
1725 Tue May 7 04:49:55 2019

new/exception_lists/keywords

OS-7125 Need mitigation of L1TF (CVE-2018-3646)

Reviewed by: Robert Mustacchi <rm@joyent.com>

Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>

```
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 # Copyright (c) 2009, 2010, Oracle and/or its affiliates. All rights reserved.
22 # Copyright 2017 Nexenta Systems, Inc.
23 # Copyright (c) 2013 by Delphix. All rights reserved.
24 # Copyright 2016 Toomas Soome <tsoome@me.com>
25 # Copyright 2018 Joyent, Inc.
26 #

28 syntax: glob
29 usr/src/cmd/boot/common/bblk_einfo.c
30 usr/src/cmd/zonestat/zonestat/zonestat.c
31 usr/src/contrib/*
32 usr/src/data/locale/data/zh_CN.UTF-8.src
33 usr/src/data/locale/data/zh_HK.UTF-8.src
34 usr/src/data/locale/data/zh_MO.UTF-8.src
35 usr/src/data/locale/data/zh_SG.UTF-8.src
36 usr/src/data/locale/data/zh_TW.UTF-8.src
37 usr/src/data/terminfo/termcap.src
38 usr/src/data/terminfo/terminfo.src
39 usr/src/data/ucode/amd/*
40 usr/src/data/ucode/intel/*
41 usr/src/test/zfs-tests/tests/functional/cli_root/zpool_upgrade/blockfiles/zfs-po
42 usr/src/test/zfs-tests/tests/functional/delegate/delegate_common.kshlib
43 usr/src/test/test-runner/cmd/run
44 usr/src/uts/common/io/iwn/fw-iw/*.ucode
```

new/usr/src/pkg/manifests/system-header.mf

1

```
*****
96439 Tue May 7 04:49:55 2019
new/usr/src/pkg/manifests/system-header.mf
OS-7125 Need mitigation of LITF (CVE-2018-3646)
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
*****
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 #
23 # Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
24 # Copyright (c) 2012 by Delphix. All rights reserved.
25 # Copyright 2013 Saso Kiselkov. All rights reserved.
26 # Copyright 2014 Garrett D'Amore <garrett@damore.org>
27 # Copyright 2018 Nexenta Systems, Inc.
28 # Copyright 2016 Hans Rosenfeld <rosenfeld@grumpf.hope-2000.org>
29 # Copyright 2019 Peter Tribble.
30 #
31 #
32 set name=pkg.fmri value=pkg:/system/header@$(PKGVERS)
33 set name=pkg.description \
34 value="SunOS C/C++ header files for general development of software"
35 set name=pkg.summary value="SunOS Header Files"
36 set name=info.classification value=org.opensolaris.category.2008:System/Core
37 set name=variant.arch value=$(ARCH)
38 dir path=usr group=sys
39 dir path=usr/include
40 $(i386_ONLY)dir path=usr/include/${ARCH64}
41 $(i386_ONLY)dir path=usr/include/${ARCH64}/sys
42 dir path=usr/include/ads
43 dir path=usr/include/arpa
44 dir path=usr/include/asm
45 dir path=usr/include/ast
46 dir path=usr/include/bsm
47 dir path=usr/include/dat
48 dir path=usr/include/des
49 dir path=usr/include/gssapi
50 dir path=usr/include/hal
51 $(i386_ONLY)dir path=usr/include/ia32
52 $(i386_ONLY)dir path=usr/include/ia32/sys
53 dir path=usr/include/inet
54 dir path=usr/include/inet/kssl
55 dir path=usr/include/ipp
56 dir path=usr/include/ipp/ipgpc
57 dir path=usr/include/iso
58 dir path=usr/include/kerberosv5
59 dir path=usr/include/libpolkit
```

new/usr/src/pkg/manifests/system-header.mf

2

```
60 dir path=usr/include/net
61 dir path=usr/include/netinet
62 dir path=usr/include/nfs
63 dir path=usr/include/protocols
64 dir path=usr/include/rpc
65 dir path=usr/include/rpcsvc
66 dir path=usr/include/sasl
67 dir path=usr/include/scsi
68 dir path=usr/include/scsi/plugins
69 dir path=usr/include/scsi/plugins/ses
70 dir path=usr/include/scsi/plugins/ses/framework
71 dir path=usr/include/scsi/plugins/ses/vendor
72 dir path=usr/include/scsi/plugins/smp
73 dir path=usr/include/scsi/plugins/smp/engine
74 dir path=usr/include/scsi/plugins/smp/framework
75 dir path=usr/include/security
76 dir path=usr/include/sharefs
77 dir path=usr/include/sys
78 dir path=usr/include/sys/av
79 dir path=usr/include/sys/contract
80 dir path=usr/include/sys/crypto
81 dir path=usr/include/sys/dktp
82 dir path=usr/include/sys/fc4
83 dir path=usr/include/sys/fm
84 dir path=usr/include/sys/fm/cpu
85 dir path=usr/include/sys/fm/fs
86 dir path=usr/include/sys/fm/io
87 $(sparc_ONLY)dir path=usr/include/sys/fpu
88 dir path=usr/include/sys/fs
89 dir path=usr/include/sys/hotplug
90 dir path=usr/include/sys/hotplug/pci
91 dir path=usr/include/sys/ib
92 dir path=usr/include/sys/ib/adapters
93 dir path=usr/include/sys/ib/adapters/hermon
94 dir path=usr/include/sys/ib/adapters/tavor
95 dir path=usr/include/sys/ib/clients
96 dir path=usr/include/sys/ib/clients/ibd
97 dir path=usr/include/sys/ib/clients/of
98 dir path=usr/include/sys/ib/clients/of/rdma
99 dir path=usr/include/sys/ib/clients/of/sol_ofs
100 dir path=usr/include/sys/ib/clients/of/sol_ucma
101 dir path=usr/include/sys/ib/clients/of/sol_umad
102 dir path=usr/include/sys/ib/clients/of/sol_uverbs
103 dir path=usr/include/sys/ib/ibnexus
104 dir path=usr/include/sys/ib/ibtl
105 dir path=usr/include/sys/ib/ibtl/impl
106 dir path=usr/include/sys/ib/mgt
107 dir path=usr/include/sys/ib/mgt/ibmf
108 dir path=usr/include/sys/iso
109 dir path=usr/include/sys/proc
110 dir path=usr/include/sys/rsm
111 $(i386_ONLY)dir path=usr/include/sys/sata group=sys
112 dir path=usr/include/sys/scsi
113 dir path=usr/include/sys/scsi/adapters
114 dir path=usr/include/sys/scsi/conf
115 dir path=usr/include/sys/scsi/generic
116 dir path=usr/include/sys/scsi/impl
117 dir path=usr/include/sys/scsi/targets
118 dir path=usr/include/sys/sysevent
119 dir path=usr/include/sys/tsol
120 dir path=usr/include/tsol
121 dir path=usr/include/uuid
122 $(sparc_ONLY)dir path=usr/include/v7
123 $(sparc_ONLY)dir path=usr/include/v7/sys
124 $(sparc_ONLY)dir path=usr/include/v9
125 $(sparc_ONLY)dir path=usr/include/v9/sys
```

```

126 dir path=usr/include/vm
127 dir path=usr/platform/group=sys
128 $(sparc_ONLY)dir path=usr/platform/SUNW,A70 group=sys
129 $(sparc_ONLY)dir path=usr/platform/SUNW,Netra-CP2300 group=sys
130 $(sparc_ONLY)dir path=usr/platform/SUNW,Netra-CP2300/include
131 $(sparc_ONLY)dir path=usr/platform/SUNW,Netra-CP3010 group=sys
132 $(sparc_ONLY)dir path=usr/platform/SUNW,Netra-CP3010/include
133 $(sparc_ONLY)dir path=usr/platform/SUNW,Netra-T12 group=sys
134 $(sparc_ONLY)dir path=usr/platform/SUNW,Netra-T4 group=sys
135 $(sparc_ONLY)dir path=usr/platform/SUNW,SPARC-Enterprise group=sys
136 $(sparc_ONLY)dir path=usr/platform/SUNW,Serverblade1 group=sys
137 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Blade-100 group=sys
138 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Blade-1000 group=sys
139 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Blade-1500 group=sys
140 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Blade-2500 group=sys
141 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Fire group=sys
142 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Fire-15000 group=sys
143 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Fire-280R group=sys
144 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Fire-480R group=sys
145 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Fire-880 group=sys
146 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Fire-V215 group=sys
147 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Fire-V240 group=sys
148 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Fire-V250 group=sys
149 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Fire-V440 group=sys
150 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Fire-V445 group=sys
151 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Fire-V490 group=sys
152 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Fire-V890 group=sys
153 $(sparc_ONLY)dir path=usr/platform/SUNW,Ultra-2 group=sys
154 $(sparc_ONLY)dir path=usr/platform/SUNW,Ultra-250 group=sys
155 $(sparc_ONLY)dir path=usr/platform/SUNW,Ultra-4 group=sys
156 $(sparc_ONLY)dir path=usr/platform/SUNW,Ultra-Enterprise group=sys
157 $(sparc_ONLY)dir path=usr/platform/SUNW,UltraSPARC-IIe-NetraCT-40 group=sys
158 $(sparc_ONLY)dir path=usr/platform/SUNW,UltraSPARC-IIe-NetraCT-60 group=sys
159 $(sparc_ONLY)dir path=usr/platform/SUNW,UltraSPARC-IIi-Netract group=sys
160 $(i386_ONLY)dir path=usr/platform/i86pc group=sys
161 $(i386_ONLY)dir path=usr/platform/i86pc/include
162 $(i386_ONLY)dir path=usr/platform/i86pc/include/sys
163 $(i386_ONLY)dir path=usr/platform/i86pc/include/vm
164 $(i386_ONLY)dir path=usr/platform/i86xpv group=sys
165 $(i386_ONLY)dir path=usr/platform/i86xpv/include
166 $(i386_ONLY)dir path=usr/platform/i86xpv/include/sys
167 $(i386_ONLY)dir path=usr/platform/i86xpv/include/vm
168 $(sparc_ONLY)dir path=usr/platform/sun4u group=sys
169 $(sparc_ONLY)dir path=usr/platform/sun4u/include
170 $(sparc_ONLY)dir path=usr/platform/sun4u/include/sys
171 $(sparc_ONLY)dir path=usr/platform/sun4u/include/sys/i2c
172 $(sparc_ONLY)dir path=usr/platform/sun4u/include/sys/i2c/clients
173 $(sparc_ONLY)dir path=usr/platform/sun4u/include/sys/i2c/misc
174 $(sparc_ONLY)dir path=usr/platform/sun4u/include/vm
175 $(sparc_ONLY)dir path=usr/platform/sun4v group=sys
176 $(sparc_ONLY)dir path=usr/platform/sun4v/include
177 $(sparc_ONLY)dir path=usr/platform/sun4v/include/sys
178 $(sparc_ONLY)dir path=usr/platform/sun4v/include/vm
179 dir path=usr/share
180 dir path=usr/share/man
181 dir path=usr/share/man/man3
182 dir path=usr/share/man/man3head
183 dir path=usr/share/man/man4
184 dir path=usr/share/man/man5
185 dir path=usr/share/man/man7i
186 dir path=usr/share/src group=sys
187 dir path=usr/share/src/uts
188 $(i386_ONLY)dir path=usr/share/src/uts/i86pc
189 $(i386_ONLY)dir path=usr/share/src/uts/i86xpv
190 $(sparc_ONLY)dir path=usr/share/src/uts/sun4u
191 $(sparc_ONLY)dir path=usr/share/src/uts/sun4v

```

```

192 dir path=usr/xpg4
193 dir path=usr/xpg4/include
194 $(i386_ONLY)file path=usr/include/$(ARCH64)/sys/kdi_regs.h
195 $(i386_ONLY)file path=usr/include/$(ARCH64)/sys/privmregs.h
196 $(i386_ONLY)file path=usr/include/$(ARCH64)/sys/privregs.h
197 file path=usr/include/ads/dsgetdc.h
198 file path=usr/include/aio.h
199 file path=usr/include/alloca.h
200 file path=usr/include/aptrace.h
201 file path=usr/include/aptrace_impl.h
202 file path=usr/include/ar.h
203 file path=usr/include/archives.h
204 file path=usr/include/arpa/ftp.h
205 file path=usr/include/arpa/inet.h
206 file path=usr/include/arpa/nameser.h
207 file path=usr/include/arpa/nameser_compat.h
208 file path=usr/include/arpa/telnet.h
209 file path=usr/include/arpa/tftp.h
210 $(i386_ONLY)file path=usr/include/asm/atomic.h
211 $(i386_ONLY)file path=usr/include/asm/bitmap.h
212 $(i386_ONLY)file path=usr/include/asm/byteorder.h
213 $(i386_ONLY)file path=usr/include/asm/clock.h
214 $(i386_ONLY)file path=usr/include/asm/cpu.h
215 $(i386_ONLY)file path=usr/include/asm/cpuvar.h
216 $(sparc_ONLY)file path=usr/include/asm/flush.h
217 $(i386_ONLY)file path=usr/include/asm/htable.h
218 $(i386_ONLY)file path=usr/include/asm/mmu.h
219 file path=usr/include/asm/sunddi.h
220 file path=usr/include/asm/thread.h
221 file path=usr/include/assert.h
222 file path=usr/include/ast/align.h
223 file path=usr/include/ast/ast.h
224 file path=usr/include/ast/ast_botch.h
225 file path=usr/include/ast/ast_ccode.h
226 file path=usr/include/ast/ast_common.h
227 file path=usr/include/ast/ast_dir.h
228 file path=usr/include/ast/ast_dirent.h
229 file path=usr/include/ast/ast_fcntl.h
230 file path=usr/include/ast/ast_float.h
231 file path=usr/include/ast/ast_fs.h
232 file path=usr/include/ast/ast_getopt.h
233 file path=usr/include/ast/ast_iconv.h
234 file path=usr/include/ast/ast_lib.h
235 file path=usr/include/ast/ast_limits.h
236 file path=usr/include/ast/ast_map.h
237 file path=usr/include/ast/ast_mmap.h
238 file path=usr/include/ast/ast_mode.h
239 file path=usr/include/ast/ast_namval.h
240 file path=usr/include/ast/ast_ndbm.h
241 file path=usr/include/ast/ast_nl_types.h
242 file path=usr/include/ast/ast_param.h
243 file path=usr/include/ast/ast_standards.h
244 file path=usr/include/ast/ast_std.h
245 file path=usr/include/ast/ast_stdio.h
246 file path=usr/include/ast/ast_sys.h
247 file path=usr/include/ast/ast_time.h
248 file path=usr/include/ast/ast_tty.h
249 file path=usr/include/ast/ast_version.h
250 file path=usr/include/ast/ast_vfork.h
251 file path=usr/include/ast/ast_wait.h
252 file path=usr/include/ast/ast_wchar.h
253 file path=usr/include/ast/ast_windows.h
254 file path=usr/include/ast/bytsex.h
255 file path=usr/include/ast/ccode.h
256 file path=usr/include/ast/cdt.h
257 file path=usr/include/ast/cmd.h

```

258 file path=usr/include/ast/cmdext.h
259 file path=usr/include/ast/debug.h
260 file path=usr/include/ast/dirent.h
261 file path=usr/include/ast/dlldefs.h
262 file path=usr/include/ast/dt.h
263 file path=usr/include/ast/endian.h
264 file path=usr/include/ast/error.h
265 file path=usr/include/ast/find.h
266 file path=usr/include/ast/fnmatch.h
267 file path=usr/include/ast/fnv.h
268 file path=usr/include/ast/fs3d.h
269 file path=usr/include/ast/fts.h
270 file path=usr/include/ast/ftw.h
271 file path=usr/include/ast/ftwalk.h
272 file path=usr/include/ast/getopt.h
273 file path=usr/include/ast/glob.h
274 file path=usr/include/ast/hash.h
275 file path=usr/include/ast/hashkey.h
276 file path=usr/include/ast/hashpart.h
277 file path=usr/include/ast/history.h
278 file path=usr/include/ast/iconv.h
279 file path=usr/include/ast/ip6.h
280 file path=usr/include/ast/lc.h
281 file path=usr/include/ast/ls.h
282 file path=usr/include/ast/magic.h
283 file path=usr/include/ast/magicid.h
284 file path=usr/include/ast/mc.h
285 file path=usr/include/ast/mime.h
286 file path=usr/include/ast/mnt.h
287 file path=usr/include/ast/modecanon.h
288 file path=usr/include/ast/modex.h
289 file path=usr/include/ast/namval.h
290 file path=usr/include/ast/nl_types.h
291 file path=usr/include/ast/nval.h
292 file path=usr/include/ast/option.h
293 file path=usr/include/ast/preroot.h
294 file path=usr/include/ast/proc.h
295 file path=usr/include/ast/prototyped.h
296 file path=usr/include/ast/re_comp.h
297 file path=usr/include/ast/recfmt.h
298 file path=usr/include/ast/regex.h
299 file path=usr/include/ast/regexp.h
300 file path=usr/include/ast/sfdisc.h
301 file path=usr/include/ast/sfio.h
302 file path=usr/include/ast/sfio_s.h
303 file path=usr/include/ast/sfio_t.h
304 file path=usr/include/ast/shcmd.h
305 file path=usr/include/ast/shell.h
306 file path=usr/include/ast/sig.h
307 file path=usr/include/ast/stack.h
308 file path=usr/include/ast/stak.h
309 file path=usr/include/ast/stdio.h
310 file path=usr/include/ast/stk.h
311 file path=usr/include/ast/sum.h
312 file path=usr/include/ast/swap.h
313 file path=usr/include/ast/tar.h
314 file path=usr/include/ast/times.h
315 file path=usr/include/ast/tm.h
316 file path=usr/include/ast/tmx.h
317 file path=usr/include/ast/tok.h
318 file path=usr/include/ast/tv.h
319 file path=usr/include/ast/usage.h
320 file path=usr/include/ast/vdb.h
321 file path=usr/include/ast/vecargs.h
322 file path=usr/include/ast/vmalloc.h
323 file path=usr/include/ast/wait.h

