4728                         * causing the system call to fail.
4729                         */
4730                        error = kill(pid, 0);
4731                        if (error) {
4732                                return (error);
4733                        }
4734                }
4735                mutex_enter(&pidlock);
```



```
4736                if (pid == 0)
4737                        proc = curproc;
4738                else if (pid < 0)
4739                        proc = pgfind(-pid);
4740                else
4741                        proc = prfind(pid);
4742                if (proc == NULL) {
4743                        mutex_exit(&pidlock);
4744                        return (ESRCH);
4745                }
4746                if (pid < 0)
4747                        pidp = proc->p_pgidp;
4748                else
4749                        pidp = proc->p_pidp;
4750                ASSERT(pidp);
4751                /*
4752                 * Get a hold on the pid structure while referencing it.
4753                 * There is a separate PID_HOLD should it be inserted
4754                 * in the list below.
4755                 */
4756                PID_HOLD(pidp);
4757                mutex_exit(&pidlock);

4759                pssp = NULL;
4760                /*
4761                 * Hold sd_lock to prevent traversal of sd_siglist while
4762                 * it is modified.
4763                 */
4764                mutex_enter(&stp->sd_lock);
4765                for (ssp = stp->sd_siglist; ssp && (ssp->ss_pid != pid);
4766                    pssp = ssp, ssp = ssp->ss_next)
4767                        ;

4769                if (ss.ss_events) {
4770                        if (ss.ss_events &
4771                            ~(S_INPUT|S_HIPRI|S_MSG|S_HANGUP|S_ERROR|
4772                            S_RDNORM|S_WRNORM|S_RDBAND|S_WRBAND|S_BANDURG)) {
4773                                mutex_exit(&stp->sd_lock);
4774                                mutex_enter(&pidlock);
4775                                PID_RELE(pidp);
4776                                mutex_exit(&pidlock);
4777                                return (EINVAL);
4778                        }
4779                        if ((ss.ss_events & S_BANDURG) &&
4780                            !(ss.ss_events & S_RDBAND)) {
4781                                mutex_exit(&stp->sd_lock);
4782                                mutex_enter(&pidlock);
4783                                PID_RELE(pidp);
4784                                mutex_exit(&pidlock);
4785                                return (EINVAL);
4786                        }

4788                        /*
4789                         * If proc not already registered, add it
4790                         * to list.
4791                         */
4792                        if (!ssp) {
4793                                ssp = kmem_alloc(sizeof (strsig_t), KM_SLEEP);
4794                                ssp->ss_pidp = pidp;
4795                                ssp->ss_pid = pid;
4796                                ssp->ss_next = NULL;
4797                                if (pssp)
4798                                        pssp->ss_next = ssp;
4799                                else
4800                                        stp->sd_siglist = ssp;
4801                                mutex_enter(&pidlock);
```

```
4802                                PID_HOLD(pidp);
4803                                mutex_exit(&pidlock);
4804                        }

4806                        /*
4807                         * Set events.
4808                         */
4809                        ssp->ss_events = ss.ss_events;
4810                } else {
4811                        /*
4812                         * Remove proc from register list.
4813                         */
4814                        if (ssp) {
4815                                mutex_enter(&pidlock);
4816                                PID_RELE(pidp);
4817                                mutex_exit(&pidlock);
4818                                if (pssp)
4819                                        pssp->ss_next = ssp->ss_next;
4820                                else
4821                                        stp->sd_siglist = ssp->ss_next;
4822                                kmem_free(ssp, sizeof (strsig_t));
4823                        } else {
4824                                mutex_exit(&stp->sd_lock);
4825                                mutex_enter(&pidlock);
4826                                PID_RELE(pidp);
4827                                mutex_exit(&pidlock);
4828                                return (EINVAL);
4829                        }
4830                }

4832                /*
4833                 * Recalculate OR of sig events.
4834                 */
4835                stp->sd_sigflags = 0;
4836                for (ssp = stp->sd_siglist; ssp; ssp = ssp->ss_next)
4837                        stp->sd_sigflags |= ssp->ss_events;
4838                mutex_exit(&stp->sd_lock);
4839                mutex_enter(&pidlock);
4840                PID_RELE(pidp);
4841                mutex_exit(&pidlock);
4842                return (0);
4843        }

4845        case I_EGETSIG:
4846                /*
4847                 * Return (in arg) the current registration of events
4848                 * for which the calling proc is to be signaled.
4849                 */
4850        {
4851                struct strsig *ssp;
4852                struct proc *proc;
4853                pid_t pid;
4854                struct pid  *pidp;
4855                struct strsigset ss;

4857                error = strcopyin((void *)arg, &ss, sizeof (ss), copyflag);
4858                if (error)
4859                        return (error);

4861                pid = ss.ss_pid;
4862                mutex_enter(&pidlock);
4863                if (pid == 0)
4864                        proc = curproc;
4865                else if (pid < 0)
4866                        proc = pgfind(-pid);
4867                else
```



```
4868                        proc = prfind(pid);
4869                if (proc == NULL) {
4870                        mutex_exit(&pidlock);
4871                        return (ESRCH);
4872                }
4873                if (pid < 0)
4874                        pidp = proc->p_pgidp;
4875                else
4876                        pidp = proc->p_pidp;

4878                /* Prevent the pidp from being reassigned */
4879                PID_HOLD(pidp);
4880                mutex_exit(&pidlock);

4882                mutex_enter(&stp->sd_lock);
4883                for (ssp = stp->sd_siglist; ssp; ssp = ssp->ss_next)
4884                        if (ssp->ss_pid == pid) {
4885                                ss.ss_pid = ssp->ss_pid;
4886                                ss.ss_events = ssp->ss_events;
4887                                error = strcopyout(&ss, (void *)arg,
4888                                    sizeof (struct strsigset), copyflag);
4889                                mutex_exit(&stp->sd_lock);
4890                                mutex_enter(&pidlock);
4891                                PID_RELE(pidp);
4892                                mutex_exit(&pidlock);
4893                                return (error);
4894                        }
4895                mutex_exit(&stp->sd_lock);
4896                mutex_enter(&pidlock);
4897                PID_RELE(pidp);
4898                mutex_exit(&pidlock);
4899                return (EINVAL);
4900        }

4902        case I_PEEK:
4903        {
4904                STRUCT_DECL(strpeek, strpeek);
4905                size_t n;
4906                mblk_t *fmp, *tmp_mp = NULL;

4908                STRUCT_INIT(strpeek, flag);

4910                error = strcopyin((void *)arg, STRUCT_BUF(strpeek),
4911                    STRUCT_SIZE(strpeek), copyflag);
4912                if (error)
4913                        return (error);

4915                mutex_enter(QLOCK(rdq));
4916                /*
4917                 * Skip the invalid messages
4918                 */
4919                for (mp = rdq->q_first; mp != NULL; mp = mp->b_next)
4920                        if (mp->b_datap->db_type != M_SIG)
4921                                break;

4923                /*
4924                 * If user has requested to peek at a high priority message
4925                 * and first message is not, return 0
4926                 */
4927                if (mp != NULL) {
4928                        if ((STRUCT_FGET(strpeek, flags) & RS_HIPRI) &&
4929                            queclass(mp) == QNORM) {
4930                                *rvalp = 0;
4931                                mutex_exit(QLOCK(rdq));
4932                                return (0);
4933                        }
```

```
4934                } else if (stp->sd_struiordq == NULL ||
4935                    (STRUCT_FGET(strpeek, flags) & RS_HIPRI)) {
4936                        /*
4937                         * No mblks to look at at the streamhead and
4938                         * 1). This isn't a synch stream or
4939                         * 2). This is a synch stream but caller wants high
4940                         *      priority messages which is not supported by
4941                         *      the synch stream. (it only supports QNORM)
4942                         */
4943                        *rvalp = 0;
4944                        mutex_exit(QLOCK(rdq));
4945                        return (0);
4946                }

4948                fmp = mp;

4950                if (mp && mp->b_datap->db_type == M_PASSFP) {
4951                        mutex_exit(QLOCK(rdq));
4952                        return (EBADMSG);
4953                }

4955                ASSERT(mp == NULL || mp->b_datap->db_type == M_PCPROTO ||
4956                    mp->b_datap->db_type == M_PROTO ||
4957                    mp->b_datap->db_type == M_DATA);

4959                if (mp && mp->b_datap->db_type == M_PCPROTO) {
4960                        STRUCT_FSET(strpeek, flags, RS_HIPRI);
4961                } else {
4962                        STRUCT_FSET(strpeek, flags, 0);
4963                }


4966                if (mp && ((tmp_mp = dupmsg(mp)) == NULL)) {
4967                        mutex_exit(QLOCK(rdq));
4968                        return (ENOSR);
4969                }
4970                mutex_exit(QLOCK(rdq));

4972                /*
4973                 * set mp = tmp_mp, so that I_PEEK processing can continue.
4974                 * tmp_mp is used to free the dup'd message.
4975                 */
4976                mp = tmp_mp;

4978                uio.uio_fmode = 0;
4979                uio.uio_extflg = UIO_COPY_CACHED;
4980                uio.uio_segflg = (copyflag == U_TO_K) ? UIO_USERSPACE :
4981                    UIO_SYSSPACE;
4982                uio.uio_limit = 0;
4983                /*
4984                 * First process PROTO blocks, if any.
4985                 * If user doesn't want to get ctl info by setting maxlen <= 0,
4986                 * then set len to -1/0 and skip control blocks part.
4987                 */
4988                if (STRUCT_FGET(strpeek, ctlbuf.maxlen) < 0)
4989                        STRUCT_FSET(strpeek, ctlbuf.len, -1);
4990                else if (STRUCT_FGET(strpeek, ctlbuf.maxlen) == 0)
4991                        STRUCT_FSET(strpeek, ctlbuf.len, 0);
4992                else {
4993                        int     ctl_part = 0;

4995                        iov.iov_base = STRUCT_FGETP(strpeek, ctlbuf.buf);
4996                        iov.iov_len = STRUCT_FGET(strpeek, ctlbuf.maxlen);
4997                        uio.uio_iov = &iov;
4998                        uio.uio_resid = iov.iov_len;
4999                        uio.uio_loffset = 0;
```



```
5000                        uio.uio_iovcnt = 1;
5001                        while (mp && mp->b_datap->db_type != M_DATA &&
5002                            uio.uio_resid >= 0) {
5003                                ASSERT(STRUCT_FGET(strpeek, flags) == 0 ?
5004                                    mp->b_datap->db_type == M_PROTO :
5005                                    mp->b_datap->db_type == M_PCPROTO);

5007                                if ((n = MIN(uio.uio_resid,
5008                                    mp->b_wptr - mp->b_rptr)) != 0 &&
5009                                    (error = uiomove((char *)mp->b_rptr, n,
5010                                    UIO_READ, &uio)) != 0) {
5011                                        freemsg(tmp_mp);
5012                                        return (error);
5013                                }
5014                                ctl_part = 1;
5015                                mp = mp->b_cont;
5016                        }
5017                        /* No ctl message */
5018                        if (ctl_part == 0)
5019                                STRUCT_FSET(strpeek, ctlbuf.len, -1);
5020                        else
5021                                STRUCT_FSET(strpeek, ctlbuf.len,
5022                                    STRUCT_FGET(strpeek, ctlbuf.maxlen) -
5023                                    uio.uio_resid);
5024                }

5026                /*
5027                 * Now process DATA blocks, if any.
5028                 * If user doesn't want to get data info by setting maxlen <= 0,
5029                 * then set len to -1/0 and skip data blocks part.
5030                 */
5031                if (STRUCT_FGET(strpeek, databuf.maxlen) < 0)
5032                        STRUCT_FSET(strpeek, databuf.len, -1);
5033                else if (STRUCT_FGET(strpeek, databuf.maxlen) == 0)
5034                        STRUCT_FSET(strpeek, databuf.len, 0);
5035                else {
5036                        int     data_part = 0;

5038                        iov.iov_base = STRUCT_FGETP(strpeek, databuf.buf);
5039                        iov.iov_len = STRUCT_FGET(strpeek, databuf.maxlen);
5040                        uio.uio_iov = &iov;
5041                        uio.uio_resid = iov.iov_len;
5042                        uio.uio_loffset = 0;
5043                        uio.uio_iovcnt = 1;
5044                        while (mp && uio.uio_resid) {
5045                                if (mp->b_datap->db_type == M_DATA) {
5046                                        if ((n = MIN(uio.uio_resid,
5047                                            mp->b_wptr - mp->b_rptr)) != 0 &&
5048                                            (error = uiomove((char *)mp->b_rptr,
5049                                            n, UIO_READ, &uio)) != 0) {
5050                                                freemsg(tmp_mp);
5051                                                return (error);
5052                                        }
5053                                        data_part = 1;
5054                                }
5055                                ASSERT(data_part == 0 ||
5056                                    mp->b_datap->db_type == M_DATA);
5057                                mp = mp->b_cont;
5058                        }
5059                        /* No data message */
5060                        if (data_part == 0)
5061                                STRUCT_FSET(strpeek, databuf.len, -1);
5062                        else
5063                                STRUCT_FSET(strpeek, databuf.len,
5064                                    STRUCT_FGET(strpeek, databuf.maxlen) -
5065                                    uio.uio_resid);
```

```
5066                }
5067                freemsg(tmp_mp);

5069                /*
5070                 * It is a synch stream and user wants to get
5071                 * data (maxlen > 0).
5072                 * uio setup is done by the codes that process DATA
5073                 * blocks above.
5074                 */
5075                if ((fmp == NULL) && STRUCT_FGET(strpeek, databuf.maxlen) > 0) {
5076                        infod_t infod;

5078                        infod.d_cmd = INFOD_COPYOUT;
5079                        infod.d_res = 0;
5080                        infod.d_uiop = &uio;
5081                        error = infonext(rdq, &infod);
5082                        if (error == EINVAL || error == EBUSY)
5083                                error = 0;
5084                        if (error)
5085                                return (error);
5086                        STRUCT_FSET(strpeek, databuf.len, STRUCT_FGET(strpeek,
5087                            databuf.maxlen) - uio.uio_resid);
5088                        if (STRUCT_FGET(strpeek, databuf.len) == 0) {
5089                                /*
5090                                 * No data found by the infonext().
5091                                 */
5092                                STRUCT_FSET(strpeek, databuf.len, -1);
5093                        }
5094                }
5095                error = strcopyout(STRUCT_BUF(strpeek), (void *)arg,
5096                    STRUCT_SIZE(strpeek), copyflag);
5097                if (error) {
5098                        return (error);
5099                }
5100                /*
5101                 * If there is no message retrieved, set return code to 0
5102                 * otherwise, set it to 1.
5103                 */
5104                if (STRUCT_FGET(strpeek, ctlbuf.len) == -1 &&
5105                    STRUCT_FGET(strpeek, databuf.len) == -1)
5106                        *rvalp = 0;
5107                else
5108                        *rvalp = 1;
5109                return (0);
5110        }

5112        case I_FDINSERT:
5113        {
5114                STRUCT_DECL(strfdinsert, strfdinsert);
5115                struct file *resftp;
5116                struct stdata *resstp;
5117                t_uscalar_t     ival;
5118                ssize_t msgsize;
5119                struct strbuf mctl;

5121                STRUCT_INIT(strfdinsert, flag);
5122                if (stp->sd_flag & STRHUP)
5123                        return (ENXIO);
5124                /*
5125                 * STRDERR, STWRERR and STPLEX tested above.
5126                 */
5127                error = strcopyin((void *)arg, STRUCT_BUF(strfdinsert),
5128                    STRUCT_SIZE(strfdinsert), copyflag);
5129                if (error)
5130                        return (error);
```



```
5132                if (STRUCT_FGET(strfdinsert, offset) < 0 ||
5133                    (STRUCT_FGET(strfdinsert, offset) %
5134                    sizeof (t_uscalar_t)) != 0)
5135                        return (EINVAL);
5136                if ((resftp = getf(STRUCT_FGET(strfdinsert, fildes))) != NULL) {
5137                        if ((resstp = resftp->f_vnode->v_stream) == NULL) {
5138                                releasef(STRUCT_FGET(strfdinsert, fildes));
5139                                return (EINVAL);
5140                        }
5141                } else
5142                        return (EINVAL);

5144                mutex_enter(&resstp->sd_lock);
5145                if (resstp->sd_flag & (STRDERR|STWRERR|STRHUP|STPLEX)) {
5146                        error = strgeterr(resstp,
5147                            STRDERR|STWRERR|STRHUP|STPLEX, 0);
5148                        if (error != 0) {
5149                                mutex_exit(&resstp->sd_lock);
5150                                releasef(STRUCT_FGET(strfdinsert, fildes));
5151                                return (error);
5152                        }
5153                }
5154                mutex_exit(&resstp->sd_lock);

5156 #ifdef  _ILP32
5157                {
5158                        queue_t *q;
5159                        queue_t *mate = NULL;

5161                        /* get read queue of stream terminus */
5162                        claimstr(resstp->sd_wrq);
5163                        for (q = resstp->sd_wrq->q_next; q->q_next != NULL;
5164                            q = q->q_next)
5165                                if (!STRMATED(resstp) && STREAM(q) != resstp &&
5166                                    mate == NULL) {
5167                                        ASSERT(q->q_qinfo->qi_srvp);
5168                                        ASSERT(_OTHERQ(q)->q_qinfo->qi_srvp);
5169                                        claimstr(q);
5170                                        mate = q;
5171                                }
5172                        q = _RD(q);
5173                        if (mate)
5174                                releasestr(mate);
5175                        releasestr(resstp->sd_wrq);
5176                        ival = (t_uscalar_t)q;
5177                }
5178 #else
5179                ival = (t_uscalar_t)getminor(resftp->f_vnode->v_rdev);
5180 #endif  /* _ILP32 */

5182                if (STRUCT_FGET(strfdinsert, ctlbuf.len) <
5183                    STRUCT_FGET(strfdinsert, offset) + sizeof (t_uscalar_t)) {
5184                        releasef(STRUCT_FGET(strfdinsert, fildes));
5185                        return (EINVAL);
5186                }

5188                /*
5189                 * Check for legal flag value.
5190                 */
5191                if (STRUCT_FGET(strfdinsert, flags) & ~RS_HIPRI) {
5192                        releasef(STRUCT_FGET(strfdinsert, fildes));
5193                        return (EINVAL);
5194                }

5196                /* get these values from those cached in the stream head */
5197                mutex_enter(QLOCK(stp->sd_wrq));
```

```
5198                rmin = stp->sd_qn_minpsz;
5199                rmax = stp->sd_qn_maxpsz;
5200                mutex_exit(QLOCK(stp->sd_wrq));

5202                /*
5203                 * Make sure ctl and data sizes together fall within
5204                 * the limits of the max and min receive packet sizes
5205                 * and do not exceed system limit.  A negative data
5206                 * length means that no data part is to be sent.
5207                 */
5208                ASSERT((rmax >= 0) || (rmax == INFPSZ));
5209                if (rmax == 0) {
5210                        releasef(STRUCT_FGET(strfdinsert, fildes));
5211                        return (ERANGE);
5212                }
5213                if ((msgsize = STRUCT_FGET(strfdinsert, databuf.len)) < 0)
5214                        msgsize = 0;
5215                if ((msgsize < rmin) ||
5216                    ((msgsize > rmax) && (rmax != INFPSZ)) ||
5217                    (STRUCT_FGET(strfdinsert, ctlbuf.len) > strctlsz)) {
5218                        releasef(STRUCT_FGET(strfdinsert, fildes));
5219                        return (ERANGE);
5220                }

5222                mutex_enter(&stp->sd_lock);
5223                while (!(STRUCT_FGET(strfdinsert, flags) & RS_HIPRI) &&
5224                    !canputnext(stp->sd_wrq)) {
5225                        if ((error = strwaitq(stp, WRITEWAIT, (ssize_t)0,
5226                            flag, -1, &done)) != 0 || done) {
5227                                mutex_exit(&stp->sd_lock);
5228                                releasef(STRUCT_FGET(strfdinsert, fildes));
5229                                return (error);
5230                        }
5231                        if ((error = i_straccess(stp, access)) != 0) {
5232                                mutex_exit(&stp->sd_lock);
5233                                releasef(
5234                                    STRUCT_FGET(strfdinsert, fildes));
5235                                return (error);
5236                        }
5237                }
5238                mutex_exit(&stp->sd_lock);

5240                /*
5241                 * Copy strfdinsert.ctlbuf into native form of
5242                 * ctlbuf to pass down into strmakemsg().
5243                 */
5244                mctl.maxlen = STRUCT_FGET(strfdinsert, ctlbuf.maxlen);
5245                mctl.len = STRUCT_FGET(strfdinsert, ctlbuf.len);
5246                mctl.buf = STRUCT_FGETP(strfdinsert, ctlbuf.buf);

5248                iov.iov_base = STRUCT_FGETP(strfdinsert, databuf.buf);
5249                iov.iov_len = STRUCT_FGET(strfdinsert, databuf.len);
5250                uio.uio_iov = &iov;
5251                uio.uio_iovcnt = 1;
5252                uio.uio_loffset = 0;
5253                uio.uio_segflg = (copyflag == U_TO_K) ? UIO_USERSPACE :
5254                    UIO_SYSSPACE;
5255                uio.uio_fmode = 0;
5256                uio.uio_extflg = UIO_COPY_CACHED;
5257                uio.uio_resid = iov.iov_len;
5258                if ((error = strmakemsg(&mctl,
5259                    &msgsize, &uio, stp,
5260                    STRUCT_FGET(strfdinsert, flags), &mp)) != 0 || !mp) {
5261                        STRUCT_FSET(strfdinsert, databuf.len, msgsize);
5262                        releasef(STRUCT_FGET(strfdinsert, fildes));
5263                        return (error);
```



```
5264                }

5266                STRUCT_FSET(strfdinsert, databuf.len, msgsize);

5268                /*
5269                 * Place the possibly reencoded queue pointer 'offset' bytes
5270                 * from the start of the control portion of the message.
5271                 */
5272                *((t_uscalar_t *)(mp->b_rptr +
5273                    STRUCT_FGET(strfdinsert, offset))) = ival;

5275                /*
5276                 * Put message downstream.
5277                 */
5278                stream_willservice(stp);
5279                putnext(stp->sd_wrq, mp);
5280                stream_runservice(stp);
5281                releasef(STRUCT_FGET(strfdinsert, fildes));
5282                return (error);
5283        }

5285        case I_SENDFD:
5286        {
5287                struct file *fp;

5289                if ((fp = getf((int)arg)) == NULL)
5290                        return (EBADF);
5291                error = do_sendfp(stp, fp, crp);
5292                if (auditing) {
5293                        audit_fdsend((int)arg, fp, error);
5294                }
5295                releasef((int)arg);
5296                return (error);
5297        }

5299        case I_RECVFD:
5300        case I_E_RECVFD:
5301        {
5302                struct k_strrecvfd *srf;
5303                int i, fd;

5305                mutex_enter(&stp->sd_lock);
5306                while (!(mp = getq(rdq))) {
5307                        if (stp->sd_flag & (STRHUP|STREOF)) {
5308                                mutex_exit(&stp->sd_lock);
5309                                return (ENXIO);
5310                        }
5311                        if ((error = strwaitq(stp, GETWAIT, (ssize_t)0,
5312                            flag, -1, &done)) != 0 || done) {
5313                                mutex_exit(&stp->sd_lock);
5314                                return (error);
5315                        }
5316                        if ((error = i_straccess(stp, access)) != 0) {
5317                                mutex_exit(&stp->sd_lock);
5318                                return (error);
5319                        }
5320                }
5321                if (mp->b_datap->db_type != M_PASSFP) {
5322                        putback(stp, rdq, mp, mp->b_band);
5323                        mutex_exit(&stp->sd_lock);
5324                        return (EBADMSG);
5325                }
5326                mutex_exit(&stp->sd_lock);

5328                srf = (struct k_strrecvfd *)mp->b_rptr;
5329                if ((fd = ufalloc(0)) == -1) {
```

```
5330                        mutex_enter(&stp->sd_lock);
5331                        putback(stp, rdq, mp, mp->b_band);
5332                        mutex_exit(&stp->sd_lock);
5333                        return (EMFILE);
5334                }
5335                if (cmd == I_RECVFD) {
5336                        struct o_strrecvfd      ostrfd;

5338                        /* check to see if uid/gid values are too large. */

5340                        if (srf->uid > (o_uid_t)USHRT_MAX ||
5341                            srf->gid > (o_gid_t)USHRT_MAX) {
5342                                mutex_enter(&stp->sd_lock);
5343                                putback(stp, rdq, mp, mp->b_band);
5344                                mutex_exit(&stp->sd_lock);
5345                                setf(fd, NULL); /* release fd entry */
5346                                return (EOVERFLOW);
5347                        }

5349                        ostrfd.fd = fd;
5350                        ostrfd.uid = (o_uid_t)srf->uid;
5351                        ostrfd.gid = (o_gid_t)srf->gid;

5353                        /* Null the filler bits */
5354                        for (i = 0; i < 8; i++)
5355                                ostrfd.fill[i] = 0;

5357                        error = strcopyout(&ostrfd, (void *)arg,
5358                            sizeof (struct o_strrecvfd), copyflag);
5359                } else {                /* I_E_RECVFD */
5360                        struct strrecvfd        strfd;

5362                        strfd.fd = fd;
5363                        strfd.uid = srf->uid;
5364                        strfd.gid = srf->gid;

5366                        /* null the filler bits */
5367                        for (i = 0; i < 8; i++)
5368                                strfd.fill[i] = 0;

5370                        error = strcopyout(&strfd, (void *)arg,
5371                            sizeof (struct strrecvfd), copyflag);
5372                }

5374                if (error) {
5375                        setf(fd, NULL); /* release fd entry */
5376                        mutex_enter(&stp->sd_lock);
5377                        putback(stp, rdq, mp, mp->b_band);
5378                        mutex_exit(&stp->sd_lock);
5379                        return (error);
5380                }
5381                if (auditing) {
5382                        audit_fdrecv(fd, srf->fp);
5383                }

5385                /*
5386                 * Always increment f_count since the freemsg() below will
5387                 * always call free_passfp() which performs a closef().
5388                 */
5389                mutex_enter(&srf->fp->f_tlock);
5390                srf->fp->f_count++;
5391                mutex_exit(&srf->fp->f_tlock);
5392                setf(fd, srf->fp);
5393                freemsg(mp);
5394                return (0);
5395        }
```



```
5397        case I_SWROPT:
5398                /*
5399                 * Set/clear the write options. arg is a bit
5400                 * mask with any of the following bits set...
5401                 *      SNDZERO - send zero length message
5402                 *      SNDPIPE - send sigpipe to process if
5403                 *              sd_werror is set and process is
5404                 *              doing a write or putmsg.
5405                 * The new stream head write options should reflect
5406                 * what is in arg.
5407                 */
5408                if (arg & ~(SNDZERO|SNDPIPE))
5409                        return (EINVAL);

5411                mutex_enter(&stp->sd_lock);
5412                stp->sd_wput_opt &= ~(SW_SIGPIPE|SW_SNDZERO);
5413                if (arg & SNDZERO)
5414                        stp->sd_wput_opt |= SW_SNDZERO;
5415                if (arg & SNDPIPE)
5416                        stp->sd_wput_opt |= SW_SIGPIPE;
5417                mutex_exit(&stp->sd_lock);
5418                return (0);

5420        case I_GWROPT:
5421        {
5422                int wropt = 0;

5424                if (stp->sd_wput_opt & SW_SNDZERO)
5425                        wropt |= SNDZERO;
5426                if (stp->sd_wput_opt & SW_SIGPIPE)
5427                        wropt |= SNDPIPE;
5428                return (strcopyout(&wropt, (void *)arg, sizeof (wropt),
5429                    copyflag));
5430        }

5432        case I_LIST:
5433                /*
5434                 * Returns all the modules found on this stream,
5435                 * upto the driver. If argument is NULL, return the
5436                 * number of modules (including driver). If argument
5437                 * is not NULL, copy the names into the structure
5438                 * provided.
5439                 */

5441        {
5442                queue_t *q;
5443                char *qname;
5444                int i, nmods;
5445                struct str_mlist *mlist;
5446                STRUCT_DECL(str_list, strlist);

5448                if (arg == NULL) { /* Return number of modules plus driver */
5449                        if (stp->sd_vnode->v_type == VFIFO)
5450                                *rvalp = stp->sd_pushcnt;
5451                        else
5452                                *rvalp = stp->sd_pushcnt + 1;
5453                        return (0);
5454                }

5456                STRUCT_INIT(strlist, flag);

5458                error = strcopyin((void *)arg, STRUCT_BUF(strlist),
5459                    STRUCT_SIZE(strlist), copyflag);
5460                if (error != 0)
5461                        return (error);
```

```
5463                mlist = STRUCT_FGETP(strlist, sl_modlist);
5464                nmods = STRUCT_FGET(strlist, sl_nmods);
5465                if (nmods <= 0)
5466                        return (EINVAL);

5468                claimstr(stp->sd_wrq);
5469                q = stp->sd_wrq;
5470                for (i = 0; i < nmods && _SAMESTR(q); i++, q = q->q_next) {
5471                        qname = Q2NAME(q->q_next);
5472                        error = strcopyout(qname, &mlist[i], strlen(qname) + 1,
5473                            copyflag);
5474                        if (error != 0) {
5475                                releasestr(stp->sd_wrq);
5476                                return (error);
5477                        }
5478                }
5479                releasestr(stp->sd_wrq);
5480                return (strcopyout(&i, (void *)arg, sizeof (int), copyflag));
5481        }

5483        case I_CKBAND:
5484        {
5485                queue_t *q;
5486                qband_t *qbp;

5488                if ((arg < 0) || (arg >= NBAND))
5489                        return (EINVAL);
5490                q = _RD(stp->sd_wrq);
5491                mutex_enter(QLOCK(q));
5492                if (arg > (int)q->q_nband) {
5493                        *rvalp = 0;
5494                } else {
5495                        if (arg == 0) {
5496                                if (q->q_first)
5497                                        *rvalp = 1;
5498                                else
5499                                        *rvalp = 0;
5500                        } else {
5501                                qbp = q->q_bandp;
5502                                while (--arg > 0)
5503                                        qbp = qbp->qb_next;
5504                                if (qbp->qb_first)
5505                                        *rvalp = 1;
5506                                else
5507                                        *rvalp = 0;
5508                        }
5509                }
5510                mutex_exit(QLOCK(q));
5511                return (0);
5512        }

5514        case I_GETBAND:
5515        {
5516                int intpri;
5517                queue_t *q;

5519                q = _RD(stp->sd_wrq);
5520                mutex_enter(QLOCK(q));
5521                mp = q->q_first;
5522                if (!mp) {
5523                        mutex_exit(QLOCK(q));
5524                        return (ENODATA);
5525                }
5526                intpri = (int)mp->b_band;
5527                error = strcopyout(&intpri, (void *)arg, sizeof (int),
```



```
5528                    copyflag);
5529                mutex_exit(QLOCK(q));
5530                return (error);
5531        }

5533        case I_ATMARK:
5534        {
5535                queue_t *q;

5537                if (arg & ~(ANYMARK|LASTMARK))
5538                        return (EINVAL);
5539                q = _RD(stp->sd_wrq);
5540                mutex_enter(&stp->sd_lock);
5541                if ((stp->sd_flag & STRATMARK) && (arg == ANYMARK)) {
5542                        *rvalp = 1;
5543                } else {
5544                        mutex_enter(QLOCK(q));
5545                        mp = q->q_first;

5547                        if (mp == NULL)
5548                                *rvalp = 0;
5549                        else if ((arg == ANYMARK) && (mp->b_flag & MSGMARK))
5550                                *rvalp = 1;
5551                        else if ((arg == LASTMARK) && (mp == stp->sd_mark))
5552                                *rvalp = 1;
5553                        else
5554                                *rvalp = 0;
5555                        mutex_exit(QLOCK(q));
5556                }
5557                mutex_exit(&stp->sd_lock);
5558                return (0);
5559        }

5561        case I_CANPUT:
5562        {
5563                char band;

5565                if ((arg < 0) || (arg >= NBAND))
5566                        return (EINVAL);
5567                band = (char)arg;
5568                *rvalp = bcanputnext(stp->sd_wrq, band);
5569                return (0);
5570        }

5572        case I_SETCLTIME:
5573        {
5574                int closetime;

5576                error = strcopyin((void *)arg, &closetime, sizeof (int),
5577                    copyflag);
5578                if (error)
5579                        return (error);
5580                if (closetime < 0)
5581                        return (EINVAL);

5583                stp->sd_closetime = closetime;
5584                return (0);
5585        }

5587        case I_GETCLTIME:
5588        {
5589                int closetime;

5591                closetime = stp->sd_closetime;
5592                return (strcopyout(&closetime, (void *)arg, sizeof (int),
5593                    copyflag));
```

```
5594         }

5596         case TIOCGSID:
5597         {
5598                 pid_t sid;

5600                 mutex_enter(&stp->sd_lock);
5601                 if (stp->sd_sidp == NULL) {
5602                         mutex_exit(&stp->sd_lock);
5603                         return (ENOTTY);
5604                 }
5605                 sid = stp->sd_sidp->pid_id;
5606                 mutex_exit(&stp->sd_lock);
5607                 return (strcopyout(&sid, (void *)arg, sizeof (pid_t),
5608                     copyflag));
5609         }

5611         case TIOCSPGRP:
5612         {
5613                 pid_t pgrp;
5614                 proc_t *q;
5615                 pid_t   sid, fg_pgid, bg_pgid;

5617                 if (error = strcopyin((void *)arg, &pgrp, sizeof (pid_t),
5618                     copyflag))
5619                         return (error);
5620                 mutex_enter(&stp->sd_lock);
5621                 mutex_enter(&pidlock);
5622                 if (stp->sd_sidp != ttoproc(curthread)->p_sessp->s_sidp) {
5623                         mutex_exit(&pidlock);
5624                         mutex_exit(&stp->sd_lock);
5625                         return (ENOTTY);
5626                 }
5627                 if (pgrp == stp->sd_pgidp->pid_id) {
5628                         mutex_exit(&pidlock);
5629                         mutex_exit(&stp->sd_lock);
5630                         return (0);
5631                 }
5632                 if (pgrp <= 0 || pgrp >= maxpid) {
5633                         mutex_exit(&pidlock);
5634                         mutex_exit(&stp->sd_lock);
5635                         return (EINVAL);
5636                 }
5637                 if ((q = pgfind(pgrp)) == NULL ||
5638                     q->p_sessp != ttoproc(curthread)->p_sessp) {
5639                         mutex_exit(&pidlock);
5640                         mutex_exit(&stp->sd_lock);
5641                         return (EPERM);
5642                 }
5643                 sid = stp->sd_sidp->pid_id;
5644                 fg_pgid = q->p_pgrp;
5645                 bg_pgid = stp->sd_pgidp->pid_id;
5646                 CL_SET_PROCESS_GROUP(curthread, sid, bg_pgid, fg_pgid);
5647                 PID_RELE(stp->sd_pgidp);
5648                 ctty_clear_sighuped();
5649                 stp->sd_pgidp = q->p_pgidp;
5650                 PID_HOLD(stp->sd_pgidp);
5651                 mutex_exit(&pidlock);
5652                 mutex_exit(&stp->sd_lock);
5653                 return (0);
5654         }

5656         case TIOCGPGRP:
5657         {
5658                 pid_t pgrp;
```



```
5660                 mutex_enter(&stp->sd_lock);
5661                 if (stp->sd_sidp == NULL) {
5662                         mutex_exit(&stp->sd_lock);
5663                         return (ENOTTY);
5664                 }
5665                 pgrp = stp->sd_pgidp->pid_id;
5666                 mutex_exit(&stp->sd_lock);
5667                 return (strcopyout(&pgrp, (void *)arg, sizeof (pid_t),
5668                     copyflag));
5669         }

5671         case TIOCSCTTY:
5672         {
5673                 return (strctty(stp));
5674         }

5676         case TIOCNOTTY:
5677         {
5678                 /* freectty() always assumes curproc. */
5679                 if (freectty(B_FALSE) != 0)
5680                         return (0);
5681                 return (ENOTTY);
5682         }

5684         case FIONBIO:
5685         case FIOASYNC:
5686                 return (0);     /* handled by the upper layer */
5687         }
5688 }

5690 /*
5691  * Custom free routine used for M_PASSFP messages.
5692  */
5693 static void
5694 free_passfp(struct k_strrecvfd *srf)
5695 {
5696         (void) closef(srf->fp);
5697         kmem_free(srf, sizeof (struct k_strrecvfd) + sizeof (frtn_t));
5698 }

5700 /* ARGSUSED */
5701 int
5702 do_sendfp(struct stdata *stp, struct file *fp, struct cred *cr)
5703 {
5704         queue_t *qp, *nextqp;
5705         struct k_strrecvfd *srf;
5706         mblk_t *mp;
5707         frtn_t *frtnp;
5708         size_t bufsize;
5709         queue_t *mate = NULL;
5710         syncq_t *sq = NULL;
5711         int retval = 0;

5713         if (stp->sd_flag & STRHUP)
5714                 return (ENXIO);

5716         claimstr(stp->sd_wrq);

5718         /* Fastpath, we have a pipe, and we are already mated, use it. */
5719         if (STRMATED(stp)) {
5720                 qp = _RD(stp->sd_mate->sd_wrq);
5721                 claimstr(qp);
5722                 mate = qp;
5723         } else { /* Not already mated. */

5725                 /*
```

```
5726                 * Walk the stream to the end of this one.
5727                 * assumes that the claimstr() will prevent
5728                 * plumbing between the stream head and the
5729                 * driver from changing
5730                 */
5731                qp = stp->sd_wrq;

5733                /*
5734                 * Loop until we reach the end of this stream.
5735                 * On completion, qp points to the write queue
5736                 * at the end of the stream, or the read queue
5737                 * at the stream head if this is a fifo.
5738                 */
5739                while (((qp = qp->q_next) != NULL) && _SAMESTR(qp))
5740                        ;

5742                /*
5743                 * Just in case we get a q_next which is NULL, but
5744                 * not at the end of the stream.  This is actually
5745                 * broken, so we set an assert to catch it in
5746                 * debug, and set an error and return if not debug.
5747                 */
5748                ASSERT(qp);
5749                if (qp == NULL) {
5750                        releasestr(stp->sd_wrq);
5751                        return (EINVAL);
5752                }

5754                /*
5755                 * Enter the syncq for the driver, so (hopefully)
5756                 * the queue values will not change on us.
5757                 * XXXX - This will only prevent the race IFF only
5758                 *   the write side modifies the q_next member, and
5759                 *   the put procedure is protected by at least
5760                 *   MT_PERQ.
5761                 */
5762                if ((sq = qp->q_syncq) != NULL)
5763                        entersq(sq, SQ_PUT);

5765                /* Now get the q_next value from this qp. */
5766                nextqp = qp->q_next;

5768                /*
5769                 * If nextqp exists and the other stream is different
5770                 * from this one claim the stream, set the mate, and
5771                 * get the read queue at the stream head of the other
5772                 * stream.  Assumes that nextqp was at least valid when
5773                 * we got it.  Hopefully the entersq of the driver
5774                 * will prevent it from changing on us.
5775                 */
5776                if ((nextqp != NULL) && (STREAM(nextqp) != stp)) {
5777                        ASSERT(qp->q_qinfo->qi_srvp);
5778                        ASSERT(_OTHERQ(qp)->q_qinfo->qi_srvp);
5779                        ASSERT(_OTHERQ(qp->q_next)->q_qinfo->qi_srvp);
5780                        claimstr(nextqp);

5782                        /* Make sure we still have a q_next */
5783                        if (nextqp != qp->q_next) {
5784                                releasestr(stp->sd_wrq);
5785                                releasestr(nextqp);
5786                                return (EINVAL);
5787                        }

5789                        qp = _RD(STREAM(nextqp)->sd_wrq);
5790                        mate = qp;
5791                }
```



```
5792                /* If we entered the synq above, leave it. */
5793                if (sq != NULL)
5794                        leavesq(sq, SQ_PUT);
5795        } /*  STRMATED(STP)  */

5797        /* XXX prevents substitution of the ops vector */
5798        if (qp->q_qinfo != &strdata && qp->q_qinfo != &fifo_strdata) {
5799                retval = EINVAL;
5800                goto out;
5801        }

5803        if (qp->q_flag & QFULL) {
5804                retval = EAGAIN;
5805                goto out;
5806        }

5808        /*
5809         * Since M_PASSFP messages include a file descriptor, we use
5810         * esballoc() and specify a custom free routine (free_passfp()) that
5811         * will close the descriptor as part of freeing the message.  For
5812         * convenience, we stash the frtn_t right after the data block.
5813         */
5814        bufsize = sizeof (struct k_strrecvfd) + sizeof (frtn_t);
5815        srf = kmem_alloc(bufsize, KM_NOSLEEP);
5816        if (srf == NULL) {
5817                retval = EAGAIN;
5818                goto out;
5819        }