324 file path=usr/include/ast/wchar.h
325 file path=usr/include/ast/wordexp.h
326 file path=usr/include/atomic.h
327 file path=usr/include/attr.h
328 file path=usr/include/auth_attr.h
329 file path=usr/include/bsm/adt.h
330 file path=usr/include/bsm/adt_event.h
331 file path=usr/include/bsm/audit.h
332 file path=usr/include/bsm/audit_kernel.h
333 file path=usr/include/bsm/audit_kevents.h
334 file path=usr/include/bsm/audit_record.h
335 file path=usr/include/bsm/audit_uevents.h
336 file path=usr/include/bsm/devices.h
337 file path=usr/include/bsm/libbsm.h
338 file path=usr/include/config_admin.h
339 file path=usr/include/cpio.h
340 file path=usr/include/crypt.h
341 file path=usr/include/cryptoutil.h
342 file path=usr/include/ctype.h
343 file path=usr/include/curses.h
344 file path=usr/include/dat/dat.h
345 file path=usr/include/dat/dat_error.h
346 file path=usr/include/dat/dat_platform_specific.h
347 file path=usr/include/dat/dat_redirection.h
348 file path=usr/include/dat/dat_registry.h
349 file path=usr/include/dat/dat_vendor_specific.h
350 file path=usr/include/dat/udat.h
351 file path=usr/include/dat/udat_config.h
352 file path=usr/include/dat/udat_redirection.h
353 file path=usr/include/dat/udat_vendor_specific.h
354 file path=usr/include/deflt.h
355 file path=usr/include/des/des.h
356 file path=usr/include/des/desdata.h
357 file path=usr/include/des/softdes.h
358 file path=usr/include/devfsadm.h
359 file path=usr/include/device_info.h
360 file path=usr/include/devid.h
361 file path=usr/include/devmgmt.h
362 file path=usr/include/devpoll.h
363 file path=usr/include/dial.h
364 file path=usr/include/dirent.h
365 file path=usr/include/dlfcn.h
366 file path=usr/include/door.h
367 file path=usr/include/elf.h
368 file path=usr/include/endian.h
369 file path=usr/include/err.h
370 file path=usr/include/errno.h
371 file path=usr/include/eti.h
372 file path=usr/include/euc.h
373 file path=usr/include/exacct.h
374 file path=usr/include/exacct_impl.h
375 file path=usr/include/exec_attr.h
376 file path=usr/include/execinfo.h
377 file path=usr/include/fatal.h
378 file path=usr/include/fcntl.h
379 file path=usr/include/float.h
380 file path=usr/include/fmtmsg.h
381 file path=usr/include/fnmatch.h
382 file path=usr/include/form.h
383 file path=usr/include/fts.h
384 file path=usr/include/ftw.h
385 file path=usr/include/gelf.h
386 file path=usr/include/getopt.h
387 file path=usr/include/getwidth.h
388 file path=usr/include/glob.h
389 file path=usr/include/grp.h

```
390 file path=usr/include/gssapi/gssapi.h
391 file path=usr/include/gssapi/gssapi_ext.h
392 file path=usr/include/hal/libhal-storage.h
393 file path=usr/include/hal/libhal.h
394 $(i386_ONLY)file path=usr/include/ia32/sys/asm_linkage.h
395 $(i386_ONLY)file path=usr/include/ia32/sys/kdi_regs.h
396 $(i386_ONLY)file path=usr/include/ia32/sys/machtypes.h
397 $(i386_ONLY)file path=usr/include/ia32/sys/privmregs.h
398 $(i386_ONLY)file path=usr/include/ia32/sys/privregs.h
399 $(i386_ONLY)file path=usr/include/ia32/sys/psw.h
400 $(i386_ONLY)file path=usr/include/ia32/sys/pte.h
401 $(i386_ONLY)file path=usr/include/ia32/sys/reg.h
402 $(i386_ONLY)file path=usr/include/ia32/sys/stack.h
403 $(i386_ONLY)file path=usr/include/ia32/sys/trap.h
404 $(i386_ONLY)file path=usr/include/ia32/sys/traptrace.h
405 file path=usr/include/iconv.h
406 file path=usr/include/idmap.h
407 file path=usr/include/ieeefp.h
408 file path=usr/include/ifaddrs.h
409 file path=usr/include/inet/arp.h
410 file path=usr/include/inet/common.h
411 file path=usr/include/inet/ip.h
412 file path=usr/include/inet/ip6.h
413 file path=usr/include/inet/ip6_asp.h
414 file path=usr/include/inet/ip_arp.h
415 file path=usr/include/inet/ip_ftable.h
416 file path=usr/include/inet/ip_if.h
417 file path=usr/include/inet/ip_ire.h
418 file path=usr/include/inet/ip_multi.h
419 file path=usr/include/inet/ip_netinfo.h
420 file path=usr/include/inet/ip_rts.h
421 file path=usr/include/inet/ip_stack.h
422 file path=usr/include/inet/ipclassifier.h
423 file path=usr/include/inet/ipdrop.h
424 file path=usr/include/inet/ipnet.h
425 file path=usr/include/inet/ipp_common.h
426 file path=usr/include/inet/kssl/ksslap.h
427 file path=usr/include/inet/led.h
428 file path=usr/include/inet/mi.h
429 file path=usr/include/inet/mib2.h
430 file path=usr/include/inet/nd.h
431 file path=usr/include/inet/optcom.h
432 file path=usr/include/inet/sctp_itf.h
433 file path=usr/include/inet/snmpcom.h
434 file path=usr/include/inet/tcp.h
435 file path=usr/include/inet/tcp_sack.h
436 file path=usr/include/inet/tcp_stack.h
437 file path=usr/include/inet/tcp_stats.h
438 file path=usr/include/inet/tunables.h
439 file path=usr/include/inet/wifi_ioctl.h
440 file path=usr/include/inttypes.h
441 file path=usr/include/ipmp.h
442 file path=usr/include/ipmp_admin.h
443 file path=usr/include/ipmp_mpathd.h
444 file path=usr/include/ipmp_query.h
445 file path=usr/include/ipp/ipgpc/ipgpc.h
446 file path=usr/include/ipp/ipp.h
447 file path=usr/include/ipp/ipp_config.h
448 file path=usr/include/ipp/ipp_impl.h
449 file path=usr/include/ipp/ippctl.h
450 file path=usr/include/iso/ctype_iso.h
451 file path=usr/include/iso/limits_iso.h
452 file path=usr/include/iso/locale_iso.h
453 file path=usr/include/iso/setjmp_iso.h
454 file path=usr/include/iso/signal_iso.h
455 file path=usr/include/iso/stdarg_c99.h
```

```
456 file path=usr/include/iso/stdarg_iso.h
457 file path=usr/include/iso/stddef_iso.h
458 file path=usr/include/iso/stdio_c99.h
459 file path=usr/include/iso/stdio_iso.h
460 file path=usr/include/iso/stdlib_c11.h
461 file path=usr/include/iso/stdlib_c99.h
462 file path=usr/include/iso/stdlib_iso.h
463 file path=usr/include/iso/string_iso.h
464 file path=usr/include/iso/time_iso.h
465 file path=usr/include/iso/wchar_c99.h
466 file path=usr/include/iso/wchar_iso.h
467 file path=usr/include/iso/wctype_iso.h
468 file path=usr/include/iso646.h
469 file path=usr/include/kerberos5/com_err.h
470 file path=usr/include/kerberos5/krb5.h
471 file path=usr/include/kerberos5/locate_plugin.h
472 file path=usr/include/kerberos5/mit-sipb-copyright.h
473 file path=usr/include/kerberos5/mit_copyright.h
474 file path=usr/include/klpd.h
475 file path=usr/include/kmfapi.h
476 file path=usr/include/kmftypes.h
477 file path=usr/include/kstat.h
478 file path=usr/include/kvm.h
479 file path=usr/include/langinfo.h
480 file path=usr/include/lastlog.h
481 file path=usr/include/lber.h
482 file path=usr/include/ldap.h
483 file path=usr/include/libcontract.h
484 file path=usr/include/libctf.h
485 file path=usr/include/libdevice.h
486 file path=usr/include/libdevinfo.h
487 file path=usr/include/libdladm.h
488 file path=usr/include/libdlbridge.h
489 file path=usr/include/libdlib.h
490 file path=usr/include/libdllink.h
491 file path=usr/include/libdlpi.h
492 file path=usr/include/libdlvlan.h
493 file path=usr/include/libelf.h
494 $(i386_ONLY)file path=usr/include/libfdisk.h
495 file path=usr/include/libfstyp.h
496 file path=usr/include/libfstyp_module.h
497 file path=usr/include/libgen.h
498 file path=usr/include/libgrubmgmt.h
499 file path=usr/include/libintl.h
500 file path=usr/include/libipmi.h
501 file path=usr/include/libipp.h
502 file path=usr/include/libnvpair.h
503 file path=usr/include/libnwam.h
504 file path=usr/include/libpolkit/libpolkit.h
505 file path=usr/include/libproc.h
506 file path=usr/include/librcm.h
507 file path=usr/include/libscf.h
508 file path=usr/include/libscf_priv.h
509 file path=usr/include/libshare.h
510 file path=usr/include/libsysevent.h
511 file path=usr/include/libsysevent_impl.h
512 file path=usr/include/libtsnet.h
513 $(sparc_ONLY)file path=usr/include/libv12n.h
514 file path=usr/include/libw.h
515 file path=usr/include/libzfs.h
516 file path=usr/include/libzfs_core.h
517 file path=usr/include/libzoneinfo.h
518 file path=usr/include/limits.h
519 file path=usr/include/linenum.h
520 file path=usr/include/link.h
521 file path=usr/include/listen.h
```

522 file path=usr/include/locale.h
523 file path=usr/include/macros.h
524 file path=usr/include/maillock.h
525 file path=usr/include/malloc.h
526 file path=usr/include/md4.h
527 file path=usr/include/md5.h
528 file path=usr/include/memory.h
529 file path=usr/include/menu.h
530 file path=usr/include/mon.h
531 file path=usr/include/monetary.h
532 file path=usr/include/mp.h
533 file path=usr/include/mqueue.h
534 file path=usr/include/mtmalloc.h
535 file path=usr/include/nan.h
536 file path=usr/include/ndbm.h
537 file path=usr/include/ndpd.h
538 file path=usr/include/net/af.h
539 file path=usr/include/net/bridge.h
540 file path=usr/include/net/if.h
541 file path=usr/include/net/if_arp.h
542 file path=usr/include/net/if_dl.h
543 file path=usr/include/net/if_types.h
544 file path=usr/include/net/pfkeyv2.h
545 file path=usr/include/net/pfpolicy.h
546 file path=usr/include/net/ppp-comp.h
547 file path=usr/include/net/ppp_defs.h
548 file path=usr/include/net/pppio.h
549 file path=usr/include/net/radix.h
550 file path=usr/include/net/route.h
551 file path=usr/include/net/trill.h
552 file path=usr/include/net/vjcompress.h
553 file path=usr/include/netconfig.h
554 file path=usr/include/netdb.h
555 file path=usr/include/netdir.h
556 file path=usr/include/netinet/arp.h
557 file path=usr/include/netinet/dhcp.h
558 file path=usr/include/netinet/dhcp6.h
559 file path=usr/include/netinet/icmp6.h
560 file path=usr/include/netinet/icmp_var.h
561 file path=usr/include/netinet/if_ether.h
562 file path=usr/include/netinet/igmp.h
563 file path=usr/include/netinet/igmp_var.h
564 file path=usr/include/netinet/in.h
565 file path=usr/include/netinet/in_pcb.h
566 file path=usr/include/netinet/in_sysm.h
567 file path=usr/include/netinet/in_var.h
568 file path=usr/include/netinet/ip.h
569 file path=usr/include/netinet/ip6.h
570 file path=usr/include/netinet/ip_icmp.h
571 file path=usr/include/netinet/ip_mroute.h
572 file path=usr/include/netinet/ip_var.h
573 file path=usr/include/netinet/pim.h
574 file path=usr/include/netinet/sctp.h
575 file path=usr/include/netinet/tcp.h
576 file path=usr/include/netinet/tcp_debug.h
577 file path=usr/include/netinet/tcp_fsm.h
578 file path=usr/include/netinet/tcp_seq.h
579 file path=usr/include/netinet/tcp_timer.h
580 file path=usr/include/netinet/tcp_var.h
581 file path=usr/include/netinet/tcpip.h
582 file path=usr/include/netinet/udp.h
583 file path=usr/include/netinet/udp_var.h
584 file path=usr/include/netinet/vrrp.h
585 file path=usr/include/nfs/auth.h
586 file path=usr/include/nfs/export.h
587 file path=usr/include/nfs/lm.h

588 file path=usr/include/nfs/mapid.h
589 file path=usr/include/nfs/mount.h
590 file path=usr/include/nfs/nfs.h
591 file path=usr/include/nfs/nfs4.h
592 file path=usr/include/nfs/nfs4_attr.h
593 file path=usr/include/nfs/nfs4_clnt.h
594 file path=usr/include/nfs/nfs4_db_impl.h
595 file path=usr/include/nfs/nfs4_idmap_impl.h
596 file path=usr/include/nfs/nfs4_kprot.h
597 file path=usr/include/nfs/nfs_acl.h
598 file path=usr/include/nfs/nfs_clnt.h
599 file path=usr/include/nfs/nfs_cmd.h
600 file path=usr/include/nfs/nfs_log.h
601 file path=usr/include/nfs/nfs_sec.h
602 file path=usr/include/nfs/nfsid_map.h
603 file path=usr/include/nfs/nfssys.h
604 file path=usr/include/nfs/rnode.h
605 file path=usr/include/nfs/rnode4.h
606 file path=usr/include/nl_types.h
607 file path=usr/include/nlist.h
608 file path=usr/include/note.h
609 file path=usr/include/nss_common.h
610 file path=usr/include/nss_dbdefs.h
611 file path=usr/include/nss_netdir.h
612 file path=usr/include/nsswitch.h
613 file path=usr/include/panel.h
614 file path=usr/include/paths.h
615 file path=usr/include/pcsample.h
616 file path=usr/include/pfmt.h
617 file path=usr/include/pkgdev.h
618 file path=usr/include/pkginfo.h
619 file path=usr/include/pkglocs.h
620 file path=usr/include/pkgstrct.h
621 file path=usr/include/pkgtrans.h
622 file path=usr/include/poll.h
623 file path=usr/include/port.h
624 file path=usr/include/priv.h
625 file path=usr/include/proc_service.h
626 file path=usr/include/procfs.h
627 file path=usr/include/prof.h
628 file path=usr/include/prof_attr.h
629 file path=usr/include/project.h
630 file path=usr/include/protocols/dumprestore.h
631 file path=usr/include/protocols/routed.h
632 file path=usr/include/protocols/rwhod.h
633 file path=usr/include/protocols/timed.h
634 file path=usr/include/pthread.h
635 file path=usr/include/pw.h
636 file path=usr/include/pwd.h
637 file path=usr/include/rcm_module.h
638 file path=usr/include/rctl.h
639 file path=usr/include/re_comp.h
640 file path=usr/include/regex.h
641 file path=usr/include/regexp.h
642 file path=usr/include/regexpr.h
643 file path=usr/include/resolv.h
644 file path=usr/include/rje.h
645 file path=usr/include/rp_plugin.h
646 file path=usr/include/rpc/auth.h
647 file path=usr/include/rpc/auth_des.h
648 file path=usr/include/rpc/auth_sys.h
649 file path=usr/include/rpc/auth_unix.h
650 file path=usr/include/rpc/bootparam.h
651 file path=usr/include/rpc/clnt.h
652 file path=usr/include/rpc/clnt_soc.h
653 file path=usr/include/rpc/clnt_stat.h

```

654 file path=usr/include/rpc/des_crypt.h
655 $(sparc_ONLY)file path=usr/include/rpc/ib.h
656 file path=usr/include/rpc/key_prot.h
657 file path=usr/include/rpc/nettype.h
658 file path=usr/include/rpc/pmap_clnt.h
659 file path=usr/include/rpc/pmap_prot.h
660 file path=usr/include/rpc/pmap_prot.x
661 file path=usr/include/rpc/pmap_rmt.h
662 file path=usr/include/rpc/rpc.h
663 file path=usr/include/rpc/rpc_com.h
664 file path=usr/include/rpc/rpc_msg.h
665 file path=usr/include/rpc/rpc_rdma.h
666 file path=usr/include/rpc/rpc_sztypes.h
667 file path=usr/include/rpc/rpcb_clnt.h
668 file path=usr/include/rpc/rpcb_prot.h
669 file path=usr/include/rpc/rpcb_prot.x
670 file path=usr/include/rpc/rpcent.h
671 file path=usr/include/rpc/rpcsec_gss.h
672 file path=usr/include/rpc/rpcsys.h
673 file path=usr/include/rpc/svc.h
674 file path=usr/include/rpc/svc_auth.h
675 file path=usr/include/rpc/svc_mt.h
676 file path=usr/include/rpc/svc_soc.h
677 file path=usr/include/rpc/types.h
678 file path=usr/include/rpc/xdr.h
679 file path=usr/include/rpcsvc/autofs_prot.h
680 file path=usr/include/rpcsvc/autofs_prot.x
681 file path=usr/include/rpcsvc/bootparam.h
682 file path=usr/include/rpcsvc/bootparam_prot.h
683 file path=usr/include/rpcsvc/bootparam_prot.x
684 file path=usr/include/rpcsvc/dbm.h
685 file path=usr/include/rpcsvc/key_prot.x
686 file path=usr/include/rpcsvc/mount.h
687 file path=usr/include/rpcsvc/mount.x
688 file path=usr/include/rpcsvc/nfs4_prot.h
689 file path=usr/include/rpcsvc/nfs4_prot.x
690 file path=usr/include/rpcsvc/nfs_acl.h
691 file path=usr/include/rpcsvc/nfs_acl.x
692 file path=usr/include/rpcsvc/nfs_prot.h
693 file path=usr/include/rpcsvc/nfs_prot.x
694 file path=usr/include/rpcsvc/nis.h
695 file path=usr/include/rpcsvc/nis.x
696 file path=usr/include/rpcsvc/nis_db.h
697 file path=usr/include/rpcsvc/nis_object.x
698 file path=usr/include/rpcsvc/nislib.h
699 file path=usr/include/rpcsvc/nlm_prot.h
700 file path=usr/include/rpcsvc/nlm_prot.x
701 file path=usr/include/rpcsvc/nsm_addr.h
702 file path=usr/include/rpcsvc/nsm_addr.x
703 file path=usr/include/rpcsvc/rpc_sztypes.h
704 file path=usr/include/rpcsvc/rpc_sztypes.x
705 file path=usr/include/rpcsvc/rquota.h
706 file path=usr/include/rpcsvc/rquota.x
707 file path=usr/include/rpcsvc/rstat.h
708 file path=usr/include/rpcsvc/rstat.x
709 file path=usr/include/rpcsvc/rusers.h
710 file path=usr/include/rpcsvc/rusers.x
711 file path=usr/include/rpcsvc/rwall.h
712 file path=usr/include/rpcsvc/rwall.x
713 file path=usr/include/rpcsvc/sm_inter.h
714 file path=usr/include/rpcsvc/sm_inter.x
715 file path=usr/include/rpcsvc/spray.h
716 file path=usr/include/rpcsvc/spray.x
717 file path=usr/include/rpcsvc/ufs_prot.h
718 file path=usr/include/rpcsvc/ufs_prot.x
719 file path=usr/include/rpcsvc/yp.x

```

```

720 file path=usr/include/rpcsvc/yp_prot.h
721 file path=usr/include/rpcsvc/ypclnt.h
722 file path=usr/include/rpcsvc/yppasswd.h
723 file path=usr/include/rpcsvc/ypupd.h
724 file path=usr/include/rsmapi.h
725 file path=usr/include/rtld_db.h
726 file path=usr/include/sac.h
727 file path=usr/include/sasl/prop.h
728 file path=usr/include/sasl/sasl.h
729 file path=usr/include/sasl/saslplug.h
730 file path=usr/include/sasl/saslutil.h
731 file path=usr/include/sched.h
732 file path=usr/include/schedctl.h
733 file path=usr/include/scsi/libscsi.h
734 file path=usr/include/scsi/libses.h
735 file path=usr/include/scsi/libses_plugin.h
736 file path=usr/include/scsi/libsmpl.h
737 file path=usr/include/scsi/libsmpl_plugin.h
738 file path=usr/include/scsi/plugins/ses/framework/libses.h
739 file path=usr/include/scsi/plugins/ses/framework/ses2.h
740 file path=usr/include/scsi/plugins/ses/framework/ses2_impl.h
741 file path=usr/include/scsi/plugins/ses/vendor/sun.h
742 file path=usr/include/sdp.h
743 file path=usr/include/search.h
744 file path=usr/include/secdb.h
745 file path=usr/include/security/auditd.h
746 file path=usr/include/security/cryptoki.h
747 file path=usr/include/security/pam_appl.h
748 file path=usr/include/security/pam_modules.h
749 file path=usr/include/security/pkcs11.h
750 file path=usr/include/security/pkcs11f.h
751 file path=usr/include/security/pkcs11t.h
752 file path=usr/include/semaphore.h
753 file path=usr/include/setjmp.h
754 file path=usr/include/sgtty.h
755 file path=usr/include/shal.h
756 file path=usr/include/sha2.h
757 file path=usr/include/shadow.h
758 file path=usr/include/sharefs/share.h
759 file path=usr/include/sharefs/sharefs.h
760 file path=usr/include/sharefs/sharet.h
761 file path=usr/include/signinfo.h
762 file path=usr/include/signal.h
763 file path=usr/include/sip.h
764 file path=usr/include/skein.h
765 file path=usr/include/smbios.h
766 file path=usr/include/spawn.h
767 $(i386_ONLY)file path=usr/include/stack_unwind.h
768 file path=usr/include/stdalign.h
769 file path=usr/include/stdarg.h
770 file path=usr/include/stdbool.h
771 file path=usr/include/stddef.h
772 file path=usr/include/stdint.h
773 file path=usr/include/stdio.h
774 file path=usr/include/stdio_ext.h
775 file path=usr/include/stdio_impl.h
776 file path=usr/include/stdio_tag.h
777 file path=usr/include/stdlib.h
778 file path=usr/include/stdnoreturn.h
779 file path=usr/include/storclass.h
780 file path=usr/include/string.h
781 file path=usr/include/strings.h
782 file path=usr/include/stropts.h
783 file path=usr/include/syms.h
784 file path=usr/include/synch.h
785 file path=usr/include/sys/acct.h

```

```

786 file path=usr/include/sys/acctctl.h
787 file path=usr/include/sys/acl.h
788 file path=usr/include/sys/acl_impl.h
789 file path=usr/include/sys/acpi_drv.h
790 file path=usr/include/sys/aio.h
791 file path=usr/include/sys/aio_impl.h
792 file path=usr/include/sys/aio_req.h
793 file path=usr/include/sys/aioch.h
794 file path=usr/include/sys/archsystem.h
795 file path=usr/include/sys/ascii.h
796 file path=usr/include/sys/asm_linkage.h
797 file path=usr/include/sys/asynch.h
798 file path=usr/include/sys/atomic.h
799 file path=usr/include/sys/attr.h
800 file path=usr/include/sys/autoconf.h
801 file path=usr/include/sys/auxv.h
802 file path=usr/include/sys/auxv_386.h
803 file path=usr/include/sys/auxv_SPARC.h
804 file path=usr/include/sys/av/iec61883.h
805 file path=usr/include/sys/avintr.h
806 file path=usr/include/sys/avl.h
807 file path=usr/include/sys/avl_impl.h
808 file path=usr/include/sys/bitmap.h
809 file path=usr/include/sys/bitset.h
810 file path=usr/include/sys/bl.h
811 file path=usr/include/sys/blkdev.h
812 file path=usr/include/sys/bofi.h
813 file path=usr/include/sys/bofi_impl.h
814 file path=usr/include/sys/bootconf.h
815 $(i386_ONLY)file path=usr/include/sys/bootregs.h
816 file path=usr/include/sys/bootstat.h
817 $(i386_ONLY)file path=usr/include/sys/bootsvcs.h
818 file path=usr/include/sys/bpp_io.h
819 file path=usr/include/sys/brand.h
820 file path=usr/include/sys/buf.h
821 file path=usr/include/sys/bufmod.h
822 file path=usr/include/sys/bustypes.h
823 file path=usr/include/sys/byteorder.h
824 file path=usr/include/sys/callb.h
825 file path=usr/include/sys/callo.h
826 file path=usr/include/sys/cap_util.h
827 file path=usr/include/sys/ccompile.h
828 file path=usr/include/sys/cdio.h
829 file path=usr/include/sys/cis.h
830 file path=usr/include/sys/cis_handlers.h
831 file path=usr/include/sys/cis_protos.h
832 file path=usr/include/sys/cladm.h
833 file path=usr/include/sys/class.h
834 file path=usr/include/sys/clconf.h
835 file path=usr/include/sys/cmhb.h
836 file path=usr/include/sys/cmn_err.h
837 $(sparc_ONLY)file path=usr/include/sys/compregs.h
838 file path=usr/include/sys/compress.h
839 file path=usr/include/sys/condvar.h
840 file path=usr/include/sys/condvar_impl.h
841 file path=usr/include/sys/conf.h
842 file path=usr/include/sys/consdev.h
843 file path=usr/include/sys/console.h
844 file path=usr/include/sys/consplat.h
845 file path=usr/include/sys/containerof.h
846 file path=usr/include/sys/contract.h
847 file path=usr/include/sys/contract/device.h
848 file path=usr/include/sys/contract/device_impl.h
849 file path=usr/include/sys/contract/process.h
850 file path=usr/include/sys/contract/process_impl.h
851 file path=usr/include/sys/contract_impl.h

```

```

852 $(i386_ONLY)file path=usr/include/sys/controlregs.h
853 file path=usr/include/sys/copyops.h
854 file path=usr/include/sys/core.h
855 file path=usr/include/sys/corectl.h
856 file path=usr/include/sys/cpc_impl.h
857 file path=usr/include/sys/cpc_pcbe.h
858 file path=usr/include/sys/cpr.h
859 file path=usr/include/sys/cpu.h
860 file path=usr/include/sys/cpu_uarray.h
861 file path=usr/include/sys/cpucaps.h
862 file path=usr/include/sys/cpucaps_impl.h
863 file path=usr/include/sys/cpupart.h
864 file path=usr/include/sys/cpuvar.h
865 file path=usr/include/sys/crc32.h
866 file path=usr/include/sys/cred.h
867 file path=usr/include/sys/cred_impl.h
868 file path=usr/include/sys/crtctl.h
869 file path=usr/include/sys/crypto/api.h
870 file path=usr/include/sys/crypto/common.h
871 file path=usr/include/sys/crypto/ioctl.h
872 file path=usr/include/sys/crypto/ioctladmin.h
873 file path=usr/include/sys/crypto/spi.h
874 file path=usr/include/sys/cs.h
875 file path=usr/include/sys/cs_priv.h
876 file path=usr/include/sys/cs_strings.h
877 file path=usr/include/sys/cs_stubs.h
878 file path=usr/include/sys/cs_types.h
879 file path=usr/include/sys/csioctl.h
880 file path=usr/include/sys/ctf.h
881 file path=usr/include/sys/ctf_api.h
882 file path=usr/include/sys/ctfs.h
883 file path=usr/include/sys/ctfs_impl.h
884 file path=usr/include/sys/ctype.h
885 file path=usr/include/sys/cyclic.h
886 file path=usr/include/sys/cyclic_impl.h
887 file path=usr/include/sys/dacf.h
888 file path=usr/include/sys/dacf_impl.h
889 file path=usr/include/sys/damap.h
890 file path=usr/include/sys/damap_impl.h
891 file path=usr/include/sys/dc_ki.h
892 file path=usr/include/sys/ddi.h
893 file path=usr/include/sys/ddi_hp.h
894 file path=usr/include/sys/ddi_hp_impl.h
895 file path=usr/include/sys/ddi_impldefs.h
896 file path=usr/include/sys/ddi_implfuncs.h
897 file path=usr/include/sys/ddi_intr.h
898 file path=usr/include/sys/ddi_intr_impl.h
899 file path=usr/include/sys/ddi_isa.h
900 file path=usr/include/sys/ddi_obsolete.h
901 file path=usr/include/sys/ddi_periodic.h
902 file path=usr/include/sys/ddidevmap.h
903 file path=usr/include/sys/ddidmareq.h
904 file path=usr/include/sys/ddifm.h
905 file path=usr/include/sys/ddifm_impl.h
906 file path=usr/include/sys/ddimapreq.h
907 file path=usr/include/sys/ddipropdefs.h
908 file path=usr/include/sys/dditypes.h
909 file path=usr/include/sys/debug.h
910 $(i386_ONLY)file path=usr/include/sys/debugreg.h
911 file path=usr/include/sys/des.h
912 file path=usr/include/sys/devcache.h
913 file path=usr/include/sys/devcache_impl.h
914 file path=usr/include/sys/devctl.h
915 file path=usr/include/sys/devfm.h
916 file path=usr/include/sys/devid_cache.h
917 file path=usr/include/sys/devinfo_impl.h

```



```

918 file path=usr/include/sys/devops.h
919 file path=usr/include/sys/devpolicy.h
920 file path=usr/include/sys/devpoll.h
921 file path=usr/include/sys/dirent.h
922 file path=usr/include/sys/disp.h
923 file path=usr/include/sys/dkbad.h
924 file path=usr/include/sys/dkio.h
925 file path=usr/include/sys/dkioc_free_util.h
926 file path=usr/include/sys/dklabel.h
927 $(sparc_ONLY)file path=usr/include/sys/dkmpio.h
928 $(i386_ONLY)file path=usr/include/sys/dktp/altctr.h
929 $(i386_ONLY)file path=usr/include/sys/dktp/cmpkt.h
930 file path=usr/include/sys/dktp/dadkio.h
931 file path=usr/include/sys/dktp/fdisk.h
932 file path=usr/include/sys/dl.h
933 file path=usr/include/sys/dld.h
934 file path=usr/include/sys/dlpi.h
935 file path=usr/include/sys/dls_mgmt.h
936 $(i386_ONLY)file path=usr/include/sys/dma_engine.h
937 file path=usr/include/sys/dma_i8237A.h
938 file path=usr/include/sys/dnlc.h
939 file path=usr/include/sys/door.h
940 file path=usr/include/sys/door_data.h
941 file path=usr/include/sys/door_impl.h
942 file path=usr/include/sys/dumphdr.h
943 file path=usr/include/sys/ecppio.h
944 file path=usr/include/sys/ecppreg.h
945 file path=usr/include/sys/ecppsys.h
946 file path=usr/include/sys/ecppvar.h
947 file path=usr/include/sys/edonr.h
948 file path=usr/include/sys/efi_partition.h
949 file path=usr/include/sys/elf.h
950 file path=usr/include/sys/elf_386.h
951 file path=usr/include/sys/elf_SPARC.h
952 file path=usr/include/sys/elf_amd64.h
953 file path=usr/include/sys/elf_notes.h
954 file path=usr/include/sys/elftypes.h
955 file path=usr/include/sys/epm.h
956 file path=usr/include/sys/epoll.h
957 file path=usr/include/sys/errno.h
958 file path=usr/include/sys/errorq.h
959 file path=usr/include/sys/errorq_impl.h
960 file path=usr/include/sys/esunddi.h
961 file path=usr/include/sys/ethernet.h
962 file path=usr/include/sys/euc.h
963 file path=usr/include/sys/eucioctl.h
964 file path=usr/include/sys/eventfd.h
965 file path=usr/include/sys/exacct.h
966 file path=usr/include/sys/exacct_catalog.h
967 file path=usr/include/sys/exacct_impl.h
968 file path=usr/include/sys/exec.h
969 file path=usr/include/sys/exechdr.h
970 file path=usr/include/sys/fault.h
971 file path=usr/include/sys/fbio.h
972 file path=usr/include/sys/fbuf.h
973 file path=usr/include/sys/fc4/fc.h
974 file path=usr/include/sys/fc4/fc_transport.h
975 file path=usr/include/sys/fc4/fcal.h
976 file path=usr/include/sys/fc4/fcal_linkapp.h
977 file path=usr/include/sys/fc4/fcal_transport.h
978 file path=usr/include/sys/fc4/fcio.h
979 file path=usr/include/sys/fc4/fcp.h
980 file path=usr/include/sys/fc4/linkapp.h
981 file path=usr/include/sys/fcntl.h
982 file path=usr/include/sys/fdbuffer.h
983 file path=usr/include/sys/fdio.h

```

```

984 $(sparc_ONLY)file path=usr/include/sys/fdreg.h
985 $(sparc_ONLY)file path=usr/include/sys/fdvar.h
986 file path=usr/include/sys/feature_tests.h
987 file path=usr/include/sys/fem.h
988 file path=usr/include/sys/file.h
989 file path=usr/include/sys/filio.h
990 $(i386_ONLY)file path=usr/include/sys/firmload.h
991 file path=usr/include/sys/flock.h
992 file path=usr/include/sys/flock_impl.h
993 $(sparc_ONLY)file path=usr/include/sys/fm/cpu/SPARC64-VI.h
994 $(sparc_ONLY)file path=usr/include/sys/fm/cpu/UltraSPARC-II.h
995 $(sparc_ONLY)file path=usr/include/sys/fm/cpu/UltraSPARC-III.h
996 $(sparc_ONLY)file path=usr/include/sys/fm/cpu/UltraSPARC-T1.h
997 file path=usr/include/sys/fm/fs/zfs.h
998 file path=usr/include/sys/fm/io/ddi.h
999 file path=usr/include/sys/fm/io/disk.h
1000 file path=usr/include/sys/fm/io/opl_mc_fm.h
1001 file path=usr/include/sys/fm/io/pci.h
1002 file path=usr/include/sys/fm/io/scsi.h
1003 file path=usr/include/sys/fm/io/sun4upci.h
1004 file path=usr/include/sys/fm/protocol.h
1005 file path=usr/include/sys/fm/util.h
1006 file path=usr/include/sys/fork.h
1007 $(i386_ONLY)file path=usr/include/sys/fp.h
1008 $(sparc_ONLY)file path=usr/include/sys/fpu/fpu_simulator.h
1009 $(sparc_ONLY)file path=usr/include/sys/fpu/fpusystem.h
1010 $(sparc_ONLY)file path=usr/include/sys/fpu/globals.h
1011 $(sparc_ONLY)file path=usr/include/sys/fpu/ieee.h
1012 file path=usr/include/sys/frame.h
1013 file path=usr/include/sys/fs/autofs.h
1014 file path=usr/include/sys/fs/decomp.h
1015 file path=usr/include/sys/fs/dv_node.h
1016 file path=usr/include/sys/fs/fifonode.h
1017 file path=usr/include/sys/fs/hsfs_isospec.h
1018 file path=usr/include/sys/fs/hsfs_node.h
1019 file path=usr/include/sys/fs/hsfs_rrip.h
1020 file path=usr/include/sys/fs/hsfs_spec.h
1021 file path=usr/include/sys/fs/hsfs_susp.h
1022 file path=usr/include/sys/fs/lofs_info.h
1023 file path=usr/include/sys/fs/lofs_node.h
1024 file path=usr/include/sys/fs/mntdata.h
1025 file path=usr/include/sys/fs/namenode.h
1026 file path=usr/include/sys/fs/pc_dir.h
1027 file path=usr/include/sys/fs/pc_fs.h
1028 file path=usr/include/sys/fs/pc_label.h
1029 file path=usr/include/sys/fs/pc_node.h
1030 file path=usr/include/sys/fs/pxfs_ki.h
1031 file path=usr/include/sys/fs/sdev_impl.h
1032 file path=usr/include/sys/fs/snode.h
1033 file path=usr/include/sys/fs/swapnode.h
1034 file path=usr/include/sys/fs/tmp.h
1035 file path=usr/include/sys/fs/tmpnode.h
1036 file path=usr/include/sys/fs/udf_inode.h
1037 file path=usr/include/sys/fs/udf_volume.h
1038 file path=usr/include/sys/fs/ufs_acl.h
1039 file path=usr/include/sys/fs/ufs_bio.h
1040 file path=usr/include/sys/fs/ufs_filio.h
1041 file path=usr/include/sys/fs/ufs_fs.h
1042 file path=usr/include/sys/fs/ufs_fsdir.h
1043 file path=usr/include/sys/fs/ufs_inode.h
1044 file path=usr/include/sys/fs/ufs_lockfs.h
1045 file path=usr/include/sys/fs/ufs_log.h
1046 file path=usr/include/sys/fs/ufs_mount.h
1047 file path=usr/include/sys/fs/ufs_panic.h
1048 file path=usr/include/sys/fs/ufs_prot.h
1049 file path=usr/include/sys/fs/ufs_quota.h

```

1050 file path=usr/include/sys/fs/ufs_snap.h
 1051 file path=usr/include/sys/fs/ufs_trans.h
 1052 file path=usr/include/sys/fs/zfs.h
 1053 file path=usr/include/sys/fs_reparse.h
 1054 file path=usr/include/sys/fs_subr.h
 1055 file path=usr/include/sys/fsid.h
 1056 \$(sparc_ONLY)file path=usr/include/sys/fsr.h
 1057 file path=usr/include/sys/fss.h
 1058 file path=usr/include/sys/fssnap.h
 1059 file path=usr/include/sys/fssnap_if.h
 1060 file path=usr/include/sys/fsspriocntl.h
 1061 file path=usr/include/sys/fstyp.h
 1062 file path=usr/include/sys/fttrace.h
 1063 file path=usr/include/sys/fx.h
 1064 file path=usr/include/sys/fxpriocntl.h
 1065 file path=usr/include/sys/gfs.h
 1066 \$(i386_ONLY)file path=usr/include/sys/gfx_private.h
 1067 file path=usr/include/sys/gld.h
 1068 file path=usr/include/sys/gldpriv.h
 1069 file path=usr/include/sys/group.h
 1070 file path=usr/include/sys/hdio.h
 1071 file path=usr/include/sys/hook.h
 1072 file path=usr/include/sys/hook_event.h
 1073 file path=usr/include/sys/hook_impl.h
 1074 file path=usr/include/sys/hotplug/hpcsvc.h
 1075 file path=usr/include/sys/hotplug/hpctrl.h
 1076 file path=usr/include/sys/hotplug/pci/pcicfg.h
 1077 file path=usr/include/sys/hotplug/pci/pcihp.h
 1078 file path=usr/include/sys/hwconf.h
 1079 \$(i386_ONLY)file path=usr/include/sys/hypervisor.h
 1080 \$(i386_ONLY)file path=usr/include/sys/i8272A.h
 1081 file path=usr/include/sys/ia.h
 1082 file path=usr/include/sys/iapriocntl.h
 1083 file path=usr/include/sys/ib/adapters/hermon/hermon_ioctl.h
 1084 file path=usr/include/sys/ib/adapters/mlnx_umap.h
 1085 file path=usr/include/sys/ib/adapters/tavor/tavor_ioctl.h
 1086 file path=usr/include/sys/ib/clients/ibd/ibd.h
 1087 file path=usr/include/sys/ib/clients/of/ofa_solaris.h
 1088 file path=usr/include/sys/ib/clients/of/ofed_kernel.h
 1089 file path=usr/include/sys/ib/clients/of/rdma/ib_addr.h
 1090 file path=usr/include/sys/ib/clients/of/rdma/ib_user_mad.h
 1091 file path=usr/include/sys/ib/clients/of/rdma/ib_user_sa.h
 1092 file path=usr/include/sys/ib/clients/of/rdma/ib_user_verbs.h
 1093 file path=usr/include/sys/ib/clients/of/rdma/ib_verbs.h
 1094 file path=usr/include/sys/ib/clients/of/rdma/rdma_cm.h
 1095 file path=usr/include/sys/ib/clients/of/rdma/rdma_user_cm.h
 1096 file path=usr/include/sys/ib/clients/of/sol_ofs/sol_cma.h
 1097 file path=usr/include/sys/ib/clients/of/sol_ofs/sol_ib_cma.h
 1098 file path=usr/include/sys/ib/clients/of/sol_ofs/sol_kverb_impl.h
 1099 file path=usr/include/sys/ib/clients/of/sol_ofs/sol_ofs_common.h
 1100 file path=usr/include/sys/ib/clients/of/sol_ucma/sol_rdma_user_cm.h
 1101 file path=usr/include/sys/ib/clients/of/sol_ucma/sol_ucma.h
 1102 file path=usr/include/sys/ib/clients/of/sol_umad/sol_umad.h
 1103 file path=usr/include/sys/ib/clients/of/sol_uverbs/sol_uverbs.h
 1104 file path=usr/include/sys/ib/clients/of/sol_uverbs/sol_uverbs2ucma.h
 1105 file path=usr/include/sys/ib/clients/of/sol_uverbs/sol_uverbs_comp.h
 1106 file path=usr/include/sys/ib/clients/of/sol_uverbs/sol_uverbs_event.h
 1107 file path=usr/include/sys/ib/clients/of/sol_uverbs/sol_uverbs_hca.h
 1108 file path=usr/include/sys/ib/clients/of/sol_uverbs/sol_uverbs_qp.h
 1109 file path=usr/include/sys/ib/ib_pkt_hdrs.h
 1110 file path=usr/include/sys/ib/ib_types.h
 1111 file path=usr/include/sys/ib/ibnex/ibnex_devctl.h
 1112 file path=usr/include/sys/ib/ibtl/ibci.h
 1113 file path=usr/include/sys/ib/ibtl/ibti.h
 1114 file path=usr/include/sys/ib/ibtl/ibti_cm.h
 1115 file path=usr/include/sys/ib/ibtl/ibti_common.h