5821        frtnp = (frtn_t *)(srf + 1);
5822        frtnp->free_arg = (caddr_t)srf;
5823        frtnp->free_func = free_passfp;

5825        mp = esballoc((uchar_t *)srf, bufsize, BPRI_MED, frtnp);
5826        if (mp == NULL) {
5827                kmem_free(srf, bufsize);
5828                retval = EAGAIN;
5829                goto out;
5830        }
5831        mp->b_wptr += sizeof (struct k_strrecvfd);
5832        mp->b_datap->db_type = M_PASSFP;

5834        srf->fp = fp;
5835        srf->uid = crgetuid(curthread->t_cred);
5836        srf->gid = crgetgid(curthread->t_cred);
5837        mutex_enter(&fp->f_tlock);
5838        fp->f_count++;
5839        mutex_exit(&fp->f_tlock);

5841        put(qp, mp);
5842 out:
5843        releasestr(stp->sd_wrq);
5844        if (mate)
5845                releasestr(mate);
5846        return (retval);
5847 }

5849 /*
5850  * Send an ioctl message downstream and wait for acknowledgement.
5851  * flags may be set to either U_TO_K or K_TO_K and a combination
5852  * of STR_NOERROR or STR_NOSIG
5853  * STR_NOSIG: Signals are essentially ignored or held and have
5854  *      no effect for the duration of the call.
5855  * STR_NOERROR: Ignores stream head read, write and hup errors.
5856  *      Additionally, if an existing ioctl times out, it is assumed
5857  *      lost and and this ioctl will continue as if the previous ioctl had
```

```
 *      finished.  ETIME may be returned if this ioctl times out (i.e.
 *      ic_timout is not INFTIM).  Non-stream head errors may be returned if
 *      the ioc_error indicates that the driver/module had problems,
 *      an EFAULT was found when accessing user data, a lack of
 *      resources, etc.
 */
int
strdoioctl(
        struct stdata *stp,
        struct strioctl *strioc,
        int fflags,             /* file flags with model info */
        int flag,
        cred_t *crp,
        int *rvalp)
{
        mblk_t *bp;
        struct iocblk *iocbp;
        struct copyreq *reqp;
        struct copyresp *resp;
        int id;
        int transparent = 0;
        int error = 0;
        int len = 0;
        caddr_t taddr;
        int copyflag = (flag & (U_TO_K | K_TO_K));
        int sigflag = (flag & STR_NOSIG);
        int errs;
        uint_t waitflags;
        boolean_t set_iocwaitne = B_FALSE;

        ASSERT(copyflag == U_TO_K || copyflag == K_TO_K);
        ASSERT((fflags & FMODELS) != 0);

        TRACE_2(TR_FAC_STREAMS_FR,
            TR_STRDOIOCTL,
            "strdoioctl:stp %p strioc %p", stp, strioc);
        if (strioc->ic_len == TRANSPARENT) {    /* send arg in M_DATA block */
                transparent = 1;
                strioc->ic_len = sizeof (intptr_t);
        }

        if (strioc->ic_len < 0 || (strmsgsz > 0 && strioc->ic_len > strmsgsz))
                return (EINVAL);

        if ((bp = allocb_cred_wait(sizeof (union ioctypes), sigflag, &error,
            crp, curproc->p_pid)) == NULL)
                        return (error);

        bzero(bp->b_wptr, sizeof (union ioctypes));

        iocbp = (struct iocblk *)bp->b_wptr;
        iocbp->ioc_count = strioc->ic_len;
        iocbp->ioc_cmd = strioc->ic_cmd;
        iocbp->ioc_flag = (fflags & FMODELS);

        crhold(crp);
        iocbp->ioc_cr = crp;
        DB_TYPE(bp) = M_IOCTL;
        bp->b_wptr += sizeof (struct iocblk);

        if (flag & STR_NOERROR)
                errs = STPLEX;
        else
                errs = STRHUP|STRDERR|STWRERR|STPLEX;

        /*
```



```
         * If there is data to copy into ioctl block, do so.
         */
        if (iocbp->ioc_count > 0) {
                if (transparent)
                        /*
                         * Note: STR_NOERROR does not have an effect
                         * in putiocd()
                         */
                        id = K_TO_K | sigflag;
                else
                        id = flag;
                if ((error = putiocd(bp, strioc->ic_dp, id, crp)) != 0) {
                        freemsg(bp);
                        crfree(crp);
                        return (error);
                }

                /*
                 * We could have slept copying in user pages.
                 * Recheck the stream head state (the other end
                 * of a pipe could have gone away).
                 */
                if (stp->sd_flag & errs) {
                        mutex_enter(&stp->sd_lock);
                        error = strgeterr(stp, errs, 0);
                        mutex_exit(&stp->sd_lock);
                        if (error != 0) {
                                freemsg(bp);
                                crfree(crp);
                                return (error);
                        }
                }
        }
        if (transparent)
                iocbp->ioc_count = TRANSPARENT;

        /*
         * Block for up to STRTIMOUT milliseconds if there is an outstanding
         * ioctl for this stream already running.  All processes
         * sleeping here will be awakened as a result of an ACK
         * or NAK being received for the outstanding ioctl, or
         * as a result of the timer expiring on the outstanding
         * ioctl (a failure), or as a result of any waiting
         * process's timer expiring (also a failure).
         */

        error = 0;
        mutex_enter(&stp->sd_lock);
        while ((stp->sd_flag & IOCWAIT) ||
            (!set_iocwaitne && (stp->sd_flag & IOCWAITNE))) {
                clock_t cv_rval;

                TRACE_0(TR_FAC_STREAMS_FR,
                    TR_STRDOIOCTL_WAIT,
                    "strdoioctl sleeps - IOCWAIT");
                cv_rval = str_cv_wait(&stp->sd_iocmonitor, &stp->sd_lock,
                    STRTIMOUT, sigflag);
                if (cv_rval <= 0) {
                        if (cv_rval == 0) {
                                error = EINTR;
                        } else {
                                if (flag & STR_NOERROR) {
                                        /*
                                         * Terminating current ioctl in
                                         * progress -- assume it got lost and
                                         * wake up the other thread so that the
```

```
5990                                     * operation completes.
5991                                     */
5992                                    if (!(stp->sd_flag & IOCWAITNE)) {
5993                                            set_iocwaitne = B_TRUE;
5994                                            stp->sd_flag |= IOCWAITNE;
5995                                            cv_broadcast(&stp->sd_monitor);
5996                                    }
5997                                    /*
5998                                     * Otherwise, there's a running
5999                                     * STR_NOERROR -- we have no choice
6000                                     * here but to wait forever (or until
6001                                     * interrupted).
6002                                     */
6003                            } else {
6004                                    /*
6005                                     * pending ioctl has caused
6006                                     * us to time out
6007                                     */
6008                                    error = ETIME;
6009                            }
6010                    }
6011            } else if ((stp->sd_flag & errs)) {
6012                    error = strgeterr(stp, errs, 0);
6013            }
6014            if (error) {
6015                    mutex_exit(&stp->sd_lock);
6016                    freemsg(bp);
6017                    crfree(crp);
6018                    return (error);
6019            }
6020    }

6022    /*
6023     * Have control of ioctl mechanism.
6024     * Send down ioctl packet and wait for response.
6025     */
6026    if (stp->sd_iocblk != (mblk_t *)-1) {
6027            freemsg(stp->sd_iocblk);
6028    }
6029    stp->sd_iocblk = NULL;

6031    /*
6032     * If this is marked with 'noerror' (internal; mostly
6033     * I_{P,}{UN,}LINK), then make sure nobody else is able to get
6034     * in here by setting IOCWAITNE.
6035     */
6036    waitflags = IOCWAIT;
6037    if (flag & STR_NOERROR)
6038            waitflags |= IOCWAITNE;

6040    stp->sd_flag |= waitflags;

6042    /*
6043     * Assign sequence number.
6044     */
6045    iocbp->ioc_id = stp->sd_iocid = getiocseqno();

6047    mutex_exit(&stp->sd_lock);

6049    TRACE_1(TR_FAC_STREAMS_FR,
6050        TR_STRDOIOCTL_PUT, "strdoioctl put: stp %p", stp);
6051    stream_willservice(stp);
6052    putnext(stp->sd_wrq, bp);
6053    stream_runservice(stp);

6055    /*
```



```
6056     * Timed wait for acknowledgment.  The wait time is limited by the
6057     * timeout value, which must be a positive integer (number of
6058     * milliseconds) to wait, or 0 (use default value of STRTIMOUT
6059     * milliseconds), or -1 (wait forever).  This will be awakened
6060     * either by an ACK/NAK message arriving, the timer expiring, or
6061     * the timer expiring on another ioctl waiting for control of the
6062     * mechanism.
6063     */
6064 waitioc:
6065    mutex_enter(&stp->sd_lock);


6068    /*
6069     * If the reply has already arrived, don't sleep.  If awakened from
6070     * the sleep, fail only if the reply has not arrived by then.
6071     * Otherwise, process the reply.
6072     */
6073    while (!stp->sd_iocblk) {
6074            clock_t cv_rval;

6076            if (stp->sd_flag & errs) {
6077                    error = strgeterr(stp, errs, 0);
6078                    if (error != 0) {
6079                            stp->sd_flag &= ~waitflags;
6080                            cv_broadcast(&stp->sd_iocmonitor);
6081                            mutex_exit(&stp->sd_lock);
6082                            crfree(crp);
6083                            return (error);
6084                    }
6085            }

6087            TRACE_0(TR_FAC_STREAMS_FR,
6088                TR_STRDOIOCTL_WAIT2,
6089                "strdoioctl sleeps awaiting reply");
6090            ASSERT(error == 0);

6092            cv_rval = str_cv_wait(&stp->sd_monitor, &stp->sd_lock,
6093                (strioc->ic_timout ?
6094                strioc->ic_timout * 1000 : STRTIMOUT), sigflag);

6096            /*
6097             * There are four possible cases here: interrupt, timeout,
6098             * wakeup by IOCWAITNE (above), or wakeup by strrput_nondata (a
6099             * valid M_IOCTL reply).
6100             *
6101             * If we've been awakened by a STR_NOERROR ioctl on some other
6102             * thread, then sd_iocblk will still be NULL, and IOCWAITNE
6103             * will be set.  Pretend as if we just timed out.  Note that
6104             * this other thread waited at least STRTIMOUT before trying to
6105             * awaken our thread, so this is indistinguishable (even for
6106             * INFTIM) from the case where we failed with ETIME waiting on
6107             * IOCWAIT in the prior loop.
6108             */
6109            if (cv_rval > 0 && !(flag & STR_NOERROR) &&
6110                stp->sd_iocblk == NULL && (stp->sd_flag & IOCWAITNE)) {
6111                    cv_rval = -1;
6112            }

6114            /*
6115             * note: STR_NOERROR does not protect
6116             * us here.. use ic_timout < 0
6117             */
6118            if (cv_rval <= 0) {
6119                    if (cv_rval == 0) {
6120                            error = EINTR;
6121                    } else {
```

```
6122                                    error =  ETIME;
6123                            }
6124                            /*
6125                             * A message could have come in after we were scheduled
6126                             * but before we were actually run.
6127                             */
6128                            bp = stp->sd_iocblk;
6129                            stp->sd_iocblk = NULL;
6130                            if (bp != NULL) {
6131                                    if ((bp->b_datap->db_type == M_COPYIN) ||
6132                                        (bp->b_datap->db_type == M_COPYOUT)) {
6133                                            mutex_exit(&stp->sd_lock);
6134                                            if (bp->b_cont) {
6135                                                    freemsg(bp->b_cont);
6136                                                    bp->b_cont = NULL;
6137                                            }
6138                                            bp->b_datap->db_type = M_IOCDATA;
6139                                            bp->b_wptr = bp->b_rptr +
6140                                                sizeof (struct copyresp);
6141                                            resp = (struct copyresp *)bp->b_rptr;
6142                                            resp->cp_rval =
6143                                                (caddr_t)1; /* failure */
6144                                            stream_willservice(stp);
6145                                            putnext(stp->sd_wrq, bp);
6146                                            stream_runservice(stp);
6147                                            mutex_enter(&stp->sd_lock);
6148                                    } else {
6149                                            freemsg(bp);
6150                                    }
6151                            }
6152                            stp->sd_flag &= ~waitflags;
6153                            cv_broadcast(&stp->sd_iocmonitor);
6154                            mutex_exit(&stp->sd_lock);
6155                            crfree(crp);
6156                            return (error);
6157                    }
6158            }
6159            bp = stp->sd_iocblk;
6160            /*
6161             * Note: it is strictly impossible to get here with sd_iocblk set to
6162             * -1.  This is because the initial loop above doesn't allow any new
6163             * ioctls into the fray until all others have passed this point.
6164             */
6165            ASSERT(bp != NULL && bp != (mblk_t *)-1);
6166            TRACE_1(TR_FAC_STREAMS_FR,
6167                TR_STRDOIOCTL_ACK, "strdoioctl got reply: bp %p", bp);
6168            if ((bp->b_datap->db_type == M_IOCACK) ||
6169                (bp->b_datap->db_type == M_IOCNAK)) {
6170                    /* for detection of duplicate ioctl replies */
6171                    stp->sd_iocblk = (mblk_t *)-1;
6172                    stp->sd_flag &= ~waitflags;
6173                    cv_broadcast(&stp->sd_iocmonitor);
6174                    mutex_exit(&stp->sd_lock);
6175            } else {
6176                    /*
6177                     * flags not cleared here because we're still doing
6178                     * copy in/out for ioctl.
6179                     */
6180                    stp->sd_iocblk = NULL;
6181                    mutex_exit(&stp->sd_lock);
6182            }


6185            /*
6186             * Have received acknowledgment.
6187             */
```



```
6189            switch (bp->b_datap->db_type) {
6190            case M_IOCACK:
6191                    /*
6192                     * Positive ack.
6193                     */
6194                    iocbp = (struct iocblk *)bp->b_rptr;

6196                    /*
6197                     * Set error if indicated.
6198                     */
6199                    if (iocbp->ioc_error) {
6200                            error = iocbp->ioc_error;
6201                            break;
6202                    }

6204                    /*
6205                     * Set return value.
6206                     */
6207                    *rvalp = iocbp->ioc_rval;

6209                    /*
6210                     * Data may have been returned in ACK message (ioc_count > 0).
6211                     * If so, copy it out to the user's buffer.
6212                     */
6213                    if (iocbp->ioc_count && !transparent) {
6214                            if (error = getiocd(bp, strioc->ic_dp, copyflag))
6215                                    break;
6216                    }
6217                    if (!transparent) {
6218                            if (len)        /* an M_COPYOUT was used with I_STR */
6219                                    strioc->ic_len = len;
6220                            else
6221                                    strioc->ic_len = (int)iocbp->ioc_count;
6222                    }
6223                    break;

6225            case M_IOCNAK:
6226                    /*
6227                     * Negative ack.
6228                     *
6229                     * The only thing to do is set error as specified
6230                     * in neg ack packet.
6231                     */
6232                    iocbp = (struct iocblk *)bp->b_rptr;

6234                    error = (iocbp->ioc_error ? iocbp->ioc_error : EINVAL);
6235                    break;

6237            case M_COPYIN:
6238                    /*
6239                     * Driver or module has requested user ioctl data.
6240                     */
6241                    reqp = (struct copyreq *)bp->b_rptr;

6243                    /*
6244                     * M_COPYIN should *never* have a message attached, though
6245                     * it's harmless if it does -- thus, panic on a DEBUG
6246                     * kernel and just free it on a non-DEBUG build.
6247                     */
6248                    ASSERT(bp->b_cont == NULL);
6249                    if (bp->b_cont != NULL) {
6250                            freemsg(bp->b_cont);
6251                            bp->b_cont = NULL;
6252                    }
```

```
6254                error = putiocd(bp, reqp->cq_addr, flag, crp);
6255                if (error && bp->b_cont) {
6256                        freemsg(bp->b_cont);
6257                        bp->b_cont = NULL;
6258                }

6260                bp->b_wptr = bp->b_rptr + sizeof (struct copyresp);
6261                bp->b_datap->db_type = M_IOCDATA;

6263                mblk_setcred(bp, crp, curproc->p_pid);
6264                resp = (struct copyresp *)bp->b_rptr;
6265                resp->cp_rval = (caddr_t)(uintptr_t)error;
6266                resp->cp_flag = (fflags & FMODELS);

6268                stream_willservice(stp);
6269                putnext(stp->sd_wrq, bp);
6270                stream_runservice(stp);

6272                if (error) {
6273                        mutex_enter(&stp->sd_lock);
6274                        stp->sd_flag &= ~waitflags;
6275                        cv_broadcast(&stp->sd_iocmonitor);
6276                        mutex_exit(&stp->sd_lock);
6277                        crfree(crp);
6278                        return (error);
6279                }

6281                goto waitioc;

6283        case M_COPYOUT:
6284                /*
6285                 * Driver or module has ioctl data for a user.
6286                 */
6287                reqp = (struct copyreq *)bp->b_rptr;
6288                ASSERT(bp->b_cont != NULL);

6290                /*
6291                 * Always (transparent or non-transparent )
6292                 * use the address specified in the request
6293                 */
6294                taddr = reqp->cq_addr;
6295                if (!transparent)
6296                        len = (int)reqp->cq_size;

6298                /* copyout data to the provided address */
6299                error = getiocd(bp, taddr, copyflag);

6301                freemsg(bp->b_cont);
6302                bp->b_cont = NULL;

6304                bp->b_wptr = bp->b_rptr + sizeof (struct copyresp);
6305                bp->b_datap->db_type = M_IOCDATA;

6307                mblk_setcred(bp, crp, curproc->p_pid);
6308                resp = (struct copyresp *)bp->b_rptr;
6309                resp->cp_rval = (caddr_t)(uintptr_t)error;
6310                resp->cp_flag = (fflags & FMODELS);

6312                stream_willservice(stp);
6313                putnext(stp->sd_wrq, bp);
6314                stream_runservice(stp);

6316                if (error) {
6317                        mutex_enter(&stp->sd_lock);
6318                        stp->sd_flag &= ~waitflags;
6319                        cv_broadcast(&stp->sd_iocmonitor);
```



```
6320                        mutex_exit(&stp->sd_lock);
6321                        crfree(crp);
6322                        return (error);
6323                }
6324                goto waitioc;

6326        default:
6327                ASSERT(0);
6328                mutex_enter(&stp->sd_lock);
6329                stp->sd_flag &= ~waitflags;
6330                cv_broadcast(&stp->sd_iocmonitor);
6331                mutex_exit(&stp->sd_lock);
6332                break;
6333        }

6335        freemsg(bp);
6336        crfree(crp);
6337        return (error);
6338 }

6340 /*
6341  * Send an M_CMD message downstream and wait for a reply.  This is a ptools
6342  * special used to retrieve information from modules/drivers a stream without
6343  * being subjected to flow control or interfering with pending messages on the
6344  * stream (e.g. an ioctl in flight).
6345  */
6346 int
6347 strdocmd(struct stdata *stp, struct strcmd *scp, cred_t *crp)
6348 {
6349        mblk_t *mp;
6350        struct cmdblk *cmdp;
6351        int error = 0;
6352        int errs = STRHUP|STRDERR|STWRERR|STPLEX;
6353        clock_t rval, timeout = STRTIMOUT;

6355        if (scp->sc_len < 0 || scp->sc_len > sizeof (scp->sc_buf) ||
6356            scp->sc_timeout < -1)
6357                return (EINVAL);

6359        if (scp->sc_timeout > 0)
6360                timeout = scp->sc_timeout * MILLISEC;

6362        if ((mp = allocb_cred(sizeof (struct cmdblk), crp,
6363            curproc->p_pid)) == NULL)
6364                return (ENOMEM);

6366        crhold(crp);

6368        cmdp = (struct cmdblk *)mp->b_wptr;
6369        cmdp->cb_cr = crp;
6370        cmdp->cb_cmd = scp->sc_cmd;
6371        cmdp->cb_len = scp->sc_len;
6372        cmdp->cb_error = 0;
6373        mp->b_wptr += sizeof (struct cmdblk);

6375        DB_TYPE(mp) = M_CMD;
6376        DB_CPID(mp) = curproc->p_pid;

6378        /*
6379         * Copy in the payload.
6380         */
6381        if (cmdp->cb_len > 0) {
6382                mp->b_cont = allocb_cred(sizeof (scp->sc_buf), crp,
6383                    curproc->p_pid);
6384                if (mp->b_cont == NULL) {
6385                        error = ENOMEM;
```

```
6386                        goto out;
6387                }
6388
6389                /* cb_len comes from sc_len, which has already been checked */
6390                ASSERT(cmdp->cb_len <= sizeof (scp->sc_buf));
6391                (void) bcopy(scp->sc_buf, mp->b_cont->b_wptr, cmdp->cb_len);
6392                mp->b_cont->b_wptr += cmdp->cb_len;
6393                DB_CPID(mp->b_cont) = curproc->p_pid;
6394        }
6395
6396        /*
6397         * Since this mechanism is strictly for ptools, and since only one
6398         * process can be grabbed at a time, we simply fail if there's
6399         * currently an operation pending.
6400         */
6401        mutex_enter(&stp->sd_lock);
6402        if (stp->sd_flag & STRCMDWAIT) {
6403                mutex_exit(&stp->sd_lock);
6404                error = EBUSY;
6405                goto out;
6406        }
6407        stp->sd_flag |= STRCMDWAIT;
6408        ASSERT(stp->sd_cmdblk == NULL);
6409        mutex_exit(&stp->sd_lock);
6410
6411        putnext(stp->sd_wrq, mp);
6412        mp = NULL;
6413
6414        /*
6415         * Timed wait for acknowledgment.  If the reply has already arrived,
6416         * don't sleep.  If awakened from the sleep, fail only if the reply
6417         * has not arrived by then.  Otherwise, process the reply.
6418         */
6419        mutex_enter(&stp->sd_lock);
6420        while (stp->sd_cmdblk == NULL) {
6421                if (stp->sd_flag & errs) {
6422                        if ((error = strgeterr(stp, errs, 0)) != 0)
6423                                goto waitout;
6424                }
6425
6426                rval = str_cv_wait(&stp->sd_monitor, &stp->sd_lock, timeout, 0);
6427                if (stp->sd_cmdblk != NULL)
6428                        break;
6429
6430                if (rval <= 0) {
6431                        error = (rval == 0) ? EINTR : ETIME;
6432                        goto waitout;
6433                }
6434        }
6435
6436        /*
6437         * We received a reply.
6438         */
6439        mp = stp->sd_cmdblk;
6440        stp->sd_cmdblk = NULL;
6441        ASSERT(mp != NULL && DB_TYPE(mp) == M_CMD);
6442        ASSERT(stp->sd_flag & STRCMDWAIT);
6443        stp->sd_flag &= ~STRCMDWAIT;
6444        mutex_exit(&stp->sd_lock);
6445
6446        cmdp = (struct cmdblk *)mp->b_rptr;
6447        if ((error = cmdp->cb_error) != 0)
6448                goto out;
6449
6450        /*
6451         * Data may have been returned in the reply (cb_len > 0).
```



```
6452         * If so, copy it out to the user's buffer.
6453         */
6454        if (cmdp->cb_len > 0) {
6455                if (mp->b_cont == NULL || MBLKL(mp->b_cont) < cmdp->cb_len) {
6456                        error = EPROTO;
6457                        goto out;
6458                }
6459
6460                cmdp->cb_len = MIN(cmdp->cb_len, sizeof (scp->sc_buf));
6461                (void) bcopy(mp->b_cont->b_rptr, scp->sc_buf, cmdp->cb_len);
6462        }
6463        scp->sc_len = cmdp->cb_len;
6464 out:
6465        freemsg(mp);
6466        crfree(crp);
6467        return (error);
6468 waitout:
6469        ASSERT(stp->sd_cmdblk == NULL);
6470        stp->sd_flag &= ~STRCMDWAIT;
6471        mutex_exit(&stp->sd_lock);
6472        crfree(crp);
6473        return (error);
6474 }
6475
6476 /*
6477  * For the SunOS keyboard driver.
6478  * Return the next available "ioctl" sequence number.
6479  * Exported, so that streams modules can send "ioctl" messages
6480  * downstream from their open routine.
6481  */
6482 int
6483 getiocseqno(void)
6484 {
6485        int     i;
6486
6487        mutex_enter(&strresources);
6488        i = ++ioc_id;
6489        mutex_exit(&strresources);
6490        return (i);
6491 }
6492
6493 /*
6494  * Get the next message from the read queue.  If the message is
6495  * priority, STRPRI will have been set by strrput().  This flag
6496  * should be reset only when the entire message at the front of the
6497  * queue as been consumed.
6498  *
6499  * NOTE: strgetmsg and kstrgetmsg have much of the logic in common.
6500  */
6501 int
6502 strgetmsg(
6503        struct vnode *vp,
6504        struct strbuf *mctl,
6505        struct strbuf *mdata,
6506        unsigned char *prip,
6507        int *flagsp,
6508        int fmode,
6509        rval_t *rvp)
6510 {
6511        struct stdata *stp;
6512        mblk_t *bp, *nbp;
6513        mblk_t *savemp = NULL;
6514        mblk_t *savemptail = NULL;
6515        uint_t old_sd_flag;
6516        int flg;
6517        int more = 0;
```

```
6518         int error = 0;
6519         char first = 1;
6520         uint_t mark;            /* Contains MSG*MARK and _LASTMARK */
6521 #define _LASTMARK       0x8000  /* Distinct from MSG*MARK */
6522         unsigned char pri = 0;
6523         queue_t *q;
6524         int     pr = 0;                 /* Partial read successful */
6525         struct uio uios;
6526         struct uio *uiop = &uios;
6527         struct iovec iovs;
6528         unsigned char type;

6530         TRACE_1(TR_FAC_STREAMS_FR, TR_STRGETMSG_ENTER,
6531             "strgetmsg:%p", vp);

6533         ASSERT(vp->v_stream);
6534         stp = vp->v_stream;
6535         rvp->r_val1 = 0;

6537         mutex_enter(&stp->sd_lock);

6539         if ((error = i_straccess(stp, JCREAD)) != 0) {
6540                 mutex_exit(&stp->sd_lock);
6541                 return (error);
6542         }

6544         if (stp->sd_flag & (STRDERR|STPLEX)) {
6545                 error = strgeterr(stp, STRDERR|STPLEX, 0);
6546                 if (error != 0) {
6547                         mutex_exit(&stp->sd_lock);
6548                         return (error);
6549                 }
6550         }
6551         mutex_exit(&stp->sd_lock);

6553         switch (*flagsp) {
6554         case MSG_HIPRI:
6555                 if (*prip != 0)
6556                         return (EINVAL);
6557                 break;

6559         case MSG_ANY:
6560         case MSG_BAND:
6561                 break;

6563         default:
6564                 return (EINVAL);
6565         }
6566         /*
6567          * Setup uio and iov for data part
6568          */
6569         iovs.iov_base = mdata->buf;
6570         iovs.iov_len = mdata->maxlen;
6571         uios.uio_iov = &iovs;
6572         uios.uio_iovcnt = 1;
6573         uios.uio_loffset = 0;
6574         uios.uio_segflg = UIO_USERSPACE;
6575         uios.uio_fmode = 0;
6576         uios.uio_extflg = UIO_COPY_CACHED;
6577         uios.uio_resid = mdata->maxlen;
6578         uios.uio_offset = 0;

6580         q = _RD(stp->sd_wrq);
6581         mutex_enter(&stp->sd_lock);
6582         old_sd_flag = stp->sd_flag;
6583         mark = 0;
```



```
6584         for (;;) {
6585                 int done = 0;
6586                 mblk_t *q_first = q->q_first;

6588                 /*
6589                  * Get the next message of appropriate priority
6590                  * from the stream head.  If the caller is interested
6591                  * in band or hipri messages, then they should already
6592                  * be enqueued at the stream head.  On the other hand
6593                  * if the caller wants normal (band 0) messages, they
6594                  * might be deferred in a synchronous stream and they
6595                  * will need to be pulled up.
6596                  *
6597                  * After we have dequeued a message, we might find that
6598                  * it was a deferred M_SIG that was enqueued at the
6599                  * stream head.  It must now be posted as part of the
6600                  * read by calling strsignal_nolock().
6601                  *
6602                  * Also note that strrput does not enqueue an M_PCSIG,
6603                  * and there cannot be more than one hipri message,
6604                  * so there was no need to have the M_PCSIG case.
6605                  *
6606                  * At some time it might be nice to try and wrap the
6607                  * functionality of kstrgetmsg() and strgetmsg() into
6608                  * a common routine so to reduce the amount of replicated
6609                  * code (since they are extremely similar).
6610                  */
6611                 if (!(*flagsp & (MSG_HIPRI|MSG_BAND))) {
6612                         /* Asking for normal, band0 data */
6613                         bp = strget(stp, q, uiop, first, &error);
6614                         ASSERT(MUTEX_HELD(&stp->sd_lock));
6615                         if (bp != NULL) {
6616                                 if (DB_TYPE(bp) == M_SIG) {
6617                                         strsignal_nolock(stp, *bp->b_rptr,
6618                                             bp->b_band);
6619                                         freemsg(bp);
6620                                         continue;
6621                                 } else {
6622                                         break;
6623                                 }
6624                         }
6625                         if (error != 0)
6626                                 goto getmout;

6628                 /*
6629                  * We can't depend on the value of STRPRI here because
6630                  * the stream head may be in transit. Therefore, we
6631                  * must look at the type of the first message to
6632                  * determine if a high priority messages is waiting
6633                  */
6634                 } else if ((*flagsp & MSG_HIPRI) && q_first != NULL &&
6635                     DB_TYPE(q_first) >= QPCTL &&
6636                     (bp = getq_noenab(q, 0)) != NULL) {
6637                         /* Asked for HIPRI and got one */
6638                         ASSERT(DB_TYPE(bp) >= QPCTL);
6639                         break;
6640                 } else if ((*flagsp & MSG_BAND) && q_first != NULL &&
6641                     ((q_first->b_band >= *prip) || DB_TYPE(q_first) >= QPCTL) &&
6642                     (bp = getq_noenab(q, 0)) != NULL) {
6643                         /*
6644                          * Asked for at least band "prip" and got either at
6645                          * least that band or a hipri message.
6646                          */
6647                         ASSERT(bp->b_band >= *prip || DB_TYPE(bp) >= QPCTL);
6648                         if (DB_TYPE(bp) == M_SIG) {
6649                                 strsignal_nolock(stp, *bp->b_rptr, bp->b_band);
```

```
6650                                freemsg(bp);
6651                                continue;
6652                        } else {
6653                                break;
6654                        }
6655                }

6657                /* No data. Time to sleep? */
6658                qbackenable(q, 0);

6660                /*
6661                 * If STRHUP or STREOF, return 0 length control and data.
6662                 * If resid is 0, then a read(fd,buf,0) was done. Do not
6663                 * sleep to satisfy this request because by default we have
6664                 * zero bytes to return.
6665                 */
6666                if ((stp->sd_flag & (STRHUP|STREOF)) || (mctl->maxlen == 0 &&
6667                    mdata->maxlen == 0)) {
6668                        mctl->len = mdata->len = 0;
6669                        *flagsp = 0;
6670                        mutex_exit(&stp->sd_lock);
6671                        return (0);
6672                }
6673                TRACE_2(TR_FAC_STREAMS_FR, TR_STRGETMSG_WAIT,
6674                    "strgetmsg calls strwaitq:%p, %p",
6675                    vp, uiop);
6676                if (((error = strwaitq(stp, GETWAIT, (ssize_t)0, fmode, -1,
6677                    &done)) != 0) || done) {
6678                        TRACE_2(TR_FAC_STREAMS_FR, TR_STRGETMSG_DONE,
6679                            "strgetmsg error or done:%p, %p",
6680                            vp, uiop);
6681                        mutex_exit(&stp->sd_lock);
6682                        return (error);
6683                }
6684                TRACE_2(TR_FAC_STREAMS_FR, TR_STRGETMSG_AWAKE,
6685                    "strgetmsg awakes:%p, %p", vp, uiop);
6686                if ((error = i_straccess(stp, JCREAD)) != 0) {
6687                        mutex_exit(&stp->sd_lock);
6688                        return (error);
6689                }
6690                first = 0;
6691        }
6692        ASSERT(bp != NULL);
6693        /*
6694         * Extract any mark information. If the message is not completely
6695         * consumed this information will be put in the mblk
6696         * that is putback.
6697         * If MSGMARKNEXT is set and the message is completely consumed
6698         * the STRATMARK flag will be set below. Likewise, if
6699         * MSGNOTMARKNEXT is set and the message is
6700         * completely consumed STRNOTATMARK will be set.
6701         */
6702        mark = bp->b_flag & (MSGMARK | MSGMARKNEXT | MSGNOTMARKNEXT);
6703        ASSERT((mark & (MSGMARKNEXT|MSGNOTMARKNEXT)) !=
6704            (MSGMARKNEXT|MSGNOTMARKNEXT));
6705        if (mark != 0 && bp == stp->sd_mark) {
6706                mark |= _LASTMARK;
6707                stp->sd_mark = NULL;
6708        }
6709        /*
6710         * keep track of the original message type and priority
6711         */
6712        pri = bp->b_band;
6713        type = bp->b_datap->db_type;
6714        if (type == M_PASSFP) {
6715                if ((mark & _LASTMARK) && (stp->sd_mark == NULL))
```



```
6716                        stp->sd_mark = bp;
6717                bp->b_flag |= mark & ~_LASTMARK;
6718                putback(stp, q, bp, pri);
6719                qbackenable(q, pri);
6720                mutex_exit(&stp->sd_lock);
6721                return (EBADMSG);
6722        }
6723        ASSERT(type != M_SIG);

6725        /*
6726         * Set this flag so strrput will not generate signals. Need to
6727         * make sure this flag is cleared before leaving this routine
6728         * else signals will stop being sent.
6729         */
6730        stp->sd_flag |= STRGETINPROG;
6731        mutex_exit(&stp->sd_lock);

6733        if (STREAM_NEEDSERVICE(stp))
6734                stream_runservice(stp);

6736        /*
6737         * Set HIPRI flag if message is priority.
6738         */
6739        if (type >= QPCTL)
6740                flg = MSG_HIPRI;
6741        else
6742                flg = MSG_BAND;

6744        /*
6745         * First process PROTO or PCPROTO blocks, if any.
6746         */
6747        if (mctl->maxlen >= 0 && type != M_DATA) {
6748                size_t  n, bcnt;
6749                char    *ubuf;

6751                bcnt = mctl->maxlen;
6752                ubuf = mctl->buf;
6753                while (bp != NULL && bp->b_datap->db_type != M_DATA) {
6754                        if ((n = MIN(bcnt, bp->b_wptr - bp->b_rptr)) != 0 &&
6755                            copyout(bp->b_rptr, ubuf, n)) {
6756                                error = EFAULT;
6757                                mutex_enter(&stp->sd_lock);
6758                                /*
6759                                 * clear stream head pri flag based on
6760                                 * first message type
6761                                 */
6762                                if (type >= QPCTL) {
6763                                        ASSERT(type == M_PCPROTO);
6764                                        stp->sd_flag &= ~STRPRI;
6765                                }
6766                                more = 0;
6767                                freemsg(bp);
6768                                goto getmout;
6769                        }
6770                        ubuf += n;
6771                        bp->b_rptr += n;
6772                        if (bp->b_rptr >= bp->b_wptr) {
6773                                nbp = bp;
6774                                bp = bp->b_cont;
6775                                freeb(nbp);
6776                        }
6777                        ASSERT(n <= bcnt);
6778                        bcnt -= n;
6779                        if (bcnt == 0)
6780                                break;
6781                }
```

```
6782                    mctl->len = mctl->maxlen - bcnt;
6783            } else
6784                    mctl->len = -1;

6786            if (bp && bp->b_datap->db_type != M_DATA) {
6787                    /*
6788                     * More PROTO blocks in msg.
6789                     */
6790                    more |= MORECTL;
6791                    savemp = bp;
6792                    while (bp && bp->b_datap->db_type != M_DATA) {
6793                            savemptail = bp;
6794                            bp = bp->b_cont;
6795                    }
6796                    savemptail->b_cont = NULL;
6797            }

6799            /*
6800             * Now process DATA blocks, if any.
6801             */
6802            if (mdata->maxlen >= 0 && bp) {
6803                    /*
6804                     * struiocopyout will consume a potential zero-length
6805                     * M_DATA even if uio_resid is zero.
6806                     */
6807                    size_t oldresid = uiop->uio_resid;

6809                    bp = struiocopyout(bp, uiop, &error);
6810                    if (error != 0) {
6811                            mutex_enter(&stp->sd_lock);
6812                            /*
6813                             * clear stream head hi pri flag based on
6814                             * first message
6815                             */
6816                            if (type >= QPCTL) {
6817                                    ASSERT(type == M_PCPROTO);
6818                                    stp->sd_flag &= ~STRPRI;
6819                            }
6820                            more = 0;
6821                            freemsg(savemp);
6822                            goto getmout;
6823                    }
6824                    /*
6825                     * (pr == 1) indicates a partial read.
6826                     */
6827                    if (oldresid > uiop->uio_resid)
6828                            pr = 1;
6829                    mdata->len = mdata->maxlen - uiop->uio_resid;
6830            } else
6831                    mdata->len = -1;

6833            if (bp) {                       /* more data blocks in msg */
6834                    more |= MOREDATA;
6835                    if (savemp)
6836                            savemptail->b_cont = bp;
6837                    else
6838                            savemp = bp;
6839            }

6841            mutex_enter(&stp->sd_lock);
6842            if (savemp) {
6843                    if (pr && (savemp->b_datap->db_type == M_DATA) &&
6844                        msgnodata(savemp)) {
6845                            /*
6846                             * Avoid queuing a zero-length tail part of
6847                             * a message. pr=1 indicates that we read some of
```



```
6848                             * the message.
6849                             */
6850                            freemsg(savemp);
6851                            more &= ~MOREDATA;
6852                            /*
6853                             * clear stream head hi pri flag based on
6854                             * first message
6855                             */
6856                            if (type >= QPCTL) {
6857                                    ASSERT(type == M_PCPROTO);
6858                                    stp->sd_flag &= ~STRPRI;
6859                            }
6860                    } else {
6861                            savemp->b_band = pri;
6862                            /*
6863                             * If the first message was HIPRI and the one we're
6864                             * putting back isn't, then clear STRPRI, otherwise
6865                             * set STRPRI again.  Note that we must set STRPRI
6866                             * again since the flush logic in strrput_nondata()
6867                             * may have cleared it while we had sd_lock dropped.
6868                             */
6869                            if (type >= QPCTL) {
6870                                    ASSERT(type == M_PCPROTO);
6871                                    if (queclass(savemp) < QPCTL)
6872                                            stp->sd_flag &= ~STRPRI;
6873                                    else
6874                                            stp->sd_flag |= STRPRI;
6875                            } else if (queclass(savemp) >= QPCTL) {
6876                                    /*
6877                                     * The first message was not a HIPRI message,
6878                                     * but the one we are about to putback is.
6879                                     * For simplicitly, we do not allow for HIPRI
6880                                     * messages to be embedded in the message
6881                                     * body, so just force it to same type as
6882                                     * first message.
6883                                     */
6884                                    ASSERT(type == M_DATA || type == M_PROTO);
6885                                    ASSERT(savemp->b_datap->db_type == M_PCPROTO);
6886                                    savemp->b_datap->db_type = type;
6887                            }
6888                            if (mark != 0) {
6889                                    savemp->b_flag |= mark & ~_LASTMARK;
6890                                    if ((mark & _LASTMARK) &&
6891                                        (stp->sd_mark == NULL)) {
6892                                            /*
6893                                             * If another marked message arrived
6894                                             * while sd_lock was not held sd_mark
6895                                             * would be non-NULL.
6896                                             */
6897                                            stp->sd_mark = savemp;
6898                                    }
6899                            }
6900                            putback(stp, q, savemp, pri);
6901                    }
6902            } else {
6903                    /*
6904                     * The complete message was consumed.
6905                     *
6906                     * If another M_PCPROTO arrived while sd_lock was not held
6907                     * it would have been discarded since STRPRI was still set.
6908                     *
6909                     * Move the MSG*MARKNEXT information
6910                     * to the stream head just in case
6911                     * the read queue becomes empty.
6912                     * clear stream head hi pri flag based on
6913                     * first message
```