1116 file path=usr/include/sys/ib/ibtl/ibtl_ci_types.h
 1117 file path=usr/include/sys/ib/ibtl/ibtl_status.h
 1118 file path=usr/include/sys/ib/ibtl/ibtl_types.h
 1119 file path=usr/include/sys/ib/ibtl/ibvti.h
 1120 file path=usr/include/sys/ib/ibtl/impl/ibtl_util.h
 1121 file path=usr/include/sys/ib/mgt/ib_dm_attr.h
 1122 file path=usr/include/sys/ib/mgt/ib_mad.h
 1123 file path=usr/include/sys/ib/mgt/ibmf/ibmf.h
 1124 file path=usr/include/sys/ib/mgt/ibmf/ibmf_msg.h
 1125 file path=usr/include/sys/ib/mgt/ibmf/ibmf_saa.h
 1126 file path=usr/include/sys/ib/mgt/ibmf/ibmf_utils.h
 1127 file path=usr/include/sys/ib/mgt/sa_recs.h
 1128 file path=usr/include/sys/ib/mgt/sm_attr.h
 1129 file path=usr/include/sys/ibpart.h
 1130 file path=usr/include/sys/id32.h
 1131 file path=usr/include/sys/id_space.h
 1132 file path=usr/include/sys/idmap.h
 1133 file path=usr/include/sys/inline.h
 1134 file path=usr/include/sys/instance.h
 1135 file path=usr/include/sys/int_const.h
 1136 file path=usr/include/sys/int_fmtio.h
 1137 file path=usr/include/sys/int_limits.h
 1138 file path=usr/include/sys/int_types.h
 1139 file path=usr/include/sys/inttypes.h
 1140 file path=usr/include/sys/ioccom.h
 1141 file path=usr/include/sys/ioctl.h
 1142 \$(i386_ONLY)file path=usr/include/sys/iomulib.h
 1143 file path=usr/include/sys/ipc.h
 1144 file path=usr/include/sys/ipc_impl.h
 1145 file path=usr/include/sys/ipc_rctl.h
 1146 file path=usr/include/sys/isa_defs.h
 1147 file path=usr/include/sys/iso/signal_iso.h
 1148 file path=usr/include/sys/jioctl.h
 1149 file path=usr/include/sys/kbd.h
 1150 file path=usr/include/sys/kbdreg.h
 1151 file path=usr/include/sys/kbio.h
 1152 file path=usr/include/sys/kcpc.h
 1153 file path=usr/include/sys/kd.h
 1154 file path=usr/include/sys/kdi.h
 1155 file path=usr/include/sys/kdi_impl.h
 1156 file path=usr/include/sys/kdi_machimpl.h
 1157 \$(i386_ONLY)file path=usr/include/sys/kdi_regs.h
 1158 file path=usr/include/sys/kiconv.h
 1159 file path=usr/include/sys/kidmap.h
 1160 file path=usr/include/sys/klpd.h
 1161 file path=usr/include/sys/klwp.h
 1162 file path=usr/include/sys/kmem.h
 1163 file path=usr/include/sys/kmem_impl.h
 1164 file path=usr/include/sys/kobj.h
 1165 file path=usr/include/sys/kobj_impl.h
 1166 file path=usr/include/sys/ksocket.h
 1167 file path=usr/include/sys/kstat.h
 1168 file path=usr/include/sys/kstr.h
 1169 file path=usr/include/sys/ksyms.h
 1170 file path=usr/include/sys/ksynch.h
 1171 file path=usr/include/sys/lc_core.h
 1172 file path=usr/include/sys/ldterm.h
 1173 file path=usr/include/sys/lgrp.h
 1174 file path=usr/include/sys/lgrp_user.h
 1175 file path=usr/include/sys/link.h
 1176 file path=usr/include/sys/list.h
 1177 file path=usr/include/sys/list_impl.h
 1178 file path=usr/include/sys/llcl.h
 1179 file path=usr/include/sys/loadavg.h
 1180 file path=usr/include/sys/localedef.h
 1181 file path=usr/include/sys/lock.h

1182 file path=usr/include/sys/lockfs.h
1183 file path=usr/include/sys/lofi.h
1184 file path=usr/include/sys/log.h
1185 file path=usr/include/sys/logindmux.h
1186 file path=usr/include/sys/lwp.h
1187 file path=usr/include/sys/lwp_timer_impl.h
1188 file path=usr/include/sys/lwp_upimutex_impl.h
1189 file path=usr/include/sys/mac.h
1190 file path=usr/include/sys/mac_ether.h
1191 file path=usr/include/sys/mac_flow.h
1192 file path=usr/include/sys/mac_provider.h
1193 file path=usr/include/sys/machelf.h
1194 file path=usr/include/sys/machlock.h
1195 file path=usr/include/sys/machsig.h
1196 file path=usr/include/sys/machtypes.h
1197 file path=usr/include/sys/map.h
1198 \$(i386_ONLY)file path=usr/include/sys/mc.h
1199 \$(i386_ONLY)file path=usr/include/sys/mc_amd.h
1200 \$(i386_ONLY)file path=usr/include/sys/mc_intel.h
1201 \$(i386_ONLY)file path=usr/include/sys/mca_amd.h
1202 \$(i386_ONLY)file path=usr/include/sys/mca_x86.h
1203 file path=usr/include/sys/mcontext.h
1204 file path=usr/include/sys/md4.h
1205 file path=usr/include/sys/md5.h
1206 file path=usr/include/sys/md5_consts.h
1207 file path=usr/include/sys/mdi_impldefs.h
1208 file path=usr/include/sys/mem.h
1209 file path=usr/include/sys/mem_config.h
1210 file path=usr/include/sys/memlist.h
1211 file path=usr/include/sys/mhd.h
1212 file path=usr/include/sys/mii.h
1213 file path=usr/include/sys/miiregs.h
1214 file path=usr/include/sys/mkdev.h
1215 file path=usr/include/sys/mman.h
1216 file path=usr/include/sys/mmapobj.h
1217 file path=usr/include/sys/mntent.h
1218 file path=usr/include/sys/mntio.h
1219 file path=usr/include/sys/mnttab.h
1220 file path=usr/include/sys/modctl.h
1221 file path=usr/include/sys/mode.h
1222 file path=usr/include/sys/model.h
1223 file path=usr/include/sys/modhash.h
1224 file path=usr/include/sys/modhash_impl.h
1225 file path=usr/include/sys/mount.h
1226 file path=usr/include/sys/mouse.h
1227 file path=usr/include/sys/msacct.h
1228 file path=usr/include/sys/msg.h
1229 file path=usr/include/sys/msg_impl.h
1230 file path=usr/include/sys/msio.h
1231 file path=usr/include/sys/msreg.h
1232 file path=usr/include/sys/mtio.h
1233 file path=usr/include/sys/multidata.h
1234 file path=usr/include/sys/mutex.h
1235 \$(i386_ONLY)file path=usr/include/sys/mutex_impl.h
1236 file path=usr/include/sys/nbmlck.h
1237 file path=usr/include/sys/ndi_impldefs.h
1238 file path=usr/include/sys/ndifm.h
1239 file path=usr/include/sys/netconfig.h
1240 file path=usr/include/sys/neti.h
1241 file path=usr/include/sys/netstack.h
1242 file path=usr/include/sys/nexusdefs.h
1243 file path=usr/include/sys/note.h
1244 file path=usr/include/sys/null.h
1245 file path=usr/include/sys/nvpair.h
1246 file path=usr/include/sys/nvpair_impl.h
1247 file path=usr/include/sys/objfs.h

1248 file path=usr/include/sys/objfs_impl.h
1249 file path=usr/include/sys/obpdefs.h
1250 file path=usr/include/sys/old_procfs.h
1251 file path=usr/include/sys/open.h
1252 file path=usr/include/sys/openpromio.h
1253 file path=usr/include/sys/panic.h
1254 file path=usr/include/sys/param.h
1255 file path=usr/include/sys/pathconf.h
1256 file path=usr/include/sys/pathname.h
1257 file path=usr/include/sys/pattr.h
1258 file path=usr/include/sys/pbio.h
1259 file path=usr/include/sys/pcb.h
1260 file path=usr/include/sys/pccard.h
1261 file path=usr/include/sys/pci.h
1262 \$(i386_ONLY)file path=usr/include/sys/pcic_reg.h
1263 \$(i386_ONLY)file path=usr/include/sys/pcic_var.h
1264 file path=usr/include/sys/pcie.h
1265 file path=usr/include/sys/pcmcia.h
1266 file path=usr/include/sys/pctypes.h
1267 file path=usr/include/sys/pfmod.h
1268 file path=usr/include/sys/pg.h
1269 file path=usr/include/sys/pghw.h
1270 file path=usr/include/sys/phymem.h
1271 \$(i386_ONLY)file path=usr/include/sys/pic.h
1272 \$(i386_ONLY)file path=usr/include/sys/pit.h
1273 file path=usr/include/sys/pkg_hash.h
1274 file path=usr/include/sys/pm.h
1275 \$(i386_ONLY)file path=usr/include/sys/pmem.h
1276 file path=usr/include/sys/policy.h
1277 file path=usr/include/sys/poll.h
1278 file path=usr/include/sys/poll_impl.h
1279 file path=usr/include/sys/pool.h
1280 file path=usr/include/sys/pool_impl.h
1281 file path=usr/include/sys/pool_pset.h
1282 file path=usr/include/sys/port.h
1283 file path=usr/include/sys/port_impl.h
1284 file path=usr/include/sys/port_kernel.h
1285 file path=usr/include/sys/ppmio.h
1286 file path=usr/include/sys/priocntl.h
1287 file path=usr/include/sys/priv.h
1288 file path=usr/include/sys/priv_const.h
1289 file path=usr/include/sys/priv_impl.h
1290 file path=usr/include/sys/priv_names.h
1291 \$(i386_ONLY)file path=usr/include/sys/privmregs.h
1292 \$(i386_ONLY)file path=usr/include/sys/privregs.h
1293 file path=usr/include/sys/prnio.h
1294 file path=usr/include/sys/proc.h
1295 file path=usr/include/sys/proc/prdata.h
1296 file path=usr/include/sys/processor.h
1297 file path=usr/include/sys/procfs.h
1298 file path=usr/include/sys/procfs_isa.h
1299 file path=usr/include/sys/procset.h
1300 file path=usr/include/sys/project.h
1301 \$(i386_ONLY)file path=usr/include/sys/prom_emul.h
1302 \$(i386_ONLY)file path=usr/include/sys/prom_isa.h
1303 \$(i386_ONLY)file path=usr/include/sys/prom_plat.h
1304 file path=usr/include/sys/promif.h
1305 file path=usr/include/sys/promimpl.h
1306 file path=usr/include/sys/protosw.h
1307 file path=usr/include/sys/prsystem.h
1308 file path=usr/include/sys/pset.h
1309 file path=usr/include/sys/psw.h
1310 \$(i386_ONLY)file path=usr/include/sys/pte.h
1311 file path=usr/include/sys/ptem.h
1312 file path=usr/include/sys/ptms.h
1313 file path=usr/include/sys/ptyvar.h

```

1314 file path=usr/include/sys/queue.h
1315 file path=usr/include/sys/raidioctl.h
1316 file path=usr/include/sys/ramdisk.h
1317 file path=usr/include/sys/random.h
1318 file path=usr/include/sys/rctl.h
1319 file path=usr/include/sys/rctl_impl.h
1320 file path=usr/include/sys/rds.h
1321 file path=usr/include/sys/reboot.h
1322 file path=usr/include/sys/refstr.h
1323 file path=usr/include/sys/refstr_impl.h
1324 file path=usr/include/sys/reg.h
1325 file path=usr/include/sys/regset.h
1326 file path=usr/include/sys/resource.h
1327 file path=usr/include/sys/rliocntl.h
1328 file path=usr/include/sys/rsm/rsm.h
1329 file path=usr/include/sys/rsm/rsm_common.h
1330 file path=usr/include/sys/rsm/rsmapi_common.h
1331 file path=usr/include/sys/rsm/rsmka_path_int.h
1332 file path=usr/include/sys/rsm/rsmmdi.h
1333 file path=usr/include/sys/rsm/rsmpi.h
1334 file path=usr/include/sys/rsm/rsmpi_driver.h
1335 file path=usr/include/sys/rt.h
1336 $(i386_ONLY)file path=usr/include/sys/rtc.h
1337 file path=usr/include/sys/rtpiocntl.h
1338 file path=usr/include/sys/rwlock.h
1339 file path=usr/include/sys/rwlock_impl.h
1340 file path=usr/include/sys/rwstlock.h
1341 file path=usr/include/sys/sad.h
1342 $(i386_ONLY)file path=usr/include/sys/sata/sata_defs.h
1343 $(i386_ONLY)file path=usr/include/sys/sata/sata_hba.h
1344 file path=usr/include/sys/schedctl.h
1345 $(sparc_ONLY)file path=usr/include/sys/scsi/adapters/ifpio.h
1346 file path=usr/include/sys/scsi/adapters/scsi_vhci.h
1347 $(sparc_ONLY)file path=usr/include/sys/scsi/adapters/sfvar.h
1348 file path=usr/include/sys/scsi/conf/autoconf.h
1349 file path=usr/include/sys/scsi/conf/device.h
1350 file path=usr/include/sys/scsi/generic/commands.h
1351 file path=usr/include/sys/scsi/generic/dad_mode.h
1352 file path=usr/include/sys/scsi/generic/inquiry.h
1353 file path=usr/include/sys/scsi/generic/message.h
1354 file path=usr/include/sys/scsi/generic/mode.h
1355 file path=usr/include/sys/scsi/generic/persist.h
1356 file path=usr/include/sys/scsi/generic/sense.h
1357 file path=usr/include/sys/scsi/generic/sff_frames.h
1358 file path=usr/include/sys/scsi/generic/smp_frames.h
1359 file path=usr/include/sys/scsi/generic/status.h
1360 file path=usr/include/sys/scsi/impl/commands.h
1361 file path=usr/include/sys/scsi/impl/inquiry.h
1362 file path=usr/include/sys/scsi/impl/mode.h
1363 file path=usr/include/sys/scsi/impl/scsi_reset_notify.h
1364 file path=usr/include/sys/scsi/impl/scsi_sas.h
1365 file path=usr/include/sys/scsi/impl/sense.h
1366 file path=usr/include/sys/scsi/impl/services.h
1367 file path=usr/include/sys/scsi/impl/smp_transport.h
1368 file path=usr/include/sys/scsi/impl/spc3_types.h
1369 file path=usr/include/sys/scsi/impl/status.h
1370 file path=usr/include/sys/scsi/impl/transport.h
1371 file path=usr/include/sys/scsi/impl/types.h
1372 file path=usr/include/sys/scsi/impl/uscsi.h
1373 file path=usr/include/sys/scsi/impl/usmp.h
1374 file path=usr/include/sys/scsi/scsi.h
1375 file path=usr/include/sys/scsi/scsi_address.h
1376 file path=usr/include/sys/scsi/scsi_ctl.h
1377 file path=usr/include/sys/scsi/scsi_fm.h
1378 file path=usr/include/sys/scsi/scsi_names.h
1379 file path=usr/include/sys/scsi/scsi_params.h

```

```

1380 file path=usr/include/sys/scsi/scsi_pkt.h
1381 file path=usr/include/sys/scsi/scsi_resource.h
1382 file path=usr/include/sys/scsi/scsi_types.h
1383 file path=usr/include/sys/scsi/scsi_watch.h
1384 file path=usr/include/sys/scsi/targets/sddef.h
1385 file path=usr/include/sys/scsi/targets/ses.h
1386 file path=usr/include/sys/scsi/targets/sesio.h
1387 file path=usr/include/sys/scsi/targets/sgendef.h
1388 file path=usr/include/sys/scsi/targets/smp.h
1389 $(sparc_ONLY)file path=usr/include/sys/scsi/targets/ssddef.h
1390 file path=usr/include/sys/scsi/targets/stdef.h
1391 file path=usr/include/sys/secflags.h
1392 $(i386_ONLY)file path=usr/include/sys/segment.h
1393 $(i386_ONLY)file path=usr/include/sys/segments.h
1394 file path=usr/include/sys/select.h
1395 file path=usr/include/sys/sem.h
1396 file path=usr/include/sys/sem_impl.h
1397 file path=usr/include/sys/semaphore.h
1398 file path=usr/include/sys/semaphore.h
1399 file path=usr/include/sys/sendfile.h
1400 $(sparc_ONLY)file path=usr/include/sys/ser_async.h
1401 file path=usr/include/sys/ser_sync.h
1402 $(sparc_ONLY)file path=usr/include/sys/ser_zscc.h
1403 file path=usr/include/sys/serializer.h
1404 file path=usr/include/sys/session.h
1405 file path=usr/include/sys/sha1.h
1406 file path=usr/include/sys/sha2.h
1407 file path=usr/include/sys/share.h
1408 file path=usr/include/sys/shm.h
1409 file path=usr/include/sys/shm_impl.h
1410 file path=usr/include/sys/sid.h
1411 file path=usr/include/sys/siginfo.h
1412 file path=usr/include/sys/signal.h
1413 file path=usr/include/sys/signalfd.h
1414 file path=usr/include/sys/skein.h
1415 file path=usr/include/sys/sleepq.h
1416 file path=usr/include/sys/smbios.h
1417 file path=usr/include/sys/smbios_impl.h
1418 file path=usr/include/sys/smedia.h
1419 file path=usr/include/sys/sobject.h
1420 $(sparc_ONLY)file path=usr/include/sys/socal_cq_defs.h
1421 $(sparc_ONLY)file path=usr/include/sys/socalio.h
1422 $(sparc_ONLY)file path=usr/include/sys/socalmap.h
1423 $(sparc_ONLY)file path=usr/include/sys/socalreg.h
1424 $(sparc_ONLY)file path=usr/include/sys/socalvar.h
1425 file path=usr/include/sys/socket.h
1426 file path=usr/include/sys/socket_impl.h
1427 file path=usr/include/sys/socket_proto.h
1428 file path=usr/include/sys/socketvar.h
1429 file path=usr/include/sys/sockio.h
1430 file path=usr/include/sys/spl.h
1431 file path=usr/include/sys/squeue.h
1432 file path=usr/include/sys/squeue_impl.h
1433 file path=usr/include/sys/sservice.h
1434 file path=usr/include/sys/stack.h
1435 file path=usr/include/sys/stat.h
1436 file path=usr/include/sys/stat_impl.h
1437 file path=usr/include/sys/statfs.h
1438 file path=usr/include/sys/statvfs.h
1439 file path=usr/include/sys/stdbool.h
1440 file path=usr/include/sys/stddef.h
1441 file path=usr/include/sys/stdint.h
1442 file path=usr/include/sys/stermio.h
1443 file path=usr/include/sys/stream.h
1444 file path=usr/include/sys/strft.h
1445 file path=usr/include/sys/strlog.h