```
6914                  *
6915                  * If the stream head was at the mark
6916                  * (STRATMARK) before we dropped sd_lock above
6917                  * and some data was consumed then we have
6918                  * moved past the mark thus STRATMARK is
6919                  * cleared. However, if a message arrived in
6920                  * strrput during the copyout above causing
6921                  * STRATMARK to be set we can not clear that
6922                  * flag.
6923                  */
6924                 if (type >= QPCTL) {
6925                         ASSERT(type == M_PCPROTO);
6926                         stp->sd_flag &= ~STRPRI;
6927                 }
6928                 if (mark & (MSGMARKNEXT|MSGNOTMARKNEXT|MSGMARK)) {
6929                         if (mark & MSGMARKNEXT) {
6930                                 stp->sd_flag &= ~STRNOTATMARK;
6931                                 stp->sd_flag |= STRATMARK;
6932                         } else if (mark & MSGNOTMARKNEXT) {
6933                                 stp->sd_flag &= ~STRATMARK;
6934                                 stp->sd_flag |= STRNOTATMARK;
6935                         } else {
6936                                 stp->sd_flag &= ~(STRATMARK|STRNOTATMARK);
6937                         }
6938                 } else if (pr && (old_sd_flag & STRATMARK)) {
6939                         stp->sd_flag &= ~STRATMARK;
6940                 }
6941         }

6943         *flagsp = flg;
6944         *prip = pri;

6946         /*
6947          * Getmsg cleanup processing - if the state of the queue has changed
6948          * some signals may need to be sent and/or poll awakened.
6949          */
6950 getmout:
6951         qbackenable(q, pri);

6953         /*
6954          * We dropped the stream head lock above. Send all M_SIG messages
6955          * before processing stream head for SIGPOLL messages.
6956          */
6957         ASSERT(MUTEX_HELD(&stp->sd_lock));
6958         while ((bp = q->q_first) != NULL &&
6959             (bp->b_datap->db_type == M_SIG)) {
6960                 /*
6961                  * sd_lock is held so the content of the read queue can not
6962                  * change.
6963                  */
6964                 bp = getq(q);
6965                 ASSERT(bp != NULL && bp->b_datap->db_type == M_SIG);

6967                 strsignal_nolock(stp, *bp->b_rptr, bp->b_band);
6968                 mutex_exit(&stp->sd_lock);
6969                 freemsg(bp);
6970                 if (STREAM_NEEDSERVICE(stp))
6971                         stream_runservice(stp);
6972                 mutex_enter(&stp->sd_lock);
6973         }

6975         /*
6976          * stream head cannot change while we make the determination
6977          * whether or not to send a signal. Drop the flag to allow strrput
6978          * to send firstmsgsigs again.
6979          */
```



```
6980         stp->sd_flag &= ~STRGETINPROG;

6982         /*
6983          * If the type of message at the front of the queue changed
6984          * due to the receive the appropriate signals and pollwakeup events
6985          * are generated. The type of changes are:
6986          *      Processed a hipri message, q_first is not hipri.
6987          *      Processed a band X message, and q_first is band Y.
6988          * The generated signals and pollwakeups are identical to what
6989          * strrput() generates should the message that is now on q_first
6990          * arrive to an empty read queue.
6991          *
6992          * Note: only strrput will send a signal for a hipri message.
6993          */
6994         if ((bp = q->q_first) != NULL && !(stp->sd_flag & STRPRI)) {
6995                 strsigset_t signals = 0;
6996                 strpollset_t pollwakeups = 0;

6998                 if (flg & MSG_HIPRI) {
6999                         /*
7000                          * Removed a hipri message. Regular data at
7001                          * the front of  the queue.
7002                          */
7003                         if (bp->b_band == 0) {
7004                                 signals = S_INPUT | S_RDNORM;
7005                                 pollwakeups = POLLIN | POLLRDNORM;
7006                         } else {
7007                                 signals = S_INPUT | S_RDBAND;
7008                                 pollwakeups = POLLIN | POLLRDBAND;
7009                         }
7010                 } else if (pri != bp->b_band) {
7011                         /*
7012                          * The band is different for the new q_first.
7013                          */
7014                         if (bp->b_band == 0) {
7015                                 signals = S_RDNORM;
7016                                 pollwakeups = POLLIN | POLLRDNORM;
7017                         } else {
7018                                 signals = S_RDBAND;
7019                                 pollwakeups = POLLIN | POLLRDBAND;
7020                         }
7021                 }

7023                 if (pollwakeups != 0) {
7024                         if (pollwakeups == (POLLIN | POLLRDNORM)) {
7025                                 if (!(stp->sd_rput_opt & SR_POLLIN))
7026                                         goto no_pollwake;
7027                                 stp->sd_rput_opt &= ~SR_POLLIN;
7028                         }
7029                         mutex_exit(&stp->sd_lock);
7030                         pollwakeup(&stp->sd_pollist, pollwakeups);
7031                         mutex_enter(&stp->sd_lock);
7032                 }
7033 no_pollwake:

7035                 if (stp->sd_sigflags & signals)
7036                         strsendsig(stp->sd_siglist, signals, bp->b_band, 0);
7037         }
7038         mutex_exit(&stp->sd_lock);

7040         rvp->r_val1 = more;
7041         return (error);
7042 #undef  _LASTMARK
7043 }

7045 /*
```

```
 * Get the next message from the read queue.  If the message is
 * priority, STRPRI will have been set by strrput().  This flag
 * should be reset only when the entire message at the front of the
 * queue as been consumed.
 *
 * If uiop is NULL all data is returned in mctlp.
 * Note that a NULL uiop implies that FNDELAY and FNONBLOCK are assumed
 * not enabled.
 * The timeout parameter is in milliseconds; -1 for infinity.
 * This routine handles the consolidation private flags:
 *      MSG_IGNERROR    Ignore any stream head error except STPLEX.
 *      MSG_DELAYERROR  Defer the error check until the queue is empty.
 *      MSG_HOLDSIG     Hold signals while waiting for data.
 *      MSG_IPEEK       Only peek at messages.
 *      MSG_DISCARDTAIL Discard the tail M_DATA part of the message
 *                      that doesn't fit.
 *      MSG_NOMARK      If the message is marked leave it on the queue.
 *
 * NOTE: strgetmsg and kstrgetmsg have much of the logic in common.
 */
int
kstrgetmsg(
        struct vnode *vp,
        mblk_t **mctlp,
        struct uio *uiop,
        unsigned char *prip,
        int *flagsp,
        clock_t timout,
        rval_t *rvp)
{
        struct stdata *stp;
        mblk_t *bp, *nbp;
        mblk_t *savemp = NULL;
        mblk_t *savemptail = NULL;
        int flags;
        uint_t old_sd_flag;
        int flg;
        int more = 0;
        int error = 0;
        char first = 1;
        uint_t mark;            /* Contains MSG*MARK and _LASTMARK */
#define _LASTMARK       0x8000  /* Distinct from MSG*MARK */
        unsigned char pri = 0;
        queue_t *q;
        int     pr = 0;                 /* Partial read successful */
        unsigned char type;

        TRACE_1(TR_FAC_STREAMS_FR, TR_KSTRGETMSG_ENTER,
            "kstrgetmsg:%p", vp);

        ASSERT(vp->v_stream);
        stp = vp->v_stream;
        rvp->r_val1 = 0;

        mutex_enter(&stp->sd_lock);

        if ((error = i_straccess(stp, JCREAD)) != 0) {
                mutex_exit(&stp->sd_lock);
                return (error);
        }

        flags = *flagsp;
        if (stp->sd_flag & (STRDERR|STPLEX)) {
                if ((stp->sd_flag & STPLEX) ||
                    (flags & (MSG_IGNERROR|MSG_DELAYERROR)) == 0) {
                        error = strgeterr(stp, STRDERR|STPLEX,
```



```
                            (flags & MSG_IPEEK));
                        if (error != 0) {
                                mutex_exit(&stp->sd_lock);
                                return (error);
                        }
                }
        }
        mutex_exit(&stp->sd_lock);

        switch (flags & (MSG_HIPRI|MSG_ANY|MSG_BAND)) {
        case MSG_HIPRI:
                if (*prip != 0)
                        return (EINVAL);
                break;

        case MSG_ANY:
        case MSG_BAND:
                break;

        default:
                return (EINVAL);
        }

retry:
        q = _RD(stp->sd_wrq);
        mutex_enter(&stp->sd_lock);
        old_sd_flag = stp->sd_flag;
        mark = 0;
        for (;;) {
                int done = 0;
                int waitflag;
                int fmode;
                mblk_t *q_first = q->q_first;

                /*
                 * This section of the code operates just like the code
                 * in strgetmsg().  There is a comment there about what
                 * is going on here.
                 */
                if (!(flags & (MSG_HIPRI|MSG_BAND))) {
                        /* Asking for normal, band0 data */
                        bp = strget(stp, q, uiop, first, &error);
                        ASSERT(MUTEX_HELD(&stp->sd_lock));
                        if (bp != NULL) {
                                if (DB_TYPE(bp) == M_SIG) {
                                        strsignal_nolock(stp, *bp->b_rptr,
                                            bp->b_band);
                                        freemsg(bp);
                                        continue;
                                } else {
                                        break;
                                }
                        }
                        if (error != 0) {
                                goto getmout;
                        }
                /*
                 * We can't depend on the value of STRPRI here because
                 * the stream head may be in transit. Therefore, we
                 * must look at the type of the first message to
                 * determine if a high priority messages is waiting
                 */
                } else if ((flags & MSG_HIPRI) && q_first != NULL &&
                    DB_TYPE(q_first) >= QPCTL &&
                    (bp = getq_noenab(q, 0)) != NULL) {
                        ASSERT(DB_TYPE(bp) >= QPCTL);
```

```
7178                    break;
7179            } else if ((flags & MSG_BAND) && q_first != NULL &&
7180                ((q_first->b_band >= *prip) || DB_TYPE(q_first) >= QPCTL) &&
7181                (bp = getq_noenab(q, 0)) != NULL) {
7182                    /*
7183                     * Asked for at least band "prip" and got either at
7184                     * least that band or a hipri message.
7185                     */
7186                    ASSERT(bp->b_band >= *prip || DB_TYPE(bp) >= QPCTL);
7187                    if (DB_TYPE(bp) == M_SIG) {
7188                            strsignal_nolock(stp, *bp->b_rptr, bp->b_band);
7189                            freemsg(bp);
7190                            continue;
7191                    } else {
7192                            break;
7193                    }
7194            }

7196            /* No data. Time to sleep? */
7197            qbackenable(q, 0);

7199            /*
7200             * Delayed error notification?
7201             */
7202            if ((stp->sd_flag & (STRDERR|STPLEX)) &&
7203                (flags & (MSG_IGNERROR|MSG_DELAYERROR)) == MSG_DELAYERROR) {
7204                    error = strgeterr(stp, STRDERR|STPLEX,
7205                        (flags & MSG_IPEEK));
7206                    if (error != 0) {
7207                            mutex_exit(&stp->sd_lock);
7208                            return (error);
7209                    }
7210            }

7212            /*
7213             * If STRHUP or STREOF, return 0 length control and data.
7214             * If a read(fd,buf,0) has been done, do not sleep, just
7215             * return.
7216             *
7217             * If mctlp == NULL and uiop == NULL, then the code will
7218             * do the strwaitq. This is an understood way of saying
7219             * sleep "polling" until a message is received.
7220             */
7221            if ((stp->sd_flag & (STRHUP|STREOF)) ||
7222                (uiop != NULL && uiop->uio_resid == 0)) {
7223                    if (mctlp != NULL)
7224                            *mctlp = NULL;
7225                    *flagsp = 0;
7226                    mutex_exit(&stp->sd_lock);
7227                    return (0);
7228            }

7230            waitflag = GETWAIT;
7231            if (flags &
7232                (MSG_HOLDSIG|MSG_IGNERROR|MSG_IPEEK|MSG_DELAYERROR)) {
7233                    if (flags & MSG_HOLDSIG)
7234                            waitflag |= STR_NOSIG;
7235                    if (flags & MSG_IGNERROR)
7236                            waitflag |= STR_NOERROR;
7237                    if (flags & MSG_IPEEK)
7238                            waitflag |= STR_PEEK;
7239                    if (flags & MSG_DELAYERROR)
7240                            waitflag |= STR_DELAYERR;
7241            }
7242            if (uiop != NULL)
7243                    fmode = uiop->uio_fmode;
```



```
7244            else
7245                    fmode = 0;

7247            TRACE_2(TR_FAC_STREAMS_FR, TR_KSTRGETMSG_WAIT,
7248                "kstrgetmsg calls strwaitq:%p, %p",
7249                vp, uiop);
7250            if (((error = strwaitq(stp, waitflag, (ssize_t)0,
7251                fmode, timout, &done))) != 0 || done) {
7252                    TRACE_2(TR_FAC_STREAMS_FR, TR_KSTRGETMSG_DONE,
7253                        "kstrgetmsg error or done:%p, %p",
7254                        vp, uiop);
7255                    mutex_exit(&stp->sd_lock);
7256                    return (error);
7257            }
7258            TRACE_2(TR_FAC_STREAMS_FR, TR_KSTRGETMSG_AWAKE,
7259                "kstrgetmsg awakes:%p, %p", vp, uiop);
7260            if ((error = i_straccess(stp, JCREAD)) != 0) {
7261                    mutex_exit(&stp->sd_lock);
7262                    return (error);
7263            }
7264            first = 0;
7265    }
7266    ASSERT(bp != NULL);
7267    /*
7268     * Extract any mark information. If the message is not completely
7269     * consumed this information will be put in the mblk
7270     * that is putback.
7271     * If MSGMARKNEXT is set and the message is completely consumed
7272     * the STRATMARK flag will be set below. Likewise, if
7273     * MSGNOTMARKNEXT is set and the message is
7274     * completely consumed STRNOTATMARK will be set.
7275     */
7276    mark = bp->b_flag & (MSGMARK | MSGMARKNEXT | MSGNOTMARKNEXT);
7277    ASSERT((mark & (MSGMARKNEXT|MSGNOTMARKNEXT)) !=
7278        (MSGMARKNEXT|MSGNOTMARKNEXT));
7279    pri = bp->b_band;
7280    if (mark != 0) {
7281            /*
7282             * If the caller doesn't want the mark return.
7283             * Used to implement MSG_WAITALL in sockets.
7284             */
7285            if (flags & MSG_NOMARK) {
7286                    putback(stp, q, bp, pri);
7287                    qbackenable(q, pri);
7288                    mutex_exit(&stp->sd_lock);
7289                    return (EWOULDBLOCK);
7290            }
7291            if (bp == stp->sd_mark) {
7292                    mark |= _LASTMARK;
7293                    stp->sd_mark = NULL;
7294            }
7295    }

7297    /*
7298     * keep track of the first message type
7299     */
7300    type = bp->b_datap->db_type;

7302    if (bp->b_datap->db_type == M_PASSFP) {
7303            if ((mark & _LASTMARK) && (stp->sd_mark == NULL))
7304                    stp->sd_mark = bp;
7305            bp->b_flag |= mark & ~_LASTMARK;
7306            putback(stp, q, bp, pri);
7307            qbackenable(q, pri);
7308            mutex_exit(&stp->sd_lock);
7309            return (EBADMSG);
```

```
7310         }
7311         ASSERT(type != M_SIG);
7312
7313         if (flags & MSG_IPEEK) {
7314                 /*
7315                  * Clear any struioflag - we do the uiomove over again
7316                  * when peeking since it simplifies the code.
7317                  *
7318                  * Dup the message and put the original back on the queue.
7319                  * If dupmsg() fails, try again with copymsg() to see if
7320                  * there is indeed a shortage of memory.  dupmsg() may fail
7321                  * if db_ref in any of the messages reaches its limit.
7322                  */
7323
7324                 if ((nbp = dupmsg(bp)) == NULL && (nbp = copymsg(bp)) == NULL) {
7325                         /*
7326                          * Restore the state of the stream head since we
7327                          * need to drop sd_lock (strwaitbuf is sleeping).
7328                          */
7329                         size_t size = msgdsize(bp);
7330
7331                         if ((mark & _LASTMARK) && (stp->sd_mark == NULL))
7332                                 stp->sd_mark = bp;
7333                         bp->b_flag |= mark & ~_LASTMARK;
7334                         putback(stp, q, bp, pri);
7335                         mutex_exit(&stp->sd_lock);
7336                         error = strwaitbuf(size, BPRI_HI);
7337                         if (error) {
7338                                 /*
7339                                  * There is no net change to the queue thus
7340                                  * no need to qbackenable.
7341                                  */
7342                                 return (error);
7343                         }
7344                         goto retry;
7345                 }
7346
7347                 if ((mark & _LASTMARK) && (stp->sd_mark == NULL))
7348                         stp->sd_mark = bp;
7349                 bp->b_flag |= mark & ~_LASTMARK;
7350                 putback(stp, q, bp, pri);
7351                 bp = nbp;
7352         }
7353
7354         /*
7355          * Set this flag so strrput will not generate signals. Need to
7356          * make sure this flag is cleared before leaving this routine
7357          * else signals will stop being sent.
7358          */
7359         stp->sd_flag |= STRGETINPROG;
7360         mutex_exit(&stp->sd_lock);
7361
7362         if ((stp->sd_rputdatafunc != NULL) && (DB_TYPE(bp) == M_DATA)) {
7363                 mblk_t *tmp, *prevmp;
7364
7365                 /*
7366                  * Put first non-data mblk back to stream head and
7367                  * cut the mblk chain so sd_rputdatafunc only sees
7368                  * M_DATA mblks. We can skip the first mblk since it
7369                  * is M_DATA according to the condition above.
7370                  */
7371                 for (prevmp = bp, tmp = bp->b_cont; tmp != NULL;
7372                     prevmp = tmp, tmp = tmp->b_cont) {
7373                         if (DB_TYPE(tmp) != M_DATA) {
7374                                 prevmp->b_cont = NULL;
7375                                 mutex_enter(&stp->sd_lock);
```



```
7376                                 putback(stp, q, tmp, tmp->b_band);
7377                                 mutex_exit(&stp->sd_lock);
7378                                 break;
7379                         }
7380                 }
7381
7382                 bp = (stp->sd_rputdatafunc)(stp->sd_vnode, bp,
7383                     NULL, NULL, NULL, NULL);
7384
7385                 if (bp == NULL)
7386                         goto retry;
7387         }
7388
7389         if (STREAM_NEEDSERVICE(stp))
7390                 stream_runservice(stp);
7391
7392         /*
7393          * Set HIPRI flag if message is priority.
7394          */
7395         if (type >= QPCTL)
7396                 flg = MSG_HIPRI;
7397         else
7398                 flg = MSG_BAND;
7399
7400         /*
7401          * First process PROTO or PCPROTO blocks, if any.
7402          */
7403         if (mctlp != NULL && type != M_DATA) {
7404                 mblk_t *nbp;
7405
7406                 *mctlp = bp;
7407                 while (bp->b_cont && bp->b_cont->b_datap->db_type != M_DATA)
7408                         bp = bp->b_cont;
7409                 nbp = bp->b_cont;
7410                 bp->b_cont = NULL;
7411                 bp = nbp;
7412         }
7413
7414         if (bp && bp->b_datap->db_type != M_DATA) {
7415                 /*
7416                  * More PROTO blocks in msg. Will only happen if mctlp is NULL.
7417                  */
7418                 more |= MORECTL;
7419                 savemp = bp;
7420                 while (bp && bp->b_datap->db_type != M_DATA) {
7421                         savemptail = bp;
7422                         bp = bp->b_cont;
7423                 }
7424                 savemptail->b_cont = NULL;
7425         }
7426
7427         /*
7428          * Now process DATA blocks, if any.
7429          */
7430         if (uiop == NULL) {
7431                 /* Append data to tail of mctlp */
7432
7433                 if (mctlp != NULL) {
7434                         mblk_t **mpp = mctlp;
7435
7436                         while (*mpp != NULL)
7437                                 mpp = &((*mpp)->b_cont);
7438                         *mpp = bp;
7439                         bp = NULL;
7440                 }
7441         } else if (uiop->uio_resid >= 0 && bp) {
```

```
7442                size_t oldresid = uiop->uio_resid;

7444                /*
7445                 * If a streams message is likely to consist
7446                 * of many small mblks, it is pulled up into
7447                 * one continuous chunk of memory.
7448                 * The size of the first mblk may be bogus because
7449                 * successive read() calls on the socket reduce
7450                 * the size of this mblk until it is exhausted
7451                 * and then the code walks on to the next. Thus
7452                 * the size of the mblk may not be the original size
7453                 * that was passed up, it's simply a remainder
7454                 * and hence can be very small without any
7455                 * implication that the packet is badly fragmented.
7456                 * So the size of the possible second mblk is
7457                 * used to spot a badly fragmented packet.
7458                 * see longer comment at top of page
7459                 * by mblk_pull_len declaration.
7460                 */

7462                if (bp->b_cont != NULL && MBLKL(bp->b_cont) < mblk_pull_len) {
7463                        (void) pullupmsg(bp, -1);
7464                }

7466                bp = struiocopyout(bp, uiop, &error);
7467                if (error != 0) {
7468                        if (mctlp != NULL) {
7469                                freemsg(*mctlp);
7470                                *mctlp = NULL;
7471                        } else
7472                                freemsg(savemp);
7473                        mutex_enter(&stp->sd_lock);
7474                        /*
7475                         * clear stream head hi pri flag based on
7476                         * first message
7477                         */
7478                        if (!(flags & MSG_IPEEK) && (type >= QPCTL)) {
7479                                ASSERT(type == M_PCPROTO);
7480                                stp->sd_flag &= ~STRPRI;
7481                        }
7482                        more = 0;
7483                        goto getmout;
7484                }
7485                /*
7486                 * (pr == 1) indicates a partial read.
7487                 */
7488                if (oldresid > uiop->uio_resid)
7489                        pr = 1;
7490        }

7492        if (bp) {                       /* more data blocks in msg */
7493                more |= MOREDATA;
7494                if (savemp)
7495                        savemptail->b_cont = bp;
7496                else
7497                        savemp = bp;
7498        }

7500        mutex_enter(&stp->sd_lock);
7501        if (savemp) {
7502                if (flags & (MSG_IPEEK|MSG_DISCARDTAIL)) {
7503                        /*
7504                         * When MSG_DISCARDTAIL is set or
7505                         * when peeking discard any tail. When peeking this
7506                         * is the tail of the dup that was copied out - the
7507                         * message has already been putback on the queue.
```



```
7508                         * Return MOREDATA to the caller even though the data
7509                         * is discarded. This is used by sockets (to
7510                         * set MSG_TRUNC).
7511                         */
7512                        freemsg(savemp);
7513                        if (!(flags & MSG_IPEEK) && (type >= QPCTL)) {
7514                                ASSERT(type == M_PCPROTO);
7515                                stp->sd_flag &= ~STRPRI;
7516                        }
7517                } else if (pr && (savemp->b_datap->db_type == M_DATA) &&
7518                    msgnodata(savemp)) {
7519                        /*
7520                         * Avoid queuing a zero-length tail part of
7521                         * a message. pr=1 indicates that we read some of
7522                         * the message.
7523                         */
7524                        freemsg(savemp);
7525                        more &= ~MOREDATA;
7526                        if (type >= QPCTL) {
7527                                ASSERT(type == M_PCPROTO);
7528                                stp->sd_flag &= ~STRPRI;
7529                        }
7530                } else {
7531                        savemp->b_band = pri;
7532                        /*
7533                         * If the first message was HIPRI and the one we're
7534                         * putting back isn't, then clear STRPRI, otherwise
7535                         * set STRPRI again.  Note that we must set STRPRI
7536                         * again since the flush logic in strrput_nondata()
7537                         * may have cleared it while we had sd_lock dropped.
7538                         */

7540                        if (type >= QPCTL) {
7541                                ASSERT(type == M_PCPROTO);
7542                                if (queclass(savemp) < QPCTL)
7543                                        stp->sd_flag &= ~STRPRI;
7544                                else
7545                                        stp->sd_flag |= STRPRI;
7546                        } else if (queclass(savemp) >= QPCTL) {
7547                                /*
7548                                 * The first message was not a HIPRI message,
7549                                 * but the one we are about to putback is.
7550                                 * For simplicitly, we do not allow for HIPRI
7551                                 * messages to be embedded in the message
7552                                 * body, so just force it to same type as
7553                                 * first message.
7554                                 */
7555                                ASSERT(type == M_DATA || type == M_PROTO);
7556                                ASSERT(savemp->b_datap->db_type == M_PCPROTO);
7557                                savemp->b_datap->db_type = type;
7558                        }
7559                        if (mark != 0) {
7560                                if ((mark & _LASTMARK) &&
7561                                    (stp->sd_mark == NULL)) {
7562                                        /*
7563                                         * If another marked message arrived
7564                                         * while sd_lock was not held sd_mark
7565                                         * would be non-NULL.
7566                                         */
7567                                        stp->sd_mark = savemp;
7568                                }
7569                                savemp->b_flag |= mark & ~_LASTMARK;
7570                        }
7571                        putback(stp, q, savemp, pri);
7572                }
7573        } else if (!(flags & MSG_IPEEK)) {
```

```
7574                 /*
7575                  * The complete message was consumed.
7576                  *
7577                  * If another M_PCPROTO arrived while sd_lock was not held
7578                  * it would have been discarded since STRPRI was still set.
7579                  *
7580                  * Move the MSG*MARKNEXT information
7581                  * to the stream head just in case
7582                  * the read queue becomes empty.
7583                  * clear stream head hi pri flag based on
7584                  * first message
7585                  *
7586                  * If the stream head was at the mark
7587                  * (STRATMARK) before we dropped sd_lock above
7588                  * and some data was consumed then we have
7589                  * moved past the mark thus STRATMARK is
7590                  * cleared. However, if a message arrived in
7591                  * strrput during the copyout above causing
7592                  * STRATMARK to be set we can not clear that
7593                  * flag.
7594                  * XXX A "perimeter" would help by single-threading strrput,
7595                  * strread, strgetmsg and kstrgetmsg.
7596                  */
7597                 if (type >= QPCTL) {
7598                         ASSERT(type == M_PCPROTO);
7599                         stp->sd_flag &= ~STRPRI;
7600                 }
7601                 if (mark & (MSGMARKNEXT|MSGNOTMARKNEXT|MSGMARK)) {
7602                         if (mark & MSGMARKNEXT) {
7603                                 stp->sd_flag &= ~STRNOTATMARK;
7604                                 stp->sd_flag |= STRATMARK;
7605                         } else if (mark & MSGNOTMARKNEXT) {
7606                                 stp->sd_flag &= ~STRATMARK;
7607                                 stp->sd_flag |= STRNOTATMARK;
7608                         } else {
7609                                 stp->sd_flag &= ~(STRATMARK|STRNOTATMARK);
7610                         }
7611                 } else if (pr && (old_sd_flag & STRATMARK)) {
7612                         stp->sd_flag &= ~STRATMARK;
7613                 }
7614         }

7616         *flagsp = flg;
7617         *prip = pri;

7619         /*
7620          * Getmsg cleanup processing - if the state of the queue has changed
7621          * some signals may need to be sent and/or poll awakened.
7622          */
7623 getmout:
7624         qbackenable(q, pri);

7626         /*
7627          * We dropped the stream head lock above. Send all M_SIG messages
7628          * before processing stream head for SIGPOLL messages.
7629          */
7630         ASSERT(MUTEX_HELD(&stp->sd_lock));
7631         while ((bp = q->q_first) != NULL &&
7632             (bp->b_datap->db_type == M_SIG)) {
7633                 /*
7634                  * sd_lock is held so the content of the read queue can not
7635                  * change.
7636                  */
7637                 bp = getq(q);
7638                 ASSERT(bp != NULL && bp->b_datap->db_type == M_SIG);
```



```
7640                 strsignal_nolock(stp, *bp->b_rptr, bp->b_band);
7641                 mutex_exit(&stp->sd_lock);
7642                 freemsg(bp);
7643                 if (STREAM_NEEDSERVICE(stp))
7644                         stream_runservice(stp);
7645                 mutex_enter(&stp->sd_lock);
7646         }

7648         /*
7649          * stream head cannot change while we make the determination
7650          * whether or not to send a signal. Drop the flag to allow strrput
7651          * to send firstmsgsigs again.
7652          */
7653         stp->sd_flag &= ~STRGETINPROG;

7655         /*
7656          * If the type of message at the front of the queue changed
7657          * due to the receive the appropriate signals and pollwakeup events
7658          * are generated. The type of changes are:
7659          *      Processed a hipri message, q_first is not hipri.
7660          *      Processed a band X message, and q_first is band Y.
7661          * The generated signals and pollwakeups are identical to what
7662          * strrput() generates should the message that is now on q_first
7663          * arrive to an empty read queue.
7664          *
7665          * Note: only strrput will send a signal for a hipri message.
7666          */
7667         if ((bp = q->q_first) != NULL && !(stp->sd_flag & STRPRI)) {
7668                 strsigset_t signals = 0;
7669                 strpollset_t pollwakeups = 0;

7671                 if (flg & MSG_HIPRI) {
7672                         /*
7673                          * Removed a hipri message. Regular data at
7674                          * the front of  the queue.
7675                          */
7676                         if (bp->b_band == 0) {
7677                                 signals = S_INPUT | S_RDNORM;
7678                                 pollwakeups = POLLIN | POLLRDNORM;
7679                         } else {
7680                                 signals = S_INPUT | S_RDBAND;
7681                                 pollwakeups = POLLIN | POLLRDBAND;
7682                         }
7683                 } else if (pri != bp->b_band) {
7684                         /*
7685                          * The band is different for the new q_first.
7686                          */
7687                         if (bp->b_band == 0) {
7688                                 signals = S_RDNORM;
7689                                 pollwakeups = POLLIN | POLLRDNORM;
7690                         } else {
7691                                 signals = S_RDBAND;
7692                                 pollwakeups = POLLIN | POLLRDBAND;
7693                         }
7694                 }

7696                 if (pollwakeups != 0) {
7697                         if (pollwakeups == (POLLIN | POLLRDNORM)) {
7698                                 if (!(stp->sd_rput_opt & SR_POLLIN))
7699                                         goto no_pollwake;
7700                                 stp->sd_rput_opt &= ~SR_POLLIN;
7701                         }
7702                         mutex_exit(&stp->sd_lock);
7703                         pollwakeup(&stp->sd_pollist, pollwakeups);
7704                         mutex_enter(&stp->sd_lock);
7705                 }
```

```
7706 no_pollwake:

7708                 if (stp->sd_sigflags & signals)
7709                         strsendsig(stp->sd_siglist, signals, bp->b_band, 0);
7710         }
7711         mutex_exit(&stp->sd_lock);

7713         rvp->r_val1 = more;
7714         return (error);
7715 #undef  _LASTMARK
7716 }

7718 /*
7719  * Put a message downstream.
7720  *
7721  * NOTE: strputmsg and kstrputmsg have much of the logic in common.
7722  */
7723 int
7724 strputmsg(
7725         struct vnode *vp,
7726         struct strbuf *mctl,
7727         struct strbuf *mdata,
7728         unsigned char pri,
7729         int flag,
7730         int fmode)
7731 {
7732         struct stdata *stp;
7733         queue_t *wqp;
7734         mblk_t *mp;
7735         ssize_t msgsize;
7736         ssize_t rmin, rmax;
7737         int error;
7738         struct uio uios;
7739         struct uio *uiop = &uios;
7740         struct iovec iovs;
7741         int xpg4 = 0;

7743         ASSERT(vp->v_stream);
7744         stp = vp->v_stream;
7745         wqp = stp->sd_wrq;

7747         /*
7748          * If it is an XPG4 application, we need to send
7749          * SIGPIPE below
7750          */

7752         xpg4 = (flag & MSG_XPG4) ? 1 : 0;
7753         flag &= ~MSG_XPG4;

7755         if (AU_AUDITING())
7756                 audit_strputmsg(vp, mctl, mdata, pri, flag, fmode);

7758         mutex_enter(&stp->sd_lock);

7760         if ((error = i_straccess(stp, JCWRITE)) != 0) {
7761                 mutex_exit(&stp->sd_lock);
7762                 return (error);
7763         }

7765         if (stp->sd_flag & (STWRERR|STRHUP|STPLEX)) {
7766                 error = strwriteable(stp, B_FALSE, xpg4);
7767                 if (error != 0) {
7768                         mutex_exit(&stp->sd_lock);
7769                         return (error);
7770                 }
7771         }
```



```
7773         mutex_exit(&stp->sd_lock);

7775         /*
7776          * Check for legal flag value.
7777          */
7778         switch (flag) {
7779         case MSG_HIPRI:
7780                 if ((mctl->len < 0) || (pri != 0))
7781                         return (EINVAL);
7782                 break;
7783         case MSG_BAND:
7784                 break;

7786         default:
7787                 return (EINVAL);
7788         }

7790         TRACE_1(TR_FAC_STREAMS_FR, TR_STRPUTMSG_IN,
7791             "strputmsg in:stp %p", stp);

7793         /* get these values from those cached in the stream head */
7794         rmin = stp->sd_qn_minpsz;
7795         rmax = stp->sd_qn_maxpsz;

7797         /*
7798          * Make sure ctl and data sizes together fall within the
7799          * limits of the max and min receive packet sizes and do
7800          * not exceed system limit.
7801          */
7802         ASSERT((rmax >= 0) || (rmax == INFPSZ));
7803         if (rmax == 0) {
7804                 return (ERANGE);
7805         }
7806         /*
7807          * Use the MAXIMUM of sd_maxblk and q_maxpsz.
7808          * Needed to prevent partial failures in the strmakedata loop.
7809          */
7810         if (stp->sd_maxblk != INFPSZ && rmax != INFPSZ && rmax < stp->sd_maxblk)
7811                 rmax = stp->sd_maxblk;

7813         if ((msgsize = mdata->len) < 0) {
7814                 msgsize = 0;
7815                 rmin = 0;       /* no range check for NULL data part */
7816         }
7817         if ((msgsize < rmin) ||
7818             ((msgsize > rmax) && (rmax != INFPSZ)) ||
7819             (mctl->len > strctlsz)) {
7820                 return (ERANGE);
7821         }

7823         /*
7824          * Setup uio and iov for data part
7825          */
7826         iovs.iov_base = mdata->buf;
7827         iovs.iov_len = msgsize;
7828         uios.uio_iov = &iovs;
7829         uios.uio_iovcnt = 1;
7830         uios.uio_loffset = 0;
7831         uios.uio_segflg = UIO_USERSPACE;
7832         uios.uio_fmode = fmode;
7833         uios.uio_extflg = UIO_COPY_DEFAULT;
7834         uios.uio_resid = msgsize;
7835         uios.uio_offset = 0;

7837         /* Ignore flow control in strput for HIPRI */
```

```
7838         if (flag & MSG_HIPRI)
7839                 flag |= MSG_IGNFLOW;
7840
7841         for (;;) {
7842                 int done = 0;
7843
7844                 /*
7845                  * strput will always free the ctl mblk - even when strput
7846                  * fails.
7847                  */
7848                 if ((error = strmakectl(mctl, flag, fmode, &mp)) != 0) {
7849                         TRACE_3(TR_FAC_STREAMS_FR, TR_STRPUTMSG_OUT,
7850                             "strputmsg out:stp %p out %d error %d",
7851                             stp, 1, error);
7852                         return (error);
7853                 }
7854                 /*
7855                  * Verify that the whole message can be transferred by
7856                  * strput.
7857                  */
7858                 ASSERT(stp->sd_maxblk == INFPSZ ||
7859                     stp->sd_maxblk >= mdata->len);
7860
7861                 msgsize = mdata->len;
7862                 error = strput(stp, mp, uiop, &msgsize, 0, pri, flag);
7863                 mdata->len = msgsize;
7864
7865                 if (error == 0)
7866                         break;
7867
7868                 if (error != EWOULDBLOCK)
7869                         goto out;
7870
7871                 mutex_enter(&stp->sd_lock);
7872                 /*
7873                  * Check for a missed wakeup.
7874                  * Needed since strput did not hold sd_lock across
7875                  * the canputnext.
7876                  */
7877                 if (bcanputnext(wqp, pri)) {
7878                         /* Try again */
7879                         mutex_exit(&stp->sd_lock);
7880                         continue;
7881                 }
7882                 TRACE_2(TR_FAC_STREAMS_FR, TR_STRPUTMSG_WAIT,
7883                     "strputmsg wait:stp %p waits pri %d", stp, pri);
7884                 if (((error = strwaitq(stp, WRITEWAIT, (ssize_t)0, fmode, -1,
7885                     &done)) != 0) || done) {
7886                         mutex_exit(&stp->sd_lock);
7887                         TRACE_3(TR_FAC_STREAMS_FR, TR_STRPUTMSG_OUT,
7888                             "strputmsg out:q %p out %d error %d",
7889                             stp, 0, error);
7890                         return (error);
7891                 }
7892                 TRACE_1(TR_FAC_STREAMS_FR, TR_STRPUTMSG_WAKE,
7893                     "strputmsg wake:stp %p wakes", stp);
7894                 if ((error = i_straccess(stp, JCWRITE)) != 0) {
7895                         mutex_exit(&stp->sd_lock);
7896                         return (error);
7897                 }
7898                 mutex_exit(&stp->sd_lock);
7899         }
7900 out:
7901         /*
7902          * For historic reasons, applications expect EAGAIN
7903          * when data mblk could not be allocated. so change
```



```
7904          * ENOMEM back to EAGAIN
7905          */
7906         if (error == ENOMEM)
7907                 error = EAGAIN;
7908         TRACE_3(TR_FAC_STREAMS_FR, TR_STRPUTMSG_OUT,
7909             "strputmsg out:stp %p out %d error %d", stp, 2, error);
7910         return (error);
7911 }
7912
7913 /*
7914  * Put a message downstream.
7915  * Can send only an M_PROTO/M_PCPROTO by passing in a NULL uiop.
7916  * The fmode flag (NDELAY, NONBLOCK) is the or of the flags in the uio
7917  * and the fmode parameter.
7918  *
7919  * This routine handles the consolidation private flags:
7920  *      MSG_IGNERROR    Ignore any stream head error except STPLEX.
7921  *      MSG_HOLDSIG     Hold signals while waiting for data.
7922  *      MSG_IGNFLOW     Don't check streams flow control.
7923  *
7924  * NOTE: strputmsg and kstrputmsg have much of the logic in common.
7925  */
7926 int
7927 kstrputmsg(
7928         struct vnode *vp,
7929         mblk_t *mctl,
7930         struct uio *uiop,
7931         ssize_t msgsize,
7932         unsigned char pri,
7933         int flag,
7934         int fmode)
7935 {
7936         struct stdata *stp;
7937         queue_t *wqp;
7938         ssize_t rmin, rmax;
7939         int error;
7940
7941         ASSERT(vp->v_stream);
7942         stp = vp->v_stream;
7943         wqp = stp->sd_wrq;
7944         if (AU_AUDITING())
7945                 audit_strputmsg(vp, NULL, NULL, pri, flag, fmode);
7946         if (mctl == NULL)
7947                 return (EINVAL);
7948
7949         mutex_enter(&stp->sd_lock);
7950
7951         if ((error = i_straccess(stp, JCWRITE)) != 0) {
7952                 mutex_exit(&stp->sd_lock);
7953                 freemsg(mctl);
7954                 return (error);
7955         }
7956
7957         if ((stp->sd_flag & STPLEX) || !(flag & MSG_IGNERROR)) {
7958                 if (stp->sd_flag & (STWRERR|STRHUP|STPLEX)) {
7959                         error = strwriteable(stp, B_FALSE, B_TRUE);
7960                         if (error != 0) {
7961                                 mutex_exit(&stp->sd_lock);
7962                                 freemsg(mctl);
7963                                 return (error);
7964                         }
7965                 }
7966         }
7967
7968         mutex_exit(&stp->sd_lock);
```

```
7970         /*
7971          * Check for legal flag value.
7972          */
7973         switch (flag & (MSG_HIPRI|MSG_BAND|MSG_ANY)) {
7974         case MSG_HIPRI:
7975                 if (pri != 0) {
7976                         freemsg(mctl);
7977                         return (EINVAL);
7978                 }
7979                 break;
7980         case MSG_BAND:
7981                 break;
7982         default:
7983                 freemsg(mctl);
7984                 return (EINVAL);
7985         }

7987         TRACE_1(TR_FAC_STREAMS_FR, TR_KSTRPUTMSG_IN,
7988             "kstrputmsg in:stp %p", stp);

7990         /* get these values from those cached in the stream head */
7991         rmin = stp->sd_qn_minpsz;
7992         rmax = stp->sd_qn_maxpsz;

7994         /*
7995          * Make sure ctl and data sizes together fall within the
7996          * limits of the max and min receive packet sizes and do
7997          * not exceed system limit.
7998          */
7999         ASSERT((rmax >= 0) || (rmax == INFPSZ));
8000         if (rmax == 0) {
8001                 freemsg(mctl);
8002                 return (ERANGE);
8003         }
8004         /*
8005          * Use the MAXIMUM of sd_maxblk and q_maxpsz.
8006          * Needed to prevent partial failures in the strmakedata loop.
8007          */
8008         if (stp->sd_maxblk != INFPSZ && rmax != INFPSZ && rmax < stp->sd_maxblk)
8009                 rmax = stp->sd_maxblk;