```

1446 file path=usr/include/sys/strmdep.h
 1447 file path=usr/include/sys/stropts.h
 1448 file path=usr/include/sys/strredir.h
 1449 file path=usr/include/sys/strstat.h
 1450 file path=usr/include/sys/strsubr.h
 1451 file path=usr/include/sys/strsun.h
 1452 file path=usr/include/sys/strtty.h
 1453 file path=usr/include/sys/sunddi.h
 1454 file path=usr/include/sys/sunldi.h
 1455 file path=usr/include/sys/sunldi_impl.h
 1456 file path=usr/include/sys/sunmdi.h
 1457 file path=usr/include/sys/sunndi.h
 1458 file path=usr/include/sys/sunpm.h
 1459 file path=usr/include/sys/suntpi.h
 1460 file path=usr/include/sys/suntty.h
 1461 file path=usr/include/sys/swap.h
 1462 file path=usr/include/sys/synch.h
 1463 file path=usr/include/sys/syscall.h
 1464 file path=usr/include/sys/sysconf.h
 1465 file path=usr/include/sys/sysconfig.h
 1466 file path=usr/include/sys/sysconfig_impl.h
 1467 file path=usr/include/sys/sysdc.h
 1468 file path=usr/include/sys/sysdc_impl.h
 1469 file path=usr/include/sys/sysevent.h
 1470 file path=usr/include/sys/sysevent/ap_driver.h
 1471 file path=usr/include/sys/sysevent/dev.h
 1472 file path=usr/include/sys/sysevent/domain.h
 1473 file path=usr/include/sys/sysevent/dr.h
 1474 file path=usr/include/sys/sysevent/env.h
 1475 file path=usr/include/sys/sysevent/eventdefs.h
 1476 file path=usr/include/sys/sysevent/ipmp.h
 1477 file path=usr/include/sys/sysevent/pwrctl.h
 1478 file path=usr/include/sys/sysevent/vrrp.h
 1479 file path=usr/include/sys/sysevent_impl.h
 1480 \$(i386_ONLY)file path=usr/include/sys/sysi86.h
 1481 file path=usr/include/sys/sysinfo.h
 1482 file path=usr/include/sys/syslog.h
 1483 file path=usr/include/sys/sysmacros.h
 1484 file path=usr/include/sys/systeminfo.h
 1485 file path=usr/include/sys/system.h
 1486 file path=usr/include/sys/t_kuser.h
 1487 file path=usr/include/sys/t_lock.h
 1488 file path=usr/include/sys/task.h
 1489 file path=usr/include/sys/taskq.h
 1490 file path=usr/include/sys/taskq_impl.h
 1491 file path=usr/include/sys/telioclt.h
 1492 file path=usr/include/sys/termio.h
 1493 file path=usr/include/sys/termios.h
 1494 file path=usr/include/sys/termiox.h
 1495 file path=usr/include/sys/thread.h
 1496 file path=usr/include/sys/ticlts.h
 1497 file path=usr/include/sys/ticots.h
 1498 file path=usr/include/sys/ticotsord.h
 1499 file path=usr/include/sys/tihdr.h
 1500 file path=usr/include/sys/time.h
 1501 file path=usr/include/sys/time_impl.h
 1502 file path=usr/include/sys/time_std_impl.h
 1503 file path=usr/include/sys/timeb.h
 1504 file path=usr/include/sys/timer.h
 1505 file path=usr/include/sys/timerfd.h
 1506 file path=usr/include/sys/times.h
 1507 file path=usr/include/sys/timex.h
 1508 file path=usr/include/sys/timod.h
 1509 file path=usr/include/sys/tirdwr.h
 1510 file path=usr/include/sys/tiuser.h
 1511 file path=usr/include/sys/tl.h

1512 file path=usr/include/sys/tnf.h
 1513 file path=usr/include/sys/tnf_com.h
 1514 file path=usr/include/sys/tnf_probe.h
 1515 file path=usr/include/sys/tnf_writer.h
 1516 file path=usr/include/sys/todio.h
 1517 file path=usr/include/sys/tpicommon.h
 1518 file path=usr/include/sys/trap.h
 1519 \$(i386_ONLY)file path=usr/include/sys/traptrace.h
 1520 file path=usr/include/sys/ts.h
 1521 file path=usr/include/sys/tsol/label.h
 1522 file path=usr/include/sys/tsol/label_macro.h
 1523 file path=usr/include/sys/tsol/priv.h
 1524 file path=usr/include/sys/tsol/tndb.h
 1525 file path=usr/include/sys/tsol/tsyscall.h
 1526 file path=usr/include/sys/tspriocntl.h
 1527 \$(i386_ONLY)file path=usr/include/sys/tss.h
 1528 file path=usr/include/sys/ttcompat.h
 1529 file path=usr/include/sys/ttold.h
 1530 file path=usr/include/sys/tty.h
 1531 file path=usr/include/sys/ttychars.h
 1532 file path=usr/include/sys/ttydev.h
 1533 \$(sparc_ONLY)file path=usr/include/sys/ttymux.h
 1534 \$(sparc_ONLY)file path=usr/include/sys/ttymuxuser.h
 1535 file path=usr/include/sys/tuneable.h
 1536 file path=usr/include/sys/turnstile.h
 1537 file path=usr/include/sys/types.h
 1538 file path=usr/include/sys/types32.h
 1539 file path=usr/include/sys/tzfile.h
 1540 file path=usr/include/sys/u8_textprep.h
 1541 file path=usr/include/sys/uadmin.h
 1542 \$(i386_ONLY)file path=usr/include/sys/ucode.h
 1543 file path=usr/include/sys/ucontext.h
 1544 file path=usr/include/sys/uiio.h
 1545 file path=usr/include/sys/ulimit.h
 1546 file path=usr/include/sys/un.h
 1547 file path=usr/include/sys/unistd.h
 1548 file path=usr/include/sys/user.h
 1549 file path=usr/include/sys/ustat.h
 1550 file path=usr/include/sys/utime.h
 1551 file path=usr/include/sys/utrap.h
 1552 file path=usr/include/sys/utsname.h
 1553 file path=usr/include/sys/utssys.h
 1554 file path=usr/include/sys/uuid.h
 1555 file path=usr/include/sys/va_impl.h
 1556 file path=usr/include/sys/va_list.h
 1557 file path=usr/include/sys/var.h
 1558 file path=usr/include/sys/varargs.h
 1559 file path=usr/include/sys/vfs.h
 1560 file path=usr/include/sys/vfs_opreg.h
 1561 file path=usr/include/sys/vfstab.h
 1562 file path=usr/include/sys/videodev2.h
 1563 file path=usr/include/sys/visual_io.h
 1564 file path=usr/include/sys/vm.h
 1565 file path=usr/include/sys/vm_usage.h
 1566 file path=usr/include/sys/vmem.h
 1567 file path=usr/include/sys/vmem_impl.h
 1568 file path=usr/include/sys/vmem_impl_user.h
 1569 file path=usr/include/sys/vmparam.h
 1570 file path=usr/include/sys/vmsystem.h
 1571 file path=usr/include/sys/vnode.h
 1572 file path=usr/include/sys/vt.h
 1573 file path=usr/include/sys/vtdaemon.h
 1574 file path=usr/include/sys/vtoc.h
 1575 file path=usr/include/sys/vtrace.h
 1576 file path=usr/include/sys/vuid_event.h
 1577 file path=usr/include/sys/vuid_queue.h

```

1578 file path=usr/include/sys/vuid_state.h
1579 file path=usr/include/sys/vuid_store.h
1580 file path=usr/include/sys/vuid_wheel.h
1581 file path=usr/include/sys/wait.h
1582 file path=usr/include/sys/waitq.h
1583 file path=usr/include/sys/watchpoint.h
1584 $(i386_ONLY)file path=usr/include/sys/x86_archext.h
1585 $(i386_ONLY)file path=usr/include/sys/xen_errno.h
1586 file path=usr/include/sys/xti_inet.h
1587 file path=usr/include/sys/xti_osi.h
1588 file path=usr/include/sys/xti_xtiopt.h
1589 file path=usr/include/sys/zcons.h
1590 file path=usr/include/sys/zmod.h
1591 file path=usr/include/sys/zone.h
1592 $(sparc_ONLY)file path=usr/include/sys/zsdev.h
1593 file path=usr/include/sys/exits.h
1594 file path=usr/include/syslog.h
1595 file path=usr/include/tar.h
1596 file path=usr/include/tcpd.h
1597 file path=usr/include/term.h
1598 file path=usr/include/termcap.h
1599 file path=usr/include/termio.h
1600 file path=usr/include/termios.h
1601 file path=usr/include/thread.h
1602 file path=usr/include/thread_db.h
1603 file path=usr/include/threads.h
1604 file path=usr/include/time.h
1605 file path=usr/include/tiuser.h
1606 file path=usr/include/tsol/label.h
1607 file path=usr/include/tzfile.h
1608 file path=usr/include/ucontext.h
1609 file path=usr/include/ucrd.h
1610 file path=usr/include/uid_stp.h
1611 file path=usr/include/ulimit.h
1612 file path=usr/include/umem.h
1613 file path=usr/include/umem_impl.h
1614 file path=usr/include/unctrl.h
1615 file path=usr/include/unistd.h
1616 file path=usr/include/user_attr.h
1617 file path=usr/include/userdefs.h
1618 file path=usr/include/ustat.h
1619 file path=usr/include/utility.h
1620 file path=usr/include/utime.h
1621 file path=usr/include/utmp.h
1622 file path=usr/include/utmpx.h
1623 file path=usr/include/uuid/uuid.h
1624 $(sparc_ONLY)file path=usr/include/v7/sys/machpcb.h
1625 $(sparc_ONLY)file path=usr/include/v7/sys/machtrap.h
1626 $(sparc_ONLY)file path=usr/include/v7/sys/mutex_impl.h
1627 $(sparc_ONLY)file path=usr/include/v7/sys/privregs.h
1628 $(sparc_ONLY)file path=usr/include/v7/sys/prom_isa.h
1629 $(sparc_ONLY)file path=usr/include/v7/sys/psr.h
1630 $(sparc_ONLY)file path=usr/include/v7/sys/traptrace.h
1631 $(sparc_ONLY)file path=usr/include/v9/sys/asi.h
1632 $(sparc_ONLY)file path=usr/include/v9/sys/machpcb.h
1633 $(sparc_ONLY)file path=usr/include/v9/sys/machtrap.h
1634 $(sparc_ONLY)file path=usr/include/v9/sys/membar.h
1635 $(sparc_ONLY)file path=usr/include/v9/sys/mutex_impl.h
1636 $(sparc_ONLY)file path=usr/include/v9/sys/privregs.h
1637 $(sparc_ONLY)file path=usr/include/v9/sys/prom_isa.h
1638 $(sparc_ONLY)file path=usr/include/v9/sys/psr_compat.h
1639 $(sparc_ONLY)file path=usr/include/v9/sys/vis_simulator.h
1640 file path=usr/include/valtools.h
1641 file path=usr/include/values.h
1642 file path=usr/include/varargs.h
1643 file path=usr/include/vm/anon.h

```

```

1644 file path=usr/include/vm/as.h
1645 file path=usr/include/vm/faultcode.h
1646 file path=usr/include/vm/hat.h
1647 file path=usr/include/vm/kpm.h
1648 file path=usr/include/vm/page.h
1649 file path=usr/include/vm/pvn.h
1650 file path=usr/include/vm/rm.h
1651 file path=usr/include/vm/seg.h
1652 file path=usr/include/vm/seg_dev.h
1653 file path=usr/include/vm/seg_enum.h
1654 file path=usr/include/vm/seg_kmem.h
1655 file path=usr/include/vm/seg_kp.h
1656 file path=usr/include/vm/seg_kpm.h
1657 file path=usr/include/vm/seg_map.h
1658 file path=usr/include/vm/seg_spt.h
1659 file path=usr/include/vm/seg_vn.h
1660 file path=usr/include/vm/vpage.h
1661 file path=usr/include/vm/vpm.h
1662 file path=usr/include/volmgt.h
1663 file path=usr/include/wait.h
1664 file path=usr/include/wchar.h
1665 file path=usr/include/wchar_impl.h
1666 file path=usr/include/wctype.h
1667 file path=usr/include/widec.h
1668 file path=usr/include/wordep.h
1669 file path=usr/include/xlocale.h
1670 file path=usr/include/xti.h
1671 file path=usr/include/xti_inet.h
1672 file path=usr/include/zone.h
1673 file path=usr/include/zonestat.h
1674 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/acpidev.h
1675 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/amd_iommu.h
1676 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/asm_misc.h
1677 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/clock.h
1678 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/cram.h
1679 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/ddi_subrdefs.h
1680 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/debug_info.h
1681 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/fastboot.h
1682 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/ht.h
1683 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/mach_mmu.h
1684 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/machclock.h
1685 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/machcpuvar.h
1686 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/machparam.h
1687 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/machprivregs.h
1688 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/machsystem.h
1689 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/machthread.h
1690 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/memnode.h
1691 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/pc_mmu.h
1692 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/psm.h
1693 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/psm_defs.h
1694 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/psm_modctl.h
1695 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/psm_types.h
1696 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/rm_platter.h
1697 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/sbd_ioctl.h
1698 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/smp_impldefs.h
1699 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/vm_machparam.h
1700 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/x_call.h
1701 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/xc_levels.h
1702 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/xsvc.h
1703 $(i386_ONLY)file path=usr/platform/i86pc/include/vm/hat_i86.h
1704 $(i386_ONLY)file path=usr/platform/i86pc/include/vm/hat_pte.h
1705 $(i386_ONLY)file path=usr/platform/i86pc/include/vm/hment.h
1706 $(i386_ONLY)file path=usr/platform/i86pc/include/vm/htable.h
1707 $(i386_ONLY)file path=usr/platform/i86pc/include/vm/kboot_mmu.h
1708 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/balloon.h
1709 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/machprivregs.h

```

```

1710 $(i386_ONLY)file path=usr/platform/i86xpv/include/sys/xen_mmu.h
1711 $(i386_ONLY)file path=usr/platform/i86xpv/include/sys/xpv_impl.h
1712 $(i386_ONLY)file path=usr/platform/i86xpv/include/vm/seg_mf.h
1713 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/ac.h
1714 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/async.h
1715 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/cheetahregs.h
1716 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/cherrystone.h
1717 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/clock.h
1718 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/cmp.h
1719 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/cpc_ultra.h
1720 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/cpr_impl.h
1721 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/cpu_impl.h
1722 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/cpu_sgnblk_defs.h
1723 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/daktari.h
1724 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/ddi_subrdefs.h
1725 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/dvma.h
1726 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/ecc_kstat.h
1727 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/eeeprom.h
1728 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/envctrl.h
1729 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/envctrl_gen.h
1730 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/envctrl_ue250.h
1731 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/envctrl_ue450.h
1732 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/envIRON.h
1733 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/errclassify.h
1734 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/fhc.h
1735 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/gpio_87317.h
1736 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/hpc3130_events.h
1737 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/ht.h
1738 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/i2c_clients/hpc3130.h
1739 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/i2c_clients/i2c_client.h
1740 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/i2c_clients/lm75.h
1741 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/i2c_clients/max1617.h
1742 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/i2c_clients/pcf8591.h
1743 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/i2c_clients/ssc050.h
1744 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/i2c/misc/i2c_svc.h
1745 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/idprom.h
1746 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/intr.h
1747 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/intreg.h
1748 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/iocache.h
1749 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/iommu.h
1750 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/ivintr.h
1751 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/lom_io.h
1752 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/machasi.h
1753 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/machclock.h
1754 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/machcpuvar.h
1755 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/machparam.h
1756 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/machsystem.h
1757 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/machthread.h
1758 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/mem_cache.h
1759 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/memlist_plat.h
1760 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/memnode.h
1761 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/mmu.h
1762 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/nexusdebug.h
1763 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/opl_hwdesc.h
1764 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/opl_module.h
1765 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/prom_debug.h
1766 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/prom_plat.h
1767 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/pte.h
1768 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/sbd_ioctl.h
1769 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/scb.h
1770 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/scsb_led.h
1771 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/simmstat.h
1772 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/spitregs.h
1773 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/sram.h
1774 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/sun4asi.h
1775 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/sysctrl.h

```

```

1776 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/sysioerr.h
1777 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/syiosbus.h
1778 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/tod.h
1779 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/todmostek.h
1780 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/trapstat.h
1781 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/traptrace.h
1782 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/vis.h
1783 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/vm_machparam.h
1784 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/x_call.h
1785 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/xc_impl.h
1786 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/zsmach.h
1787 $(sparc_ONLY)file path=usr/platform/sun4u/include/vm/hat_sfmmu.h
1788 $(sparc_ONLY)file path=usr/platform/sun4u/include/vm/mach_sfmmu.h
1789 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/clock.h
1790 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/cmp.h
1791 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/cpc_ultra.h
1792 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/cpu_sgnblk_defs.h
1793 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/ddi_subrdefs.h
1794 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/ds_pri.h
1795 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/ds_snmp.h
1796 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/dvma.h
1797 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/eeeprom.h
1798 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/fcode.h
1799 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/hsvc.h
1800 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/ht.h
1801 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/hypervisor_api.h
1802 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/idprom.h
1803 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/intr.h
1804 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/intreg.h
1805 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/ivintr.h
1806 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/machasi.h
1807 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/machclock.h
1808 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/machcpuvar.h
1809 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/machintreg.h
1810 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/machparam.h
1811 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/machsystem.h
1812 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/machthread.h
1813 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/memlist_plat.h
1814 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/memnode.h
1815 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/mmu.h
1816 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/nexusdebug.h
1817 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/niagaraasi.h
1818 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/niagararegs.h
1819 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/ntwdt.h
1820 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/pri.h
1821 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/prom_debug.h
1822 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/prom_plat.h
1823 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/pte.h
1824 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/qcn.h
1825 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/scb.h
1826 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/soft_state.h
1827 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/sun4asi.h
1828 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/tod.h
1829 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/trapstat.h
1830 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/traptrace.h
1831 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/vis.h
1832 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/vm_machparam.h
1833 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/x_call.h
1834 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/xc_impl.h
1835 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/zsmach.h
1836 $(sparc_ONLY)file path=usr/platform/sun4v/include/vm/hat_sfmmu.h
1837 $(sparc_ONLY)file path=usr/platform/sun4v/include/vm/mach_sfmmu.h
1838 file path=usr/share/man/man3head/acct.h.3head
1839 file path=usr/share/man/man3head/aio.h.3head
1840 file path=usr/share/man/man3head/ar.h.3head
1841 file path=usr/share/man/man3head/archives.h.3head