8011         if (uiop == NULL) {
8012                 msgsize = -1;
8013                 rmin = -1;      /* no range check for NULL data part */
8014         } else {
8015                 /* Use uio flags as well as the fmode parameter flags */
8016                 fmode |= uiop->uio_fmode;

8018                 if ((msgsize < rmin) ||
8019                     ((msgsize > rmax) && (rmax != INFPSZ))) {
8020                         freemsg(mctl);
8021                         return (ERANGE);
8022                 }
8023         }

8025         /* Ignore flow control in strput for HIPRI */
8026         if (flag & MSG_HIPRI)
8027                 flag |= MSG_IGNFLOW;

8029         for (;;) {
8030                 int done = 0;
8031                 int waitflag;
8032                 mblk_t *mp;

8034                 /*
8035                  * strput will always free the ctl mblk - even when strput
```



```
8036                  * fails. If MSG_IGNFLOW is set then any error returned
8037                  * will cause us to break the loop, so we don't need a copy
8038                  * of the message. If MSG_IGNFLOW is not set, then we can
8039                  * get hit by flow control and be forced to try again. In
8040                  * this case we need to have a copy of the message. We
8041                  * do this using copymsg since the message may get modified
8042                  * by something below us.
8043                  *
8044                  * We've observed that many TPI providers do not check db_ref
8045                  * on the control messages but blindly reuse them for the
8046                  * T_OK_ACK/T_ERROR_ACK. Thus using copymsg is more
8047                  * friendly to such providers than using dupmsg. Also, note
8048                  * that sockfs uses MSG_IGNFLOW for all TPI control messages.
8049                  * Only data messages are subject to flow control, hence
8050                  * subject to this copymsg.
8051                  */
8052                 if (flag & MSG_IGNFLOW) {
8053                         mp = mctl;
8054                         mctl = NULL;
8055                 } else {
8056                         do {
8057                                 /*
8058                                  * If a message has a free pointer, the message
8059                                  * must be dupmsg to maintain this pointer.
8060                                  * Code using this facility must be sure
8061                                  * that modules below will not change the
8062                                  * contents of the dblk without checking db_ref
8063                                  * first. If db_ref is > 1, then the module
8064                                  * needs to do a copymsg first. Otherwise,
8065                                  * the contents of the dblk may become
8066                                  * inconsistent because the freesmg/freeb below
8067                                  * may end up calling atomic_add_32_nv.
8068                                  * The atomic_add_32_nv in freeb (accessing
8069                                  * all of db_ref, db_type, db_flags, and
8070                                  * db_struioflag) does not prevent other threads
8071                                  * from concurrently trying to modify e.g.
8072                                  * db_type.
8073                                  */
8074                                 if (mctl->b_datap->db_frtnp != NULL)
8075                                         mp = dupmsg(mctl);
8076                                 else
8077                                         mp = copymsg(mctl);

8079                                 if (mp != NULL)
8080                                         break;

8082                                 error = strwaitbuf(msgdsize(mctl), BPRI_MED);
8083                                 if (error) {
8084                                         freemsg(mctl);
8085                                         return (error);
8086                                 }
8087                         } while (mp == NULL);
8088                 }
8089                 /*
8090                  * Verify that all of msgsize can be transferred by
8091                  * strput.
8092                  */
8093                 ASSERT(stp->sd_maxblk == INFPSZ || stp->sd_maxblk >= msgsize);
8094                 error = strput(stp, mp, uiop, &msgsize, 0, pri, flag);
8095                 if (error == 0)
8096                         break;

8098                 if (error != EWOULDBLOCK)
8099                         goto out;

8101                 /*
```

```
8102                  * IF MSG_IGNFLOW is set we should have broken out of loop
8103                  * above.
8104                  */
8105                 ASSERT(!(flag & MSG_IGNFLOW));
8106                 mutex_enter(&stp->sd_lock);
8107                 /*
8108                  * Check for a missed wakeup.
8109                  * Needed since strput did not hold sd_lock across
8110                  * the canputnext.
8111                  */
8112                 if (bcanputnext(wqp, pri)) {
8113                         /* Try again */
8114                         mutex_exit(&stp->sd_lock);
8115                         continue;
8116                 }
8117                 TRACE_2(TR_FAC_STREAMS_FR, TR_KSTRPUTMSG_WAIT,
8118                     "kstrputmsg wait:stp %p waits pri %d", stp, pri);

8120                 waitflag = WRITEWAIT;
8121                 if (flag & (MSG_HOLDSIG|MSG_IGNERROR)) {
8122                         if (flag & MSG_HOLDSIG)
8123                                 waitflag |= STR_NOSIG;
8124                         if (flag & MSG_IGNERROR)
8125                                 waitflag |= STR_NOERROR;
8126                 }
8127                 if (((error = strwaitq(stp, waitflag,
8128                     (ssize_t)0, fmode, -1, &done)) != 0) || done) {
8129                         mutex_exit(&stp->sd_lock);
8130                         TRACE_3(TR_FAC_STREAMS_FR, TR_KSTRPUTMSG_OUT,
8131                             "kstrputmsg out:stp %p out %d error %d",
8132                             stp, 0, error);
8133                         freemsg(mctl);
8134                         return (error);
8135                 }
8136                 TRACE_1(TR_FAC_STREAMS_FR, TR_KSTRPUTMSG_WAKE,
8137                     "kstrputmsg wake:stp %p wakes", stp);
8138                 if ((error = i_straccess(stp, JCWRITE)) != 0) {
8139                         mutex_exit(&stp->sd_lock);
8140                         freemsg(mctl);
8141                         return (error);
8142                 }
8143                 mutex_exit(&stp->sd_lock);
8144         }
8145 out:
8146         freemsg(mctl);
8147         /*
8148          * For historic reasons, applications expect EAGAIN
8149          * when data mblk could not be allocated. so change
8150          * ENOMEM back to EAGAIN
8151          */
8152         if (error == ENOMEM)
8153                 error = EAGAIN;
8154         TRACE_3(TR_FAC_STREAMS_FR, TR_KSTRPUTMSG_OUT,
8155             "kstrputmsg out:stp %p out %d error %d", stp, 2, error);
8156         return (error);
8157 }

8159 /*
8160  * Determines whether the necessary conditions are set on a stream
8161  * for it to be readable, writeable, or have exceptions.
8162  *
8163  * strpoll handles the consolidation private events:
8164  *      POLLNOERR       Do not return POLLERR even if there are stream
8165  *                      head errors.
8166  *                      Used by sockfs.
8167  *      POLLRDDATA      Do not return POLLIN unless at least one message on
```



```
8168  *                      the queue contains one or more M_DATA mblks. Thus
8169  *                      when this flag is set a queue with only
8170  *                      M_PROTO/M_PCPROTO mblks does not return POLLIN.
8171  *                      Used by sockfs to ignore T_EXDATA_IND messages.
8172  *
8173  * Note: POLLRDDATA assumes that synch streams only return messages with
8174  * an M_DATA attached (i.e. not messages consisting of only
8175  * an M_PROTO/M_PCPROTO part).
8176  */
8177 int
8178 strpoll(
8179         struct stdata *stp,
8180         short events_arg,
8181         int anyyet,
8182         short *reventsp,
8183         struct pollhead **phpp)
8184 {
8185         int events = (ushort_t)events_arg;
8186         int retevents = 0;
8187         mblk_t *mp;
8188         qband_t *qbp;
8189         long sd_flags = stp->sd_flag;
8190         int headlocked = 0;

8192         /*
8193          * For performance, a single 'if' tests for most possible edge
8194          * conditions in one shot
8195          */
8196         if (sd_flags & (STPLEX | STRDERR | STWRERR)) {
8197                 if (sd_flags & STPLEX) {
8198                         *reventsp = POLLNVAL;
8199                         return (EINVAL);
8200                 }
8201                 if (((events & (POLLIN | POLLRDNORM | POLLRDBAND | POLLPRI)) &&
8202                     (sd_flags & STRDERR)) ||
8203                     ((events & (POLLOUT | POLLWRNORM | POLLWRBAND)) &&
8204                     (sd_flags & STWRERR))) {
8205                         if (!(events & POLLNOERR)) {
8206                                 *reventsp = POLLERR;
8207                                 return (0);
8208                         }
8209                 }
8210         }
8211         if (sd_flags & STRHUP) {
8212                 retevents |= POLLHUP;
8213         } else if (events & (POLLWRNORM | POLLWRBAND)) {
8214                 queue_t *tq;
8215                 queue_t *qp = stp->sd_wrq;

8217                 claimstr(qp);
8218                 /* Find next module forward that has a service procedure */
8219                 tq = qp->q_next->q_nfsrv;
8220                 ASSERT(tq != NULL);

8222                 polllock(&stp->sd_pollist, QLOCK(tq));
8223                 if (events & POLLWRNORM) {
8224                         queue_t *sqp;

8226                         if (tq->q_flag & QFULL)
8227                                 /* ensure backq svc procedure runs */
8228                                 tq->q_flag |= QWANTW;
8229                         else if ((sqp = stp->sd_struiowrq) != NULL) {
8230                                 /* Check sync stream barrier write q */
8231                                 mutex_exit(QLOCK(tq));
8232                                 polllock(&stp->sd_pollist, QLOCK(sqp));
8233                                 if (sqp->q_flag & QFULL)
```

```
8234                                    /* ensure pollwakeup() is done */
8235                                    sqp->q_flag |= QWANTWSYNC;
8236                            else
8237                                    retevents |= POLLOUT;
8238                            /* More write events to process ??? */
8239                            if (! (events & POLLWRBAND)) {
8240                                    mutex_exit(QLOCK(sqp));
8241                                    releasestr(qp);
8242                                    goto chkrd;
8243                            }
8244                            mutex_exit(QLOCK(sqp));
8245                            polllock(&stp->sd_pollist, QLOCK(tq));
8246                    } else
8247                            retevents |= POLLOUT;
8248            }
8249            if (events & POLLWRBAND) {
8250                    qbp = tq->q_bandp;
8251                    if (qbp) {
8252                            while (qbp) {
8253                                    if (qbp->qb_flag & QB_FULL)
8254                                            qbp->qb_flag |= QB_WANTW;
8255                                    else
8256                                            retevents |= POLLWRBAND;
8257                                    qbp = qbp->qb_next;
8258                            }
8259                    } else {
8260                            retevents |= POLLWRBAND;
8261                    }
8262            }
8263            mutex_exit(QLOCK(tq));
8264            releasestr(qp);
8265    }
8266 chkrd:
8267    if (sd_flags & STRPRI) {
8268            retevents |= (events & POLLPRI);
8269    } else if (events & (POLLRDNORM | POLLRDBAND | POLLIN)) {
8270            queue_t *qp = _RD(stp->sd_wrq);
8271            int normevents = (events & (POLLIN | POLLRDNORM));

8273            /*
8274             * Note: Need to do polllock() here since ps_lock may be
8275             * held. See bug 4191544.
8276             */
8277            polllock(&stp->sd_pollist, &stp->sd_lock);
8278            headlocked = 1;
8279            mp = qp->q_first;
8280            while (mp) {
8281                    /*
8282                     * For POLLRDDATA we scan b_cont and b_next until we
8283                     * find an M_DATA.
8284                     */
8285                    if ((events & POLLRDDATA) &&
8286                        mp->b_datap->db_type != M_DATA) {
8287                            mblk_t *nmp = mp->b_cont;

8289                            while (nmp != NULL &&
8290                                nmp->b_datap->db_type != M_DATA)
8291                                    nmp = nmp->b_cont;
8292                            if (nmp == NULL) {
8293                                    mp = mp->b_next;
8294                                    continue;
8295                            }
8296                    }
8297                    if (mp->b_band == 0)
8298                            retevents |= normevents;
8299                    else
```



```
8300                                    retevents |= (events & (POLLIN | POLLRDBAND));
8301                            break;
8302                    }
8303                    if (! (retevents & normevents) &&
8304                        (stp->sd_wakeq & RSLEEP)) {
8305                            /*
8306                             * Sync stream barrier read queue has data.
8307                             */
8308                            retevents |= normevents;
8309                    }
8310                    /* Treat eof as normal data */
8311                    if (sd_flags & STREOF)
8312                            retevents |= normevents;
8313            }

8315            *reventsp = (short)retevents;
8316            if (retevents) {
8317                    if (headlocked)
8318                            mutex_exit(&stp->sd_lock);
8319                    return (0);
8320            }

8322            /*
8323             * If poll() has not found any events yet, set up event cell
8324             * to wake up the poll if a requested event occurs on this
8325             * stream.  Check for collisions with outstanding poll requests.
8326             */
8327            if (!anyyet) {
8328                    *phpp = &stp->sd_pollist;
8329                    if (headlocked == 0) {
8330                            polllock(&stp->sd_pollist, &stp->sd_lock);
8331                            headlocked = 1;
8332                    }
8333                    stp->sd_rput_opt |= SR_POLLIN;
8334            }
8335            if (headlocked)
8336                    mutex_exit(&stp->sd_lock);
8337            return (0);
8338 }

8340 /*
8341  * The purpose of putback() is to assure sleeping polls/reads
8342  * are awakened when there are no new messages arriving at the,
8343  * stream head, and a message is placed back on the read queue.
8344  *
8345  * sd_lock must be held when messages are placed back on stream
8346  * head.  (getq() holds sd_lock when it removes messages from
8347  * the queue)
8348  */

8350 static void
8351 putback(struct stdata *stp, queue_t *q, mblk_t *bp, int band)
8352 {
8353            mblk_t  *qfirst;
8354            ASSERT(MUTEX_HELD(&stp->sd_lock));

8356            /*
8357             * As a result of lock-step ordering around q_lock and sd_lock,
8358             * it's possible for function calls like putnext() and
8359             * canputnext() to get an inaccurate picture of how much
8360             * data is really being processed at the stream head.
8361             * We only consolidate with existing messages on the queue
8362             * if the length of the message we want to put back is smaller
8363             * than the queue hiwater mark.
8364             */
8365            if ((stp->sd_rput_opt & SR_CONSOL_DATA) &&
```

```
8366             (DB_TYPE(bp) == M_DATA) && ((qfirst = q->q_first) != NULL) &&
8367             (DB_TYPE(qfirst) == M_DATA) &&
8368             ((qfirst->b_flag & (MSGMARK|MSGDELIM)) == 0) &&
8369             ((bp->b_flag & (MSGMARK|MSGDELIM|MSGMARKNEXT)) == 0) &&
8370             (mp_cont_len(bp, NULL) < q->q_hiwat)) {
8371                 /*
8372                  * We use the same logic as defined in strrput()
8373                  * but in reverse as we are putting back onto the
8374                  * queue and want to retain byte ordering.
8375                  * Consolidate M_DATA messages with M_DATA ONLY.
8376                  * strrput() allows the consolidation of M_DATA onto
8377                  * M_PROTO | M_PCPROTO but not the other way round.
8378                  *
8379                  * The consolidation does not take place if the message
8380                  * we are returning to the queue is marked with either
8381                  * of the marks or the delim flag or if q_first
8382                  * is marked with MSGMARK. The MSGMARK check is needed to
8383                  * handle the odd semantics of MSGMARK where essentially
8384                  * the whole message is to be treated as marked.
8385                  * Carry any MSGMARKNEXT and MSGNOTMARKNEXT from q_first
8386                  * to the front of the b_cont chain.
8387                  */
8388                 rmvq_noenab(q, qfirst);

8390                 /*
8391                  * The first message in the b_cont list
8392                  * tracks MSGMARKNEXT and MSGNOTMARKNEXT.
8393                  * We need to handle the case where we
8394                  * are appending:
8395                  *
8396                  * 1) a MSGMARKNEXT to a MSGNOTMARKNEXT.
8397                  * 2) a MSGMARKNEXT to a plain message.
8398                  * 3) a MSGNOTMARKNEXT to a plain message
8399                  * 4) a MSGNOTMARKNEXT to a MSGNOTMARKNEXT
8400                  *    message.
8401                  *
8402                  * Thus we never append a MSGMARKNEXT or
8403                  * MSGNOTMARKNEXT to a MSGMARKNEXT message.
8404                  */
8405                 if (qfirst->b_flag & MSGMARKNEXT) {
8406                         bp->b_flag |= MSGMARKNEXT;
8407                         bp->b_flag &= ~MSGNOTMARKNEXT;
8408                         qfirst->b_flag &= ~MSGMARKNEXT;
8409                 } else if (qfirst->b_flag & MSGNOTMARKNEXT) {
8410                         bp->b_flag |= MSGNOTMARKNEXT;
8411                         qfirst->b_flag &= ~MSGNOTMARKNEXT;
8412                 }

8414                 linkb(bp, qfirst);
8415         }
8416         (void) putbq(q, bp);

8418         /*
8419          * A message may have come in when the sd_lock was dropped in the
8420          * calling routine. If this is the case and STR*ATMARK info was
8421          * received, need to move that from the stream head to the q_last
8422          * so that SIOCATMARK can return the proper value.
8423          */
8424         if (stp->sd_flag & (STRATMARK | STRNOTATMARK)) {
8425                 unsigned short *flagp = &q->q_last->b_flag;
8426                 uint_t b_flag = (uint_t)*flagp;

8428                 if (stp->sd_flag & STRATMARK) {
8429                         b_flag &= ~MSGNOTMARKNEXT;
8430                         b_flag |= MSGMARKNEXT;
8431                         stp->sd_flag &= ~STRATMARK;
```



```
8432                 } else {
8433                         b_flag &= ~MSGMARKNEXT;
8434                         b_flag |= MSGNOTMARKNEXT;
8435                         stp->sd_flag &= ~STRNOTATMARK;
8436                 }
8437                 *flagp = (unsigned short) b_flag;
8438         }

8440 #ifdef  DEBUG
8441         /*
8442          * Make sure that the flags are not messed up.
8443          */
8444         {
8445                 mblk_t *mp;
8446                 mp = q->q_last;
8447                 while (mp != NULL) {
8448                         ASSERT((mp->b_flag & (MSGMARKNEXT|MSGNOTMARKNEXT)) !=
8449                             (MSGMARKNEXT|MSGNOTMARKNEXT));
8450                         mp = mp->b_cont;
8451                 }
8452         }
8453 #endif
8454         if (q->q_first == bp) {
8455                 short pollevents;

8457                 if (stp->sd_flag & RSLEEP) {
8458                         stp->sd_flag &= ~RSLEEP;
8459                         cv_broadcast(&q->q_wait);
8460                 }
8461                 if (stp->sd_flag & STRPRI) {
8462                         pollevents = POLLPRI;
8463                 } else {
8464                         if (band == 0) {
8465                                 if (!(stp->sd_rput_opt & SR_POLLIN))
8466                                         return;
8467                                 stp->sd_rput_opt &= ~SR_POLLIN;
8468                                 pollevents = POLLIN | POLLRDNORM;
8469                         } else {
8470                                 pollevents = POLLIN | POLLRDBAND;
8471                         }
8472                 }
8473                 mutex_exit(&stp->sd_lock);
8474                 pollwakeup(&stp->sd_pollist, pollevents);
8475                 mutex_enter(&stp->sd_lock);
8476         }
8477 }

8479 /*
8480  * Return the held vnode attached to the stream head of a
8481  * given queue
8482  * It is the responsibility of the calling routine to ensure
8483  * that the queue does not go away (e.g. pop).
8484  */
8485 vnode_t *
8486 strq2vp(queue_t *qp)
8487 {
8488         vnode_t *vp;
8489         vp = STREAM(qp)->sd_vnode;
8490         ASSERT(vp != NULL);
8491         VN_HOLD(vp);
8492         return (vp);
8493 }

8495 /*
8496  * return the stream head write queue for the given vp
8497  * It is the responsibility of the calling routine to ensure
```

```
8498  * that the stream or vnode do not close.
8499  */
8500 queue_t *
8501 strvp2wq(vnode_t *vp)
8502 {
8503         ASSERT(vp->v_stream != NULL);
8504         return (vp->v_stream->sd_wrq);
8505 }

8507 /*
8508  * pollwakeup stream head
8509  * It is the responsibility of the calling routine to ensure
8510  * that the stream or vnode do not close.
8511  */
8512 void
8513 strpollwakeup(vnode_t *vp, short event)
8514 {
8515         ASSERT(vp->v_stream);
8516         pollwakeup(&vp->v_stream->sd_pollist, event);
8517 }

8519 /*
8520  * Mate the stream heads of two vnodes together. If the two vnodes are the
8521  * same, we just make the write-side point at the read-side -- otherwise,
8522  * we do a full mate.  Only works on vnodes associated with streams that are
8523  * still being built and thus have only a stream head.
8524  */
8525 void
8526 strmate(vnode_t *vp1, vnode_t *vp2)
8527 {
8528         queue_t *wrq1 = strvp2wq(vp1);
8529         queue_t *wrq2 = strvp2wq(vp2);

8531         /*
8532          * Verify that there are no modules on the stream yet.  We also
8533          * rely on the stream head always having a service procedure to
8534          * avoid tweaking q_nfsrv.
8535          */
8536         ASSERT(wrq1->q_next == NULL && wrq2->q_next == NULL);
8537         ASSERT(wrq1->q_qinfo->qi_srvp != NULL);
8538         ASSERT(wrq2->q_qinfo->qi_srvp != NULL);

8540         /*
8541          * If the queues are the same, just twist; otherwise do a full mate.
8542          */
8543         if (wrq1 == wrq2) {
8544                 wrq1->q_next = _RD(wrq1);
8545         } else {
8546                 wrq1->q_next = _RD(wrq2);
8547                 wrq2->q_next = _RD(wrq1);
8548                 STREAM(wrq1)->sd_mate = STREAM(wrq2);
8549                 STREAM(wrq1)->sd_flag |= STRMATE;
8550                 STREAM(wrq2)->sd_mate = STREAM(wrq1);
8551                 STREAM(wrq2)->sd_flag |= STRMATE;
8552         }
8553 }

8555 /*
8556  * XXX will go away when console is correctly fixed.
8557  * Clean up the console PIDS, from previous I_SETSIG,
8558  * called only for cnopen which never calls strclean().
8559  */
8560 void
8561 str_cn_clean(struct vnode *vp)
8562 {
8563         strsig_t *ssp, *pssp, *tssp;
```



```
8564         struct stdata *stp;
8565         struct pid  *pidp;
8566         int update = 0;

8568         ASSERT(vp->v_stream);
8569         stp = vp->v_stream;
8570         pssp = NULL;
8571         mutex_enter(&stp->sd_lock);
8572         ssp = stp->sd_siglist;
8573         while (ssp) {
8574                 mutex_enter(&pidlock);
8575                 pidp = ssp->ss_pidp;
8576                 /*
8577                  * Get rid of PID if the proc is gone.
8578                  */
8579                 if (pidp->pid_prinactive) {
8580                         tssp = ssp->ss_next;
8581                         if (pssp)
8582                                 pssp->ss_next = tssp;
8583                         else
8584                                 stp->sd_siglist = tssp;
8585                         ASSERT(pidp->pid_ref <= 1);
8586                         PID_RELE(ssp->ss_pidp);
8587                         mutex_exit(&pidlock);
8588                         kmem_free(ssp, sizeof (strsig_t));
8589                         update = 1;
8590                         ssp = tssp;
8591                         continue;
8592                 } else
8593                         mutex_exit(&pidlock);
8594                 pssp = ssp;
8595                 ssp = ssp->ss_next;
8596         }
8597         if (update) {
8598                 stp->sd_sigflags = 0;
8599                 for (ssp = stp->sd_siglist; ssp; ssp = ssp->ss_next)
8600                         stp->sd_sigflags |= ssp->ss_events;
8601         }
8602         mutex_exit(&stp->sd_lock);
8603 }

8605 /*
8606  * Return B_TRUE if there is data in the message, B_FALSE otherwise.
8607  */
8608 static boolean_t
8609 msghasdata(mblk_t *bp)
8610 {
8611         for (; bp; bp = bp->b_cont)
8612                 if (bp->b_datap->db_type == M_DATA) {
8613                         ASSERT(bp->b_wptr >= bp->b_rptr);
8614                         if (bp->b_wptr > bp->b_rptr)
8615                                 return (B_TRUE);
8616                 }
8617         return (B_FALSE);
8618 }
```

```
**********************************************************
    5581 Tue Jan 14 16:17:27 2014
new/usr/src/uts/common/sys/ptms.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright (c) 1988, 2010, Oracle and/or its affiliates. All rights reserved.
 */
/*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
/*        All Rights Reserved   */


#ifndef _SYS_PTMS_H
#define _SYS_PTMS_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

#endif /* ! codereview */
#ifdef __cplusplus
extern "C" {
#endif

#ifdef _KERNEL

/*
 * Structures and definitions supporting the pseudo terminal
 * drivers. This structure is private and should not be used by any
 * applications.
 */
struct pt_ttys {
        queue_t *ptm_rdq;       /* master's read queue pointer */
        queue_t *pts_rdq;       /* slave's read queue pointer */
        mblk_t  *pt_nullmsg;    /* 0-bytes message block for pts close */
        pid_t    pt_pid;        /* process id (for debugging) */
        minor_t  pt_minor;      /* Minor number of this pty */
        int      pt_refcnt;     /* reference count for ptm_rdq/pts_rdq uses */
        ushort_t pt_state;      /* state of master/slave pair */
        kcondvar_t pt_cv;       /* condition variable for exclusive access */
        kmutex_t pt_lock;       /* Per-element lock */
        zoneid_t pt_zoneid;     /* Zone membership for this pty */
        uid_t    pt_ruid;       /* Real owner of pty */
        gid_t    pt_rgid;       /* Real group owner of pty */
};

/*
 * pt_state values
```



```
 */
#define PTLOCK          0x01    /* master/slave pair is locked */
#define PTMOPEN         0x02    /* master side is open */
#define PTSOPEN         0x04    /* slave side is open */
#define PTSTTY          0x08    /* slave side is tty */

/*
 * Multi-threading primitives.
 * Values of pt_refcnt: -1 if a writer is accessing the struct
 *                      0  if no one is reading or writing
 *                      > 0 equals to the number of readers accessing the struct
 */
#define PT_ENTER_READ(p) {                      \
        mutex_enter(&(p)->pt_lock);             \
        while ((p)->pt_refcnt < 0)              \
                cv_wait(&((p)->pt_cv), &(p)->pt_lock);  \
        (p)->pt_refcnt++;                       \
        mutex_exit(&(p)->pt_lock);              \
}

#define PT_ENTER_WRITE(p) {                     \
        mutex_enter(&(p)->pt_lock);             \
        while ((p)->pt_refcnt != 0)             \
                cv_wait(&((p)->pt_cv), &(p)->pt_lock);  \
        (p)->pt_refcnt = -1;                    \
        mutex_exit(&(p)->pt_lock);              \
}

#define PT_EXIT_READ(p) {                       \
        mutex_enter(&(p)->pt_lock);             \
        ASSERT((p)->pt_refcnt > 0);             \
        if ((--((p)->pt_refcnt)) == 0)          \
                cv_broadcast(&(p)->pt_cv);      \
        mutex_exit(&(p)->pt_lock);              \
}

#define PT_EXIT_WRITE(p) {                      \
        mutex_enter(&(p)->pt_lock);             \
        ASSERT((p)->pt_refcnt == -1);           \
        (p)->pt_refcnt = 0;                     \
        cv_broadcast(&(p)->pt_cv);              \
        mutex_exit(&(p)->pt_lock);              \
}

/*
 * ptms_lock and pt_cnt are defined in ptms_conf.c
 */
extern kmutex_t         ptms_lock;
extern dev_info_t       *pts_dip;       /* private copy of devinfo ptr */

extern void ptms_init(void);
extern struct pt_ttys *pt_ttys_alloc(void);
extern void ptms_close(struct pt_ttys *, uint_t);
extern struct pt_ttys *ptms_minor2ptty(minor_t);
extern int ptms_attach_slave(void);
extern int ptms_minor_valid(minor_t ptmin, uid_t *uid, gid_t *gid);
extern int ptms_minor_exists(minor_t ptmin);
extern void ptms_set_owner(minor_t ptmin, uid_t uid, gid_t gid);
extern major_t ptms_slave_attached(void);

#ifdef DEBUG
extern void ptms_log(char *, uint_t);
extern void ptms_logp(char *, uintptr_t);
#define DDBG(a, b) ptms_log(a, b)
#define DDBGP(a, b) ptms_logp(a, b)
#else
```

```
#define DDBG(a, b)
#define DDBGP(a, b)
#endif

typedef struct __ptmptsopencb_arg *ptmptsopencb_arg_t;
typedef struct ptmptsopencb {
        boolean_t               (*ppocb_func)(ptmptsopencb_arg_t);
        ptmptsopencb_arg_t      ppocb_arg;
} ptmptsopencb_t;

#endif /* ! codereview */
#endif /* _KERNEL */

typedef struct pt_own {
        uid_t   pto_ruid;
        gid_t   pto_rgid;
} pt_own_t;

/*
 * ioctl commands
 *
 *  ISPTM: Determines whether the file descriptor is that of an open master
 *         device. Return code of zero indicates that the file descriptor
 *         represents master device.
 *
 * UNLKPT: Unlocks the master and slave devices.  It returns 0 on success. On
 *         failure, the errno is set to EINVAL indicating that the master
 *         device is not open.
 *
 *  ZONEPT: Sets the zoneid of the pair of master and slave devices.  It
 *          returns 0 upon success.  Used to force a pty 'into' a zone upon
 *          zone entry.
 *
 * PT_OWNER: Sets uid and gid for slave device.  It returns 0 on success.
 *
 */
#define ISPTM           (('P'<<8)|1)    /* query for master */
#define UNLKPT          (('P'<<8)|2)    /* unlock master/slave pair */
#define PTSSTTY         (('P'<<8)|3)    /* set tty flag */
#define ZONEPT          (('P'<<8)|4)    /* set zone of master/slave pair */
#define OWNERPT         (('P'<<8)|5)    /* set owner/group for slave device */

#ifdef _KERNEL
/*
 * kernel ioctl commands
 *
 * PTMPTSOPENCB: Returns a callback function pointer and opaque argument.
 *            The return value of the callback function when it's invoked
 *            with the opaque argument passed to it will indicate if the
 *            pts slave device is currently open.
 */
#define PTMPTSOPENCB    (('P'<<8)|6)    /* check if the slave is open */

#endif /* _KERNEL */
#endif /* ! codereview */

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_PTMS_H */
```

```
**********************************************************
   16793 Tue Jan 14 16:17:27 2014
new/usr/src/uts/common/sys/termios.h
Bring back LX zones.
**********************************************************
______unchanged_portion_omitted_

  94 /*
  95  * POSIX termios functions
  96  * These functions get mapped into ioctls.
  97  */

  99 #ifndef _KERNEL

 101 #if defined(__STDC__)

 103 extern speed_t cfgetospeed(const struct termios *);
 104 extern int cfsetospeed(struct termios *, speed_t);
 105 extern speed_t cfgetispeed(const struct termios *);
 106 extern int cfsetispeed(struct termios *, speed_t);
 107 extern int tcgetattr(int, struct termios *);
 108 extern int tcsetattr(int, int, const struct termios *);
 109 extern int tcsendbreak(int, int);
 110 extern int tcdrain(int);
 111 extern int tcflush(int, int);
 112 extern int tcflow(int, int);

 114 #else

 116 extern speed_t cfgetospeed();
 117 extern int cfsetospeed();
 118 extern speed_t cfgetispeed();
 119 extern int cfsetispeed();
 120 extern int tcgetattr();
 121 extern int tcsetattr();
 122 extern int tcsendbreak();
 123 extern int tcdrain();
 124 extern int tcflush();
 125 extern int tcflow();

 127 #endif /* __STDC__ */

 129 #if !defined(__XOPEN_OR_POSIX) || defined(_XPG4_2) || defined(__EXTENSIONS__)

 131 #if defined(__STDC__)
 132 extern pid_t tcgetsid(int);
 133 #else
 134 extern pid_t tcgetsid();
 135 #endif /* __STDC__ */

 137 #endif /* !defined(__XOPEN_OR_POSIX) || defined(_XPG4_2) ... */

 139 #endif

 141 /* control characters */
 142 #define VINTR   0
 143 #define VQUIT   1
 144 #define VERASE  2
 145 #define VKILL   3
 146 #define VEOF    4
 147 #define VEOL    5
 148 #if !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__)
 149 #define VEOL2   6
 150 #endif /* !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__) */
 151 #define VMIN    4
 152 #define VTIME   5
```



```
 153 #if !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__)
 154 #define VSWTCH  7
 155 #endif /* !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__) */
 156 #define VSTART          8
 157 #define VSTOP           9
 158 #define VSUSP           10
 159 #if !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__)
 160 #define VDSUSP          11
 161 #define VREPRINT        12
 162 #define VDISCARD        13
 163 #define VWERASE         14
 164 #define VLNEXT          15
 165 /* 16 thru 19 reserved for future use */

 167 /*
 168  * control characters form Xenix termio.h
 169  */
 170 #define VCEOF   NCC             /* RESERVED true EOF char (V7 compatability) */
 171 #define VCEOL   (NCC + 1)       /* RESERVED true EOL char */

 173 #define CNUL    0
 174 #define CDEL    0177

 176 /* S5 default control chars */
 177 /* CINTR, CERASE and CKILL modified to SunOS traditional values */
 178 #define CESC    '\\'
 179 #define CINTR   CTRL('c')
 180 #define CQUIT   034     /* FS, cntl | */
 181 #define CERASE  0177    /* DEL */
 182 #define CKILL   CTRL('u')
 183 #define CEOT    04
 184 #define CEOL    0
 185 #define CEOL2   0
 186 #define CEOF    04      /* cntl d */
 187 #define CSTART  021     /* cntl q */
 188 #define CSTOP   023     /* cntl s */
 189 #define CSWTCH  032     /* cntl z */
 190 #define CNSWTCH 0
 191 #define CSUSP   CTRL('z')
 192 #define CDSUSP  CTRL('y')
 193 #define CRPRNT  CTRL('r')
 194 #define CFLUSH  CTRL('o')
 195 #define CWERASE CTRL('w')
 196 #define CLNEXT  CTRL('v')
 197 #endif /* !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__) */


 200 /* input modes */
 201 #define IGNBRK  0000001
 202 #define BRKINT  0000002
 203 #define IGNPAR  0000004
 204 #define PARMRK  0000010
 205 #define INPCK   0000020
 206 #define ISTRIP  0000040
 207 #define INLCR   0000100
 208 #define IGNCR   0000200
 209 #define ICRNL   0000400
 210 #if !defined(_POSIX_C_SOURCE) || \
 211         (defined(_XOPEN_SOURCE) && !defined(_XPG6)) || \
 212         defined(__EXTENSIONS__)
 213 #define IUCLC   0001000
 214 #endif /* !defined(_POSIX_C_SOURCE) || defined(_XOPEN_SOURCE)... */
 215 #define IXON    0002000
 216 #if !defined(_POSIX_C_SOURCE) || defined(_XOPEN_SOURCE) || \
 217         defined(__EXTENSIONS__)
 218 #define IXANY   0004000
```

```
219 #endif /* !defined(_POSIX_C_SOURCE) || defined(_XOPEN_SOURCE)... */
220 #define IXOFF   0010000
221 #if !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__)
222 #define IMAXBEL 0020000
223 #define DOSMODE 0100000  /* for 386 compatibility */
224 #endif /* !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__) */

226 /* output modes */
227 #define OPOST   0000001
228 #if !defined(_POSIX_C_SOURCE) || defined(_XOPEN_SOURCE) || \
229         defined(__EXTENSIONS__)
230 #if !defined(_XPG6) || defined(__EXTENSIONS__)
231 #define OLCUC   0000002
232 #endif
233 #define ONLCR   0000004
234 #define OCRNL   0000010
235 #define ONOCR   0000020
236 #define ONLRET  0000040
237 #define OFILL   0000100
238 #define OFDEL   0000200
239 #define NLDLY   0000400
240 #define NL0     0
241 #define NL1     0000400
242 #define CRDLY   0003000
243 #define CR0     0
244 #define CR1     0001000
245 #define CR2     0002000
246 #define CR3     0003000
247 #define TABDLY  0014000
248 #define TAB0    0
249 #define TAB1    0004000
250 #define TAB2    0010000
251 #define TAB3    0014000
252 #endif /* !defined(_POSIX_C_SOURCE) || defined(_XOPEN_SOURCE)... */
253 #if !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__)
254 #define XTABS   0014000
255 #endif /* !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__) */
256 #if !defined(_POSIX_C_SOURCE) || defined(_XOPEN_SOURCE) || \
257         defined(__EXTENSIONS__)
258 #define BSDLY   0020000
259 #define BS0     0
260 #define BS1     0020000
261 #define VTDLY   0040000
262 #define VT0     0
263 #define VT1     0040000
264 #define FFDLY   0100000
265 #define FF0     0
266 #define FF1     0100000
267 #endif /* !defined(_POSIX_C_SOURCE) || defined(_XOPEN_SOURCE)... */
268 #if !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__)
269 #define PAGEOUT 0200000
270 #define WRAP    0400000

272 /* control modes */
273 #define CBAUD   0000017
274 #endif /* !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__) */
275 #define CSIZE   0000060
276 #define CS5     0
277 #define CS6     0000020
278 #define CS7     0000040
279 #define CS8     0000060
280 #define CSTOPB  0000100
281 #define CREAD   0000200
282 #define PARENB  0000400
283 #define PARODD  0001000
284 #define HUPCL   0002000
```



```
285 #define CLOCAL  0004000
286 #if !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__)
287 #define RCV1EN  0010000
288 #define XMT1EN  0020000
289 #define LOBLK   0040000
290 #define XCLUDE  0100000         /* *V7* exclusive use coming fron XENIX */
291 #define CRTSXOFF 010000000000
292 #define CRTSCTS 020000000000
293 #define CIBAUD  03600000
294 #define PAREXT  04000000
295 #define CBAUDEXT 010000000
296 #define CIBAUDEXT 020000000

298 /*
299  * 4.4BSD hardware flow control flags
300  */
301 #define CRTS_IFLOW 010000000000
302 #define CCTS_OFLOW 020000000000

304 #endif /* !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__) */

306 /* line discipline 0 modes */
307 #define ISIG    0000001
308 #define ICANON  0000002
309 #if !defined(_POSIX_C_SOURCE) || \
310         (defined(_XOPEN_SOURCE) && !defined(_XPG6)) || \
311         defined(__EXTENSIONS__)
312 #define XCASE   0000004
313 #endif /* !defined(_POSIX_C_SOURCE) || defined(_XOPEN_SOURCE)... */
314 #define ECHO    0000010
315 #define ECHOE   0000020
316 #define ECHOK   0000040
317 #define ECHONL  0000100
318 #define NOFLSH  0000200
319 #define TOSTOP  0000400
320 #if !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__)
321 #define ECHOCTL 0001000
322 #define ECHOPRT 0002000
323 #define ECHOKE  0004000
324 #define DEFECHO 0010000
325 #define FLUSHO  0020000
326 #define PENDIN  0040000
327 #endif /* !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__) */

329 #define IEXTEN  0100000  /* POSIX flag - enable POSIX extensions */
330 #define _TIOC   ('T'<<8)

332 #if !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__)

334 #define TIOC    _TIOC

336 #define TCGETA  (_TIOC|1)
337 #define TCSETA  (_TIOC|2)
338 #define TCSETAW (_TIOC|3)
339 #define TCSETAF (_TIOC|4)
340 #define TCSBRK  (_TIOC|5)
341 #define TCXONC  (_TIOC|6)
342 #define TCFLSH  (_TIOC|7)

344 /* Slots reserved for 386/XENIX compatibility - keyboard control */

346 #define TIOCKBON        (_TIOC|8)
347 #define TIOCKBOF        (_TIOC|9)
348 #define KBENABLED       (_TIOC|10)

350 #ifndef IOCTYPE
```

```
351 #define IOCTYPE 0xff00
352 #endif

354 #define TCDSET  (_TIOC|32)
355 #define RTS_TOG (_TIOC|33)      /* 386 - "RTS" toggle define 8A1 protocol */

357 #define TIOCGWINSZ (_TIOC|104)
358 #define TIOCSWINSZ (_TIOC|103)

360 /*
361  * Softcarrier ioctls
362  */
363 #define TIOCGSOFTCAR (_TIOC|105)
364 #define TIOCSSOFTCAR (_TIOC|106)


367 /* termios ioctls */

369 #define TCGETS          (_TIOC|13)
370 #define TCSETS          (_TIOC|14)
371 #endif /* !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__) */
372 #define TCSANOW         (_TIOC|14) /* same as TCSETS */
373 #if !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__)
374 #define TCSETSW         (_TIOC|15)
375 #endif /* !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__) */
376 #define TCSADRAIN       (_TIOC|15) /* same as TCSETSW */
377 #if !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__)
378 #define TCSETSF         (_TIOC|16)