```

```

1842 file path=usr/share/man/man3head/assert.h.3head
1843 file path=usr/share/man/man3head/complex.h.3head
1844 file path=usr/share/man/man3head/cpio.h.3head
1845 file path=usr/share/man/man3head/dirent.h.3head
1846 file path=usr/share/man/man3head/ndian.h.3head
1847 file path=usr/share/man/man3head/errno.h.3head
1848 file path=usr/share/man/man3head/fcntl.h.3head
1849 file path=usr/share/man/man3head/fenv.h.3head
1850 file path=usr/share/man/man3head/float.h.3head
1851 file path=usr/share/man/man3head/floatpoint.h.3head
1852 file path=usr/share/man/man3head/fmtmsg.h.3head
1853 file path=usr/share/man/man3head/fnmatch.h.3head
1854 file path=usr/share/man/man3head/ftw.h.3head
1855 file path=usr/share/man/man3head/glob.h.3head
1856 file path=usr/share/man/man3head/grp.h.3head
1857 file path=usr/share/man/man3head/iconv.h.3head
1858 file path=usr/share/man/man3head/if.h.3head
1859 file path=usr/share/man/man3head/in.h.3head
1860 file path=usr/share/man/man3head/inet.h.3head
1861 file path=usr/share/man/man3head/inttypes.h.3head
1862 file path=usr/share/man/man3head/ipc.h.3head
1863 file path=usr/share/man/man3head/iso646.h.3head
1864 file path=usr/share/man/man3head/langinfo.h.3head
1865 file path=usr/share/man/man3head/libgen.h.3head
1866 file path=usr/share/man/man3head/libintl.h.3head
1867 file path=usr/share/man/man3head/limits.h.3head
1868 file path=usr/share/man/man3head/locale.h.3head
1869 file path=usr/share/man/man3head/math.h.3head
1870 file path=usr/share/man/man3head/mman.h.3head
1871 file path=usr/share/man/man3head/monetary.h.3head
1872 file path=usr/share/man/man3head/mqueue.h.3head
1873 file path=usr/share/man/man3head/msg.h.3head
1874 file path=usr/share/man/man3head/ndbm.h.3head
1875 file path=usr/share/man/man3head/netdb.h.3head
1876 file path=usr/share/man/man3head/nl_types.h.3head
1877 file path=usr/share/man/man3head/poll.h.3head
1878 file path=usr/share/man/man3head/pthread.h.3head
1879 file path=usr/share/man/man3head/pwd.h.3head
1880 file path=usr/share/man/man3head/queue.h.3head
1881 file path=usr/share/man/man3head/regex.h.3head
1882 file path=usr/share/man/man3head/resource.h.3head
1883 file path=usr/share/man/man3head/sched.h.3head
1884 file path=usr/share/man/man3head/search.h.3head
1885 file path=usr/share/man/man3head/select.h.3head
1886 file path=usr/share/man/man3head/sem.h.3head
1887 file path=usr/share/man/man3head/semaphore.h.3head
1888 file path=usr/share/man/man3head/setjmp.h.3head
1889 file path=usr/share/man/man3head/shm.h.3head
1890 file path=usr/share/man/man3head/signinfo.h.3head
1891 file path=usr/share/man/man3head/signal.h.3head
1892 file path=usr/share/man/man3head/socket.h.3head
1893 file path=usr/share/man/man3head/spawn.h.3head
1894 file path=usr/share/man/man3head/stat.h.3head
1895 file path=usr/share/man/man3head/statvfs.h.3head
1896 file path=usr/share/man/man3head/stdbool.h.3head
1897 file path=usr/share/man/man3head/stddef.h.3head
1898 file path=usr/share/man/man3head/stdint.h.3head
1899 file path=usr/share/man/man3head/stdio.h.3head
1900 file path=usr/share/man/man3head/stdlib.h.3head
1901 file path=usr/share/man/man3head/string.h.3head
1902 file path=usr/share/man/man3head/strings.h.3head
1903 file path=usr/share/man/man3head/stropts.h.3head
1904 file path=usr/share/man/man3head/syslog.h.3head
1905 file path=usr/share/man/man3head/tar.h.3head
1906 file path=usr/share/man/man3head/tcp.h.3head
1907 file path=usr/share/man/man3head/termios.h.3head

```

```

1908 file path=usr/share/man/man3head/tgmath.h.3head
1909 file path=usr/share/man/man3head/time.h.3head
1910 file path=usr/share/man/man3head/timeb.h.3head
1911 file path=usr/share/man/man3head/times.h.3head
1912 file path=usr/share/man/man3head/types.h.3head
1913 file path=usr/share/man/man3head/types32.h.3head
1914 file path=usr/share/man/man3head/ucontext.h.3head
1915 file path=usr/share/man/man3head/uio.h.3head
1916 file path=usr/share/man/man3head/ulimit.h.3head
1917 file path=usr/share/man/man3head/un.h.3head
1918 file path=usr/share/man/man3head/unistd.h.3head
1919 file path=usr/share/man/man3head/utime.h.3head
1920 file path=usr/share/man/man3head/utmpx.h.3head
1921 file path=usr/share/man/man3head/utsname.h.3head
1922 file path=usr/share/man/man3head/values.h.3head
1923 file path=usr/share/man/man3head/wait.h.3head
1924 file path=usr/share/man/man3head/wchar.h.3head
1925 file path=usr/share/man/man3head/wctype.h.3head
1926 file path=usr/share/man/man3head/wordexp.h.3head
1927 file path=usr/share/man/man3head/xlocale.h.3head
1928 file path=usr/share/man/man4/note.4
1929 file path=usr/share/man/man5/prof.5
1930 file path=usr/share/man/man7i/cdio.7i
1931 file path=usr/share/man/man7i/dkio.7i
1932 file path=usr/share/man/man7i/fbio.7i
1933 file path=usr/share/man/man7i/fdio.7i
1934 file path=usr/share/man/man7i/hdio.7i
1935 file path=usr/share/man/man7i/iecd1883.7i
1936 file path=usr/share/man/man7i/mhd.7i
1937 file path=usr/share/man/man7i/mtio.7i
1938 file path=usr/share/man/man7i/prnio.7i
1939 file path=usr/share/man/man7i/quotactl.7i
1940 file path=usr/share/man/man7i/sesio.7i
1941 file path=usr/share/man/man7i/sockio.7i
1942 file path=usr/share/man/man7i/streamio.7i
1943 file path=usr/share/man/man7i/termio.7i
1944 file path=usr/share/man/man7i/termiox.7i
1945 file path=usr/share/man/man7i/uscsi.7i
1946 file path=usr/share/man/man7i/visual_io.7i
1947 file path=usr/share/man/man7i/vt.7i
1948 file path=usr/xpg4/include/curses.h
1949 file path=usr/xpg4/include/term.h
1950 file path=usr/xpg4/include/unctrl.h
1951 legacy pkg=SUNWhea \
1952 desc="SunOS C/C++ header files for general development of software" \
1953 name="SunOS Header Files"
1954 license cr_Sun license=cr_Sun
1955 license lic_CDDL license=lic_CDDL
1956 license license_in_headers license=license_in_headers
1957 license usr/src/lib/pkcs11/include/THIRDPARTYLICENSE \
1958 license=usr/src/lib/pkcs11/include/THIRDPARTYLICENSE
1959 license usr/src/uts/common/sys/THIRDPARTYLICENSE.firmload \
1960 license=usr/src/uts/common/sys/THIRDPARTYLICENSE.firmload
1961 link path=usr/include/iso/assert_iso.h target=../assert.h
1962 link path=usr/include/iso/errno_iso.h target=../errno.h
1963 link path=usr/include/iso/float_iso.h target=../float.h
1964 link path=usr/include/iso/iso646_iso.h target=../iso646.h
1965 $(sparc_ONLY)link path=usr/platform/SUNW,A70/include target=../sun4u/include
1966 $(sparc_ONLY)link path=usr/platform/SUNW,Netra-T12/include \
1967 target=../sun4u/include
1968 $(sparc_ONLY)link path=usr/platform/SUNW,Netra-T4/include \
1969 target=../sun4u/include
1970 $(sparc_ONLY)link path=usr/platform/SUNW,SPARC-Enterprise/include \
1971 target=../sun4u/include
1972 $(sparc_ONLY)link path=usr/platform/SUNW,Serverblad1/include \
1973 target=../sun4u/include

```



```

1974 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Blade-100/include \
1975     target=../sun4u/include
1976 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Blade-1000/include \
1977     target=../sun4u/include
1978 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Blade-1500/include \
1979     target=../sun4u/include
1980 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Blade-2500/include \
1981     target=../sun4u/include
1982 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Fire-15000/include \
1983     target=../sun4u/include
1984 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Fire-280R/include \
1985     target=../sun4u/include
1986 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Fire-480R/include \
1987     target=../sun4u/include
1988 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Fire-880/include \
1989     target=../sun4u/include
1990 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Fire-V215/include \
1991     target=../sun4u/include
1992 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Fire-V240/include \
1993     target=../sun4u/include
1994 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Fire-V250/include \
1995     target=../sun4u/include
1996 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Fire-V440/include \
1997     target=../sun4u/include
1998 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Fire-V445/include \
1999     target=../sun4u/include
2000 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Fire-V490/include \
2001     target=../sun4u/include
2002 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Fire-V890/include \
2003     target=../sun4u/include
2004 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Fire/include \
2005     target=../sun4u/include
2006 $(sparc_ONLY)link path=usr/platform/SUNW,Ultra-2/include \
2007     target=../sun4u/include
2008 $(sparc_ONLY)link path=usr/platform/SUNW,Ultra-250/include \
2009     target=../sun4u/include
2010 $(sparc_ONLY)link path=usr/platform/SUNW,Ultra-4/include \
2011     target=../sun4u/include
2012 $(sparc_ONLY)link path=usr/platform/SUNW,Ultra-Enterprise/include \
2013     target=../sun4u/include
2014 $(sparc_ONLY)link path=usr/platform/SUNW,UltraSPARC-IIe-NetraCT-40/include \
2015     target=../sun4u/include
2016 $(sparc_ONLY)link path=usr/platform/SUNW,UltraSPARC-IIe-NetraCT-60/include \
2017     target=../sun4u/include
2018 $(sparc_ONLY)link path=usr/platform/SUNW,UltraSPARC-IIi-Netract/include \
2019     target=../sun4u/include
2020 link path=usr/share/man/man3head/LIST_CLASS_ENTRY.3head target=queue.h.3head
2021 link path=usr/share/man/man3head/LIST_CLASS_HEAD.3head target=queue.h.3head
2022 link path=usr/share/man/man3head/LIST_CONCAT.3head target=queue.h.3head
2023 link path=usr/share/man/man3head/LIST_EMPTY.3head target=queue.h.3head
2024 link path=usr/share/man/man3head/LIST_ENTRY.3head target=queue.h.3head
2025 link path=usr/share/man/man3head/LIST_FIRST.3head target=queue.h.3head
2026 link path=usr/share/man/man3head/LIST_FOREACH.3head target=queue.h.3head
2027 link path=usr/share/man/man3head/LIST_FOREACH_FROM.3head target=queue.h.3head
2028 link path=usr/share/man/man3head/LIST_FOREACH_FROM_SAFE.3head \
2029     target=queue.h.3head
2030 link path=usr/share/man/man3head/LIST_FOREACH_SAFE.3head target=queue.h.3head
2031 link path=usr/share/man/man3head/LIST_HEAD.3head target=queue.h.3head
2032 link path=usr/share/man/man3head/LIST_HEAD_INITIALIZER.3head \
2033     target=queue.h.3head
2034 link path=usr/share/man/man3head/LIST_INIT.3head target=queue.h.3head
2035 link path=usr/share/man/man3head/LIST_INSERT_AFTER.3head target=queue.h.3head
2036 link path=usr/share/man/man3head/LIST_INSERT_BEFORE.3head target=queue.h.3head
2037 link path=usr/share/man/man3head/LIST_INSERT_HEAD.3head target=queue.h.3head
2038 link path=usr/share/man/man3head/LIST_NEXT.3head target=queue.h.3head
2039 link path=usr/share/man/man3head/LIST_PREV.3head target=queue.h.3head

```

```

2040 link path=usr/share/man/man3head/LIST_REMOVE.3head target=queue.h.3head
2041 link path=usr/share/man/man3head/LIST_SWAP.3head target=queue.h.3head
2042 link path=usr/share/man/man3head/SLIST_CLASS_ENTRY.3head target=queue.h.3head
2043 link path=usr/share/man/man3head/SLIST_CLASS_HEAD.3head target=queue.h.3head
2044 link path=usr/share/man/man3head/SLIST_CONCAT.3head target=queue.h.3head
2045 link path=usr/share/man/man3head/SLIST_EMPTY.3head target=queue.h.3head
2046 link path=usr/share/man/man3head/SLIST_ENTRY.3head target=queue.h.3head
2047 link path=usr/share/man/man3head/SLIST_FIRST.3head target=queue.h.3head
2048 link path=usr/share/man/man3head/SLIST_FOREACH.3head target=queue.h.3head
2049 link path=usr/share/man/man3head/SLIST_FOREACH_FROM.3head target=queue.h.3head
2050 link path=usr/share/man/man3head/SLIST_FOREACH_FROM_SAFE.3head \
2051     target=queue.h.3head
2052 link path=usr/share/man/man3head/SLIST_FOREACH_SAFE.3head target=queue.h.3head
2053 link path=usr/share/man/man3head/SLIST_HEAD.3head target=queue.h.3head
2054 link path=usr/share/man/man3head/SLIST_HEAD_INITIALIZER.3head \
2055     target=queue.h.3head
2056 link path=usr/share/man/man3head/SLIST_INIT.3head target=queue.h.3head
2057 link path=usr/share/man/man3head/SLIST_INSERT_AFTER.3head target=queue.h.3head
2058 link path=usr/share/man/man3head/SLIST_INSERT_HEAD.3head target=queue.h.3head
2059 link path=usr/share/man/man3head/SLIST_NEXT.3head target=queue.h.3head
2060 link path=usr/share/man/man3head/SLIST_REMOVE.3head target=queue.h.3head
2061 link path=usr/share/man/man3head/SLIST_REMOVE_AFTER.3head target=queue.h.3head
2062 link path=usr/share/man/man3head/SLIST_REMOVE_HEAD.3head target=queue.h.3head
2063 link path=usr/share/man/man3head/SLIST_SWAP.3head target=queue.h.3head
2064 link path=usr/share/man/man3head/STAILQ_CLASS_ENTRY.3head target=queue.h.3head
2065 link path=usr/share/man/man3head/STAILQ_CLASS_HEAD.3head target=queue.h.3head
2066 link path=usr/share/man/man3head/STAILQ_CONCAT.3head target=queue.h.3head
2067 link path=usr/share/man/man3head/STAILQ_EMPTY.3head target=queue.h.3head
2068 link path=usr/share/man/man3head/STAILQ_ENTRY.3head target=queue.h.3head
2069 link path=usr/share/man/man3head/STAILQ_FIRST.3head target=queue.h.3head
2070 link path=usr/share/man/man3head/STAILQ_FOREACH.3head target=queue.h.3head
2071 link path=usr/share/man/man3head/STAILQ_FOREACH_FROM.3head \
2072     target=queue.h.3head
2073 link path=usr/share/man/man3head/STAILQ_FOREACH_FROM_SAFE.3head \
2074     target=queue.h.3head
2075 link path=usr/share/man/man3head/STAILQ_FOREACH_SAFE.3head \
2076     target=queue.h.3head
2077 link path=usr/share/man/man3head/STAILQ_HEAD.3head target=queue.h.3head
2078 link path=usr/share/man/man3head/STAILQ_HEAD_INITIALIZER.3head \
2079     target=queue.h.3head
2080 link path=usr/share/man/man3head/STAILQ_INIT.3head target=queue.h.3head
2081 link path=usr/share/man/man3head/STAILQ_INSERT_AFTER.3head \
2082     target=queue.h.3head
2083 link path=usr/share/man/man3head/STAILQ_INSERT_HEAD.3head target=queue.h.3head
2084 link path=usr/share/man/man3head/STAILQ_INSERT_TAIL.3head target=queue.h.3head
2085 link path=usr/share/man/man3head/STAILQ_LAST.3head target=queue.h.3head
2086 link path=usr/share/man/man3head/STAILQ_NEXT.3head target=queue.h.3head
2087 link path=usr/share/man/man3head/STAILQ_REMOVE.3head target=queue.h.3head
2088 link path=usr/share/man/man3head/STAILQ_REMOVE_AFTER.3head \
2089     target=queue.h.3head
2090 link path=usr/share/man/man3head/STAILQ_REMOVE_HEAD.3head target=queue.h.3head
2091 link path=usr/share/man/man3head/STAILQ_SWAP.3head target=queue.h.3head
2092 link path=usr/share/man/man3head/STAILQ_CLASS_ENTRY.3head target=queue.h.3head
2093 link path=usr/share/man/man3head/STAILQ_CLASS_HEAD.3head target=queue.h.3head
2094 link path=usr/share/man/man3head/STAILQ_CONCAT.3head target=queue.h.3head
2095 link path=usr/share/man/man3head/STAILQ_EMPTY.3head target=queue.h.3head
2096 link path=usr/share/man/man3head/STAILQ_ENTRY.3head target=queue.h.3head
2097 link path=usr/share/man/man3head/STAILQ_FIRST.3head target=queue.h.3head
2098 link path=usr/share/man/man3head/STAILQ_FOREACH.3head target=queue.h.3head
2099 link path=usr/share/man/man3head/STAILQ_FOREACH_FROM.3head target=queue.h.3head
2100 link path=usr/share/man/man3head/STAILQ_FOREACH_FROM_SAFE.3head \
2101     target=queue.h.3head
2102 link path=usr/share/man/man3head/STAILQ_FOREACH_REVERSE.3head \
2103     target=queue.h.3head
2104 link path=usr/share/man/man3head/STAILQ_FOREACH_REVERSE_FROM.3head \
2105     target=queue.h.3head