380  /*
381  * linux terminal ioctls we need to be aware of
382  */
383 #define TIOCSETLD       (_TIOC|123)     /* set line discipline parms */
384 #define TIOCGETLD       (_TIOC|124)     /* get line discipline parms */

386 /*
387  * The VMIN and VTIME and solaris overlap with VEOF and VEOL - This is
388  * perfectly legal except, linux expects them to be separate. So we keep
389  * them separately.
390  */
391 struct lx_cc {
392         unsigned char veof;     /* veof value */
393         unsigned char veol;     /* veol value */
394         unsigned char vmin;     /* vmin value */
395         unsigned char vtime;    /* vtime value */
396 };

398 #endif /* ! codereview */
399 /*
400  * NTP PPS ioctls
401  */
402 #define TIOCGPPS        (_TIOC|125)
403 #define TIOCSPPS        (_TIOC|126)
404 #define TIOCGPPSEV      (_TIOC|127)

406 /* Argument filled in by TIOCGPPSEV */
407 struct ppsclockev {
408         struct timeval tv;
409         uint_t serial;
410 };

412 #if defined(_SYSCALL32)
413 struct ppsclockev32 {
414         struct timeval32 tv;
415         uint32_t serial;
416 };
```



```
417 #endif /* _SYSCALL32 */

419 #endif /* !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__) */

421 #define TCSAFLUSH       (_TIOC|16) /* same as TCSETSF */

423 /* termios option flags */

425 #define TCIFLUSH        0  /* flush data received but not read */
426 #define TCOFLUSH        1  /* flush data written but not transmitted */
427 #define TCIOFLUSH       2  /* flush both data both input and output queues */

429 #define TCOOFF          0  /* suspend output */
430 #define TCOON           1  /* restart suspended output */
431 #define TCIOFF          2  /* suspend input */
432 #define TCION           3  /* restart suspended input */

434 /* TIOC ioctls for BSD, ptys, job control and modem control */

436 #if !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__)
437 #define tIOC    ('t'<<8)
438 #endif /* !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__) */

440 /* Slots for 386/XENIX compatibility */
441 /* BSD includes these ioctls in ttold.h */

443 #ifndef _SYS_TTOLD_H

445 #if !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__)
446 #define TIOCGETD        (tIOC|0)
447 #define TIOCSETD        (tIOC|1)
448 #define TIOCHPCL        (tIOC|2)
449 #define TIOCGETP        (tIOC|8)
450 #define TIOCSETP        (tIOC|9)
451 #define TIOCSETN        (tIOC|10)
452 #define TIOCEXCL        (tIOC|13)
453 #define TIOCNXCL        (tIOC|14)
454 #define TIOCFLUSH       (tIOC|16)
455 #define TIOCSETC        (tIOC|17)
456 #define TIOCGETC        (tIOC|18)
457 /*
458  * BSD ioctls that are not the same as XENIX are included here.
459  * There are also some relevant ioctls from SUN/BSD sys/ttycom.h
460  * BSD pty ioctls like TIOCPKT are not supported in SVR4.
461  */

463 #define TIOCLBIS        (tIOC|127)      /* bis local mode bits */
464 #define TIOCLBIC        (tIOC|126)      /* bic local mode bits */
465 #define TIOCLSET        (tIOC|125)      /* set entire local mode word */
466 #define TIOCLGET        (tIOC|124)      /* get local modes */
467 #define TIOCSBRK        (tIOC|123)      /* set break bit */
468 #define TIOCCBRK        (tIOC|122)      /* clear break bit */
469 #define TIOCSDTR        (tIOC|121)      /* set data terminal ready */
470 #define TIOCCDTR        (tIOC|120)      /* clear data terminal ready */
471 #define TIOCSLTC        (tIOC|117)      /* set local special chars */
472 #define TIOCGLTC        (tIOC|116)      /* get local special chars */
473 #define TIOCOUTQ        (tIOC|115)      /* driver output queue size */
474 #define TIOCNOTTY       (tIOC|113)      /* void tty association */
475 #define TIOCSCTTY       (tIOC|132)      /* get a ctty */
476 #define TIOCSTOP        (tIOC|111)      /* stop output, like ^S */
477 #define TIOCSTART       (tIOC|110)      /* start output, like ^Q */
478 #define TIOCSILOOP      (tIOC|109)      /* private to Sun; do not use */
479 #define TIOCCILOOP      (tIOC|108)      /* private to Sun; do not use */

481 #endif /* !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__) */
```

```
#endif /* end _SYS_TTOLD_H */

/* POSIX job control ioctls */

#if !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__)
#define TIOCGPGRP       (tIOC|20)       /* get pgrp of tty */
#define TIOCSPGRP       (tIOC|21)       /* set pgrp of tty */
#define TIOCGSID        (tIOC|22)       /* get session id on ctty */

/* Miscellaneous */
#define TIOCSTI         (tIOC|23)       /* simulate terminal input */

/* Modem control */
#define TIOCMSET        (tIOC|26)       /* set all modem bits */
#define TIOCMBIS        (tIOC|27)       /* bis modem bits */
#define TIOCMBIC        (tIOC|28)       /* bic modem bits */
#define TIOCMGET        (tIOC|29)       /* get all modem bits */
#define         TIOCM_LE        0001            /* line enable */
#define         TIOCM_DTR       0002            /* data terminal ready */
#define         TIOCM_RTS       0004            /* request to send */
#define         TIOCM_ST        0010            /* secondary transmit */
#define         TIOCM_SR        0020            /* secondary receive */
#define         TIOCM_CTS       0040            /* clear to send */
#define         TIOCM_CAR       0100            /* carrier detect */
#define         TIOCM_CD        TIOCM_CAR
#define         TIOCM_RNG       0200            /* ring */
#define         TIOCM_RI        TIOCM_RNG
#define         TIOCM_DSR       0400            /* data set ready */

/* pseudo-tty */

#define TIOCREMOTE      (tIOC|30)       /* remote input editing */
#define TIOCSIGNAL      (tIOC|31)       /* pty: send signal to slave */


/* Some more 386 xenix stuff */

#define LDIOC   ('D'<<8)

#define LDOPEN  (LDIOC|0)
#define LDCLOSE (LDIOC|1)
#define LDCHG   (LDIOC|2)
#define LDGETT  (LDIOC|8)
#define LDSETT  (LDIOC|9)

/* Slots for 386 compatibility */

#define LDSMAP  (LDIOC|110)
#define LDGMAP  (LDIOC|111)
#define LDNMAP  (LDIOC|112)
#define LDEMAP  (LDIOC|113)
#define LDDMAP  (LDIOC|114)

/*
 * These are retained for 386/XENIX compatibility.
 */

#define DIOC            ('d'<<8)
#define DIOCGETP        (DIOC|8)                /* V7 */
#define DIOCSETP        (DIOC|9)                /* V7 */

/*
 * Returns a non-zero value if there
 * are characters in the input queue.
 *
 * XXX - somebody is confused here.  V7 had no such "ioctl", although XENIX may
```



```
 * have added it; 4BSD had FIONREAD, which returned the number of characters
 * waiting, and was supposed to work on all descriptors (i.e., every driver
 * should make a stab at implementing it).
 */
#define FIORDCHK        (('f'<<8)|3)            /* V7 */
#endif /* !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__) */

/*
 * Speeds
 */
#define B0      0
#define B50     1
#define B75     2
#define B110    3
#define B134    4
#define B150    5
#define B200    6
#define B300    7
#define B600    8
#define B1200   9
#define B1800   10
#define B2400   11
#define B4800   12
#define B9600   13
#define B19200  14
#define B38400  15
#define B57600  16
#define B76800  17
#define B115200 18
#define B153600 19
#define B230400 20
#define B307200 21
#define B460800 22
#define B921600 23

#ifndef _SYS_TTOLD_H

#if !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__)
/* Windowing structure to support JWINSIZE/TIOCSWINSZ/TIOCGWINSZ */
struct winsize {
        unsigned short ws_row;  /* rows, in characters */
        unsigned short ws_col;  /* columns, in character */
        unsigned short ws_xpixel;       /* horizontal size, pixels */
        unsigned short ws_ypixel;       /* vertical size, pixels */
};
#endif /* !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__) */

#endif /* end _SYS_TTOLD_H */

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_TERMIOS_H */
```

```
**********************************************************
   21021 Tue Jan 14 16:17:27 2014
new/usr/src/uts/i86pc/ml/syscall_asm.s
Bring back LX zones.
**********************************************************
_____unchanged_portion_omitted_
```

```
634 #endif  /* __lint */

636 #if defined(__lint)
637 /*
638  * System call via an int80.  This entry point is only used by the Linux
639  * application environment.  Unlike the sysenter path, there is no default
640  * action to take if no callback is registered for this process.
641  */
642 void
643 sys_int80()
644 {}

646 #else   /* __lint */

648         ENTRY_NP(brand_sys_int80)
649         BRAND_CALLBACK(BRAND_CB_INT80)

651         ALTENTRY(sys_int80)
652         /*
653          * We hit an int80, but this process isn't of a brand with an int80
654          * handler.  Bad process!  Make it look as if the INT failed.
655          * Modify %eip to point before the INT, push the expected error
656          * code and fake a GP fault.
657          *
658          */
659         subl    $2, (%esp)      /* int insn 2-bytes */
660         pushl   $_CONST(_MUL(T_INT80, GATE_DESC_SIZE) + 2)
661         jmp     gptrap                  / GP fault
662         SET_SIZE(sys_int80)
663         SET_SIZE(brand_sys_int80)
664 #endif /* ! codereview */

666 /*
667  * Declare a uintptr_t which covers the entire pc range of syscall
668  * handlers for the stack walkers that need this.
669  */
670         .align  CPTRSIZE
671         .globl  _allsyscalls_size
672         .type   _allsyscalls_size, @object
673 _allsyscalls_size:
674         .NWORD  . - _allsyscalls
675         SET_SIZE(_allsyscalls_size)

677 #endif  /* __lint */

679 /*
680  * These are the thread context handlers for lwps using sysenter/sysexit.
681  */

683 #if defined(__lint)

685 /*ARGSUSED*/
686 void
687 sep_save(void *ksp)
688 {}

690 /*ARGSUSED*/
691 void
692 sep_restore(void *ksp)
693 {}
```



```
695 #else   /* __lint */

697         /*
698          * setting this value to zero as we switch away causes the
699          * stack-pointer-on-sysenter to be NULL, ensuring that we
700          * don't silently corrupt another (preempted) thread stack
701          * when running an lwp that (somehow) didn't get sep_restore'd
702          */
703         ENTRY_NP(sep_save)
704         xorl    %edx, %edx
705         xorl    %eax, %eax
706         movl    $MSR_INTC_SEP_ESP, %ecx
707         wrmsr
708         ret
709         SET_SIZE(sep_save)

711         /*
712          * Update the kernel stack pointer as we resume onto this cpu.
713          */
714         ENTRY_NP(sep_restore)
715         movl    4(%esp), %eax                   /* per-lwp kernel sp */
716         xorl    %edx, %edx
717         movl    $MSR_INTC_SEP_ESP, %ecx
718         wrmsr
719         ret
720         SET_SIZE(sep_restore)

722 #endif  /* __lint */

724 /*
725  * Call syscall().  Called from trap() on watchpoint at lcall 0,7
726  */

728 #if defined(__lint)

730 void
731 watch_syscall(void)
732 {}

734 #else   /* __lint */

736         ENTRY_NP(watch_syscall)
737         CLI(%eax)
738         movl    %gs:CPU_THREAD, %ebx
739         movl    T_STACK(%ebx), %esp             / switch to the thread stack
740         movl    REGOFF_EAX(%esp), %eax          / recover original syscall#
741         jmp     _watch_do_syscall
742         SET_SIZE(watch_syscall)

744 #endif  /* __lint */
```

```
**********************************************************
   38450 Tue Jan 14 16:17:28 2014
new/usr/src/uts/i86pc/ml/syscall_asm_amd64.s
Bring back LX zones.
**********************************************************
______unchanged_portion_omitted_

1161 #endif  /* __lint */
1162
1163 #if defined(__lint)
1164 /*
1165  * System call via an int80.  This entry point is only used by the Linux
1166  * application environment.  Unlike the other entry points, there is no
1167  * default action to take if no callback is registered for this process.
1168  */
1169 void
1170 sys_int80()
1171 {}

1173 #else   /* __lint */

1175         ENTRY_NP(brand_sys_int80)
1176         SWAPGS                          /* kernel gsbase */
1177         XPV_TRAP_POP
1178         BRAND_CALLBACK(BRAND_CB_INT80, BRAND_URET_FROM_INTR_STACK())
1179         SWAPGS                          /* user gsbase */
1180         jmp     nopop_int80

1182         ENTRY_NP(sys_int80)
1183         /*
1184          * We hit an int80, but this process isn't of a brand with an int80
1185          * handler.  Bad process!  Make it look as if the INT failed.
1186          * Modify %rip to point before the INT, push the expected error
1187          * code and fake a GP fault. Note on 64-bit hypervisor we need
1188          * to undo the XPV_TRAP_POP and push rcx and r11 back on the stack
1189          * because gptrap will pop them again with its own XPV_TRAP_POP.
1190          */
1191         XPV_TRAP_POP
1192 nopop_int80:
1193         subq    $2, (%rsp)      /* int insn 2-bytes */
1194         pushq   $_CONST(_MUL(T_INT80, GATE_DESC_SIZE) + 2)
1195 #if defined(__xpv)
1196         push    %r11
1197         push    %rcx
1198 #endif
1199         jmp     gptrap                  / GP fault
1200         SET_SIZE(sys_int80)
1201         SET_SIZE(brand_sys_int80)
1202 #endif  /* __lint */

1204 #endif /* ! codereview */

1206 /*
1207  * This is the destination of the "int $T_SYSCALLINT" interrupt gate, used by
1208  * the generic i386 libc to do system calls. We do a small amount of setup
1209  * before jumping into the existing sys_syscall32 path.
1210  */
1211 #if defined(__lint)

1213 /*ARGSUSED*/
1214 void
1215 sys_syscall_int()
1216 {}

1218 #else   /* __lint */
```



```
1220         ENTRY_NP(brand_sys_syscall_int)
1221         SWAPGS                          /* kernel gsbase */
1222         XPV_TRAP_POP
1223         BRAND_CALLBACK(BRAND_CB_INT91, BRAND_URET_FROM_INTR_STACK())
1224         jmp     nopop_syscall_int

1226         ALTENTRY(sys_syscall_int)
1227         SWAPGS                          /* kernel gsbase */
1228         XPV_TRAP_POP

1230 nopop_syscall_int:
1231         movq    %gs:CPU_THREAD, %r15
1232         movq    T_STACK(%r15), %rsp
1233         movl    %eax, %eax
1234         /*
1235          * Set t_post_sys on this thread to force ourselves out via the slow
1236          * path. It might be possible at some later date to optimize this out
1237          * and use a faster return mechanism.
1238          */
1239         movb    $1, T_POST_SYS(%r15)
1240         CLEAN_CS
1241         jmp     _syscall32_save
1242         /*
1243          * There should be no instructions between this label and SWAPGS/IRET
1244          * or we could end up breaking branded zone support. See the usage of
1245          * this label in lx_brand_int80_callback and sn1_brand_int91_callback
1246          * for examples.
1247          */
1248         ALTENTRY(sys_sysint_swapgs_iret)
1249         SWAPGS                          /* user gsbase */
1250         IRET
1251         /*NOTREACHED*/
1252         SET_SIZE(sys_sysint_swapgs_iret)
1253         SET_SIZE(sys_syscall_int)
1254         SET_SIZE(brand_sys_syscall_int)

1256 #endif  /* __lint */
1257
1258 /*
1259  * Legacy 32-bit applications and old libc implementations do lcalls;
1260  * we should never get here because the LDT entry containing the syscall
1261  * segment descriptor has the "segment present" bit cleared, which means
1262  * we end up processing those system calls in trap() via a not-present trap.
1263  *
1264  * We do it this way because a call gate unhelpfully does -nothing- to the
1265  * interrupt flag bit, so an interrupt can run us just after the lcall
1266  * completes, but just before the swapgs takes effect.   Thus the INTR_PUSH and
1267  * INTR_POP paths would have to be slightly more complex to dance around
1268  * this problem, and end up depending explicitly on the first
1269  * instruction of this handler being either swapgs or cli.
1270  */

1272 #if defined(__lint)

1274 /*ARGSUSED*/
1275 void
1276 sys_lcall32()
1277 {}

1279 #else   /* __lint */

1281         ENTRY_NP(sys_lcall32)
1282         SWAPGS                          /* kernel gsbase */
1283         pushq   $0
1284         pushq   %rbp
1285         movq    %rsp, %rbp
```

```
1286         leaq    __lcall_panic_str(%rip), %rdi
1287         xorl    %eax, %eax
1288         call    panic
1289         SET_SIZE(sys_lcall32)

1291 __lcall_panic_str:
1292         .string "sys_lcall32: shouldn't be here!"

1294 /*
1295  * Declare a uintptr_t which covers the entire pc range of syscall
1296  * handlers for the stack walkers that need this.
1297  */
1298         .align  CPTRSIZE
1299         .globl  _allsyscalls_size
1300         .type   _allsyscalls_size, @object
1301 _allsyscalls_size:
1302         .NWORD  . - _allsyscalls
1303         SET_SIZE(_allsyscalls_size)

1305 #endif  /* __lint */

1307 /*
1308  * These are the thread context handlers for lwps using sysenter/sysexit.
1309  */

1311 #if defined(__lint)

1313 /*ARGSUSED*/
1314 void
1315 sep_save(void *ksp)
1316 {}

1318 /*ARGSUSED*/
1319 void
1320 sep_restore(void *ksp)
1321 {}

1323 #else   /* __lint */

1325         /*
1326          * setting this value to zero as we switch away causes the
1327          * stack-pointer-on-sysenter to be NULL, ensuring that we
1328          * don't silently corrupt another (preempted) thread stack
1329          * when running an lwp that (somehow) didn't get sep_restore'd
1330          */
1331         ENTRY_NP(sep_save)
1332         xorl    %edx, %edx
1333         xorl    %eax, %eax
1334         movl    $MSR_INTC_SEP_ESP, %ecx
1335         wrmsr
1336         ret
1337         SET_SIZE(sep_save)

1339         /*
1340          * Update the kernel stack pointer as we resume onto this cpu.
1341          */
1342         ENTRY_NP(sep_restore)
1343         movq    %rdi, %rdx
1344         shrq    $32, %rdx
1345         movl    %edi, %eax
1346         movl    $MSR_INTC_SEP_ESP, %ecx
1347         wrmsr
1348         ret
1349         SET_SIZE(sep_restore)

1351 #endif  /* __lint */
```

```
**********************************************************
   27569 Tue Jan 14 16:17:28 2014
new/usr/src/uts/i86pc/sys/apic.h
Bring back LX zones.
**********************************************************
______unchanged_portion_omitted_

 294 /*
 295  * intr_type definitions
 296  */
 297 #define IO_INTR_INT     0x00
 298 #define IO_INTR_NMI     0x01
 299 #define IO_INTR_SMI     0x02
 300 #define IO_INTR_EXTINT  0x03

 302 /*
 303  * destination APIC ID
 304  */
 305 #define INTR_ALL_APIC           0xff


 308 /* local vector table                                                */
 309 #define AV_MASK         0x10000

 311 /* interrupt command register 32-63                                  */
 312 #define AV_TOALL        0x7fffffff
 313 #define AV_HIGH_ORDER   0x40000000
 314 #define AV_IM_OFF       0x40000000

 316 /* interrupt command register 0-31                                   */
 317 #define AV_DELIV_MODE   0x700

 319 #define AV_FIXED        0x000
 320 #define AV_LOPRI        0x100
 321 #define AV_SMI          0x200
 322 #define AV_REMOTE       0x300
 323 #define AV_NMI          0x400
 324 #define AV_RESET        0x500
 325 #define AV_STARTUP      0x600
 326 #define AV_EXTINT       0x700

 328 #define AV_PDEST        0x000
 329 #define AV_LDEST        0x800

 331 /* IO & Local APIC Bit Definitions */
 332 #define RDT_VECTOR(x)   ((uchar_t)((x) & 0xFF))
 333 #define AV_PENDING      0x1000
 334 #define AV_ACTIVE_LOW   0x2000          /* only for integrated APIC */
 335 #define AV_REMOTE_IRR   0x4000          /* IOAPIC RDT-specific */
 336 #define AV_LEVEL        0x8000
 337 #define AV_DEASSERT     AV_LEVEL
 338 #define AV_ASSERT       0xc000

 340 #define AV_READ_PENDING 0x10000
 341 #define AV_REMOTE_STATUS        0x20000 /* 1 = valid, 0 = invalid */

 343 #define AV_SH_SELF              0x40000 /* Short hand for self */
 344 #define AV_SH_ALL_INCSELF       0x80000 /* All processors */
 345 #define AV_SH_ALL_EXCSELF       0xc0000 /* All excluding self */
 346 /* spurious interrupt vector register                                */
 347 #define AV_UNIT_ENABLE  0x100

 349 #define APIC_MAXVAL     0xffffffffUL
 350 #define APIC_TIME_MIN   0x5000
 351 #define APIC_TIME_COUNT 0x4000
```



```
 353 /*
 354  * Range of the low byte value in apic_tick before starting calibration
 355  */
 356 #define APIC_LB_MIN     0x60
 357 #define APIC_LB_MAX     0xe0

 359 #define APIC_MAX_VECTOR         255
 360 #define APIC_RESV_VECT          0x00
 361 #define APIC_RESV_IRQ           0xfe
 362 #define APIC_BASE_VECT          0x20    /* This will come in as interrupt 0 */
 363 #define APIC_AVAIL_VECTOR       (APIC_MAX_VECTOR+1-APIC_BASE_VECT)
 364 #define APIC_VECTOR_PER_IPL     0x10    /* # of vectors before PRI changes */
 365 #define APIC_VECTOR(ipl)        (apic_ipltopri[ipl] | APIC_RESV_VECT)
 366 #define APIC_VECTOR_MASK        0x0f
 367 #define APIC_HI_PRI_VECTS       2       /* vects reserved for hi pri reqs */
 368 #define APIC_IPL_MASK           0xf0
 369 #define APIC_IPL_SHIFT          4       /* >> to get ipl part of vector */
 370 #define APIC_FIRST_FREE_IRQ     0x10
 371 #define APIC_MAX_ISA_IRQ        15
 372 #define APIC_IPL0               0x0f    /* let IDLE_IPL be the lowest */
 373 #define APIC_IDLE_IPL           0x00

 375 #define APIC_MASK_ALL           0xf0    /* Mask all interrupts */

 377 /* spurious interrupt vector                                         */
 378 #define APIC_SPUR_INTR          0xFF

 380 /* special or reserve vectors */
 381 #define APIC_CHECK_RESERVE_VECTORS(v) \
 382         (((v) == T_FASTTRAP) || ((v) == APIC_SPUR_INTR) || \
 383         ((v) == T_SYSCALLINT) || ((v) == T_DTRACE_RET) || ((v) == 0x80))
 383         ((v) == T_SYSCALLINT) || ((v) == T_DTRACE_RET))

 385 /* cmos shutdown code for BIOS                                       */
 386 #define BIOS_SHUTDOWN           0x0a

 388 /* define the entry types for BIOS information tables as defined in PC+MP */
 389 #define APIC_CPU_ENTRY          0
 390 #define APIC_BUS_ENTRY          1
 391 #define APIC_IO_ENTRY           2
 392 #define APIC_IO_INTR_ENTRY      3
 393 #define APIC_LOCAL_INTR_ENTRY   4
 394 #define APIC_MPTBL_ADDR         (639 * 1024)
 395 /*
 396  * The MP Floating Point structure could be in 1st 1KB of EBDA or last KB
 397  * of system base memory or in ROM between 0xF0000 and 0xFFFFF
 398  */
 399 #define MPFPS_RAM_WIN_LEN       1024
 400 #define MPFPS_ROM_WIN_START     (uint32_t)0xf0000
 401 #define MPFPS_ROM_WIN_LEN       0x10000

 403 #define EISA_LEVEL_CNTL         0x4D0

 405 /* definitions for apic_irq_table */
 406 #define FREE_INDEX              (short)-1       /* empty slot */
 407 #define RESERVE_INDEX           (short)-2       /* ipi, softintr, clkintr */
 408 #define ACPI_INDEX              (short)-3       /* ACPI */
 409 #define MSI_INDEX               (short)-4       /* MSI */
 410 #define MSIX_INDEX              (short)-5       /* MSI-X */
 411 #define DEFAULT_INDEX           (short)0x7FFF
 412         /* biggest positive no. to avoid conflict with actual index */

 414 #define APIC_IS_MSI_OR_MSIX_INDEX(index) \
 415         ((index) == MSI_INDEX || (index) == MSIX_INDEX)

 417 /*
```

```
 418  * definitions for MSI Address
 419  */
 420 #define MSI_ADDR_HDR            APIC_LOCAL_ADDR
 421 #define MSI_ADDR_DEST_SHIFT     12      /* Destination CPU's apic id */
 422 #define MSI_ADDR_RH_FIXED       0x0     /* Redirection Hint Fixed */
 423 #define MSI_ADDR_RH_LOPRI       0x1     /* Redirection Hint Lowest priority */
 424 #define MSI_ADDR_RH_SHIFT       3
 425 #define MSI_ADDR_DM_PHYSICAL    0x0     /* Physical Destination Mode */
 426 #define MSI_ADDR_DM_LOGICAL     0x1     /* Logical Destination Mode */
 427 #define MSI_ADDR_DM_SHIFT       2

 429 /*
 430  * TM is either edge or level.
 431  */
 432 #define TRIGGER_MODE_EDGE               0x0     /* edge sensitive */
 433 #define TRIGGER_MODE_LEVEL              0x1     /* level sensitive */

 435 /*
 436  * definitions for MSI Data
 437  */
 438 #define MSI_DATA_DELIVERY_FIXED         0x0     /* Fixed delivery */
 439 #define MSI_DATA_DELIVERY_LOPRI         0x1     /* Lowest priority delivery */
 440 #define MSI_DATA_DELIVERY_SMI           0x2
 441 #define MSI_DATA_DELIVERY_NMI           0x4
 442 #define MSI_DATA_DELIVERY_INIT          0x5
 443 #define MSI_DATA_DELIVERY_EXTINT        0x7
 444 #define MSI_DATA_DELIVERY_SHIFT         8
 445 #define MSI_DATA_TM_EDGE                TRIGGER_MODE_EDGE
 446 #define MSI_DATA_TM_LEVEL               TRIGGER_MODE_LEVEL
 447 #define MSI_DATA_TM_SHIFT               15
 448 #define MSI_DATA_LEVEL_DEASSERT         0x0
 449 #define MSI_DATA_LEVEL_ASSERT           0x1     /* Edge always assert */
 450 #define MSI_DATA_LEVEL_SHIFT            14

 452 /*
 453  * use to define each irq setup by the apic
 454  */
 455 typedef struct  apic_irq {
 456         short   airq_mps_intr_index;    /* index into mps interrupt entries */
 457                                         /*  table */
 458         uchar_t airq_intin_no;
 459         uchar_t airq_ioapicindex;
 460         dev_info_t      *airq_dip; /* device corresponding to this interrupt */
 461         /*
 462          * IRQ could be shared (in H/W) in which case dip & major will be
 463          * for the one that was last added at this level. We cannot keep a
 464          * linked list as delspl does not tell us which device has just
 465          * been unloaded. For most servers where we are worried about
 466          * performance, interrupt should not be shared & should not be
 467          * a problem. This does not cause any correctness issue - dip is
 468          * used only as an optimisation to avoid going thru all the tables
 469          * in translate IRQ (which is always called twice due to brokenness
 470          * in the way IPLs are determined for devices). major is used only
 471          * to bind interrupts corresponding to the same device on the same
 472          * CPU. Not finding major will just cause it to be potentially bound
 473          * to another CPU.
 474          */
 475         major_t airq_major;     /* major number corresponding to the device */
 476         ushort_t airq_rdt_entry;        /* level, polarity & trig mode */
 477         uint32_t airq_cpu;              /* target CPU, non-reserved IRQ only */
 478         uint32_t airq_temp_cpu;   /* non-reserved IRQ only, for disable_intr */
 479         uchar_t airq_vector;            /* Vector chosen for this irq */
 480         uchar_t airq_share;             /* number of interrupts at this irq */
 481         uchar_t airq_share_id;          /* id to identify source from irqno */
 482         uchar_t airq_ipl;               /* The ipl at which this is handled */
 483         iflag_t airq_iflag;             /* interrupt flag */
```



```
 484         uchar_t airq_origirq;           /* original irq passed in */
 485         uint_t  airq_busy;              /* How frequently did clock find */
 486                                         /* us in this */
 487         struct apic_irq *airq_next;     /* chain of intpts sharing a vector */
 488         void            *airq_intrmap_private; /* intr remap private data */
 489 } apic_irq_t;
_____unchanged_portion_omitted_
```

```
**********************************************************
    4530 Tue Jan 14 16:17:29 2014
new/usr/src/uts/intel/Makefile
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
# uts/intel/Makefile
#
# Copyright (c) 1999, 2010, Oracle and/or its affiliates. All rights reserved.
#
#       This makefile drives the production of all implementation architecture
#       independent modules for Intel processors.

UTSBASE = ..

include Makefile.intel

LINT_KMODS_X1   = $(LINT_KMODS:nsmb=)
LINT_KMODS_X2   = $(LINT_KMODS_X1:smbfs=)
LINT_KMODLIBS   = $(LINT_KMODS_X2:e1000g=)
LINT_LIBS       = $(LINT_LIB) $(GEN_LINT_LIB) \
                  $(LINT_KMODLIBS:%=$(LINT_LIB_DIR)/llib-l%.ln)

#
# dprov is delivered in the SUNWcryptoint package.
#
DRV_KMODS       += dprov

#
#
def             :=      TARGET= def
def.prereq      :=      TARGET= def
all             :=      TARGET= all
all.prereq      :=      TARGET= all
install         :=      TARGET= install
install.prereq  :=      TARGET= all
clean           :=      TARGET= clean
clobber         :=      TARGET= clobber
lint            :=      TARGET= lint
lint.prereq     :=      TARGET= lint
modlintlib      :=      TARGET= modlintlib
modlist         :=      TARGET= modlist
modlist         :=      NO_STATE= -K $$MODSTATE$$$$
clean.lint      :=      TARGET= clean.lint
check           :=      TARGET= check
install_h       :=      TARGET= install_h
install_h.prereq        :=      TARGET= install_h
```



```
.KEEP_STATE:

.PARALLEL:      $(PARALLEL_KMODS) $(XMODS) config $(LINT_DEPS)

def all install clean clobber modlist: genassym $(KMODS) $(XMODS) config
def all install clean clobber modlist: $(KMODS) $(XMODS) config

clobber: clobber.targ

#
# Privilege constants
#
# NOTE: The rules for generating priv_const.c file are shared between all
# processor architectures and and should be kept in sync. If they are changed in
# this file make sure that sparc rules are updated as well.
#
PRIVS_C = $(SRC)/uts/common/os/priv_const.c

$(PRIVS_C): $(PRIVS_AWK) $(PRIVS_DEF)
        $(NAWK) -f $(PRIVS_AWK) < $(PRIVS_DEF) cfile=$@

CLOBBERFILES += $(PRIVS_C)

#
# Prerequisites
#
# The uts/Makefile defines build parallelism for x86 platforms such that i86pc,
# i86xpv and intel are all built in parallel. This requires building certain
# parts before the parallel build can start. The uts/Makefile appends the
# '.prereq' string to the original target and executes this Makefile to build
# any prerequisites needed before the full parallel build can start. After that
# make continues with normal targets.
#
# Any build prerequisites for x86 builds should be described here.
#
# genassym is used to build intel/dtrace and genunix, so it should be built
# first.
#
# priv_const.c is required to build genunix.
#
# genunix is used by everyone to ctf-merge with. Genunix is CTF-merged with
#   intel/ip so as a side effect this dependency builds intel/ip as part of the
#   prerequisites.
#
# intel/dtrace depends on i86pc/genassym, so we need to build both
# i86pc/genassym and intel/genassym.
#
all.prereq install.prereq def.prereq: genassym genunix FRC
all.prereq install.prereq def.prereq: genunix FRC
        @cd ../i86pc/genassym; pwd; $(MAKE) $(@:%.prereq=%)

#
# i86pc lint libraries should be built first
#
lint.prereq: FRC
        @cd ../i86pc; pwd; $(MAKE) $(NO_STATE) lint

#
# Nothing to do for any other prerequisite targets.
#
%.prereq:

genunix: $(PRIVS_C)

modlintlib clean.lint: $(LINT_KMODS) $(XMODS)
```

```
127 genassym $(KMODS) $(SUBDIRS) config:    FRC
127 $(KMODS) $(SUBDIRS) config:     FRC
128         @cd $@; pwd; $(MAKE) $(NO_STATE) $(TARGET)

130 $(XMODS):       FRC
131         @if [ -f $@/Makefile  ]; then \
132                 cd $@; pwd; $(MAKE) $(NO_STATE) $(TARGET); \
133         else \
134                 true; \
135         fi

137 install_h check:        FRC
138         @cd sys; pwd; $(MAKE) $(TARGET)
139         @cd asm; pwd; $(MAKE) $(TARGET)
140         @cd ia32/sys; pwd; $(MAKE) $(TARGET)
141         @cd amd64/sys; pwd; $(MAKE) $(TARGET)

143 #
144 # Work-around to disable acpica global crosscheck lint warnings
145 #
146 LGREP.intel =   grep -v 'intel/io/acpica'

148 #
149 #       Full kernel lint target.
150 #
151 LINT_TARGET     = globallint

153 # workaround for multiply defined errors
154 globallint := LINTFLAGS += -erroff=E_NAME_MULTIPLY_DEF2

156 globallint:
157         @pwd
158         @-$(ECHO) "\nFULL KERNEL: global crosschecks:"
159         @-$(LINT) $(LINTFLAGS) $(LINT_LIBS) 2>&1 | $(LGREP.intel) | $(LGREP.2)

161 lint:   modlintlib .WAIT $(LINT_DEPS)

163 include ../Makefile.targ
```

```
**********************************************************
    7186 Tue Jan 14 16:17:29 2014
new/usr/src/uts/intel/Makefile.files
Bring back LX zones.
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright (c) 1999, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright (c) 2012, Joyent, Inc. All rights reserved.
#

#
#       This Makefile defines all file modules and build rules for the
# directory uts/intel and its children. These are the source files which
# are specific to x86 processor architectures.
#

#
#       Core (unix) objects
#
CORE_OBJS +=            \
        arch_kdi.o      \
        copy.o          \
        copy_subr.o     \
        cpc_subr.o      \
        ddi_arch.o      \
        ddi_i86.o       \
        ddi_i86_asm.o   \
        desctbls.o      \
        desctbls_asm.o  \
        exception.o     \
        float.o         \
        fmsmb.o         \
        fpu.o           \
        i86_subr.o      \
        lock_prim.o     \
        ovbcopy.o       \
        polled_io.o     \
        sseblk.o        \
        sundep.o        \
        swtch.o         \
        sysi86.o

#
# 64-bit multiply/divide compiler helper routines
# used only for ia32
```



```
#

SPECIAL_OBJS_32 +=      \
        muldiv.o

#
#       Generic-unix Module
#
GENUNIX_OBJS +=         \
        archdep.o       \
        getcontext.o    \
        install_utrap.o \
        lwp_private.o   \
        prom_enter.o    \
        prom_exit.o     \
        prom_panic.o    \
        sendsig.o       \
        syscall.o


#
#       PROM Routines
#
GENUNIX_OBJS +=         \
        prom_env.o      \
        prom_emul.o     \
        prom_getchar.o  \
        prom_init.o     \
        prom_node.o     \
        prom_printf.o   \
        prom_prop.o     \
        prom_putchar.o  \
        prom_reboot.o   \
        prom_version.o

#
#       file system modules
#
CORE_OBJS +=            \
        prmachdep.o

LX_PROC_OBJS += \
        lx_prsubr.o     \
        lx_prvfsops.o   \
        lx_prvnops.o

LX_AUTOFS_OBJS +=       \
        lx_autofs.o

#endif /* ! codereview */
#
#       ZFS file system module
#
ZFS_OBJS +=             \
        spa_boot.o

#
#       Decompression code
#
CORE_OBJS += decompress.o

#
#       Microcode utilities
#
CORE_OBJS += ucode_utils.o
```

```
#
#       Driver modules
#
AGPGART_OBJS += agpgart.o agp_kstat.o
AGPTARGET_OBJS += agptarget.o
AMD64GART_OBJS += amd64_gart.o
ARCMSR_OBJS += arcmsr.o
ATA_OBJS += $(GHD_OBJS) ata_blacklist.o ata_common.o ata_disk.o \
        ata_dma.o atapi.o atapi_fsm.o ata_debug.o \
        sil3xxx.o
BSCBUS_OBJS += bscbus.o
BSCV_OBJS += bscv.o
CMDK_OBJS += cmdk.o
CMLB_OBJS += cmlb.o
CPUNEX_OBJS += cpunex.o
DADK_OBJS += dadk.o
DCOPY_OBJS += dcopy.o
DNET_OBJS += dnet.o dnet_mii.o
FD_OBJS += fd.o
GDA_OBJS += gda.o
GHD_OBJS += ghd.o ghd_debug.o ghd_dma.o ghd_queue.o ghd_scsa.o \
        ghd_scsi.o ghd_timer.o ghd_waitq.o ghd_gcmd.o
I915_OBJS += i915_dma.o i915_drv.o i915_irq.o i915_mem.o \
        i915_gem.o i915_gem_debug.o i915_gem_tiling.o
NSKERN_OBJS += nsc_asm.o
PCICFG_OBJS += pcicfg.o
PCI_PCINEXUS_OBJS += pci_pci.o
PCIEB_OBJS += pcieb_x86.o
PIT_BEEP_OBJS += pit_beep.o
POWER_OBJS += power.o
PCI_AUTOCONFIG_OBJS += pci_autoconfig.o pci_boot.o pcie_nvidia.o \
                       pci_memlist.o pci_resource.o
RADEON_OBJS += r300_cmdbuf.o radeon_cp.o radeon_drv.o \
                       radeon_state.o radeon_irq.o radeon_mem.o
SD_OBJS += sd.o sd_xbuf.o

HECI_OBJS +=            \
        heci_init.o     \
        heci_intr.o     \
        heci_interface.o \
        io_heci.o       \
        heci_main.o

STRATEGY_OBJS += strategy.o
UCODE_OBJS += ucode_drv.o
VGATEXT_OBJS += vgatext.o vgasubr.o

#
#       Kernel linker
#
KRTLD_OBJS +=           \
        bootrd.o        \
        ufsops.o        \
        hsfs.o          \
        doreloc.o       \
        kobj_boot.o     \
        kobj_convrelstr.o \
        kobj_crt.o      \
        kobj_isa.o      \
        kobj_reloc.o

#
#       misc. modules
#
ACPICA_OBJS     += dbcmds.o dbdisply.o \
                   dbexec.o dbfileio.o dbhistry.o dbinput.o dbstats.o \
```



```
                   dbutils.o dbxface.o evevent.o evgpe.o evgpeblk.o \
                   evmisc.o evregion.o evrgnini.o evsci.o evxface.o \
                   evxfevnt.o evxfregn.o hwacpi.o hwgpe.o hwregs.o \
                   hwsleep.o hwtimer.o dsfield.o dsinit.o dsmethod.o \
                   dsmthdat.o dsobject.o dsopcode.o dsutils.o dswexec.o \
                   dswload.o dswscope.o dswstate.o exconfig.o exconvrt.o \
                   excreate.o exdump.o exfield.o exfldio.o exmisc.o \
                   exmutex.o exnames.o exoparg1.o exoparg2.o exoparg3.o \
                   exoparg6.o exprep.o exregion.o exresnte.o exresolv.o \
                   exresop.o exstore.o exstoren.o exstorob.o exsystem.o \
                   exutils.o psargs.o psopcode.o psparse.o psscope.o \
                   pstree.o psutils.o pswalk.o psxface.o nsaccess.o \
                   nsalloc.o nsdump.o nsdumpdv.o nseval.o nsinit.o \
                   nsload.o nsnames.o nsobject.o nsparse.o nssearch.o \
                   nsutils.o nswalk.o nsxfeval.o nsxfname.o nsxfobj.o \
                   rsaddr.o rscalc.o rscreate.o rsdump.o \
                   rsinfo.o rsio.o rsirq.o rslist.o rsmemory.o rsmisc.o \
                   rsutils.o rsxface.o tbfadt.o tbfind.o tbinstal.o  \
                   tbutils.o tbxface.o tbxfroot.o \
                   utalloc.o utclib.o utcopy.o utdebug.o utdelete.o \
                   uteval.o utglobal.o utinit.o utmath.o utmisc.o \
                   utobject.o utresrc.o utxface.o acpica.o acpi_enum.o \
                   master_ops.o osl.o osl_ml.o acpica_ec.o utcache.o \
                   utmutex.o utstate.o dmbuffer.o dmnames.o dmobject.o \
                   dmopcode.o dmresrc.o dmresrcl.o dmresrcs.o dmutils.o \
                   dmwalk.o psloop.o nspredef.o hwxface.o hwvalid.o \
                   utlock.o utids.o nsrepair.o nsrepair2.o \
                   dbmethod.o dbnames.o dsargs.o dscontrol.o dswload2.o \
                   evglock.o evgpeinit.o evgpeutil.o evxfgpe.o exdebug.o \
                   hwpci.o utdecode.o utosi.o utxferror.o


AGP_OBJS += agpmaster.o
FBT_OBJS += fbt.o
SDT_OBJS += sdt.o

#
#       AMD8111 NIC driver module
#
AMD8111S_OBJS += amd8111s_main.o amd8111s_hw.o

#
#       Pentium Performance Counter BackEnd module
#
P123_PCBE_OBJS = p123_pcbe.o

#
#       Pentium 4 Performance Counter BackEnd module
#
P4_PCBE_OBJS = p4_pcbe.o

#
#       AMD Opteron/Athlon64 Performance Counter BackEnd module
#
OPTERON_PCBE_OBJS = opteron_pcbe.o

#
#       Intel Core Architecture Performance Counter BackEnd module
#
CORE_PCBE_OBJS = core_pcbe.o

#
#       AMR module
#
AMR_OBJS = amr.o
```

```
260 #
261 #       IPMI module
262 IPMI_OBJS +=    ipmi_main.o ipmi.o ipmi_kcs.o

264 #
265 #       IOMMULIB module
266 #
267 IOMMULIB_OBJS = iommulib.o

269 #
270 #       Brand modules
271 #
272 SN1_BRAND_OBJS  =       sn1_brand.o sn1_brand_asm.o
273 S10_BRAND_OBJS  =       s10_brand.o s10_brand_asm.o
274 LX_BRAND_OBJS  =                \
275         lx_brand.o              \
276         lx_brand_asm.o          \
277         lx_brk.o                \
278         lx_clone.o              \
279         lx_futex.o              \
280         lx_getpid.o             \
281         lx_id.o                 \
282         lx_kill.o               \
283         lx_misc.o               \
284         lx_modify_ldt.o         \
285         lx_pid.o                \
286         lx_sched.o              \
287         lx_signum.o             \
288         lx_syscall.o            \
289         lx_sysinfo.o            \
290         lx_thread_area.o
291 #endif /* ! codereview */

293 #
294 #       special files
295 #
296 MODSTUB_OBJ +=          \
297         modstubs.o

299 BOOTDEV_OBJS +=         \
300         bootdev.o

302 INC_PATH        += -I$(UTSBASE)/intel


305 CPR_INTEL_OBJS +=       cpr_intel.o

307 #
308 # AMD family 0xf memory controller module
309 #
310 include $(SRC)/common/mc/mc-amd/Makefile.mcamd
311 MCAMD_OBJS      += \
312         $(MCAMD_CMN_OBJS) \
313         mcamd_drv.o \
314         mcamd_dimmcfg.o \
315         mcamd_subr.o \
316         mcamd_pcicfg.o

318 #
319 # Intel Nehalem memory controller module
320 #
321 INTEL_NHM_OBJS += \
322         nhm_init.o \
323         mem_addr.o \
324         intel_nhmdrv.o \
325         nhm_pci_cfg.o \
```



```
326         dimm_topo.o \
327         intel_nhm.o

329 #
330 # Intel 5000/5100/5400/7300 chipset memory controller hub (MCH) module
331 #
332 INTEL_NB5000_OBJS += \
333         intel_nb5000.o \
334         intel_nbdrv.o \
335         dimm_addr.o \
336         nb_pci_cfg.o \
337         nb5000_init.o
```

```
**********************************************************
   16141 Tue Jan 14 16:17:29 2014
new/usr/src/uts/intel/Makefile.intel
test
**********************************************************
```

```
   1 # CDDL HEADER START
   2 #
   3 # The contents of this file are subject to the terms of the
   4 # Common Development and Distribution License (the "License").
   5 # You may not use this file except in compliance with the License.
   6 #
   7 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   8 # or http://www.opensolaris.org/os/licensing.
   9 # See the License for the specific language governing permissions
  10 # and limitations under the License.
  11 #
  12 # When distributing Covered Code, include this CDDL HEADER in each
  13 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  14 # If applicable, add the following below this CDDL HEADER, with the
  15 # fields enclosed by brackets "[]" replaced with your own identifying
  16 # information: Portions Copyright [yyyy] [name of copyright owner]
  17 #
  18 # CDDL HEADER END
  19 #

  21 # Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
  22 # Copyright (c) 2012 Nexenta Systems, Inc. All rights reserved.
  23 # Copyright (c) 2013 Andrew Stormont.  All rights reserved.