```

```

2106 link path=usr/share/man/man3head/TAILQ_FOREACH_REVERSE_FROM_SAFE.3head \
2107     target=queue.h.3head
2108 link path=usr/share/man/man3head/TAILQ_FOREACH_REVERSE_SAFE.3head \
2109     target=queue.h.3head
2110 link path=usr/share/man/man3head/TAILQ_FOREACH_SAFE.3head target=queue.h.3head
2111 link path=usr/share/man/man3head/TAILQ_HEAD.3head target=queue.h.3head
2112 link path=usr/share/man/man3head/TAILQ_HEAD_INITIALIZER.3head \
2113     target=queue.h.3head
2114 link path=usr/share/man/man3head/TAILQ_INIT.3head target=queue.h.3head
2115 link path=usr/share/man/man3head/TAILQ_INSERT_AFTER.3head target=queue.h.3head
2116 link path=usr/share/man/man3head/TAILQ_INSERT_BEFORE.3head \
2117     target=queue.h.3head
2118 link path=usr/share/man/man3head/TAILQ_INSERT_HEAD.3head target=queue.h.3head
2119 link path=usr/share/man/man3head/TAILQ_INSERT_TAIL.3head target=queue.h.3head
2120 link path=usr/share/man/man3head/TAILQ_LAST.3head target=queue.h.3head
2121 link path=usr/share/man/man3head/TAILQ_NEXT.3head target=queue.h.3head
2122 link path=usr/share/man/man3head/TAILQ_PREV.3head target=queue.h.3head
2123 link path=usr/share/man/man3head/TAILQ_REMOVE.3head target=queue.h.3head
2124 link path=usr/share/man/man3head/TAILQ_SWAP.3head target=queue.h.3head
2125 link path=usr/share/man/man3head/acct.3head target=acct.h.3head
2126 link path=usr/share/man/man3head/aio.3head target=aio.h.3head
2127 link path=usr/share/man/man3head/ar.3head target=ar.h.3head
2128 link path=usr/share/man/man3head/archives.3head target=archives.h.3head
2129 link path=usr/share/man/man3head/assert.3head target=assert.h.3head
2130 link path=usr/share/man/man3head/complex.3head target=complex.h.3head
2131 link path=usr/share/man/man3head/cpio.3head target=cpio.h.3head
2132 link path=usr/share/man/man3head/dirent.3head target=dirent.h.3head
2133 link path=usr/share/man/man3head/errno.3head target=errno.h.3head
2134 link path=usr/share/man/man3head/fcntl.3head target=fcntl.h.3head
2135 link path=usr/share/man/man3head/fenv.3head target=fenv.h.3head
2136 link path=usr/share/man/man3head/float.3head target=float.h.3head
2137 link path=usr/share/man/man3head/floatingpoint.3head \
2138     target=floatingpoint.h.3head
2139 link path=usr/share/man/man3head/fmtmsg.3head target=fmtmsg.h.3head
2140 link path=usr/share/man/man3head/fnmatch.3head target=fnmatch.h.3head
2141 link path=usr/share/man/man3head/ftw.3head target=ftw.h.3head
2142 link path=usr/share/man/man3head/glob.3head target=glob.h.3head
2143 link path=usr/share/man/man3head/grp.3head target=grp.h.3head
2144 link path=usr/share/man/man3head/iconv.3head target=iconv.h.3head
2145 link path=usr/share/man/man3head/if.3head target=if.h.3head
2146 link path=usr/share/man/man3head/in.3head target=in.h.3head
2147 link path=usr/share/man/man3head/inet.3head target=inet.h.3head
2148 link path=usr/share/man/man3head/inttypes.3head target=inttypes.h.3head
2149 link path=usr/share/man/man3head/ipc.3head target=ipc.h.3head
2150 link path=usr/share/man/man3head/iso646.3head target=iso646.h.3head
2151 link path=usr/share/man/man3head/langinfo.3head target=langinfo.h.3head
2152 link path=usr/share/man/man3head/libgen.3head target=libgen.h.3head
2153 link path=usr/share/man/man3head/libintl.3head target=libintl.h.3head
2154 link path=usr/share/man/man3head/limits.3head target=limits.h.3head
2155 link path=usr/share/man/man3head/locale.3head target=locale.h.3head
2156 link path=usr/share/man/man3head/math.3head target=math.h.3head
2157 link path=usr/share/man/man3head/mman.3head target=mman.h.3head
2158 link path=usr/share/man/man3head/monetary.3head target=monetary.h.3head
2159 link path=usr/share/man/man3head/mqueue.3head target=mqueue.h.3head
2160 link path=usr/share/man/man3head/msg.3head target=msg.h.3head
2161 link path=usr/share/man/man3head/ndbm.3head target=ndbm.h.3head
2162 link path=usr/share/man/man3head/netdb.3head target=netdb.h.3head
2163 link path=usr/share/man/man3head/nl_types.3head target=nl_types.h.3head
2164 link path=usr/share/man/man3head/poll.3head target=poll.h.3head
2165 link path=usr/share/man/man3head/pthread.3head target=pthread.h.3head
2166 link path=usr/share/man/man3head/pwd.3head target=pwd.h.3head
2167 link path=usr/share/man/man3head/regex.3head target=regex.h.3head
2168 link path=usr/share/man/man3head/resource.3head target=resource.h.3head
2169 link path=usr/share/man/man3head/sched.3head target=sched.h.3head
2170 link path=usr/share/man/man3head/search.3head target=search.h.3head
2171 link path=usr/share/man/man3head/select.3head target=select.h.3head

```

```

2172 link path=usr/share/man/man3head/sem.3head target=sem.h.3head
2173 link path=usr/share/man/man3head/semaphore.3head target=semaphore.h.3head
2174 link path=usr/share/man/man3head/setjmp.3head target=setjmp.h.3head
2175 link path=usr/share/man/man3head/shm.3head target=shm.h.3head
2176 link path=usr/share/man/man3head/signinfo.3head target=signinfo.h.3head
2177 link path=usr/share/man/man3head/signal.3head target=signal.h.3head
2178 link path=usr/share/man/man3head/socket.3head target=socket.h.3head
2179 link path=usr/share/man/man3head/spawn.3head target=spawn.h.3head
2180 link path=usr/share/man/man3head/stat.3head target=stat.h.3head
2181 link path=usr/share/man/man3head/statvfs.3head target=statvfs.h.3head
2182 link path=usr/share/man/man3head/stdbool.3head target=stdbool.h.3head
2183 link path=usr/share/man/man3head/stddef.3head target=stddef.h.3head
2184 link path=usr/share/man/man3head/stdint.3head target=stdint.h.3head
2185 link path=usr/share/man/man3head/stdio.3head target=stdio.h.3head
2186 link path=usr/share/man/man3head/stdlib.3head target=stdlib.h.3head
2187 link path=usr/share/man/man3head/string.3head target=string.h.3head
2188 link path=usr/share/man/man3head/strings.3head target=strings.h.3head
2189 link path=usr/share/man/man3head/stropts.3head target=stropts.h.3head
2190 link path=usr/share/man/man3head/syslog.3head target=syslog.h.3head
2191 link path=usr/share/man/man3head/tar.3head target=tar.h.3head
2192 link path=usr/share/man/man3head/tcp.3head target=cp.h.3head
2193 link path=usr/share/man/man3head/termios.3head target=termios.h.3head
2194 link path=usr/share/man/man3head/tgmath.3head target=tgmath.h.3head
2195 link path=usr/share/man/man3head/time.3head target=time.h.3head
2196 link path=usr/share/man/man3head/timeb.3head target=timeb.h.3head
2197 link path=usr/share/man/man3head/times.3head target=times.h.3head
2198 link path=usr/share/man/man3head/types.3head target=types.h.3head
2199 link path=usr/share/man/man3head/types32.3head target=types32.h.3head
2200 link path=usr/share/man/man3head/ucontext.3head target=ucontext.h.3head
2201 link path=usr/share/man/man3head/uio.3head target=uio.h.3head
2202 link path=usr/share/man/man3head/ulimit.3head target=ulimit.h.3head
2203 link path=usr/share/man/man3head/un.3head target=un.h.3head
2204 link path=usr/share/man/man3head/unistd.3head target=unistd.h.3head
2205 link path=usr/share/man/man3head/utime.3head target=utime.h.3head
2206 link path=usr/share/man/man3head/utmpx.3head target=utmpx.h.3head
2207 link path=usr/share/man/man3head/utsname.3head target=utsname.h.3head
2208 link path=usr/share/man/man3head/values.3head target=values.h.3head
2209 link path=usr/share/man/man3head/wait.3head target=wait.h.3head
2210 link path=usr/share/man/man3head/wchar.3head target=wchar.h.3head
2211 link path=usr/share/man/man3head/wctype.3head target=wctype.h.3head
2212 link path=usr/share/man/man3head/wordexp.3head target=wordexp.h.3head
2213 link path=usr/share/man/man3head/xlocale.3head target=xlocale.h.3head
2214 $(i386_ONLY)link path=usr/share/src/uts/i86pc/sys \
2215     target=../../../../platform/i86pc/include/sys
2216 $(i386_ONLY)link path=usr/share/src/uts/i86pc/vm \
2217     target=../../../../platform/i86pc/include/vm
2218 $(i386_ONLY)link path=usr/share/src/uts/i86pc/vm \
2219     target=../../../../platform/i86pc/vm/include/sys
2220 $(i386_ONLY)link path=usr/share/src/uts/i86pc/vm \
2221     target=../../../../platform/i86pc/vm/include/vm
2222 $(sparc_ONLY)link path=usr/share/src/uts/sun4u/sys \
2223     target=../../../../platform/sun4u/include/sys
2224 $(sparc_ONLY)link path=usr/share/src/uts/sun4u/vm \
2225     target=../../../../platform/sun4u/include/vm
2226 $(sparc_ONLY)link path=usr/share/src/uts/sun4v/sys \
2227     target=../../../../platform/sun4v/include/sys
2228 $(sparc_ONLY)link path=usr/share/src/uts/sun4v/vm \
2229     target=../../../../platform/sun4v/include/vm

```

```

*****
30332 Tue May 7 04:49:55 2019
new/usr/src/uts/common/disp/cpupart.c
OS-7125 Need mitigation of L1TF (CVE-2018-3646)
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
*****
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) 1996, 2010, Oracle and/or its affiliates. All rights reserved.
23 * Copyright 2018 Joyent, Inc.
24 * Copyright (c) 2017 by Delphix. All rights reserved.
25 */

26 #include <sys/types.h>
27 #include <sys/system.h>
28 #include <sys/cmn_err.h>
29 #include <sys/cpuvar.h>
30 #include <sys/thread.h>
31 #include <sys/disp.h>
32 #include <sys/kmem.h>
33 #include <sys/debug.h>
34 #include <sys/cpupart.h>
35 #include <sys/pset.h>
36 #include <sys/var.h>
37 #include <sys/cyclic.h>
38 #include <sys/lgrp.h>
39 #include <sys/pghw.h>
40 #include <sys/loadavg.h>
41 #include <sys/class.h>
42 #include <sys/fss.h>
43 #include <sys/pool.h>
44 #include <sys/pool_pset.h>
45 #include <sys/policy.h>

47 /*
48 * Calling pool_lock() protects the pools configuration, which includes
49 * CPU partitions. cpu_lock protects the CPU partition list, and prevents
50 * partitions from being created or destroyed while the lock is held.
51 * The lock ordering with respect to related locks is:
52 *
53 *   pool_lock() ---> cpu_lock ---> pidlock --> p_lock
54 *
55 * Blocking memory allocations may be made while holding "pool_lock"
56 * or cpu_lock.
57 */

```

```

59 /*
60 * The cp_default partition is allocated statically, but its lgroup load average
61 * (lpl) list is allocated dynamically after kmem subsystem is initialized. This
62 * saves some memory since the space allocated reflects the actual number of
63 * lgroups supported by the platform. The lgrp facility provides a temporary
64 * space to hold lpl information during system bootstrap.
65 */

67 cpupart_t          *cp_list_head;
68 cpupart_t          cp_default;
69 static cpupartid_t cp_id_next;
70 uint_t             cp_numparts;
71 uint_t             cp_numparts_nonempty;

73 /*
74 * Need to limit total number of partitions to avoid slowing down the
75 * clock code too much. The clock code traverses the list of
76 * partitions and needs to be able to execute in a reasonable amount
77 * of time (less than 1/hz seconds). The maximum is sized based on
78 * max_ncpus so it shouldn't be a problem unless there are large
79 * numbers of empty partitions.
80 */
81 static uint_t      cp_max_numparts;

83 /*
84 * Processor sets and CPU partitions are different but related concepts.
85 * A processor set is a user-level abstraction allowing users to create
86 * sets of CPUs and bind threads exclusively to those sets. A CPU
87 * partition is a kernel dispatcher object consisting of a set of CPUs
88 * and a global dispatch queue. The processor set abstraction is
89 * implemented via a CPU partition, and currently there is a 1-1
90 * mapping between processor sets and partitions (excluding the default
91 * partition, which is not visible as a processor set). Hence, the
92 * numbering for processor sets and CPU partitions is identical. This
93 * may not always be true in the future, and these macros could become
94 * less trivial if we support e.g. a processor set containing multiple
95 * CPU partitions.
96 */
97 #define PSTOCP(psid) ((cpupartid_t)((psid) == PS_NONE ? CP_DEFAULT : (psid)))
98 #define CPTOPS(cpid) ((psetid_t)((cpid) == CP_DEFAULT ? PS_NONE : (cpid)))

100 static int cpupart_unbind_threads(cpupart_t *, boolean_t);

102 /*
103 * Find a CPU partition given a processor set ID.
104 */
105 static cpupart_t *
106 cpupart_find_all(psetid_t psid)
107 {
108     cpupart_t *cp;
109     cpupartid_t cpid = PSTOCP(psid);

111     ASSERT(MUTEX_HELD(&cpu_lock));

113     /* default partition not visible as a processor set */
114     if (psid == CP_DEFAULT)
115         return (NULL);

117     if (psid == PS_MYID)
118         return (curthread->t_cpupart);

120     cp = cp_list_head;
121     do {
122         if (cp->cp_id == cpid)
123             return (cp);
124         cp = cp->cp_next;

```

```

125     } while (cp != cp_list_head);
126     return (NULL);
127 }
_____unchanged_portion_omitted_____

320 static int
321 cpupart_move_cpu(cpu_t *cp, cpupart_t *newpp, int forced)
322 {
323     cpupart_t *oldpp;
324     cpu_t *ncp, *newlist;
325     kthread_t *t;
326     int move_threads = 1;
327     lgrp_id_t lgrp_id;
328     proc_t *p;
329     int lgrp_diff_lpl;
330     lpl_t *cpu_lpl;
331     int ret;
332     boolean_t unbind_all_threads = (forced != 0);

334     ASSERT(MUTEX_HELD(&cpu_lock));
335     ASSERT(newpp != NULL);

337     oldpp = cp->cpu_part;
338     ASSERT(oldpp != NULL);
339     ASSERT(oldpp->cp_ncpus > 0);

341     if (newpp == oldpp) {
342         /*
343          * Don't need to do anything.
344          */
345         return (0);
346     }

348     cpu_state_change_notify(cp->cpu_id, CPU_CPUPART_OUT);

350     if (!disp_bound_partition(cp, 0)) {
351         /*
352          * Don't need to move threads if there are no threads in
353          * the partition. Note that threads can't enter the
354          * partition while we're holding cpu_lock.
355          */
356         move_threads = 0;
357     } else if (oldpp->cp_ncpus == 1) {
358         /*
359          * The last CPU is removed from a partition which has threads
360          * running in it. Some of these threads may be bound to this
361          * CPU.
362          *
363          * Attempt to unbind threads from the CPU and from the processor
364          * set. Note that no threads should be bound to this CPU since
365          * cpupart_move_threads will refuse to move bound threads to
366          * other CPUs.
367          */
368         (void) cpu_unbind(oldpp->cp_cpulist->cpu_id, B_FALSE);
369         (void) cpupart_unbind_threads(oldpp, B_FALSE);

371         if (!disp_bound_partition(cp, 0)) {
372             /*
373              * No bound threads in this partition any more
374              */
375             move_threads = 0;
376         } else {
377             /*
378              * There are still threads bound to the partition
379              */

```

```

380         cpu_state_change_notify(cp->cpu_id, CPU_CPUPART_IN);
381         return (EBUSY);
382     }
383 }

385 /*
386  * If forced flag is set unbind any threads from this CPU.
387  * Otherwise unbind soft-bound threads only.
388  */
389 if ((ret = cpu_unbind(cp->cpu_id, unbind_all_threads)) != 0) {
390     cpu_state_change_notify(cp->cpu_id, CPU_CPUPART_IN);
391     return (ret);
392 }

394 /*
395  * Stop further threads weak binding to this cpu.
396  */
397 cpu_inmotion = cp;
398 membar_enter();

400 /*
401  * Notify the Processor Groups subsystem that the CPU
402  * will be moving cpu partitions. This is done before any
403  * CPUs are paused to provide an opportunity for any
404  * needed memory allocations.
405  */
406 pg_cpupart_out(cp, oldpp);
407 pg_cpupart_in(cp, newpp);

409 again:
410 if (move_threads) {
411     int loop_count;
412     /*
413      * Check for threads strong or weak bound to this CPU.
414      */
415     for (loop_count = 0; disp_bound_threads(cp, 0); loop_count++) {
416         if (loop_count >= 5) {
417             cpu_state_change_notify(cp->cpu_id,
418                 CPU_CPUPART_IN);
419             pg_cpupart_out(cp, newpp);
420             pg_cpupart_in(cp, oldpp);
421             cpu_inmotion = NULL;
422             return (EBUSY); /* some threads still bound */
423         }
424         delay(1);
425     }
426 }

428 /*
429  * Before we actually start changing data structures, notify
430  * the cyclic subsystem that we want to move this CPU out of its
431  * partition.
432  */
433 if (!cyclic_move_out(cp)) {
434     /*
435      * This CPU must be the last CPU in a processor set with
436      * a bound cyclic.
437      */
438     cpu_state_change_notify(cp->cpu_id, CPU_CPUPART_IN);
439     pg_cpupart_out(cp, newpp);
440     pg_cpupart_in(cp, oldpp);
441     cpu_inmotion = NULL;
442     return (EBUSY);
443 }

445 pause_cpus(cp, NULL);