  25 #
  26 #       This makefile contains the common definitions for all intel
  27 #       implementation architecture independent modules.
  28 #

  30 #
  31 #       Machine type (implementation architecture):
  32 #
  33 PLATFORM         = i86pc

  35 #
  36 #       Everybody needs to know how to build modstubs.o and to locate unix.o.
  37 #       Note that unix.o must currently be selected from among the possible
  38 #       "implementation architectures". Note further, that unix.o is only
  39 #       used as an optional error check for undefines so (theoretically)
  40 #       any "implementation architectures" could be used. We choose i86pc
  41 #       because it is the reference port.
  42 #
  43 UNIX_DIR         = $(UTSBASE)/i86pc/unix
  44 GENLIB_DIR       = $(UTSBASE)/intel/genunix
  45 GENASSYM_DIR     = $(UTSBASE)/intel/genassym
  46 #endif /* ! codereview */
  47 IPDRV_DIR        = $(UTSBASE)/intel/ip
  48 MODSTUBS_DIR     = $(UNIX_DIR)
  49 DSF_DIR          = $(UTSBASE)/$(PLATFORM)/genassym
  50 LINTS_DIR        = $(OBJS_DIR)
  51 LINT_LIB_DIR     = $(UTSBASE)/intel/lint-libs/$(OBJS_DIR)

  53 UNIX_O           = $(UNIX_DIR)/$(OBJS_DIR)/unix.o
  54 GENLIB           = $(GENLIB_DIR)/$(OBJS_DIR)/libgenunix.so
  55 MODSTUBS_O       = $(MODSTUBS_DIR)/$(OBJS_DIR)/modstubs.o
  56 LINT_LIB         = $(UTSBASE)/i86pc/lint-libs/$(OBJS_DIR)/llib-lunix.ln
  57 GEN_LINT_LIB     = $(UTSBASE)/intel/lint-libs/$(OBJS_DIR)/llib-lgenunix.ln

  59 #
  60 #       Include the makefiles which define build rule templates, the
  61 #       collection of files per module, and a few specific flags. Note
```



```
  62 #       that order is significant, just as with an include path. The
  63 #       first build rule template which matches the files name will be
  64 #       used. By including these in order from most machine dependent
  65 #       to most machine independent, we allow a machine dependent file
  66 #       to be used in preference over a machine independent version
  67 #       (Such as a machine specific optimization, which preserves the
  68 #       interfaces.)
  69 #
  70 include $(UTSBASE)/intel/Makefile.files
  71 include $(UTSBASE)/common/Makefile.files

  73 #
  74 # ----- TRANSITIONAL SECTION --------------------------------------------------
  75 #

  77 #
  78 #       Not everything which *should* be a module is a module yet. The
  79 #       following is a list of such objects which are currently part of
  80 #       genunix but which might someday become kmods.  This must be
  81 #       defined before we include Makefile.uts, or else genunix's build
  82 #       won't be as parallel as we might like.
  83 #
  84 NOT_YET_KMODS    = $(OLDPTY_OBJS) $(PTY_OBJS) $(VCONS_CONF_OBJS) $(MOD_OBJS)

  86 #
  87 # ----- END OF TRANSITIONAL SECTION -------------------------------------------
  88 #
  89 #       Include machine independent rules. Note that this does not imply
  90 #       that the resulting module from rules in Makefile.uts is machine
  91 #       independent. Only that the build rules are machine independent.
  92 #
  93 include $(UTSBASE)/Makefile.uts

  95 #
  96 #       The following must be defined for all implementations:
  97 #
  98 MODSTUBS               = $(UTSBASE)/intel/ia32/ml/modstubs.s

 100 #
 101 #       Define supported builds
 102 #
 103 DEF_BUILDS             = $(DEF_BUILDS64) $(DEF_BUILDS32)
 104 ALL_BUILDS             = $(ALL_BUILDS64) $(ALL_BUILDS32)

 106 #
 107 #       x86 or amd64 inline templates
 108 #
 109 INLINES_32             = $(UTSBASE)/intel/ia32/ml/ia32.il
 110 INLINES_64             = $(UTSBASE)/intel/amd64/ml/amd64.il
 111 INLINES                += $(INLINES_$(CLASS))

 113 #
 114 #       kernel-specific optimizations; override default in Makefile.master
 115 #

 117 CFLAGS_XARCH_32        = $(i386_CFLAGS)
 118 CFLAGS_XARCH_64        = $(amd64_CFLAGS)
 119 CFLAGS_XARCH           = $(CFLAGS_XARCH_$(CLASS))

 121 COPTFLAG_32            = $(COPTFLAG)
 122 COPTFLAG_64            = $(COPTFLAG64)
 123 COPTIMIZE              = $(COPTFLAG_$(CLASS))

 125 CFLAGS                 = $(CFLAGS_XARCH)
 126 CFLAGS                 += $(COPTIMIZE)
 127 CFLAGS                 += $(INLINES) -D_ASM_INLINES
```

```
128 CFLAGS                  += $(CCMODE)
129 CFLAGS                  += $(SPACEFLAG)
130 CFLAGS                  += $(CCUNBOUND)
131 CFLAGS                  += $(CFLAGS_uts)
132 CFLAGS                  += -xstrconst

134 ASFLAGS_XARCH_32        = $(i386_ASFLAGS)
135 ASFLAGS_XARCH_64        = $(amd64_ASFLAGS)
136 ASFLAGS_XARCH           = $(ASFLAGS_XARCH_$(CLASS))

138 ASFLAGS                 += $(ASFLAGS_XARCH)

140 #
141 #       Define the base directory for installation.
142 #
143 BASE_INS_DIR    = $(ROOT)

145 #
146 #       Debugging level
147 #
148 #       Special knowledge of which special debugging options affect which
149 #       file is used to optimize the build if these flags are changed.
150 #
151 DEBUG_DEFS_OBJ32        =
152 DEBUG_DEFS_DBG32        = -DDEBUG
153 DEBUG_DEFS_OBJ64        =
154 DEBUG_DEFS_DBG64        = -DDEBUG
155 DEBUG_DEFS              = $(DEBUG_DEFS_$(BUILD_TYPE))

157 DEBUG_COND_OBJ32        = $(POUND_SIGN)
158 DEBUG_COND_DBG32        =
159 DEBUG_COND_OBJ64        = $(POUND_SIGN)
160 DEBUG_COND_DBG64        =
161 IF_DEBUG_OBJ            = $(DEBUG_COND_$(BUILD_TYPE))$(OBJS_DIR)/

163 $(IF_DEBUG_OBJ)syscall.o        :=      DEBUG_DEFS      += -DSYSCALLTRACE
164 $(IF_DEBUG_OBJ)clock.o          :=      DEBUG_DEFS      += -DKSLICE=1

166 #
167 #       Collect the preprocessor definitions to be associated with *all*
168 #       files.
169 #
170 ALL_DEFS         = $(DEBUG_DEFS) $(OPTION_DEFS)

172 #
173 #       The kernels modules which are "implementation architecture"
174 #       specific for this machine are enumerated below. Note that most
175 #       of these modules must exist (in one form or another) for each
176 #       architecture.
177 #
178 #       Common Drivers (usually pseudo drivers) (/kernel/drv)
179 #       DRV_KMODS are built both 32-bit and 64-bit
180 #       DRV_KMODS_32 are built only 32-bit
181 #       DRV_KMODS_64 are built only 64-bit
182 #
183 DRV_KMODS       += aac
184 DRV_KMODS       += aggr
185 DRV_KMODS       += ahci
186 DRV_KMODS       += amd64_gart
187 DRV_KMODS       += amr
188 DRV_KMODS       += agpgart
189 DRV_KMODS       += srn
190 DRV_KMODS       += agptarget
191 DRV_KMODS       += arn
192 DRV_KMODS       += arp
193 DRV_KMODS       += asy
```



```
194 DRV_KMODS       += ata
195 DRV_KMODS       += ath
196 DRV_KMODS       += atu
197 DRV_KMODS       += audio
198 DRV_KMODS       += audio1575
199 DRV_KMODS       += audio810
200 DRV_KMODS       += audiocmi
201 DRV_KMODS       += audiocmihd
202 DRV_KMODS       += audioemu10k
203 DRV_KMODS       += audioens
204 DRV_KMODS       += audiohd
205 DRV_KMODS       += audioixp
206 DRV_KMODS       += audiols
207 DRV_KMODS       += audiop16x
208 DRV_KMODS       += audiopci
209 DRV_KMODS       += audiosolo
210 DRV_KMODS       += audiots
211 DRV_KMODS       += audiovia823x
212 DRV_KMODS_32    += audiovia97
213 DRV_KMODS       += bl
214 DRV_KMODS       += blkdev
215 DRV_KMODS       += bge
216 DRV_KMODS       += bofi
217 DRV_KMODS       += bpf
218 DRV_KMODS       += bridge
219 DRV_KMODS       += bscbus
220 DRV_KMODS       += bscv
221 DRV_KMODS       += chxge
222 DRV_KMODS       += cxgbe
223 DRV_KMODS       += ntxn
224 DRV_KMODS       += myri10ge
225 DRV_KMODS       += clone
226 DRV_KMODS       += cmdk
227 DRV_KMODS       += cn
228 DRV_KMODS       += conskbd
229 DRV_KMODS       += consms
230 DRV_KMODS       += cpqary3
231 DRV_KMODS       += cpuid
232 DRV_KMODS       += cpunex
233 DRV_KMODS       += crypto
234 DRV_KMODS       += cryptoadm
235 DRV_KMODS       += dca
236 DRV_KMODS       += devinfo
237 DRV_KMODS       += dld
238 DRV_KMODS       += dlpistub
239 DRV_KMODS_32    += dnet
240 DRV_KMODS       += dump
241 DRV_KMODS       += ecpp
242 DRV_KMODS       += emlxs
243 DRV_KMODS       += fd
244 DRV_KMODS       += fdc
245 DRV_KMODS       += fm
246 DRV_KMODS       += fssnap
247 DRV_KMODS       += hxge
248 DRV_KMODS       += i8042
249 DRV_KMODS       += i915
250 DRV_KMODS       += icmp
251 DRV_KMODS       += icmp6
252 DRV_KMODS       += intel_nb5000
253 DRV_KMODS       += intel_nhm
254 DRV_KMODS       += ip
255 DRV_KMODS       += ip6
256 DRV_KMODS       += ipd
257 DRV_KMODS       += ipf
258 DRV_KMODS       += ipnet
259 DRV_KMODS       += ippctl
```

```
260 DRV_KMODS       += ipsecah
261 DRV_KMODS       += ipsecesp
262 DRV_KMODS       += ipw
263 DRV_KMODS       += iwh
264 DRV_KMODS       += iwi
265 DRV_KMODS       += iwk
266 DRV_KMODS       += iwp
267 DRV_KMODS       += iwscn
268 DRV_KMODS       += kb8042
269 DRV_KMODS       += keysock
270 DRV_KMODS       += kssl
271 DRV_KMODS       += kstat
272 DRV_KMODS       += ksyms
273 DRV_KMODS       += kmdb
274 DRV_KMODS       += llc1
275 DRV_KMODS       += lofi
276 DRV_KMODS       += log
277 DRV_KMODS       += logindmux
278 DRV_KMODS       += mega_sas
279 DRV_KMODS       += mc-amd
280 DRV_KMODS       += mm
281 DRV_KMODS       += mouse8042
282 DRV_KMODS       += mpt_sas
283 DRV_KMODS       += mr_sas
284 DRV_KMODS       += mwl
285 DRV_KMODS       += nca
286 DRV_KMODS       += nsmb
287 DRV_KMODS       += nulldriver
288 DRV_KMODS       += nv_sata
289 DRV_KMODS       += nxge
290 DRV_KMODS       += oce
291 DRV_KMODS       += openeepr
292 DRV_KMODS       += pci_pci
293 DRV_KMODS       += pcic
294 DRV_KMODS       += pcieb
295 DRV_KMODS       += physmem
296 DRV_KMODS       += pit_beep
297 DRV_KMODS       += pm
298 DRV_KMODS       += poll
299 DRV_KMODS       += pool
300 DRV_KMODS       += power
301 DRV_KMODS       += pseudo
302 DRV_KMODS       += ptc
303 DRV_KMODS       += ptm
304 DRV_KMODS       += pts
305 DRV_KMODS       += ptsl
306 DRV_KMODS       += qlge
307 DRV_KMODS       += radeon
308 DRV_KMODS       += ral
309 DRV_KMODS       += ramdisk
310 DRV_KMODS       += random
311 DRV_KMODS       += rds
312 DRV_KMODS       += rdsv3
313 DRV_KMODS       += rpcib
314 DRV_KMODS       += rsm
315 DRV_KMODS       += rts
316 DRV_KMODS       += rtw
317 DRV_KMODS       += rum
318 DRV_KMODS       += rwd
319 DRV_KMODS       += rwn
320 DRV_KMODS       += sad
321 DRV_KMODS       += sd
322 DRV_KMODS       += sdhost
323 DRV_KMODS       += sgen
324 DRV_KMODS       += si3124
325 DRV_KMODS       += smbios
```



```
326 DRV_KMODS       += softmac
327 DRV_KMODS       += spdsock
328 DRV_KMODS       += smbsrv
329 DRV_KMODS       += smp
330 DRV_KMODS       += sppp
331 DRV_KMODS       += sppptun
332 DRV_KMODS       += srpt
333 DRV_KMODS       += st
334 DRV_KMODS       += sy
335 DRV_KMODS       += sysevent
336 DRV_KMODS       += sysmsg
337 DRV_KMODS       += tcp
338 DRV_KMODS       += tcp6
339 DRV_KMODS       += tl
340 DRV_KMODS       += tnf
341 DRV_KMODS       += tpm
342 DRV_KMODS       += trill
343 DRV_KMODS       += udp
344 DRV_KMODS       += udp6
345 DRV_KMODS       += ucode
346 DRV_KMODS       += ural
347 DRV_KMODS       += uath
348 DRV_KMODS       += urtw
349 DRV_KMODS       += vgatext
350 DRV_KMODS       += heci
351 DRV_KMODS       += vnic
352 DRV_KMODS       += vscan
353 DRV_KMODS       += wc
354 DRV_KMODS       += winlock
355 DRV_KMODS       += wpi
356 DRV_KMODS       += xge
357 DRV_KMODS       += yge
358 DRV_KMODS       += zcons
359 DRV_KMODS       += zyd
360 DRV_KMODS       += simnet
361 DRV_KMODS       += stmf
362 DRV_KMODS       += stmf_sbd
363 DRV_KMODS       += fct
364 DRV_KMODS       += fcoe
365 DRV_KMODS       += fcoet
366 DRV_KMODS       += fcoei
367 DRV_KMODS       += qlt
368 DRV_KMODS       += iscsit
369 DRV_KMODS       += pppt
370 DRV_KMODS       += ncall nsctl sdbc nskern sv
371 DRV_KMODS       += ii rdc rdcsrv rdcstub
372 DRV_KMODS       += iptun

374 #
375 # Common code drivers
376 #

378 DRV_KMODS       += afe
379 DRV_KMODS       += atge
380 DRV_KMODS       += bfe
381 DRV_KMODS       += dmfe
382 DRV_KMODS       += e1000g
383 DRV_KMODS       += efe
384 DRV_KMODS       += elxl
385 DRV_KMODS       += hme
386 DRV_KMODS       += mxfe
387 DRV_KMODS       += nge
388 DRV_KMODS       += pcn
389 DRV_KMODS       += rge
390 DRV_KMODS       += rtls
391 DRV_KMODS       += sfe
```

```
392 DRV_KMODS       += amd8111s
393 DRV_KMODS       += igb
394 DRV_KMODS       += ipmi
395 DRV_KMODS       += iprb
396 DRV_KMODS       += ixgbe
397 DRV_KMODS       += vr

399 #
400 # Virtio drivers
401 #

403 # Virtio core
404 DRV_KMODS       += virtio

406 # Virtio block driver
407 DRV_KMODS       += vioblk

409 #
410 #       DTrace and DTrace Providers
411 #
412 DRV_KMODS       += dtrace
413 DRV_KMODS       += fbt
414 DRV_KMODS       += lockstat
415 DRV_KMODS       += profile
416 DRV_KMODS       += sdt
417 DRV_KMODS       += systrace
418 DRV_KMODS       += fasttrap
419 DRV_KMODS       += dcpc

421 #
422 #       I/O framework test drivers
423 #
424 DRV_KMODS       += pshot
425 DRV_KMODS       += gen_drv
426 DRV_KMODS       += tvhci tphci tclient
427 DRV_KMODS       += emul64

429 #
430 #       Machine Specific Driver Modules (/kernel/drv):
431 #
432 DRV_KMODS       += options
433 DRV_KMODS       += scsi_vhci
434 DRV_KMODS       += pmcs
435 DRV_KMODS       += pmcs8001fw
436 DRV_KMODS       += arcmsr
437 DRV_KMODS       += fcp
438 DRV_KMODS       += fcip
439 DRV_KMODS       += fcsm
440 DRV_KMODS       += fp
441 DRV_KMODS       += qlc
442 DRV_KMODS       += iscsi

444 #
445 #       PCMCIA specific module(s)
446 #
447 DRV_KMODS       += pcs
448 MISC_KMODS      += cardbus

450 #
451 #       SCSI Enclosure Services driver
452 #
453 DRV_KMODS       += ses

455 #
456 #       USB specific modules
457 #
```



```
458 DRV_KMODS       += hid
459 DRV_KMODS       += hwarc hwahc
460 DRV_KMODS       += hubd
461 DRV_KMODS       += uhci
462 DRV_KMODS       += ehci
463 DRV_KMODS       += ohci
464 DRV_KMODS       += usb_mid
465 DRV_KMODS       += usb_ia
466 DRV_KMODS       += scsa2usb
467 DRV_KMODS       += usbprn
468 DRV_KMODS       += ugen
469 DRV_KMODS       += usbser
470 DRV_KMODS       += usbsacm
471 DRV_KMODS       += usbsksp
472 DRV_KMODS       += usbsprl
473 DRV_KMODS       += usb_ac
474 DRV_KMODS       += usb_as
475 DRV_KMODS       += usbskel
476 DRV_KMODS       += usbvc
477 DRV_KMODS       += usbftdi
478 DRV_KMODS       += wusb_df
479 DRV_KMODS       += wusb_ca
480 DRV_KMODS       += usbecm

482 #
483 #       1394 modules
484 #
485 MISC_KMODS      += s1394 sbp2
486 DRV_KMODS       += hci1394 scsa1394
487 DRV_KMODS       += av1394
488 DRV_KMODS       += dcam1394

490 #
491 #       InfiniBand pseudo drivers
492 #
493 DRV_KMODS       += ib ibp eibnx eoib rdsib sdp iser daplt hermon tavor sol_ucma
494 DRV_KMODS       += sol_umad

496 #
497 #       LVM modules
498 #
499 DRV_KMODS       += md
500 MISC_KMODS      += md_stripe md_hotspares md_mirror md_raid md_trans md_notify
501 MISC_KMODS      += md_sp

503 #
504 #       Brand modules
505 #
506 BRAND_KMODS     += sn1_brand s10_brand lx_brand
507 DRV_KMODS       += lx_systrace lx_ptm lx_audio
508 STRMOD_KMODS    += ldlinux
 45 BRAND_KMODS     += sn1_brand s10_brand

510 #
511 #       Exec Class Modules (/kernel/exec):
512 #
513 EXEC_KMODS      += elfexec intpexec shbinexec javaexec

515 #
516 #       Scheduling Class Modules (/kernel/sched):
517 #
518 SCHED_KMODS     += IA RT TS RT_DPTBL TS_DPTBL FSS FX FX_DPTBL SDC

520 #
521 #       File System Modules (/kernel/fs):
522 #
```

```
523 FS_KMODS        += autofs cachefs ctfs dcfs dev devfs fdfs fifofs hsfs lofs
524 FS_KMODS        += lx_afs lx_proc mntfs namefs nfs objfs zfs zut
 61 FS_KMODS        += mntfs namefs nfs objfs zfs zut
525 FS_KMODS        += pcfs procfs sockfs specfs tmpfs udfs ufs sharefs
526 FS_KMODS        += smbfs

528 #
529 #       Streams Modules (/kernel/strmod):
530 #
531 STRMOD_KMODS    += bufmod connld dedump ldterm pckt pfmod pipemod
532 STRMOD_KMODS    += ptem redirmod rpcmod rlmod telmod timod
533 STRMOD_KMODS    += spppasyn spppcomp
534 STRMOD_KMODS    += tirdwr ttcompat
535 STRMOD_KMODS    += usbkbm
536 STRMOD_KMODS    += usbms
537 STRMOD_KMODS    += usbwcm
538 STRMOD_KMODS    += usb_ah
539 STRMOD_KMODS    += drcompat
540 STRMOD_KMODS    += cryptmod
541 STRMOD_KMODS    += vuid2ps2
542 STRMOD_KMODS    += vuid3ps2
543 STRMOD_KMODS    += vuidm3p
544 STRMOD_KMODS    += vuidm4p
545 STRMOD_KMODS    += vuidm5p

547 #
548 #       'System' Modules (/kernel/sys):
549 #
550 SYS_KMODS       += c2audit
551 SYS_KMODS       += doorfs
552 SYS_KMODS       += exacctsys
553 SYS_KMODS       += inst_sync
554 SYS_KMODS       += kaio
555 SYS_KMODS       += msgsys
556 SYS_KMODS       += pipe
557 SYS_KMODS       += portfs
558 SYS_KMODS       += pset
559 SYS_KMODS       += semsys
560 SYS_KMODS       += shmsys
561 SYS_KMODS       += sysacct
562 SYS_KMODS       += acctctl

564 #
565 #       'Misc' Modules (/kernel/misc)
566 #       MISC_KMODS are built both 32-bit and 64-bit
567 #       MISC_KMODS_32 are built only 32-bit
568 #       MISC_KMODS_64 are built only 64-bit
569 #
570 MISC_KMODS      += ac97
571 MISC_KMODS      += acpica
572 MISC_KMODS      += agpmaster
573 MISC_KMODS      += bignum
574 MISC_KMODS      += bootdev
575 MISC_KMODS      += busra
576 MISC_KMODS      += cmlb
577 MISC_KMODS      += consconfig
578 MISC_KMODS      += ctf
579 MISC_KMODS      += dadk
580 MISC_KMODS      += dcopy
581 MISC_KMODS      += dls
582 MISC_KMODS      += drm
583 MISC_KMODS      += fssnap_if
584 MISC_KMODS      += gda
585 MISC_KMODS      += gld
586 MISC_KMODS      += hidparser
587 MISC_KMODS      += hook
```



```
588 MISC_KMODS      += hpcsvc
589 MISC_KMODS      += ibcm
590 MISC_KMODS      += ibdm
591 MISC_KMODS      += ibdma
592 MISC_KMODS      += ibmf
593 MISC_KMODS      += ibtl
594 MISC_KMODS      += idm
595 MISC_KMODS      += idmap
596 MISC_KMODS      += iommulib
597 MISC_KMODS      += ipc
598 MISC_KMODS      += kbtrans
599 MISC_KMODS      += kcf
600 MISC_KMODS      += kgssapi
601 MISC_KMODS      += kmech_dummy
602 MISC_KMODS      += kmech_krb5
603 MISC_KMODS      += ksocket
604 MISC_KMODS      += mac
605 MISC_KMODS      += mii
606 MISC_KMODS      += mwlfw
607 MISC_KMODS      += net80211
608 MISC_KMODS      += nfs_dlboot
609 MISC_KMODS      += nfssrv
610 MISC_KMODS      += neti
611 MISC_KMODS      += pci_autoconfig
612 MISC_KMODS      += pcicfg
613 MISC_KMODS      += pcihp
614 MISC_KMODS      += pcmcia
615 MISC_KMODS      += rpcsec
616 MISC_KMODS      += rpcsec_gss
617 MISC_KMODS      += rsmops
618 MISC_KMODS      += sata
619 MISC_KMODS      += scsi
620 MISC_KMODS      += sda
621 MISC_KMODS      += sol_ofs
622 MISC_KMODS      += spuni
623 MISC_KMODS      += strategy
624 MISC_KMODS      += strplumb
625 MISC_KMODS      += tem
626 MISC_KMODS      += tlimod
627 MISC_KMODS      += usba usba10 usbs49_fw
628 MISC_KMODS      += scsi_vhci_f_sym_hds
629 MISC_KMODS      += scsi_vhci_f_sym
630 MISC_KMODS      += scsi_vhci_f_tpgs
631 MISC_KMODS      += scsi_vhci_f_asym_sun
632 MISC_KMODS      += scsi_vhci_f_tape
633 MISC_KMODS      += scsi_vhci_f_tpgs_tape
634 MISC_KMODS      += fctl
635 MISC_KMODS      += emlxs_fw
636 MISC_KMODS      += qlc_fw_2200
637 MISC_KMODS      += qlc_fw_2300
638 MISC_KMODS      += qlc_fw_2400
639 MISC_KMODS      += qlc_fw_2500
640 MISC_KMODS      += qlc_fw_6322
641 MISC_KMODS      += qlc_fw_8100
642 MISC_KMODS      += hwa1480_fw
643 MISC_KMODS      += uathfw
644 MISC_KMODS      += uwba

646 MISC_KMODS      += klmmod klmops

648 #
649 #       Software Cryptographic Providers (/kernel/crypto):
650 #
651 CRYPTO_KMODS    += aes
652 CRYPTO_KMODS    += arcfour
653 CRYPTO_KMODS    += blowfish
```

```
654 CRYPTO_KMODS    += des
655 CRYPTO_KMODS    += ecc
656 CRYPTO_KMODS    += md4
657 CRYPTO_KMODS    += md5
658 CRYPTO_KMODS    += rsa
659 CRYPTO_KMODS    += sha1
660 CRYPTO_KMODS    += sha2
661 CRYPTO_KMODS    += swrand

663 #
664 #       IP Policy Modules (/kernel/ipp)
665 #
666 IPP_KMODS       += dlcosmk
667 IPP_KMODS       += flowacct
668 IPP_KMODS       += ipgpc
669 IPP_KMODS       += dscpmk
670 IPP_KMODS       += tokenmt
671 IPP_KMODS       += tswtclmt

673 #
674 #       generic-unix module (/kernel/genunix):
675 #
676 GENUNIX_KMODS   += genunix

678 #
679 #       Modules eXcluded from the product:
680 #

682 #
683 #       'Dacf' Modules (/kernel/dacf):
684 #

686 #
687 #       Performance Counter BackEnd modules (/usr/kernel/pcbe)
688 #
689 PCBE_KMODS      += p123_pcbe p4_pcbe opteron_pcbe core_pcbe

691 #
692 #       MAC-Type Plugin Modules (/kernel/mac)
693 #
694 MAC_KMODS       += mac_6to4
695 MAC_KMODS       += mac_ether
696 MAC_KMODS       += mac_ipv4
697 MAC_KMODS       += mac_ipv6
698 MAC_KMODS       += mac_wifi
699 MAC_KMODS       += mac_ib

701 #
702 # socketmod (kernel/socketmod)
703 #
704 SOCKET_KMODS    += sockpfp
705 SOCKET_KMODS    += socksctp
706 SOCKET_KMODS    += socksdp
707 SOCKET_KMODS    += sockrds
708 SOCKET_KMODS    += ksslf

710 #
711 #       kiconv modules (/kernel/kiconv):
712 #
713 KICONV_KMODS    += kiconv_emea kiconv_ja kiconv_ko kiconv_sc kiconv_tc

715 #
716 #       'Dacf' Modules (/kernel/dacf):
717 #
718 DACF_KMODS      += net_dacf
```



```
720 #
721 # Ensure that the variable member of the cpu_t (cpu_m) is defined
722 # for the lint builds so as not to cause lint errors during the
723 # global cross check.
724 #
725 LINTFLAGS       += -D_MACHDEP -I$(UTSBASE)/i86pc
```

```
**********************************************************
    4169 Tue Jan 14 16:17:30 2014
new/usr/src/uts/intel/brand/lx/lx_brand_asm.s
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#if defined(__lint)

#include <sys/systm.h>

#else   /* __lint */

#include "genassym.h"
#include "../common/brand_asm.h"

#endif  /* __lint */

#ifdef  __lint

void
lx_brand_int80_callback(void)
{
}

#else   /* __lint */

#if defined(__amd64)

/*
 * See "64-BIT INTERPOSITION STACK" in brand_asm.h.
 */
ENTRY(lx_brand_int80_callback)
        GET_PROCP(SP_REG, 0, %r15)
        movq    P_ZONE(%r15), %r15              /* grab the zone pointer */
        /* grab the 'max syscall num' for this process from 'zone brand data' */
        movq    ZONE_BRAND_DATA(%r15), %r15     /* grab the zone brand ptr */
        movl    LXZD_MAX_SYSCALL(%r15), %r15d   /* get the 'max sysnum' word */
        cmpq    %r15, %rax                      /* is 0 <= syscall <= MAX? */
        jbe     0f                              /* yes, syscall is OK */
        xorl    %eax, %eax                      /* no, zero syscall number */
0:
.lx_brand_int80_patch_point:
        jmp     .lx_brand_int80_notrace

.lx_brand_int80_notrace:
        CALC_TABLE_ADDR(%r15, L_HANDLER)
1:
        movq    %r15, %rax
        GET_V(%rsp, 0, V_SSP, %rsp)     /* restore intr. stack pointer */
        xchgq   (%rsp), %rax            /* swap %rax and return addr */
        jmp     sys_sysint_swapgs_iret

.lx_brand_int80_trace:
        /*
         * If tracing is active, we vector to an alternate trace-enabling
         * handler table instead.
         */
        CALC_TABLE_ADDR(%r15, L_TRACEHANDLER)
        jmp     1b
SET_SIZE(lx_brand_int80_callback)

#define PATCH_POINT     _CONST(.lx_brand_int80_patch_point + 1)
#define PATCH_VAL       _CONST(.lx_brand_int80_trace - .lx_brand_int80_notrace)

ENTRY(lx_brand_int80_enable)
        movl    $1, lx_systrace_brand_enabled(%rip)
        movq    $PATCH_POINT, %r8
        movb    $PATCH_VAL, (%r8)
        ret
SET_SIZE(lx_brand_int80_enable)

ENTRY(lx_brand_int80_disable)
        movq    $PATCH_POINT, %r8
        movb    $0, (%r8)
        movl    $0, lx_systrace_brand_enabled(%rip)
        ret
SET_SIZE(lx_brand_int80_disable)


#elif defined(__i386)

/*
 * See "32-BIT INTERPOSITION STACK" in brand_asm.h.
 */
ENTRY(lx_brand_int80_callback)
        GET_PROCP(SP_REG, 0, %ebx)
        movl    P_ZONE(%ebx), %ebx              /* grab the zone pointer */
        /* grab the 'max syscall num' for this process from 'zone brand data' */
        movl    ZONE_BRAND_DATA(%ebx), %ebx     /* grab the zone brand data */
        movl    LXZD_MAX_SYSCALL(%ebx), %ebx    /* get the max sysnum */

        cmpl    %ebx, %eax                      /* is 0 <= syscall <= MAX? */
        jbe     0f                              /* yes, syscall is OK */
        xorl    %eax, %eax                      /* no, zero syscall number */
0:

.lx_brand_int80_patch_point:
        jmp     .lx_brand_int80_notrace

.lx_brand_int80_notrace:
        CALC_TABLE_ADDR(%ebx, L_HANDLER)

1:
        movl    %ebx, %eax
        GET_V(%esp, 0, V_U_EBX, %ebx)           /* restore scratch register */
        addl    $V_END, %esp            /* restore intr. stack ptr */
```

```
128         xchgl   (%esp), %eax            /* swap new and orig. return addrs */
129         jmp     nopop_sys_rtt_syscall

131 .lx_brand_int80_trace:
132         CALC_TABLE_ADDR(%ebx, L_TRACEHANDLER)
133         jmp     1b
134 SET_SIZE(lx_brand_int80_callback)


137 #define PATCH_POINT     _CONST(.lx_brand_int80_patch_point + 1)
138 #define PATCH_VAL       _CONST(.lx_brand_int80_trace - .lx_brand_int80_notrace)

140 ENTRY(lx_brand_int80_enable)
141         pushl   %ebx
142         pushl   %eax
143         movl    $1, lx_systrace_brand_enabled
144         movl    $PATCH_POINT, %ebx
145         movl    $PATCH_VAL, %eax
146         movb    %al, (%ebx)
147         popl    %eax
148         popl    %ebx
149         ret
150 SET_SIZE(lx_brand_int80_enable)

152 ENTRY(lx_brand_int80_disable)
153         pushl   %ebx
154         movl    $PATCH_POINT, %ebx
155         movb    $0, (%ebx)
156         movl    $0, lx_systrace_brand_enabled
157         popl    %ebx
158         ret
159 SET_SIZE(lx_brand_int80_disable)

161 #endif  /* __i386 */
162 #endif  /* __lint */
163 #endif /* ! codereview */
```

```
**********************************************************
    1818 Tue Jan 14 16:17:30 2014
new/usr/src/uts/intel/genassym/Makefile
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "%Z%%M% %I%     %E% SMI"
#
#       This makefile drives the production of genassym.h through
#       compile time intialized data.
#
#       intel architecture dependent
#


#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../..

GENASSYM_H      = $(GENASSYM_DIR)/$(OBJS_DIR)/genassym.h
OFFSETS_SRC     = $(GENASSYM_DIR)/offsets.in

#
#       Include common rules.
#
include $(UTSBASE)/intel/Makefile.intel

#
#       Define targets
#
ALL_TARGET      = $(GENASSYM_H)

INC_PATH        += -I$(UTSBASE)/common/brand/lx

#
#       Overrides
#
CLEANFILES      = Nothing_to_remove
CLOBBERFILES    = $(GENASSYM_H) Nothing_to_remove

#
#       Default build targets.
#
```



```
.KEEP_STATE:

def:            $(DEF_DEPS)

all:            $(ALL_DEPS)

clean:          $(CLEAN_DEPS)

clobber:        $(CLOBBER_DEPS)

clean.lint:

install:        def

#
# Create genassym.h
#
$(GENASSYM_H): $(OFFSETS_SRC)
        $(OFFSETS_CREATE) <$(OFFSETS_SRC) >$@

#
#       Include common targets.
#
include $(UTSBASE)/intel/Makefile.targ
#endif /* ! codereview */
```

****
1165 Tue Jan 14 16:17:30 2014
new/usr/src/uts/intel/genassym/offsets.in
Bring back LX zones.
****

```
\
\ CDDL HEADER START
\
\ The contents of this file are subject to the terms of the
\ Common Development and Distribution License (the "License").
\ You may not use this file except in compliance with the License.
\
\ You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
\ or http://www.opensolaris.org/os/licensing.
\ See the License for the specific language governing permissions
\ and limitations under the License.
\
\ When distributing Covered Code, include this CDDL HEADER in each
\ file and include the License file at usr/src/OPENSOLARIS.LICENSE.
\ If applicable, add the following below this CDDL HEADER, with the
\ fields enclosed by brackets "[]" replaced with your own identifying
\ information: Portions Copyright [yyyy] [name of copyright owner]
\
\ CDDL HEADER END
\
\
\ Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
\ Use is subject to license terms.
\

\
\ offsets.in: input file to produce the architecture-dependent genassym.h
\ using the ctfstabs program
\

#ifndef _GENASSYM
#define _GENASSYM
#endif

#include <sys/lx_brand.h>

lx_proc_data
        l_handler
        l_tracehandler
        l_traceflag

lx_zone_data
        lxzd_max_syscall
#endif /* ! codereview */
```

```
**********************************************************
   40308 Tue Jan 14 16:17:30 2014
new/usr/src/uts/intel/ia32/os/archdep.c
Bring back LX zones.
**********************************************************
______unchanged_portion_omitted_

 607 /*
 608  * Protect segment registers from non-user privilege levels and GDT selectors
 609  * other than USER_CS, USER_DS and lwp FS and GS values.  If the segment
 610  * selector is non-null and not USER_CS/USER_DS, we make sure that the
 611  * TI bit is set to point into the LDT and that the RPL is set to 3.
 612  *
 613  * Since struct regs stores each 16-bit segment register as a 32-bit greg_t, we
 614  * also explicitly zero the top 16 bits since they may be coming from the
 615  * user's address space via setcontext(2) or /proc.
 616  *
 617  * Note about null selector. When running on the hypervisor if we allow a
 618  * process to set its %cs to null selector with RPL of 0 the hypervisor will
 619  * crash the domain. If running on bare metal we would get a #gp fault and
 620  * be able to kill the process and continue on. Therefore we make sure to
 621  * force RPL to SEL_UPL even for null selector when setting %cs.
 622  */

 624 #if defined(IS_CS) || defined(IS_NOT_CS)
 625 #error  "IS_CS and IS_NOT_CS already defined"
 626 #endif

 628 #define IS_CS           1
 629 #define IS_NOT_CS       0

 631 /*ARGSUSED*/
 632 static greg_t
 633 fix_segreg(greg_t sr, int iscs, model_t datamodel)
 634 {
 635         kthread_t *t = curthread;
 636
 637 #endif /* ! codereview */
 638         switch (sr &= 0xffff) {

 640         case 0:
 641                 if (iscs == IS_CS)
 642                         return (0 | SEL_UPL);
 643                 else
 644                         return (0);

 646 #if defined(__amd64)
 647         /*
 648          * If lwp attempts to switch data model then force their
 649          * code selector to be null selector.
 650          */
 651         case U32CS_SEL:
 652                 if (datamodel == DATAMODEL_NATIVE)
 653                         return (0 | SEL_UPL);
 654                 else
 655                         return (sr);

 657         case UCS_SEL:
 658                 if (datamodel == DATAMODEL_ILP32)
 659                         return (0 | SEL_UPL);
 660 #elif defined(__i386)
 661         case UCS_SEL:
 662 #endif
 663         /*FALLTHROUGH*/
 664         case UDS_SEL:
 665         case LWPFS_SEL:
```



```
 666         case LWPGS_SEL:
 667         case SEL_UPL:
 668                 return (sr);
 669         default:
 670                 break;
 671         }

 673         /*
 674          * Allow this process's brand to do any necessary segment register
 675          * manipulation.
 676          */
 677         if (PROC_IS_BRANDED(t->t_procp) && BRMOP(t->t_procp)->b_fixsegreg) {
 678                 greg_t bsr = BRMOP(t->t_procp)->b_fixsegreg(sr, datamodel);

 680                 if (bsr == 0 && iscs == IS_CS)
 681                         return (0 | SEL_UPL);
 682                 else
 683                         return (bsr);
 684 #endif /* ! codereview */
 685         }

 687         /*
 688          * Force it into the LDT in ring 3 for 32-bit processes, which by
 689          * default do not have an LDT, so that any attempt to use an invalid
 690          * selector will reference the (non-existant) LDT, and cause a #gp
 691          * fault for the process.
 692          *
 693          * 64-bit processes get the null gdt selector since they
 694          * are not allowed to have a private LDT.
 695          */
 696 #if defined(__amd64)
 697         if (datamodel == DATAMODEL_ILP32) {
 698                 return (sr | SEL_TI_LDT | SEL_UPL);
 699         } else {
 700                 if (iscs == IS_CS)
 701                         return (0 | SEL_UPL);
 702                 else
 703                         return (0);
 704         }

 706 #elif defined(__i386)
 707         return (sr | SEL_TI_LDT | SEL_UPL);
 708 #endif
 709 }

 711 /*
 712  * Set general registers.
 713  */
 714 void
 715 setgregs(klwp_t *lwp, gregset_t grp)
 716 {
 717         struct regs *rp = lwptoregs(lwp);
 718         model_t datamodel = lwp_getdatamodel(lwp);

 720 #if defined(__amd64)
 721         struct pcb *pcb = &lwp->lwp_pcb;
 722         int thisthread = lwptot(lwp) == curthread;

 724         if (datamodel == DATAMODEL_NATIVE) {

 726                 if (thisthread)
 727                         (void) save_syscall_args();     /* copy the args */

 729                 rp->r_rdi = grp[REG_RDI];
 730                 rp->r_rsi = grp[REG_RSI];
 731                 rp->r_rdx = grp[REG_RDX];
```

```
732                 rp->r_rcx = grp[REG_RCX];
733                 rp->r_r8 = grp[REG_R8];
734                 rp->r_r9 = grp[REG_R9];
735                 rp->r_rax = grp[REG_RAX];
736                 rp->r_rbx = grp[REG_RBX];
737                 rp->r_rbp = grp[REG_RBP];
738                 rp->r_r10 = grp[REG_R10];
739                 rp->r_r11 = grp[REG_R11];
740                 rp->r_r12 = grp[REG_R12];
741                 rp->r_r13 = grp[REG_R13];
742                 rp->r_r14 = grp[REG_R14];
743                 rp->r_r15 = grp[REG_R15];
744                 rp->r_trapno = grp[REG_TRAPNO];
745                 rp->r_err = grp[REG_ERR];
746                 rp->r_rip = grp[REG_RIP];
747                 /*
748                  * Setting %cs or %ss to anything else is quietly but
749                  * quite definitely forbidden!
750                  */
751                 rp->r_cs = UCS_SEL;
752                 rp->r_ss = UDS_SEL;
753                 rp->r_rsp = grp[REG_RSP];

755                 if (thisthread)
756                         kpreempt_disable();

758                 pcb->pcb_ds = UDS_SEL;
759                 pcb->pcb_es = UDS_SEL;

761                 /*
762                  * 64-bit processes -are- allowed to set their fsbase/gsbase
763                  * values directly, but only if they're using the segment
764                  * selectors that allow that semantic.
765                  *
766                  * (32-bit processes must use lwp_set_private().)
767                  */
768                 pcb->pcb_fsbase = grp[REG_FSBASE];
769                 pcb->pcb_gsbase = grp[REG_GSBASE];
770                 pcb->pcb_fs = fix_segreg(grp[REG_FS], IS_NOT_CS, datamodel);
771                 pcb->pcb_gs = fix_segreg(grp[REG_GS], IS_NOT_CS, datamodel);

773                 /*
774                  * Ensure that we go out via update_sregs
775                  */
776                 pcb->pcb_rupdate = 1;
777                 lwptot(lwp)->t_post_sys = 1;
778                 if (thisthread)
779                         kpreempt_enable();
780 #if defined(_SYSCALL32_IMPL)
781         } else {
782                 rp->r_rdi = (uint32_t)grp[REG_RDI];
783                 rp->r_rsi = (uint32_t)grp[REG_RSI];
784                 rp->r_rdx = (uint32_t)grp[REG_RDX];
785                 rp->r_rcx = (uint32_t)grp[REG_RCX];
786                 rp->r_rax = (uint32_t)grp[REG_RAX];
787                 rp->r_rbx = (uint32_t)grp[REG_RBX];
788                 rp->r_rbp = (uint32_t)grp[REG_RBP];
789                 rp->r_trapno = (uint32_t)grp[REG_TRAPNO];
790                 rp->r_err = (uint32_t)grp[REG_ERR];
791                 rp->r_rip = (uint32_t)grp[REG_RIP];

793                 rp->r_cs = fix_segreg(grp[REG_CS], IS_CS, datamodel);
794                 rp->r_ss = fix_segreg(grp[REG_DS], IS_NOT_CS, datamodel);

796                 rp->r_rsp = (uint32_t)grp[REG_RSP];
```



```
798                 if (thisthread)
799                         kpreempt_disable();

801                 pcb->pcb_ds = fix_segreg(grp[REG_DS], IS_NOT_CS, datamodel);
802                 pcb->pcb_es = fix_segreg(grp[REG_ES], IS_NOT_CS, datamodel);

804                 /*
805                  * (See fsbase/gsbase commentary above)
806                  */
807                 pcb->pcb_fs = fix_segreg(grp[REG_FS], IS_NOT_CS, datamodel);
808                 pcb->pcb_gs = fix_segreg(grp[REG_GS], IS_NOT_CS, datamodel);

810                 /*
811                  * Ensure that we go out via update_sregs
812                  */
813                 pcb->pcb_rupdate = 1;
814                 lwptot(lwp)->t_post_sys = 1;
815                 if (thisthread)
816                         kpreempt_enable();
817 #endif
818         }

820         /*
821          * Only certain bits of the flags register can be modified.
822          */
823         rp->r_rfl = (rp->r_rfl & ~PSL_USERMASK) |
824             (grp[REG_RFL] & PSL_USERMASK);

826 #elif defined(__i386)

828         /*
829          * Only certain bits of the flags register can be modified.
830          */
831         grp[EFL] = (rp->r_efl & ~PSL_USERMASK) | (grp[EFL] & PSL_USERMASK);

833         /*
834          * Copy saved registers from user stack.
835          */
836         bcopy(grp, &rp->r_gs, sizeof (gregset_t));

838         rp->r_cs = fix_segreg(rp->r_cs, IS_CS, datamodel);
839         rp->r_ss = fix_segreg(rp->r_ss, IS_NOT_CS, datamodel);
840         rp->r_ds = fix_segreg(rp->r_ds, IS_NOT_CS, datamodel);
841         rp->r_es = fix_segreg(rp->r_es, IS_NOT_CS, datamodel);
842         rp->r_fs = fix_segreg(rp->r_fs, IS_NOT_CS, datamodel);
843         rp->r_gs = fix_segreg(rp->r_gs, IS_NOT_CS, datamodel);

845 #endif  /* __i386 */
846 }

848 /*
849  * Determine whether eip is likely to have an interrupt frame
850  * on the stack.  We do this by comparing the address to the
851  * range of addresses spanned by several well-known routines.
852  */
853 extern void _interrupt();
854 extern void _allsyscalls();
855 extern void _cmntrap();
856 extern void fakesoftint();

858 extern size_t _interrupt_size;
859 extern size_t _allsyscalls_size;
860 extern size_t _cmntrap_size;
861 extern size_t _fakesoftint_size;

863 /*
```

```
 * Get a pc-only stacktrace.  Used for kmem_alloc() buffer ownership tracking.
 * Returns MIN(current stack depth, pcstack_limit).
 */
int
getpcstack(pc_t *pcstack, int pcstack_limit)
{
        struct frame *fp = (struct frame *)getfp();
        struct frame *nextfp, *minfp, *stacktop;
        int depth = 0;
        int on_intr;
        uintptr_t pc;

        if ((on_intr = CPU_ON_INTR(CPU)) != 0)
                stacktop = (struct frame *)(CPU->cpu_intr_stack + SA(MINFRAME));
        else
                stacktop = (struct frame *)curthread->t_stk;
        minfp = fp;

        pc = ((struct regs *)fp)->r_pc;

        while (depth < pcstack_limit) {
                nextfp = (struct frame *)fp->fr_savfp;
                pc = fp->fr_savpc;
                if (nextfp <= minfp || nextfp >= stacktop) {
                        if (on_intr) {
                                /*
                                 * Hop from interrupt stack to thread stack.
                                 */
                                stacktop = (struct frame *)curthread->t_stk;
                                minfp = (struct frame *)curthread->t_stkbase;
                                on_intr = 0;
                                continue;
                        }
                        break;
                }
                pcstack[depth++] = (pc_t)pc;
                fp = nextfp;
                minfp = fp;
        }
        return (depth);
}

/*
 * The following ELF header fields are defined as processor-specific
 * in the V8 ABI:
 *
 *      e_ident[EI_DATA]        encoding of the processor-specific
 *                              data in the object file
 *      e_machine               processor identification
 *      e_flags                 processor-specific flags associated
 *                              with the file
 */

/*
 * The value of at_flags reflects a platform's cpu module support.
 * at_flags is used to check for allowing a binary to execute and
 * is passed as the value of the AT_FLAGS auxiliary vector.
 */
int at_flags = 0;

/*
 * Check the processor-specific fields of an ELF header.
 *
 * returns 1 if the fields are valid, 0 otherwise
 */
/*ARGSUSED2*/
```



```
int
elfheadcheck(
        unsigned char e_data,
        Elf32_Half e_machine,
        Elf32_Word e_flags)
{
        if (e_data != ELFDATA2LSB)
                return (0);
#if defined(__amd64)
        if (e_machine == EM_AMD64)
                return (1);
#endif
        return (e_machine == EM_386);
}

uint_t auxv_hwcap_include = 0;  /* patch to enable unrecognized features */
uint_t auxv_hwcap_include_2 = 0;        /* second word */
uint_t auxv_hwcap_exclude = 0;  /* patch for broken cpus, debugging */
uint_t auxv_hwcap_exclude_2 = 0;        /* second word */
#if defined(_SYSCALL32_IMPL)
uint_t auxv_hwcap32_include = 0;        /* ditto for 32-bit apps */
uint_t auxv_hwcap32_include_2 = 0;      /* ditto for 32-bit apps */
uint_t auxv_hwcap32_exclude = 0;        /* ditto for 32-bit apps */
uint_t auxv_hwcap32_exclude_2 = 0;      /* ditto for 32-bit apps */
#endif

/*
 * Gather information about the processor and place it into auxv_hwcap
 * so that it can be exported to the linker via the aux vector.
 *
 * We use this seemingly complicated mechanism so that we can ensure
 * that /etc/system can be used to override what the system can or
 * cannot discover for itself.
 */
void
bind_hwcap(void)
{
        uint_t cpu_hwcap_flags[2];
        cpuid_pass4(NULL, cpu_hwcap_flags);

        auxv_hwcap = (auxv_hwcap_include | cpu_hwcap_flags[0]) &
            ~auxv_hwcap_exclude;
        auxv_hwcap_2 = (auxv_hwcap_include_2 | cpu_hwcap_flags[1]) &
            ~auxv_hwcap_exclude_2;

#if defined(__amd64)
        /*
         * On AMD processors, sysenter just doesn't work at all
         * when the kernel is in long mode.  On IA-32e processors
         * it does, but there's no real point in all the alternate
         * mechanism when syscall works on both.
         *
         * Besides, the kernel's sysenter handler is expecting a
         * 32-bit lwp ...
         */
        auxv_hwcap &= ~AV_386_SEP;
#else
        /*
         * 32-bit processes can -always- use the lahf/sahf instructions
         */
        auxv_hwcap |= AV_386_AHF;
#endif

        if (auxv_hwcap_include || auxv_hwcap_exclude || auxv_hwcap_include_2 ||
            auxv_hwcap_exclude_2) {
                /*
```

```
 996                  * The below assignment is regrettably required to get lint
 997                  * to accept the validity of our format string.  The format
 998                  * string is in fact valid, but whatever intelligence in lint
 999                  * understands the cmn_err()-specific %b appears to have an
1000                  * off-by-one error:  it (mistakenly) complains about bit
1001                  * number 32 (even though this is explicitly permitted).
1002                  * Normally, one would will away such warnings with a "LINTED"
1003                  * directive, but for reasons unclear and unknown, lint
1004                  * refuses to be assuaged in this case.  Fortunately, lint
1005                  * doesn't pretend to have solved the Halting Problem --
1006                  * and as soon as the format string is programmatic, it
1007                  * knows enough to shut up.
1008                  */
1009                 char *fmt = "?user ABI extensions: %b\n";
1010                 cmn_err(CE_CONT, fmt, auxv_hwcap, FMT_AV_386);
1011                 fmt = "?user ABI extensions (word 2): %b\n";
1012                 cmn_err(CE_CONT, fmt, auxv_hwcap_2, FMT_AV_386_2);
1013         }

1015 #if defined(_SYSCALL32_IMPL)
1016         auxv_hwcap32 = (auxv_hwcap32_include | cpu_hwcap_flags[0]) &
1017             ~auxv_hwcap32_exclude;
1018         auxv_hwcap32_2 = (auxv_hwcap32_include_2 | cpu_hwcap_flags[1]) &
1019             ~auxv_hwcap32_exclude_2;

1021 #if defined(__amd64)
1022         /*
1023          * If this is an amd64 architecture machine from Intel, then
1024          * syscall -doesn't- work in compatibility mode, only sysenter does.
1025          *
1026          * Sigh.
1027          */
1028         if (!cpuid_syscall32_insn(NULL))
1029                 auxv_hwcap32 &= ~AV_386_AMD_SYSC;

1031         /*
1032          * 32-bit processes can -always- use the lahf/sahf instructions
1033          */
1034         auxv_hwcap32 |= AV_386_AHF;
1035 #endif

1037         if (auxv_hwcap32_include || auxv_hwcap32_exclude ||
1038             auxv_hwcap32_include_2 || auxv_hwcap32_exclude_2) {
1039                 /*
1040                  * See the block comment in the cmn_err() of auxv_hwcap, above.
1041                  */
1042                 char *fmt = "?32-bit user ABI extensions: %b\n";
1043                 cmn_err(CE_CONT, fmt, auxv_hwcap32, FMT_AV_386);
1044                 fmt = "?32-bit user ABI extensions (word 2): %b\n";
1045                 cmn_err(CE_CONT, fmt, auxv_hwcap32_2, FMT_AV_386_2);
1046         }
1047 #endif
1048 }

1050 /*
1051  *      sync_icache() - this is called
1052  *      in proc/fs/prusrio.c. x86 has an unified cache and therefore
1053  *      this is a nop.
1054  */
1055 /* ARGSUSED */
1056 void
1057 sync_icache(caddr_t addr, uint_t len)
1058 {
1059         /* Do nothing for now */
1060 }
```



```
1062 /*ARGSUSED*/
1063 void
1064 sync_data_memory(caddr_t va, size_t len)
1065 {
1066         /* Not implemented for this platform */
1067 }

1069 int
1070 __ipltospl(int ipl)
1071 {
1072         return (ipltospl(ipl));
1073 }

1075 /*
1076  * The panic code invokes panic_saveregs() to record the contents of a
1077  * regs structure into the specified panic_data structure for debuggers.
1078  */
1079 void
1080 panic_saveregs(panic_data_t *pdp, struct regs *rp)
1081 {
1082         panic_nv_t *pnv = PANICNVGET(pdp);

1084         struct cregs    creg;

1086         getcregs(&creg);

1088 #if defined(__amd64)
1089         PANICNVADD(pnv, "rdi", rp->r_rdi);
1090         PANICNVADD(pnv, "rsi", rp->r_rsi);
1091         PANICNVADD(pnv, "rdx", rp->r_rdx);
1092         PANICNVADD(pnv, "rcx", rp->r_rcx);
1093         PANICNVADD(pnv, "r8", rp->r_r8);
1094         PANICNVADD(pnv, "r9", rp->r_r9);
1095         PANICNVADD(pnv, "rax", rp->r_rax);
1096         PANICNVADD(pnv, "rbx", rp->r_rbx);
1097         PANICNVADD(pnv, "rbp", rp->r_rbp);
1098         PANICNVADD(pnv, "r10", rp->r_r10);
1099         PANICNVADD(pnv, "r11", rp->r_r11);
1100         PANICNVADD(pnv, "r12", rp->r_r12);
1101         PANICNVADD(pnv, "r13", rp->r_r13);
1102         PANICNVADD(pnv, "r14", rp->r_r14);
1103         PANICNVADD(pnv, "r15", rp->r_r15);
1104         PANICNVADD(pnv, "fsbase", rdmsr(MSR_AMD_FSBASE));
1105         PANICNVADD(pnv, "gsbase", rdmsr(MSR_AMD_GSBASE));
1106         PANICNVADD(pnv, "ds", rp->r_ds);
1107         PANICNVADD(pnv, "es", rp->r_es);
1108         PANICNVADD(pnv, "fs", rp->r_fs);
1109         PANICNVADD(pnv, "gs", rp->r_gs);
1110         PANICNVADD(pnv, "trapno", rp->r_trapno);
1111         PANICNVADD(pnv, "err", rp->r_err);
1112         PANICNVADD(pnv, "rip", rp->r_rip);
1113         PANICNVADD(pnv, "cs", rp->r_cs);
1114         PANICNVADD(pnv, "rflags", rp->r_rfl);
1115         PANICNVADD(pnv, "rsp", rp->r_rsp);
1116         PANICNVADD(pnv, "ss", rp->r_ss);
1117         PANICNVADD(pnv, "gdt_hi", (uint64_t)(creg.cr_gdt._l[3]));
1118         PANICNVADD(pnv, "gdt_lo", (uint64_t)(creg.cr_gdt._l[0]));
1119         PANICNVADD(pnv, "idt_hi", (uint64_t)(creg.cr_idt._l[3]));
1120         PANICNVADD(pnv, "idt_lo", (uint64_t)(creg.cr_idt._l[0]));
1121 #elif defined(__i386)
1122         PANICNVADD(pnv, "gs", (uint32_t)rp->r_gs);
1123         PANICNVADD(pnv, "fs", (uint32_t)rp->r_fs);
1124         PANICNVADD(pnv, "es", (uint32_t)rp->r_es);
1125         PANICNVADD(pnv, "ds", (uint32_t)rp->r_ds);
1126         PANICNVADD(pnv, "edi", (uint32_t)rp->r_edi);
1127         PANICNVADD(pnv, "esi", (uint32_t)rp->r_esi);
```

```
1128         PANICNVADD(pnv, "ebp", (uint32_t)rp->r_ebp);
1129         PANICNVADD(pnv, "esp", (uint32_t)rp->r_esp);
1130         PANICNVADD(pnv, "ebx", (uint32_t)rp->r_ebx);
1131         PANICNVADD(pnv, "edx", (uint32_t)rp->r_edx);
1132         PANICNVADD(pnv, "ecx", (uint32_t)rp->r_ecx);
1133         PANICNVADD(pnv, "eax", (uint32_t)rp->r_eax);
1134         PANICNVADD(pnv, "trapno", (uint32_t)rp->r_trapno);
1135         PANICNVADD(pnv, "err", (uint32_t)rp->r_err);
1136         PANICNVADD(pnv, "eip", (uint32_t)rp->r_eip);
1137         PANICNVADD(pnv, "cs", (uint32_t)rp->r_cs);
1138         PANICNVADD(pnv, "eflags", (uint32_t)rp->r_efl);
1139         PANICNVADD(pnv, "uesp", (uint32_t)rp->r_uesp);
1140         PANICNVADD(pnv, "ss", (uint32_t)rp->r_ss);
1141         PANICNVADD(pnv, "gdt", creg.cr_gdt);
1142         PANICNVADD(pnv, "idt", creg.cr_idt);
1143 #endif  /* __i386 */

1145         PANICNVADD(pnv, "ldt", creg.cr_ldt);
1146         PANICNVADD(pnv, "task", creg.cr_task);
1147         PANICNVADD(pnv, "cr0", creg.cr_cr0);
1148         PANICNVADD(pnv, "cr2", creg.cr_cr2);
1149         PANICNVADD(pnv, "cr3", creg.cr_cr3);
1150         if (creg.cr_cr4)
1151                 PANICNVADD(pnv, "cr4", creg.cr_cr4);

1153         PANICNVSET(pdp, pnv);
1154 }

1156 #define TR_ARG_MAX 6    /* Max args to print, same as SPARC */

1158 #if !defined(__amd64)

1160 /*
1161  * Given a return address (%eip), determine the likely number of arguments
1162  * that were pushed on the stack prior to its execution.  We do this by
1163  * expecting that a typical call sequence consists of pushing arguments on
1164  * the stack, executing a call instruction, and then performing an add
1165  * on %esp to restore it to the value prior to pushing the arguments for
1166  * the call.  We attempt to detect such an add, and divide the addend
1167  * by the size of a word to determine the number of pushed arguments.
1168  *
1169  * If we do not find such an add, we punt and return TR_ARG_MAX. It is not
1170  * possible to reliably determine if a function took no arguments (i.e. was
1171  * void) because assembler routines do not reliably perform an add on %esp
1172  * immediately upon returning (eg. _sys_call()), so returning TR_ARG_MAX is
1173  * safer than returning 0.
1174  */
1175 static ulong_t
1176 argcount(uintptr_t eip)
1177 {
1178         const uint8_t *ins = (const uint8_t *)eip;
1179         ulong_t n;

1181         enum {
1182                 M_MODRM_ESP = 0xc4,     /* Mod/RM byte indicates %esp */
1183                 M_ADD_IMM32 = 0x81,     /* ADD imm32 to r/m32 */
1184                 M_ADD_IMM8  = 0x83      /* ADD imm8 to r/m32 */
1185         };

1187         if (eip < KERNELBASE || ins[1] != M_MODRM_ESP)
1188                 return (TR_ARG_MAX);

1190         switch (ins[0]) {
1191         case M_ADD_IMM32:
1192                 n = ins[2] + (ins[3] << 8) + (ins[4] << 16) + (ins[5] << 24);
1193                 break;
```



```
1195         case M_ADD_IMM8:
1196                 n = ins[2];
1197                 break;

1199         default:
1200                 return (TR_ARG_MAX);
1201         }

1203         n /= sizeof (long);
1204         return (MIN(n, TR_ARG_MAX));
1205 }

1207 #endif  /* !__amd64 */

1209 /*
1210  * Print a stack backtrace using the specified frame pointer.  We delay two
1211  * seconds before continuing, unless this is the panic traceback.
1212  * If we are in the process of panicking, we also attempt to write the
1213  * stack backtrace to a staticly assigned buffer, to allow the panic
1214  * code to find it and write it in to uncompressed pages within the
1215  * system crash dump.
1216  * Note that the frame for the starting stack pointer value is omitted because
1217  * the corresponding %eip is not known.
1218  */

1220 extern char *dump_stack_scratch;

1222 #if defined(__amd64)

1224 void
1225 traceback(caddr_t fpreg)
1226 {
1227         struct frame    *fp = (struct frame *)fpreg;
1228         struct frame    *nextfp;
1229         uintptr_t       pc, nextpc;
1230         ulong_t         off;
1231         char            args[TR_ARG_MAX * 2 + 16], *sym;
1232         uint_t    offset = 0;
1233         uint_t    next_offset = 0;
1234         char        stack_buffer[1024];

1236         if (!panicstr)
1237                 printf("traceback: %%fp = %p\n", (void *)fp);

1239         if (panicstr && !dump_stack_scratch) {
1240                 printf("Warning - stack not written to the dump buffer\n");
1241         }

1243         fp = (struct frame *)plat_traceback(fpreg);
1244         if ((uintptr_t)fp < KERNELBASE)
1245                 goto out;

1247         pc = fp->fr_savpc;
1248         fp = (struct frame *)fp->fr_savfp;

1250         while ((uintptr_t)fp >= KERNELBASE) {
1251                 /*
1252                  * XX64 Until port is complete tolerate 8-byte aligned
1253                  * frame pointers but flag with a warning so they can
1254                  * be fixed.
1255                  */
1256                 if (((uintptr_t)fp & (STACK_ALIGN - 1)) != 0) {
1257                         if (((uintptr_t)fp & (8 - 1)) == 0) {
1258                                 printf("  >> warning! 8-byte"
1259                                     " aligned %%fp = %p\n", (void *)fp);
```

```
1260                        } else {
1261                                printf(
1262                                    "  >> mis-aligned %%fp = %p\n", (void *)fp);
1263                                break;
1264                        }
1265                }

1267                args[0] = '\0';
1268                nextpc = (uintptr_t)fp->fr_savpc;
1269                nextfp = (struct frame *)fp->fr_savfp;
1270                if ((sym = kobj_getsymname(pc, &off)) != NULL) {
1271                        printf("%016lx %s:%s+%lx (%s)\n", (uintptr_t)fp,
1272                            mod_containing_pc((caddr_t)pc), sym, off, args);
1273                        (void) snprintf(stack_buffer, sizeof (stack_buffer),
1274                            "%s:%s+%lx (%s) | ",
1275                            mod_containing_pc((caddr_t)pc), sym, off, args);
1276                } else {
1277                        printf("%016lx %lx (%s)\n",
1278                            (uintptr_t)fp, pc, args);
1279                        (void) snprintf(stack_buffer, sizeof (stack_buffer),
1280                            "%lx (%s) | ", pc, args);
1281                }

1283                if (panicstr && dump_stack_scratch) {
1284                        next_offset = offset + strlen(stack_buffer);
1285                        if (next_offset < STACK_BUF_SIZE) {
1286                                bcopy(stack_buffer, dump_stack_scratch + offset,
1287                                    strlen(stack_buffer));
1288                                offset = next_offset;
1289                        } else {
1290                                /*
1291                                 * In attempting to save the panic stack
1292                                 * to the dumpbuf we have overflowed that area.
1293                                 * Print a warning and continue to printf the
1294                                 * stack to the msgbuf
1295                                 */
1296                                printf("Warning: stack in the dump buffer"
1297                                    " may be incomplete\n");
1298                                offset = next_offset;
1299                        }
1300                }

1302                pc = nextpc;
1303                fp = nextfp;
1304        }
1305 out:
1306        if (!panicstr) {
1307                printf("end of traceback\n");
1308                DELAY(2 * MICROSEC);
1309        } else if (dump_stack_scratch) {
1310                dump_stack_scratch[offset] = '\0';
1311        }
1312 }

1314 #elif defined(__i386)

1316 void
1317 traceback(caddr_t fpreg)
1318 {
1319        struct frame *fp = (struct frame *)fpreg;
1320        struct frame *nextfp, *minfp, *stacktop;
1321        uintptr_t pc, nextpc;
1322        uint_t    offset = 0;
1323        uint_t    next_offset = 0;
1324        char        stack_buffer[1024];
```



```
1326        cpu_t *cpu;

1328        /*
1329         * args[] holds TR_ARG_MAX hex long args, plus ", " or '\0'.
1330         */
1331        char args[TR_ARG_MAX * 2 + 8], *p;

1333        int on_intr;
1334        ulong_t off;
1335        char *sym;

1337        if (!panicstr)
1338                printf("traceback: %%fp = %p\n", (void *)fp);

1340        if (panicstr && !dump_stack_scratch) {
1341                printf("Warning - stack not written to the dumpbuf\n");
1342        }

1344        /*
1345         * If we are panicking, all high-level interrupt information in
1346         * CPU was overwritten.  panic_cpu has the correct values.
1347         */
1348        kpreempt_disable();                     /* prevent migration */

1350        cpu = (panicstr && CPU->cpu_id == panic_cpu.cpu_id)? &panic_cpu : CPU;

1352        if ((on_intr = CPU_ON_INTR(cpu)) != 0)
1353                stacktop = (struct frame *)(cpu->cpu_intr_stack + SA(MINFRAME));
1354        else
1355                stacktop = (struct frame *)curthread->t_stk;

1357        kpreempt_enable();

1359        fp = (struct frame *)plat_traceback(fpreg);
1360        if ((uintptr_t)fp < KERNELBASE)
1361                goto out;

1363        minfp = fp;     /* Baseline minimum frame pointer */
1364        pc = fp->fr_savpc;
1365        fp = (struct frame *)fp->fr_savfp;

1367        while ((uintptr_t)fp >= KERNELBASE) {
1368                ulong_t argc;
1369                long *argv;

1371                if (fp <= minfp || fp >= stacktop) {
1372                        if (on_intr) {
1373                                /*
1374                                 * Hop from interrupt stack to thread stack.
1375                                 */
1376                                stacktop = (struct frame *)curthread->t_stk;
1377                                minfp = (struct frame *)curthread->t_stkbase;
1378                                on_intr = 0;
1379                                continue;
1380                        }
1381                        break; /* we're outside of the expected range */
1382                }

1384                if ((uintptr_t)fp & (STACK_ALIGN - 1)) {
1385                        printf("  >> mis-aligned %%fp = %p\n", (void *)fp);
1386                        break;
1387                }

1389                nextpc = fp->fr_savpc;
1390                nextfp = (struct frame *)fp->fr_savfp;
1391                argc = argcount(nextpc);
```

```
1392                argv = (long *)((char *)fp + sizeof (struct frame));

1394                args[0] = '\0';
1395                p = args;
1396                while (argc-- > 0 && argv < (long *)stacktop) {
1397                        p += snprintf(p, args + sizeof (args) - p,
1398                            "%s%lx", (p == args) ? "" : ", ", *argv++);
1399                }

1401                if ((sym = kobj_getsymname(pc, &off)) != NULL) {
1402                        printf("%08lx %s:%s+%lx (%s)\n", (uintptr_t)fp,
1403                            mod_containing_pc((caddr_t)pc), sym, off, args);
1404                        (void) snprintf(stack_buffer, sizeof (stack_buffer),
1405                            "%s:%s+%lx (%s) | ",
1406                            mod_containing_pc((caddr_t)pc), sym, off, args);

1408                } else {
1409                        printf("%08lx %lx (%s)\n",
1410                            (uintptr_t)fp, pc, args);
1411                        (void) snprintf(stack_buffer, sizeof (stack_buffer),
1412                            "%lx (%s) | ", pc, args);

1414                }

1416                if (panicstr && dump_stack_scratch) {
1417                        next_offset = offset + strlen(stack_buffer);
1418                        if (next_offset < STACK_BUF_SIZE) {
1419                                bcopy(stack_buffer, dump_stack_scratch + offset,
1420                                    strlen(stack_buffer));
1421                                offset = next_offset;
1422                        } else {
1423                                /*
1424                                 * In attempting to save the panic stack
1425                                 * to the dumpbuf we have overflowed that area.
1426                                 * Print a warning and continue to printf the
1427                                 * stack to the msgbuf
1428                                 */
1429                                printf("Warning: stack in the dumpbuf"
1430                                    " may be incomplete\n");
1431                                offset = next_offset;
1432                        }
1433                }

1435                minfp = fp;
1436                pc = nextpc;
1437                fp = nextfp;
1438        }
1439 out:
1440        if (!panicstr) {
1441                printf("end of traceback\n");
1442                DELAY(2 * MICROSEC);
1443        } else if (dump_stack_scratch) {
1444                dump_stack_scratch[offset] = '\0';
1445        }

1447 }

1449 #endif  /* __i386 */

1451 /*
1452  * Generate a stack backtrace from a saved register set.
1453  */
1454 void
1455 traceregs(struct regs *rp)
1456 {
1457        traceback((caddr_t)rp->r_fp);
```



```
1458 }

1460 void
1461 exec_set_sp(size_t stksize)
1462 {
1463        klwp_t *lwp = ttolwp(curthread);

1465        lwptoregs(lwp)->r_sp = (uintptr_t)curproc->p_usrstack - stksize;
1466 }

1468 hrtime_t
1469 gethrtime_waitfree(void)
1470 {
1471        return (dtrace_gethrtime());
1472 }

1474 hrtime_t
1475 gethrtime(void)
1476 {
1477        return (gethrtimef());
1478 }

1480 hrtime_t
1481 gethrtime_unscaled(void)
1482 {
1483        return (gethrtimeunscaledf());
1484 }

1486 void
1487 scalehrtime(hrtime_t *hrt)
1488 {
1489        scalehrtimef(hrt);
1490 }

1492 uint64_t
1493 unscalehrtime(hrtime_t nsecs)
1494 {
1495        return (unscalehrtimef(nsecs));
1496 }

1498 void
1499 gethrestime(timespec_t *tp)
1500 {
1501        gethrestimef(tp);
1502 }

1504 #if defined(__amd64)
1505 /*
1506  * Part of the implementation of hres_tick(); this routine is
1507  * easier in C than assembler .. called with the hres_lock held.
1508  *
1509  * XX64 Many of these timekeeping variables need to be extern'ed in a header
1510  */

1512 #include <sys/time.h>
1513 #include <sys/machlock.h>

1515 extern int one_sec;
1516 extern int max_hres_adj;

1518 void
1519 __adj_hrestime(void)
1520 {
1521        long long adj;

1523        if (hrestime_adj == 0)
```

```
1524                 adj = 0;
1525         else if (hrestime_adj > 0) {
1526                 if (hrestime_adj < max_hres_adj)
1527                         adj = hrestime_adj;
1528                 else
1529                         adj = max_hres_adj;
1530         } else {
1531                 if (hrestime_adj < -max_hres_adj)
1532                         adj = -max_hres_adj;
1533                 else
1534                         adj = hrestime_adj;
1535         }

1537         timedelta -= adj;
1538         hrestime_adj = timedelta;
1539         hrestime.tv_nsec += adj;

1541         while (hrestime.tv_nsec >= NANOSEC) {
1542                 one_sec++;
1543                 hrestime.tv_sec++;
1544                 hrestime.tv_nsec -= NANOSEC;
1545         }
1546 }
1547 #endif

1549 /*
1550  * Wrapper functions to maintain backwards compability
1551  */
1552 int
1553 xcopyin(const void *uaddr, void *kaddr, size_t count)
1554 {
1555         return (xcopyin_nta(uaddr, kaddr, count, UIO_COPY_CACHED));
1556 }

1558 int
1559 xcopyout(const void *kaddr, void *uaddr, size_t count)
1560 {
1561         return (xcopyout_nta(kaddr, uaddr, count, UIO_COPY_CACHED));
1562 }
```

```
**********************************************************
   35407 Tue Jan 14 16:17:31 2014
new/usr/src/uts/intel/ia32/os/desctbls.c
Bring back LX zones.
**********************************************************
______unchanged_portion_omitted_

 160 /*
 161  * The brand infrastructure interposes on two handlers, and we use one as a
 162  * NULL signpost.
 163  */
 164 static struct interposing_handler brand_tbl[3];
 164 static struct interposing_handler brand_tbl[2];

 166 /*
 167  * software prototypes for default local descriptor table
 168  */

 170 /*
 171  * Routines for loading segment descriptors in format the hardware
 172  * can understand.
 173  */

 175 #if defined(__amd64)

 177 /*
 178  * In long mode we have the new L or long mode attribute bit
 179  * for code segments. Only the conforming bit in type is used along
 180  * with descriptor priority and present bits. Default operand size must
 181  * be zero when in long mode. In 32-bit compatibility mode all fields
 182  * are treated as in legacy mode. For data segments while in long mode
 183  * only the present bit is loaded.
 184  */
 185 void
 186 set_usegd(user_desc_t *dp, uint_t lmode, void *base, size_t size,
 187     uint_t type, uint_t dpl, uint_t gran, uint_t defopsz)
 188 {
 189         ASSERT(lmode == SDP_SHORT || lmode == SDP_LONG);

 191         /*
 192          * 64-bit long mode.
 193          */
 194         if (lmode == SDP_LONG)
 195                 dp->usd_def32 = 0;              /* 32-bit operands only */
 196         else
 197                 /*
 198                  * 32-bit compatibility mode.
 199                  */
 200                 dp->usd_def32 = defopsz;        /* 0 = 16, 1 = 32-bit ops */

 202         dp->usd_long = lmode;   /* 64-bit mode */
 203         dp->usd_type = type;
 204         dp->usd_dpl = dpl;
 205         dp->usd_p = 1;
 206         dp->usd_gran = gran;            /* 0 = bytes, 1 = pages */

 208         dp->usd_lobase = (uintptr_t)base;
 209         dp->usd_midbase = (uintptr_t)base >> 16;
 210         dp->usd_hibase = (uintptr_t)base >> (16 + 8);
 211         dp->usd_lolimit = size;
 212         dp->usd_hilimit = (uintptr_t)size >> 16;
 213 }
______unchanged_portion_omitted_

 900 #endif  /* __xpv */
 901 #endif  /* __i386 */
```



```
 903 /*
 904  * Build kernel IDT.
 905  *
 906  * Note that for amd64 we pretty much require every gate to be an interrupt
 907  * gate which blocks interrupts atomically on entry; that's because of our
 908  * dependency on using 'swapgs' every time we come into the kernel to find
 909  * the cpu structure. If we get interrupted just before doing that, %cs could
 910  * be in kernel mode (so that the trap prolog doesn't do a swapgs), but
 911  * %gsbase is really still pointing at something in userland. Bad things will
 912  * ensue. We also use interrupt gates for i386 as well even though this is not
 913  * required for some traps.
 914  *
 915  * Perhaps they should have invented a trap gate that does an atomic swapgs?
 916  */
 917 static void
 918 init_idt_common(gate_desc_t *idt)
 919 {
 920         set_gatesegd(&idt[T_ZERODIV], &div0trap, KCS_SEL, SDT_SYSIGT, TRP_KPL,
 921             0);
 922         set_gatesegd(&idt[T_SGLSTP], &dbgtrap, KCS_SEL, SDT_SYSIGT, TRP_KPL,
 923             0);
 924         set_gatesegd(&idt[T_NMIFLT], &nmiint, KCS_SEL, SDT_SYSIGT, TRP_KPL,
 925             0);
 926         set_gatesegd(&idt[T_BPTFLT], &brktrap, KCS_SEL, SDT_SYSIGT, TRP_UPL,
 927             0);
 928         set_gatesegd(&idt[T_OVFLW], &ovflotrap, KCS_SEL, SDT_SYSIGT, TRP_UPL,
 929             0);
 930         set_gatesegd(&idt[T_BOUNDFLT], &boundstrap, KCS_SEL, SDT_SYSIGT,
 931             TRP_KPL, 0);
 932         set_gatesegd(&idt[T_ILLINST], &invoptrap, KCS_SEL, SDT_SYSIGT, TRP_KPL,
 933             0);
 934         set_gatesegd(&idt[T_NOEXTFLT], &ndptrap,  KCS_SEL, SDT_SYSIGT, TRP_KPL,
 935             0);

 937         /*
 938          * double fault handler.
 939          *
 940          * Note that on the hypervisor a guest does not receive #df faults.
 941          * Instead a failsafe event is injected into the guest if its selectors
 942          * and/or stack is in a broken state. See xen_failsafe_callback.
 943          */
 944 #if !defined(__xpv)
 945 #if defined(__amd64)

 947         set_gatesegd(&idt[T_DBLFLT], &syserrtrap, KCS_SEL, SDT_SYSIGT, TRP_KPL,
 948             T_DBLFLT);

 950 #elif defined(__i386)

 952         /*
 953          * task gate required.
 954          */
 955         set_gatesegd(&idt[T_DBLFLT], NULL, DFTSS_SEL, SDT_SYSTASKGT, TRP_KPL,
 956             0);

 958 #endif  /* __i386 */
 959 #endif  /* !__xpv */

 961         /*
 962          * T_EXTOVRFLT coprocessor-segment-overrun not supported.
 963          */

 965         set_gatesegd(&idt[T_TSSFLT], &invtsstrap, KCS_SEL, SDT_SYSIGT, TRP_KPL,
 966             0);
 967         set_gatesegd(&idt[T_SEGFLT], &segnptrap, KCS_SEL, SDT_SYSIGT, TRP_KPL,
```

```
 968             0);
 969         set_gatesegd(&idt[T_STKFLT], &stktrap, KCS_SEL, SDT_SYSIGT, TRP_KPL, 0);
 970         set_gatesegd(&idt[T_GPFLT], &gptrap, KCS_SEL, SDT_SYSIGT, TRP_KPL, 0);
 971         set_gatesegd(&idt[T_PGFLT], &pftrap, KCS_SEL, SDT_SYSIGT, TRP_KPL, 0);
 972         set_gatesegd(&idt[T_EXTERRFLT], &ndperr, KCS_SEL, SDT_SYSIGT, TRP_KPL,
 973             0);
 974         set_gatesegd(&idt[T_ALIGNMENT], &achktrap, KCS_SEL, SDT_SYSIGT,
 975             TRP_KPL, 0);
 976         set_gatesegd(&idt[T_MCE], &mcetrap, KCS_SEL, SDT_SYSIGT, TRP_KPL, 0);
 977         set_gatesegd(&idt[T_SIMDFPE], &xmtrap, KCS_SEL, SDT_SYSIGT, TRP_KPL, 0);

 979         /*
 980          * install "int80" handler at, well, 0x80.
 981          */
 982         set_gatesegd(&idt0[T_INT80], &sys_int80, KCS_SEL, SDT_SYSIGT, TRP_UPL,
 983             0);

 985         /*
 986 #endif /* ! codereview */
 987          * install fast trap handler at 210.
 988          */
 989         set_gatesegd(&idt[T_FASTTRAP], &fasttrap, KCS_SEL, SDT_SYSIGT, TRP_UPL,
 990             0);

 992         /*
 993          * System call handler.
 994          */
 995 #if defined(__amd64)
 996         set_gatesegd(&idt[T_SYSCALLINT], &sys_syscall_int, KCS_SEL, SDT_SYSIGT,
 997             TRP_UPL, 0);

 999 #elif defined(__i386)
1000         set_gatesegd(&idt[T_SYSCALLINT], &sys_call, KCS_SEL, SDT_SYSIGT,
1001             TRP_UPL, 0);
1002 #endif  /* __i386 */

1004         /*
1005          * Install the DTrace interrupt handler for the pid provider.
1006          */
1007         set_gatesegd(&idt[T_DTRACE_RET], &dtrace_ret, KCS_SEL,
1008             SDT_SYSIGT, TRP_UPL, 0);

1010         /*
1011 -        * Prepare interposing descriptors for the branded "int80"
1012 -        * and syscall handlers and cache copies of the default
1013 -        * descriptors.
1014          */
1015         brand_tbl[0].ih_inum = T_INT80;
1016         brand_tbl[0].ih_default_desc = idt0[T_INT80];
1017         set_gatesegd(&(brand_tbl[0].ih_interp_desc), &brand_sys_int80, KCS_SEL,
1018             SDT_SYSIGT, TRP_UPL, 0);

1020         brand_tbl[1].ih_inum = T_SYSCALLINT;
1021         brand_tbl[1].ih_default_desc = idt0[T_SYSCALLINT];
 980          * Prepare interposing descriptor for the syscall handler
 981          * and cache copy of the default descriptor.
 982          */
 983         brand_tbl[0].ih_inum = T_SYSCALLINT;
 984         brand_tbl[0].ih_default_desc = idt0[T_SYSCALLINT];

1023 #if defined(__amd64)
1024         set_gatesegd(&(brand_tbl[1].ih_interp_desc), &brand_sys_syscall_int,
 987         set_gatesegd(&(brand_tbl[0].ih_interp_desc), &brand_sys_syscall_int,
1025             KCS_SEL, SDT_SYSIGT, TRP_UPL, 0);
1026 #elif defined(__i386)
1027         set_gatesegd(&(brand_tbl[1].ih_interp_desc), &brand_sys_call,
```



```
 990         set_gatesegd(&(brand_tbl[0].ih_interp_desc), &brand_sys_call,
1028             KCS_SEL, SDT_SYSIGT, TRP_UPL, 0);
1029 #endif  /* __i386 */

1031         brand_tbl[2].ih_inum = 0;
 994         brand_tbl[1].ih_inum = 0;
1032 }
______unchanged_portion_omitted_
```

```
**********************************************************
    2317 Tue Jan 14 16:17:31 2014
new/usr/src/uts/intel/ldlinux/Makefile
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# uts/intel/ldlinux/Makefile
#
# Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
#       This makefile drives the production of the ldlinux streams kernel
#       module.
#
#       intel architecture dependent
#


#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../..

#
#       Define the module and object file sets.
#
MODULE          = ldlinux
OBJECTS         = $(LDLINUX_OBJS:%=$(OBJS_DIR)/%)
LINTS           = $(LDLINUX_OBJS:%.o=$(LINTS_DIR)/%.ln)
ROOTMODULE      = $(USR_STRMOD_DIR)/$(MODULE)

#
#       Include common rules.
#
include $(UTSBASE)/intel/Makefile.intel

#
#       Define targets
#
ALL_TARGET      = $(BINARY)
LINT_TARGET     = $(MODULE).lint
INSTALL_TARGET  = $(BINARY) $(ROOTMODULE)

CPPFLAGS        += -I$(UTSBASE)/common/brand/lx

#
#       Overrides.
```



```
#
CFLAGS          += $(CCVERBOSE)

#
# For now, disable these lint checks; maintainers should endeavor
# to investigate and remove these for maximum lint coverage.
# Please do not carry these forward to new Makefiles.
#
LINTTAGS        += -erroff=E_BAD_PTR_CAST_ALIGN

#
#       Default build targets.
#
.KEEP_STATE:

def:            $(DEF_DEPS)

all:            $(ALL_DEPS)

clean:          $(CLEAN_DEPS)

clobber:        $(CLOBBER_DEPS)

lint:           $(LINT_DEPS)

modlintlib:     $(MODLINTLIB_DEPS)

clean.lint:     $(CLEAN_LINT_DEPS)

install:        $(INSTALL_DEPS)

#
#       Include common targets.
#
include $(UTSBASE)/intel/Makefile.targ

$(OBJS_DIR)/%.o:        $(UTSBASE)/common/brand/lx/io/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(LINTS_DIR)/%.ln:      $(UTSBASE)/common/brand/lx/io/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))
#endif /* ! codereview */
```

```
**********************************************************
    2361 Tue Jan 14 16:17:31 2014
new/usr/src/uts/intel/lx_afs/Makefile
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

#
#       This makefile drives the production of the lxproc file system
#       kernel module.
#
#       i86 architecture dependent
#


#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../..

#
#       Define the module and object file sets.
#
#       Note that the name of the actual filesystem is lx_afs and
#       not lx_autofs.  This is becase filesystem names are stupidly
#       limited to 8 characters.
#
MODULE          = lx_afs
OBJECTS         = $(LX_AUTOFS_OBJS:%=$(OBJS_DIR)/%)
LINTS           = $(LX_AUTOFS_OBJS:%.o=$(LINTS_DIR)/%.ln)
ROOTMODULE      = $(USR_FS_DIR)/$(MODULE)

INC_PATH        += -I$(UTSBASE)/common/brand/lx

#
#       Include common rules.
#
include $(UTSBASE)/intel/Makefile.intel

#
#       Define targets
#
ALL_TARGET      = $(BINARY)
```



```
LINT_TARGET     = $(MODULE).lint
INSTALL_TARGET  = $(BINARY) $(ROOTMODULE)

#
#       Overrides.
#
CFLAGS          += $(CCVERBOSE)
LDFLAGS         += -dy

#
# For now, disable these lint checks; maintainers should endeavor
# to investigate and remove these for maximum lint coverage.
# Please do not carry these forward to new Makefiles.
#
LINTTAGS        += -erroff=E_ASSIGN_NARROW_CONV

#
#       Default build targets.
#
.KEEP_STATE:

def:            $(DEF_DEPS)

all:            $(ALL_DEPS)

clean:          $(CLEAN_DEPS)

clobber:        $(CLOBBER_DEPS)

lint:           $(LINT_DEPS)

modlintlib:     $(MODLINTLIB_DEPS)

clean.lint:     $(CLEAN_LINT_DEPS)

install:        $(INSTALL_DEPS)

#
#       Include common targets.
#
include $(UTSBASE)/intel/Makefile.targ

#
#       Include brand-specific rules
#

include $(UTSBASE)/intel/lx_afs/Makefile.rules
#endif /* ! codereview */
```

```
**********************************************************
    1207 Tue Jan 14 16:17:31 2014
new/usr/src/uts/intel/lx_afs/Makefile.rules
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "%Z%%M% %I%     %E% SMI"
#

#
#       Section 1a: C object build rules
#
$(OBJS_DIR)/%.o:                $(UTSBASE)/common/brand/lx/autofs/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

#
#       Section 1b: Lint 'object' build rules.
#
$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/brand/lx/autofs/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))
#endif /* ! codereview */
```

```
**********************************************************
    2420 Tue Jan 14 16:17:32 2014
new/usr/src/uts/intel/lx_audio/Makefile
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# uts/intel/lx_audio/Makefile
#
# Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
#       This makefile drives the production of the lx_audio driver
#
#       intel architecture dependent
#


#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../..

#
#       Define the module and object file sets.
#
MODULE          = lx_audio
OBJECTS         = $(LX_AUDIO_OBJS:%=$(OBJS_DIR)/%)
LINTS           = $(LX_AUDIO_OBJS:%.o=$(LINTS_DIR)/%.ln)
ROOTMODULE      = $(USR_DRV_DIR)/$(MODULE)
CONF_SRCDIR     = $(UTSBASE)/common/brand/lx/io

#
#       Include common rules.
#
include $(UTSBASE)/intel/Makefile.intel

#
#       Define targets
#
ALL_TARGET      = $(BINARY) $(SRC_CONFILE)
LINT_TARGET     = $(MODULE).lint
INSTALL_TARGET  = $(BINARY) $(ROOTMODULE) $(ROOT_CONFFILE)

CPPFLAGS        += -I$(UTSBASE)/common/brand/lx

#
# For now, disable these lint checks; maintainers should endeavor
```



```
# to investigate and remove these for maximum lint coverage.
# Please do not carry these forward to new Makefiles.
#
LINTTAGS        += -erroff=E_BAD_PTR_CAST_ALIGN
LINTTAGS        += -erroff=E_ASSIGN_NARROW_CONV
LINTTAGS        += -erroff=E_SUSPICIOUS_COMPARISON

#
#       Default build targets.
#
.KEEP_STATE:

def:            $(DEF_DEPS)

all:            $(ALL_DEPS)

clean:          $(CLEAN_DEPS)

clobber:        $(CLOBBER_DEPS)

lint:           $(LINT_DEPS)

modlintlib:     $(MODLINTLIB_DEPS)

clean.lint:     $(CLEAN_LINT_DEPS)

install:        $(INSTALL_DEPS)

#
#       Include common targets.
#
include $(UTSBASE)/intel/Makefile.targ

$(OBJS_DIR)/%.o:        $(UTSBASE)/common/brand/lx/io/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(LINTS_DIR)/%.ln:      $(UTSBASE)/common/brand/lx/io/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))
#endif /* ! codereview */
```

```
**********************************************************
    2401 Tue Jan 14 16:17:32 2014
new/usr/src/uts/intel/lx_brand/Makefile
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
#       This makefile drives the production of the kernel component of
#       the lx brand
#


#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE         = ../..

#
#       Path to where brand common sources live
#
LX_CMN          = $(SRC)/common/brand/lx

#
#       Define the module and object file sets.
#
MODULE          = lx_brand
OBJECTS         = $(LX_BRAND_OBJS:%=$(OBJS_DIR)/%)
LINTS           = $(LX_BRAND_OBJS:%.o=$(LINTS_DIR)/%.ln)
ROOTMODULE      = $(USR_BRAND_DIR)/$(MODULE)

#
#       Include common rules.
#
include $(UTSBASE)/intel/Makefile.intel

#
#       Define targets
#
ALL_TARGET      = $(BINARY)
LINT_TARGET     = $(MODULE).lint
INSTALL_TARGET  = $(BINARY) $(ROOTMODULE)

INC_PATH        += -I$(UTSBASE)/common/brand/lx -I$(LX_CMN)
AS_INC_PATH     += -I$(UTSBASE)/i86pc/genassym/$(OBJS_DIR)
```



```
#
# lint pass one enforcement
#
CFLAGS          += $(CCVERBOSE)

LDFLAGS         += -dy -Nexec/elfexec

#
# For now, disable these lint checks; maintainers should endeavor
# to investigate and remove these for maximum lint coverage.
# Please do not carry these forward to new Makefiles.
#
LINTTAGS        += -erroff=E_ASSIGN_NARROW_CONV
LINTTAGS        += -erroff=E_SUSPICIOUS_COMPARISON

#
#       Default build targets.
#
.KEEP_STATE:

def:            $(DEF_DEPS)

all:            $(ALL_DEPS)

clean:          $(CLEAN_DEPS)

clobber:        $(CLOBBER_DEPS)

lint:           $(LINT_DEPS)

modlintlib:     $(MODLINTLIB_DEPS)

clean.lint:     $(CLEAN_LINT_DEPS)

install:        $(INSTALL_DEPS)

#
#       Include common targets.
#
include $(UTSBASE)/intel/Makefile.targ

#
#       Include brand-specific rules
#

include $(UTSBASE)/intel/lx_brand/Makefile.rules
#endif /* ! codereview */
```

```
**********************************************************
    2428 Tue Jan 14 16:17:32 2014
new/usr/src/uts/intel/lx_brand/Makefile.rules
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
#ident  "%Z%%M% %I%     %E% SMI"
#

#
#       Section 1a: C object build rules
#
$(OBJS_DIR_OBJ64)/%.o:          $(UTSBASE)/common/brand/lx/os/%.c
        $(COMPILE.c) -D_ELF32_COMPAT -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR_DBG64)/%.o:          $(UTSBASE)/common/brand/lx/os/%.c
        $(COMPILE.c) -D_ELF32_COMPAT -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR_OBJ64)/%.o:          $(UTSBASE)/common/brand/lx/syscall/%.c
        $(COMPILE.c) -D_ELF32_COMPAT -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR_DBG64)/%.o:          $(UTSBASE)/common/brand/lx/syscall/%.c
        $(COMPILE.c) -D_ELF32_COMPAT -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR_OBJ64)/%.o:          $(UTSBASE)/intel/brand/lx/%.s
        $(COMPILE.s) -D_ELF32_COMPAT -o $@ $<

$(OBJS_DIR_OBJ64)/%.o:          $(LX_CMN)/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR_DBG64)/%.o:          $(UTSBASE)/intel/brand/lx/%.s
        $(COMPILE.s) -D_ELF32_COMPAT -o $@ $<

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/brand/lx/os/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/brand/lx/syscall/%.c
```



```
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(LX_CMN)/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/brand/lx/%.s
        $(COMPILE.s) -o $@ $<

#
#       Section 1b: Lint ‘object’ build rules.
#
$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/brand/lx/os/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/brand/lx/syscall/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(LX_CMN)/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/brand/lx/%.s
        @($(LHEAD) $(LINT.s) $< $(LTAIL))
#endif /* ! codereview */
```

```
**********************************************************
    2432 Tue Jan 14 16:17:32 2014
new/usr/src/uts/intel/lx_proc/Makefile
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# uts/intel/lx_proc/Makefile
#
# Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
#       This makefile drives the production of the lxproc file system
#       kernel module.
#
#       i86 architecture dependent
#


#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../..

#
#       Path to where brand common sources live
#
LX_CMN          = $(SRC)/common/brand/lx

#
#       Define the module and object file sets.
#
MODULE          = lx_proc
OBJECTS         = $(LX_PROC_OBJS:%=$(OBJS_DIR)/%)
LINTS           = $(LX_PROC_OBJS:%.o=$(LINTS_DIR)/%.ln)
ROOTMODULE      = $(USR_FS_DIR)/$(MODULE)

INC_PATH        += -I$(UTSBASE)/common/brand/lx -I$(LX_CMN)

#
#       Include common rules.
#
include $(UTSBASE)/intel/Makefile.intel

#
#       Define targets
#
ALL_TARGET      = $(BINARY)
```



```
LINT_TARGET     = $(MODULE).lint
INSTALL_TARGET  = $(BINARY) $(ROOTMODULE)

#
#       Overrides.
#
CFLAGS          += $(CCVERBOSE)

#
#       Depends on procfs and lx_brand
#
LDFLAGS         += -dy -Nfs/procfs -Nbrand/lx_brand

#
# For now, disable these lint checks; maintainers should endeavor
# to investigate and remove these for maximum lint coverage.
# Please do not carry these forward to new Makefiles.
#
LINTTAGS        += -erroff=E_PTRDIFF_OVERFLOW
LINTTAGS        += -erroff=E_ASSIGN_NARROW_CONV

#
#       Default build targets.
#
.KEEP_STATE:

def:            $(DEF_DEPS)

all:            $(ALL_DEPS)

clean:          $(CLEAN_DEPS)

clobber:        $(CLOBBER_DEPS)

lint:           $(LINT_DEPS)

modlintlib:     $(MODLINTLIB_DEPS)

clean.lint:     $(CLEAN_LINT_DEPS)

install:        $(INSTALL_DEPS)

#
#       Include common targets.
#
include $(UTSBASE)/intel/Makefile.targ

#
#       Include brand-specific rules
#

include $(UTSBASE)/intel/lx_proc/Makefile.rules
#endif /* ! codereview */
```

```
**********************************************************
    1203 Tue Jan 14 16:17:32 2014
new/usr/src/uts/intel/lx_proc/Makefile.rules
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
#ident  "%Z%%M% %I%     %E% SMI"

#
#       Section 1a: C object build rules
#
$(OBJS_DIR)/%.o:                $(UTSBASE)/common/brand/lx/procfs/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

#
#       Section 1b: Lint ‘object’ build rules.
#
$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/brand/lx/procfs/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))
#endif /* ! codereview */
```

```
**********************************************************
    2098 Tue Jan 14 16:17:32 2014
new/usr/src/uts/intel/lx_ptm/Makefile
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# uts/intel/lx_ptm/Makefile
#
# Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
#       This makefile drives the production of the lx_ptm driver
#
#       intel architecture dependent
#

#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../..

#
#       Define the module and object file sets.
#
MODULE          = lx_ptm
OBJECTS         = $(LX_PTM_OBJS:%=$(OBJS_DIR)/%)
LINTS           = $(LX_PTM_OBJS:%.o=$(LINTS_DIR)/%.ln)
ROOTMODULE      = $(USR_DRV_DIR)/$(MODULE)
CONF_SRCDIR     = $(UTSBASE)/common/brand/lx/io

#
#       Include common rules.
#
include $(UTSBASE)/intel/Makefile.intel

#
#       Define targets
#
ALL_TARGET      = $(BINARY) $(SRC_CONFILE)
LINT_TARGET     = $(MODULE).lint
INSTALL_TARGET  = $(BINARY) $(ROOTMODULE) $(ROOT_CONFFILE)

CPPFLAGS        += -I$(UTSBASE)/common/brand/lx

#
#       Default build targets.
```



```
#
.KEEP_STATE:

def:            $(DEF_DEPS)

all:            $(ALL_DEPS)

clean:          $(CLEAN_DEPS)

clobber:        $(CLOBBER_DEPS)

lint:           $(LINT_DEPS)

modlintlib:     $(MODLINTLIB_DEPS)

clean.lint:     $(CLEAN_LINT_DEPS)

install:        $(INSTALL_DEPS)

#
#       Include common targets.
#
include $(UTSBASE)/intel/Makefile.targ

$(OBJS_DIR)/%.o:        $(UTSBASE)/common/brand/lx/io/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(LINTS_DIR)/%.ln:      $(UTSBASE)/common/brand/lx/io/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))
#endif /* ! codereview */
```

```
**********************************************************
    2179 Tue Jan 14 16:17:33 2014
new/usr/src/uts/intel/lx_systrace/Makefile
Bring back LX zones.
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

UTSBASE = ../..

MODULE          = lx_systrace
OBJECTS         = $(LX_SYSTRACE_OBJS:%=$(OBJS_DIR)/%)
LINTS           = $(LX_SYSTRACE_OBJS:%.o=$(LINTS_DIR)/%.ln)
ROOTMODULE      = $(USR_DRV_DIR)/$(MODULE)
ROOTLINK        = $(USR_DTRACE_DIR)/$(MODULE)
CONF_SRCDIR     = $(UTSBASE)/common/brand/lx/dtrace

include $(UTSBASE)/intel/Makefile.intel

ALL_TARGET      = $(BINARY) $(SRC_CONFILE)
LINT_TARGET     = $(MODULE).lint
INSTALL_TARGET  = $(BINARY) $(ROOTMODULE) $(ROOTLINK) $(ROOT_CONFFILE)

CPPFLAGS += -I$(UTSBASE)/common/brand/lx

LDFLAGS         += -dy -Ndrv/dtrace -Nbrand/lx_brand

#
# For now, disable these lint checks; maintainers should endeavor
# to investigate and remove these for maximum lint coverage.
# Please do not carry these forward to new Makefiles.
#
LINTTAGS        += -erroff=E_STATIC_UNUSED

.KEEP_STATE:

def:            $(DEF_DEPS)

all:            $(ALL_DEPS)

clean:          $(CLEAN_DEPS)

clobber:        $(CLOBBER_DEPS)
```



```
lint:           $(LINT_DEPS)

modlintlib:     $(MODLINTLIB_DEPS)

clean.lint:     $(CLEAN_LINT_DEPS)

install:        $(INSTALL_DEPS)

$(ROOTLINK):    $(USR_DTRACE_DIR) $(ROOTMODULE)
        -$(RM) $@; ln $(ROOTMODULE) $@

include $(UTSBASE)/intel/Makefile.targ

$(OBJS_DIR)/%.o:        $(UTSBASE)/common/brand/lx/dtrace/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(LINTS_DIR)/%.ln:      $(UTSBASE)/common/brand/lx/dtrace/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))
#endif /* ! codereview */
```

```
**********************************************************
    1486 Tue Jan 14 16:17:33 2014
new/usr/src/uts/intel/sys/machbrand.h
Bring back LX zones.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright (c) 2006, 2010, Oracle and/or its affiliates. All rights reserved.
 */

#ifndef _SYS_MACHBRAND_H
#define _SYS_MACHBRAND_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

#endif /* ! codereview */
#ifdef __cplusplus
extern "C" {
#endif

#ifndef _ASM

#include <sys/model.h>

struct brand_mach_ops {
        void    (*b_sysenter)(void);
        void    (*b_int80)(void);
#endif /* ! codereview */
        void    (*b_int91)(void);
        void    (*b_syscall)(void);
        void    (*b_syscall32)(void);
        greg_t  (*b_fixsegreg)(greg_t, model_t);
#endif /* ! codereview */
};

#endif  /* _ASM */

#define BRAND_CB_SYSENTER       0
#define BRAND_CB_INT80          1
#define BRAND_CB_INT91          2
#define BRAND_CB_SYSCALL        3
#define BRAND_CB_SYSCALL32      4
#define BRAND_CB_INT91          1
#define BRAND_CB_SYSCALL        2
#define BRAND_CB_SYSCALL32      3

#ifdef  __cplusplus
```



```
}
______unchanged_portion_omitted_
```

```
**********************************************************
   24269 Tue Jan 14 16:17:33 2014
new/usr/src/uts/intel/sys/segments.h
Bring back LX zones.
**********************************************************
______unchanged_portion_omitted_

 383 #define GATESEG_GETOFFSET(sgd)  ((uintptr_t)((sgd)->sgd_looffset |      \
 384                                 (sgd)->sgd_hioffset << 16 |             \
 385                                 (uint64_t)((sgd)->sgd_hi64offset) << 32))

 387 #endif  /* __amd64 */

 389 /*
 390  * functions for initializing and updating segment descriptors.
 391  */
 392 #if defined(__amd64)

 394 extern void set_usegd(user_desc_t *, uint_t, void *, size_t, uint_t, uint_t,
 395     uint_t, uint_t);

 397 #elif defined(__i386)

 399 extern void set_usegd(user_desc_t *, void *, size_t, uint_t, uint_t,
 400     uint_t, uint_t);

 402 #endif  /* __i386 */

 404 extern void set_gatesegd(gate_desc_t *, void (*)(void), selector_t,
 405     uint_t, uint_t, uint_t);

 407 extern void set_syssegd(system_desc_t *, void *, size_t, uint_t, uint_t);

 409 extern void *get_ssd_base(system_desc_t *);

 411 extern void gdt_update_usegd(uint_t, user_desc_t *);

 413 extern int ldt_update_segd(user_desc_t *, user_desc_t *);

 415 #if defined(__xpv)

 417 extern int xen_idt_to_trap_info(uint_t, gate_desc_t *, void *);
 418 extern void xen_idt_write(gate_desc_t *, uint_t);

 420 #endif  /* __xen */

 422 void init_boot_gdt(user_desc_t *);

 424 #endif  /* _ASM */

 426 /*
 427  * Common segment parameter defintions for granularity, default
 428  * operand size and operaton mode.
 429  */
 430 #define SDP_BYTES       0       /* segment limit scaled to bytes */
 431 #define SDP_PAGES       1       /* segment limit scaled to pages */
 432 #define SDP_OP32        1       /* code and data default operand = 32 bits */
 433 #define SDP_LONG        1       /* long mode code segment (64 bits) */
 434 #define SDP_SHORT       0       /* compat/legacy code segment (32 bits) */
 435 /*
 436  * System segments and gate types.
 437  *
 438  * In long mode i386 32-bit ldt, tss, call, interrupt and trap gate
 439  * types are redefined into 64-bit equivalents.
 440  */
 441 #define SDT_SYSNULL      0      /* system null */
```



```
 442 #define SDT_SYS286TSS    1      /* system 286 TSS available */
 443 #define SDT_SYSLDT       2      /* system local descriptor table */
 444 #define SDT_SYS286BSY    3      /* system 286 TSS busy */
 445 #define SDT_SYS286CGT    4      /* system 286 call gate */
 446 #define SDT_SYSTASKGT    5      /* system task gate */
 447 #define SDT_SYS286IGT    6      /* system 286 interrupt gate */
 448 #define SDT_SYS286TGT    7      /* system 286 trap gate */
 449 #define SDT_SYSNULL2     8      /* system null again */
 450 #define SDT_SYSTSS       9      /* system TSS available */
 451 #define SDT_SYSNULL3    10      /* system null again */
 452 #define SDT_SYSTSSBSY   11      /* system TSS busy */
 453 #define SDT_SYSCGT      12      /* system call gate */
 454 #define SDT_SYSNULL4    13      /* system null again */
 455 #define SDT_SYSIGT      14      /* system interrupt gate */
 456 #define SDT_SYSTGT      15      /* system trap gate */

 458 /*
 459  * Memory segment types.
 460  *
 461  * While in long mode expand-down, writable and accessed type field
 462  * attributes are ignored. Only the conforming bit is loaded by hardware
 463  * for long mode code segment descriptors.
 464  */
 465 #define SDT_MEMRO       16      /* read only */
 466 #define SDT_MEMROA      17      /* read only accessed */
 467 #define SDT_MEMRW       18      /* read write */
 468 #define SDT_MEMRWA      19      /* read write accessed */
 469 #define SDT_MEMROD      20      /* read only expand dwn limit */
 470 #define SDT_MEMRODA     21      /* read only expand dwn limit accessed */
 471 #define SDT_MEMRWD      22      /* read write expand dwn limit */
 472 #define SDT_MEMRWDA     23      /* read write expand dwn limit accessed */
 473 #define SDT_MEME        24      /* execute only */
 474 #define SDT_MEMEA       25      /* execute only accessed */
 475 #define SDT_MEMER       26      /* execute read */
 476 #define SDT_MEMERA      27      /* execute read accessed */
 477 #define SDT_MEMEC       28      /* execute only conforming */
 478 #define SDT_MEMEAC      29      /* execute only accessed conforming */
 479 #define SDT_MEMERC      30      /* execute read conforming */
 480 #define SDT_MEMERAC     31      /* execute read accessed conforming */

 482 /*
 483  * Entries in the Interrupt Descriptor Table (IDT)
 484  */
 485 #define IDT_DE          0       /* #DE: Divide Error */
 486 #define IDT_DB          1       /* #DB: Debug */
 487 #define IDT_NMI         2       /* Nonmaskable External Interrupt */
 488 #define IDT_BP          3       /* #BP: Breakpoint */
 489 #define IDT_OF          4       /* #OF: Overflow */
 490 #define IDT_BR          5       /* #BR: Bound Range Exceeded */
 491 #define IDT_UD          6       /* #UD: Undefined/Invalid Opcode */
 492 #define IDT_NM          7       /* #NM: No Math Coprocessor */
 493 #define IDT_DF          8       /* #DF: Double Fault */
 494 #define IDT_FPUGP       9       /* Coprocessor Segment Overrun */
 495 #define IDT_TS          10      /* #TS: Invalid TSS */
 496 #define IDT_NP          11      /* #NP: Segment Not Present */
 497 #define IDT_SS          12      /* #SS: Stack Segment Fault */
 498 #define IDT_GP          13      /* #GP: General Protection Fault */
 499 #define IDT_PF          14      /* #PF: Page Fault */
 500 #define IDT_MF          16      /* #MF: FPU Floating-Point Error */
 501 #define IDT_AC          17      /* #AC: Alignment Check */
 502 #define IDT_MC          18      /* #MC: Machine Check */
 503 #define IDT_XF          19      /* #XF: SIMD Floating-Point Exception */
 504 #define NIDT            256     /* size in entries of IDT */

 506 /*
 507  * Entries in the Global Descriptor Table (GDT)
```

```
 508  *
 509  * We make sure to space the system descriptors (LDT's, TSS')
 510  * such that they are double gdt slot aligned. This is because
 511  * in long mode system segment decriptors expand to 128 bits.
 512  *
 513  * GDT_LWPFS and GDT_LWPGS must be the same for both 32 and 64-bit
 514  * kernels. See setup_context in libc. 64-bit processes must set
 515  * %fs or %gs to null selector to use 64-bit fsbase or gsbase
 516  * respectively.
 517  */
 518 #define GDT_NULL        0       /* null */
 519 #define GDT_B32DATA     1       /* dboot 32 bit data descriptor */
 520 #define GDT_B32CODE     2       /* dboot 32 bit code descriptor */
 521 #define GDT_B16CODE     3       /* bios call 16 bit code descriptor */
 522 #define GDT_B16DATA     4       /* bios call 16 bit data descriptor */
 523 #define GDT_B64CODE     5       /* dboot 64 bit code descriptor */
 524 #define GDT_BGSTMP      7       /* kmdb descriptor only used early in boot */

 526 #if defined(__amd64)

 528 #define GDT_KCODE       6       /* kernel code seg %cs */
 529 #define GDT_KDATA       7       /* kernel data seg %ds */
 530 #define GDT_U32CODE     8       /* 32-bit process on 64-bit kernel %cs */
 531 #define GDT_UDATA       9       /* user data seg %ds (32 and 64 bit) */
 532 #define GDT_UCODE       10      /* native user code  seg %cs */
 533 #define GDT_LDT         12      /* LDT for current process */
 534 #define GDT_KTSS        14      /* kernel tss */
 535 #define GDT_FS          GDT_NULL /* kernel %fs segment selector */
 536 #define GDT_GS          GDT_NULL /* kernel %gs segment selector */
 537 #define GDT_LWPFS       55      /* lwp private %fs segment selector (32-bit) */
 538 #define GDT_LWPGS       56      /* lwp private %gs segment selector (32-bit) */
 539 #define GDT_BRANDMIN    57      /* first entry in GDT for brand usage */
 540 #define GDT_BRANDMAX    61      /* last entry in GDT for brand usage */
 541 #define NGDT            62      /* number of entries in GDT */

 543 /*
 544  * This selector is only used in the temporary GDT used to bring additional
 545  * CPUs from 16-bit real mode into long mode in real_mode_start().
 546  */
 547 #define TEMPGDT_KCODE64 1       /* 64-bit code selector */

 549 #elif defined(__i386)

 551 #define GDT_LDT         40      /* LDT for current process */
 552 #define GDT_KTSS        42      /* kernel tss */
 553 #define GDT_KCODE       43      /* kernel code seg %cs */
 554 #define GDT_KDATA       44      /* kernel data seg %ds */
 555 #define GDT_UCODE       45      /* native user code  seg %cs */
 556 #define GDT_UDATA       46      /* user data seg %ds (32 and 64 bit) */
 557 #define GDT_DBFLT       47      /* double fault #DF selector */
 558 #define GDT_FS          53      /* kernel %fs segment selector */
 559 #define GDT_GS          54      /* kernel %gs segment selector */
 560 #define GDT_LWPFS       55      /* lwp private %fs segment selector */
 561 #define GDT_LWPGS       56      /* lwp private %gs segment selector */
 562 #define GDT_BRANDMIN    57      /* first entry in GDT for brand usage */
 563 #define GDT_BRANDMAX    61      /* last entry in GDT for brand usage */
 564 #if !defined(__xpv)
 565 #define NGDT            90      /* number of entries in GDT */
 566 #else
 567 #define NGDT            512     /* single 4K page for the hypervisor */
 568 #endif

 570 #endif  /* __i386 */

 572 /*
 573  * Convenient selector definitions.
```



```
 574  */

 576 /*
 577  * XXPV 64 bit Xen only allows the guest %cs/%ss be the private ones it
 578  * provides, not the ones we create for ourselves.  See FLAT_RING3_CS64 in
 579  * public/arch-x86_64.h
 580  *
 581  * 64-bit Xen runs paravirtual guests in ring 3 but emulates them running in
 582  * ring 0 by clearing CPL in %cs value pushed on guest exception stacks.
 583  * Therefore we will have KCS_SEL value indicate ring 0 and use that everywhere
 584  * in the kernel. But in the few files where we initialize segment registers or
 585  * create and update descriptors we will explicity OR in SEL_KPL (ring 3) for
 586  * kernel %cs. See desctbls.c for an example.
 587  */

 589 #if defined(__xpv) && defined(__amd64)
 590 #define KCS_SEL         0xe030          /* FLAT_RING3_CS64 & 0xFFF0 */
 591 #define KDS_SEL         0xe02b          /* FLAT_RING3_SS64 */
 592 #else
 593 #define KCS_SEL         SEL_GDT(GDT_KCODE, SEL_KPL)
 594 #define KDS_SEL         SEL_GDT(GDT_KDATA, SEL_KPL)
 595 #endif

 597 #define UCS_SEL         SEL_GDT(GDT_UCODE, SEL_UPL)
 598 #if defined(__amd64)
 599 #define TEMP_CS64_SEL   SEL_GDT(TEMPGDT_KCODE64, SEL_KPL)
 600 #define U32CS_SEL       SEL_GDT(GDT_U32CODE, SEL_UPL)
 601 #endif

 603 #define UDS_SEL         SEL_GDT(GDT_UDATA, SEL_UPL)
 604 #define ULDT_SEL        SEL_GDT(GDT_LDT, SEL_KPL)
 605 #define KTSS_SEL        SEL_GDT(GDT_KTSS, SEL_KPL)
 606 #define DFTSS_SEL       SEL_GDT(GDT_DBFLT, SEL_KPL)
 607 #define KFS_SEL         0
 608 #define KGS_SEL         SEL_GDT(GDT_GS, SEL_KPL)
 609 #define LWPFS_SEL       SEL_GDT(GDT_LWPFS, SEL_UPL)
 610 #define LWPGS_SEL       SEL_GDT(GDT_LWPGS, SEL_UPL)
 611 #define BRANDMIN_SEL    SEL_GDT(GDT_BRANDMIN, SEL_UPL)
 612 #define BRANDMAX_SEL    SEL_GDT(GDT_BRANDMAX, SEL_UPL)

 614 #define B64CODE_SEL     SEL_GDT(GDT_B64CODE, SEL_KPL)
 615 #define B32CODE_SEL     SEL_GDT(GDT_B32CODE, SEL_KPL)
 616 #define B32DATA_SEL     SEL_GDT(GDT_B32DATA, SEL_KPL)
 617 #define B16CODE_SEL     SEL_GDT(GDT_B16CODE, SEL_KPL)
 618 #define B16DATA_SEL     SEL_GDT(GDT_B16DATA, SEL_KPL)

 620 /*
 621  * Temporary %gs descriptor used by kmdb with -d option. Only lives
 622  * in boot's GDT and is not copied into kernel's GDT from boot.
 623  */
 624 #define KMDBGS_SEL      SEL_GDT(GDT_BGSTMP, SEL_KPL)

 626 /*
 627  * Selector used for kdi_idt when kmdb has taken over the IDT.
 628  */
 629 #if defined(__amd64)
 630 #define KMDBCODE_SEL    B64CODE_SEL
 631 #else
 632 #define KMDBCODE_SEL    B32CODE_SEL
 633 #endif

 635 /*
 636  * Entries in default Local Descriptor Table (LDT) for every process.
 637  */
 638 #define LDT_SYSCALL     0       /* call gate for libc.a (obsolete) */
 639 #define LDT_SIGCALL     1       /* EOL me, call gate for static sigreturn */
```

```
640 #define LDT_RESVD1      2       /* old user %cs */
641 #define LDT_RESVD2      3       /* old user %ds */
642 #define LDT_ALTSYSCALL  4       /* alternate call gate for system calls */
643 #define LDT_ALTSIGCALL  5       /* EOL me, alternate call gate for sigreturn */
644 #define LDT_UDBASE      6       /* user descriptor base index */
645 #define MINNLDT         512     /* Current min solaris ldt size (1 4K page) */
646 #define MAXNLDT         8192    /* max solaris ldt size (16 4K pages) */

648 #ifndef _ASM

650 extern  gate_desc_t     *idt0;
651 extern  desctbr_t       idt0_default_reg;
652 extern  user_desc_t     *gdt0;

654 extern user_desc_t      zero_udesc;
655 extern user_desc_t      null_udesc;
656 extern system_desc_t    null_sdesc;

658 #if defined(__amd64)
659 extern user_desc_t      zero_u32desc;
660 #endif
661 #if defined(__amd64)
662 extern user_desc_t      ucs_on;
663 extern user_desc_t      ucs_off;
664 extern user_desc_t      ucs32_on;
665 extern user_desc_t      ucs32_off;
666 #endif  /* __amd64 */

668 extern tss_t *ktss0;

670 #if defined(__i386)
671 extern tss_t *dftss0;
672 #endif  /* __i386 */

674 extern void div0trap(), dbgtrap(), nmiint(), brktrap(), ovflotrap();
675 extern void boundstrap(), invoptrap(), ndptrap();
676 #if !defined(__xpv)
677 extern void syserrtrap();
678 #endif
679 extern void invaltrap(), invtsstrap(), segnptrap(), stktrap();
680 extern void gptrap(), pftrap(), ndperr();
681 extern void overrun(), resvtrap();
682 extern void _start(), cmnint();
683 extern void achktrap(), mcetrap();
684 extern void xmtrap();
685 extern void fasttrap();
686 extern void sys_int80();
687 extern void brand_sys_int80();
688 #endif /* ! codereview */
689 extern void dtrace_ret();

691 #if !defined(__amd64)
692 extern void pentium_pftrap();
693 #endif

695 #endif /* _ASM */

697 #ifdef  __cplusplus
698 }
699 #endif

701 #endif /* _SYS_SEGMENTS_H */
```