```

```

447     if (move_threads) {
448         /*
449          * The thread on cpu before the pause thread may have read
450          * cpu_inmotion before we raised the barrier above. Check
451          * again.
452          */
453         if (disp_bound_threads(cp, 1)) {
454             start_cpus();
455             goto again;
456         }
457     }
458 }
459
460 /*
461  * Now that CPUs are paused, let the PG subsystem perform
462  * any necessary data structure updates.
463  */
464 pg_cpupart_move(cp, oldpp, newpp);
465
466 /* save this cpu's lgroup -- it'll be the same in the new partition */
467 lgrp_id = cp->cpu_lpl->lpl_lgrp_id;
468
469 cpu_lpl = cp->cpu_lpl;
470 /*
471  * let the lgroup framework know cp has left the partition
472  */
473 lgrp_config(LGRP_CONFIG_CPUPART_DEL, (uintptr_t)cp, lgrp_id);
474
475 /* move out of old partition */
476 oldpp->cp_ncpus--;
477 if (oldpp->cp_ncpus > 0) {
478     ncp = cp->cpu_prev_part->cpu_next_part = cp->cpu_next_part;
479     cp->cpu_next_part->cpu_prev_part = cp->cpu_prev_part;
480     if (oldpp->cp_cpulist == cp) {
481         oldpp->cp_cpulist = ncp;
482     }
483 } else {
484     ncp = oldpp->cp_cpulist = NULL;
485     cp_numparts_nonempty--;
486     ASSERT(cp_numparts_nonempty != 0);
487 }
488 oldpp->cp_gen++;
489
490 /* move into new partition */
491 newlist = newpp->cp_cpulist;
492 if (newlist == NULL) {
493     newpp->cp_cpulist = cp->cpu_next_part = cp->cpu_prev_part = cp;
494     cp_numparts_nonempty++;
495     ASSERT(cp_numparts_nonempty != 0);
496 } else {
497     cp->cpu_next_part = newlist;
498     cp->cpu_prev_part = newlist->cpu_prev_part;
499     newlist->cpu_prev_part->cpu_next_part = cp;
500     newlist->cpu_prev_part = cp;
501 }
502 cp->cpu_part = newpp;
503 newpp->cp_ncpus++;
504 newpp->cp_gen++;
505
506 ASSERT(bitset_is_null(&newpp->cp_haltset));
507 ASSERT(bitset_is_null(&oldpp->cp_haltset));
508
509 /*
510  * let the lgroup framework know cp has entered the partition

```

```

512     */
513     lgrp_config(LGRP_CONFIG_CPUPART_ADD, (uintptr_t)cp, lgrp_id);
514
515     /*
516     * If necessary, move threads off processor.
517     */
518     if (move_threads) {
519         ASSERT(ncp != NULL);
520
521         /*
522          * Walk thru the active process list to look for
523          * threads that need to have a new home lgroup,
524          * or the last CPU they run on is the same CPU
525          * being moved out of the partition.
526          */
527
528         for (p = practive; p != NULL; p = p->p_next) {
529             t = p->p_tlist;
530
531             if (t == NULL)
532                 continue;
533
534             lgrp_diff_lpl = 0;
535
536             do {
537                 ASSERT(t->t_lpl != NULL);
538
539                 /*
540                  * Update the count of how many threads are
541                  * in this CPU's lgroup but have a different lpl
542                  */
543
544                 if (t->t_lpl != cpu_lpl &&
545                     t->t_lpl->lpl_lgrp_id == lgrp_id)
546                     lgrp_diff_lpl++;
547
548                 /*
549                  * If the lgroup that t is assigned to no
550                  * longer has any CPUs in t's partition,
551                  * we'll have to choose a new lgroup for t.
552                  */
553
554                 if (!LGRP_CPUS_IN_PART(t->t_lpl->lpl_lgrp_id,
555                     t->t_cpupart)) {
556                     lgrp_move_thread(t,
557                         lgrp_choose(t, t->t_cpupart), 0);
558                 }
559
560                 /*
561                  * make sure lpl points to our own partition
562                  */
563                 ASSERT(t->t_lpl >= t->t_cpupart->cp_lgrp_loads &&
564                     (t->t_lpl < t->t_cpupart->cp_lgrp_loads +
565                     t->t_cpupart->cp_nlgrp_loads));
566
567                 ASSERT(t->t_lpl->lpl_ncpu > 0);
568
569                 /* Update CPU last ran on if it was this CPU */
570                 if (t->t_cpu == cp && t->t_cpupart == oldpp &&
571                     t->t_bound_cpu != cp) {
572                     t->t_cpu = disp_lowpri_cpu(ncp, t,
573                         t->t_pri);
574                     t->t_cpu = disp_lowpri_cpu(ncp,
575                         t->t_lpl, t->t_pri, NULL);
576                 }

```

```

576         t = t->t_forw;
577     } while (t != p->p_tlist);

579     /*
580     * Didn't find any threads in the same lgroup as this
581     * CPU with a different lpl, so remove the lgroup from
582     * the process lgroup bitmask.
583     */

585     if (lgrp_diff_lpl)
586         klggrpset_del(p->p_lgrpset, lgrpid);
587 }

589 /*
590 * Walk thread list looking for threads that need to be
591 * rehomed, since there are some threads that are not in
592 * their process's p_tlist.
593 */

595 t = curthread;

597 do {
598     ASSERT(t != NULL && t->t_lpl != NULL);

600     /*
601     * If the lgroup that t is assigned to no
602     * longer has any CPUs in t's partition,
603     * we'll have to choose a new lgroup for t.
604     * Also, choose best lgroup for home when
605     * thread has specified lgroup affinities,
606     * since there may be an lgroup with more
607     * affinity available after moving CPUs
608     * around.
609     */
610     if (!LGRP_CPUS_IN_PART(t->t_lpl->lpl_lgrpid,
611         t->t_cpupart) || t->t_lgrp_affinity) {
612         lgrp_move_thread(t,
613             lgrp_choose(t, t->t_cpupart), 1);
614     }

616     /* make sure lpl points to our own partition */
617     ASSERT((t->t_lpl >= t->t_cpupart->cp_lgrploads) &&
618         (t->t_lpl < t->t_cpupart->cp_lgrploads +
619             t->t_cpupart->cp_nlgrploads));

621     ASSERT(t->t_lpl->lpl_ncpu > 0);

623     /* Update CPU last ran on if it was this CPU */
624     if (t->t_cpu == cp && t->t_cpupart == oldpp &&
625         t->t_bound_cpu != cp) {
626         t->t_cpu = disp_lowpri_cpu(ncp, t,
627             t->t_pri);
626         t->t_cpu = disp_lowpri_cpu(ncp, t->t_lpl,
627             t->t_pri, NULL);
628     }

630     t = t->t_next;
631 } while (t != curthread);

633 /*
634 * Clear off the CPU's run queue, and the kp queue if the
635 * partition is now empty.
636 */
637 disp_cpu_inactive(cp);

639 /*

```

```

640         * Make cp switch to a thread from the new partition.
641         */
642         cp->cpu_runrun = 1;
643         cp->cpu_kprunrun = 1;
644     }

646     cpu_inmotion = NULL;
647     start_cpus();

649     /*
650     * Let anyone interested know that cpu has been added to the set.
651     */
652     cpu_state_change_notify(cp->cpu_id, CPU_CPUPART_IN);

654     /*
655     * Now let the cyclic subsystem know that it can reshuffle cyclics
656     * bound to the new processor set.
657     */
658     cyclic_move_in(cp);

660     return (0);
661 }

```

unchanged portion omitted

```

*****
70269 Tue May 7 04:49:56 2019
new/usr/src/uts/common/disp/disp.c
OS-7125 Need mitigation of L1TF (CVE-2018-3646)
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
*****
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 2009 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */
25
26 /*
27 * Copyright (c) 2018, Joyent, Inc. All rights reserved.
28 */
29
30 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
31 /*      All Rights Reserved      */
32
33 #include <sys/types.h>
34 #include <sys/param.h>
35 #include <sys/symmacros.h>
36 #include <sys/signal.h>
37 #include <sys/user.h>
38 #include <sys/sysm.h>
39 #include <sys/sysinfo.h>
40 #include <sys/var.h>
41 #include <sys/errno.h>
42 #include <sys/cmn_err.h>
43 #include <sys/debug.h>
44 #include <sys/inline.h>
45 #include <sys/disp.h>
46 #include <sys/class.h>
47 #include <sys/bitmap.h>
48 #include <sys/kmem.h>
49 #include <sys/cpuvar.h>
50 #include <sys/vtrace.h>
51 #include <sys/tnf.h>
52 #include <sys/cpupart.h>
53 #include <sys/lgrp.h>
54 #include <sys/pg.h>
55 #include <sys/cmt.h>
56 #include <sys/bitset.h>
57 #include <sys/schedctl.h>
58 #include <sys/atomic.h>

```

```

60 #include <sys/dtrace.h>
61 #include <sys/sdt.h>
62 #include <sys/archsystem.h>
63 #include <sys/ht.h>
64
65 #include <vm/as.h>
66
67 #define BOUND_CPU      0x1
68 #define BOUND_PARTITION 0x2
69 #define BOUND_INTR     0x4
70
71 /* Dispatch queue allocation structure and functions */
72 struct disp_queue_info {
73     disp_t *dp;
74     dispq_t *olddispq;
75     dispq_t *newdispq;
76     ulong_t *olddgactmap;
77     ulong_t *newdgactmap;
78     int     oldnglobpris;
79 };
80
81 /*
82  * _____
83  * unchanged portion omitted
84  * _____
85  */
86
87 #define CPU_IDLING(pri) ((pri) == -1)
88
89 static void
90 cpu_resched(cpu_t *cp, pri_t tpri)
91 {
92     int     call_poke_cpu = 0;
93     pri_t   cpupri = cp->cpu_dispatch_pri;
94
95     if (cpupri != CPU_IDLE_PRI && cpupri < tpri) {
96         if (!CPU_IDLING(cpupri) && (cpupri < tpri)) {
97             TRACE_2(TR_FAC_DISP, TR_CPU_RESCHED,
98                 "CPU_RESCHED:Tpri %d CpuPri %d", tpri, cpupri);
99             if (tpri >= upreemptpri && cp->cpu_runrun == 0) {
100                 cp->cpu_runrun = 1;
101                 aston(cp->cpu_dispthread);
102                 if (tpri < kpreemptpri && cp != CPU)
103                     call_poke_cpu = 1;
104             }
105             if (tpri >= kpreemptpri && cp->cpu_kprunrun == 0) {
106                 cp->cpu_kprunrun = 1;
107                 if (cp != CPU)
108                     call_poke_cpu = 1;
109             }
110         }
111     }
112
113     /*
114      * Propagate cpu_runrun, and cpu_kprunrun to global visibility.
115      */
116     membar_enter();
117
118     if (call_poke_cpu)
119         poke_cpu(cp->cpu_id);
120 }
121
122 /*
123  * setbackdq() keeps runqs balanced such that the difference in length
124  * between the chosen runq and the next one is no more than RUNQ_MAX_DIFF.
125  * For threads with priorities below RUNQ_MATCH_PRI levels, the runq's lengths
126  * must match. When per-thread TS_RUNQMATCH flag is set, setbackdq() will
127  * try to keep runqs perfectly balanced regardless of the thread priority.
128  */
129 #define RUNQ_MATCH_PRI 16 /* pri below which queue lengths must match */
130 #define RUNQ_MAX_DIFF 2  /* maximum runq length difference */
131 #define RUNQ_LEN(cp, pri) ((cp)->cpu_disp->disp_q[pri].dq_sruncnt)

```

```

1165 /*
1166 * Macro that evaluates to true if it is likely that the thread has cache
1167 * warmth. This is based on the amount of time that has elapsed since the
1168 * thread last ran. If that amount of time is less than "rechoose_interval"
1169 * ticks, then we decide that the thread has enough cache warmth to warrant
1170 * some affinity for t->t_cpu.
1171 */
1172 #define THREAD_HAS_CACHE_WARMTH(thread) \
1173     ((thread == curthread) || \
1174      ((ddi_get_lbolt() - thread->t_disp_time) <= rechoose_interval))
1175 /*
1176 * Put the specified thread on the back of the dispatcher
1177 * queue corresponding to its current priority.
1178 *
1179 * Called with the thread in transition, onproc or stopped state
1180 * and locked (transition implies locked) and at high spl.
1181 * Returns with the thread in TS_RUN state and still locked.
1182 */
1183 void
1184 setbackdq(kthread_t *tp)
1185 {
1186     dispq_t *dq;
1187     disp_t      *dp;
1188     cpu_t      *cp;
1189     pri_t      tpri;
1190     int        bound;
1191     boolean_t   self;

1193     ASSERT(THREAD_LOCK_HELD(tp));
1194     ASSERT((tp->t_schedflag & TS_ALLSTART) == 0);
1195     ASSERT(!thread_on_queue(tp)); /* make sure tp isn't on a runq */

1197     /*
1198     * If thread is "swapped" or on the swap queue don't
1199     * queue it, but wake sched.
1200     */
1201     if ((tp->t_schedflag & (TS_LOAD | TS_ON_SWAPQ)) != TS_LOAD) {
1202         disp_swapped_setrun(tp);
1203         return;
1204     }

1206     self = (tp == curthread);

1208     if (tp->t_bound_cpu || tp->t_weakbound_cpu)
1209         bound = 1;
1210     else
1211         bound = 0;

1213     tpri = DISP_PRIO(tp);
1214     if (ncpus == 1)
1215         cp = tp->t_cpu;
1216     else if (!bound) {
1217         if (tpri >= kqcpri) {
1218             setkpdq(tp, SETKP_BACK);
1219             return;
1220         }

1222     /*
1223     * We'll generally let this thread continue to run where
1224     * it last ran...but will consider migration if:
1225     * - The thread probably doesn't have much cache warmth.
1226     * - HT exclusion would prefer us to run elsewhere
1227     * - We thread probably doesn't have much cache warmth.
1228     * - The CPU where it last ran is the target of an offline
1229     * request.

```

```

1229     * - The thread last ran outside its home lgroup.
1230     * - The thread last ran outside it's home lgroup.
1231     */
1232     if ((!THREAD_HAS_CACHE_WARMTH(tp)) ||
1233         !ht_should_run(tp, tp->t_cpu) ||
1234         !LGRP_CONTAINS_CPU(tp->t_lpl->lpl_lgrp, tp->t_cpu)) {
1235         cp = disp_lowpri_cpu(tp->t_cpu, tp, tpri);
1236         (tp->t_cpu == cpu_inmotion) {
1237             cp = disp_lowpri_cpu(tp->t_cpu, tp->t_lpl, tpri, NULL);
1238         } else if (!LGRP_CONTAINS_CPU(tp->t_lpl->lpl_lgrp, tp->t_cpu)) {
1239             cp = disp_lowpri_cpu(tp->t_cpu, tp->t_lpl, tpri,
1240                                 self ? tp->t_cpu : NULL);
1241         } else {
1242             cp = tp->t_cpu;
1243         }

1244     if (tp->t_cpupart == cp->cpu_part) {
1245         int qlen;

1247         /*
1248         * Perform any CMT load balancing
1249         */
1250         cp = cmt_balance(tp, cp);

1252         /*
1253         * Balance across the run queues
1254         */
1255         qlen = RUNQ_LEN(cp, tpri);
1256         if (tpri >= RUNQ_MATCH_PRI &&
1257             !(tp->t_schedflag & TS_RUNQMATCH))
1258             qlen -= RUNQ_MAX_DIFF;
1259         if (qlen > 0) {
1260             cpu_t *newcp;

1262             if (tp->t_lpl->lpl_lgrp == LGRP_ROOTID) {
1263                 newcp = cp->cpu_next_part;
1264             } else if ((newcp = cp->cpu_next_lpl) == cp) {
1265                 newcp = cp->cpu_next_part;
1266             }

1268             if (ht_should_run(tp, newcp) &&
1269                 RUNQ_LEN(newcp, tpri) < qlen) {
1270                 if (RUNQ_LEN(newcp, tpri) < qlen) {
1271                     DTRACE_PROBE3(runq_balance,
1272                                   kthread_t *, tp,
1273                                   cpu_t *, cp, cpu_t *, newcp);
1274                     cp = newcp;
1275                 }
1276             }
1277         } else {
1278             /*
1279             * Migrate to a cpu in the new partition.
1280             */
1281             cp = disp_lowpri_cpu(tp->t_cpupart->cp_cpulist, tp,
1282                                 tp->t_pri);
1283             cp = disp_lowpri_cpu(tp->t_cpupart->cp_cpulist,
1284                                 tp->t_lpl, tp->t_pri, NULL);
1285         }
1286     }
1287     ASSERT((cp->cpu_flags & CPU_QUIESCED) == 0);
1288 } else {
1289     /*
1290     * It is possible that t_weakbound_cpu != t_bound_cpu (for
1291     * a short time until weak binding that existed when the
1292     * strong binding was established has dropped) so we must
1293     * favour weak binding over strong.

```



```

1286         */
1287         cp = tp->t_weakbound_cpu ?
1288             tp->t_weakbound_cpu : tp->t_bound_cpu;
1289     }
1290     /*
1291     * A thread that is ONPROC may be temporarily placed on the run queue
1292     * but then chosen to run again by disp.  If the thread we're placing on
1293     * the queue is in TS_ONPROC state, don't set its t_waitrq until a
1294     * replacement process is actually scheduled in swtch().  In this
1295     * situation, curthread is the only thread that could be in the ONPROC
1296     * state.
1297     */
1298     if ((!self) && (tp->t_waitrq == 0)) {
1299         hrttime_t curtime;
1300
1301         curtime = gethrtime_unscaled();
1302         (void) cpu_update_pct(tp, curtime);
1303         tp->t_waitrq = curtime;
1304     } else {
1305         (void) cpu_update_pct(tp, gethrtime_unscaled());
1306     }
1307
1308     dp = cp->cpu_disp;
1309     disp_lock_enter_high(&dp->disp_lock);
1310
1311     DTRACE_SCHED3(enqueue, kthread_t *, tp, disp_t *, dp, int, 0);
1312     TRACE_3(TR_FAC_DISP, TR_BACKQ, "setbackdq:pri %d cpu %p tid %p",
1313         tpri, cp, tp);
1314
1315 #ifndef NPROBE
1316     /* Kernel probe */
1317     if (tnf_tracing_active)
1318         tnf_thread_queue(tp, cp, tpri);
1319 #endif /* NPROBE */
1320
1321     ASSERT(tpri >= 0 && tpri < dp->disp_npri);
1322
1323     THREAD_RUN(tp, &dp->disp_lock); /* set t_state to TS_RUN */
1324     tp->t_disp_queue = dp;
1325     tp->t_link = NULL;
1326
1327     dq = &dp->disp_q[tpri];
1328     dp->disp_nrunnable++;
1329     if (!bound)
1330         dp->disp_steal = 0;
1331     membar_enter();
1332
1333     if (dq->dq_sruncnt++ != 0) {
1334         ASSERT(dq->dq_first != NULL);
1335         dq->dq_last->t_link = tp;
1336         dq->dq_last = tp;
1337     } else {
1338         ASSERT(dq->dq_first == NULL);
1339         ASSERT(dq->dq_last == NULL);
1340         dq->dq_first = dq->dq_last = tp;
1341         BT_SET(dp->disp_qactmap, tpri);
1342         if (tpri > dp->disp_maxrunpri) {
1343             dp->disp_maxrunpri = tpri;
1344             membar_enter();
1345             cpu_resched(cp, tpri);
1346         }
1347     }
1348
1349     if (!bound && tpri > dp->disp_max_unbound_pri) {
1350         if (self && dp->disp_max_unbound_pri == -1 && cp == CPU) {
1351             /*

```

```

1352         * If there are no other unbound threads on the
1353         * run queue, don't allow other CPUs to steal
1354         * this thread while we are in the middle of a
1355         * context switch. We may just switch to it
1356         * again right away. CPU_DISP_DONTSTEAL is cleared
1357         * in swtch and swtch_to.
1358         */
1359         cp->cpu_disp_flags |= CPU_DISP_DONTSTEAL;
1360     }
1361     dp->disp_max_unbound_pri = tpri;
1362 }
1363 (*disp_enq_thread)(cp, bound);
1364 }
1365
1366 /*
1367 * Put the specified thread on the front of the dispatcher
1368 * queue corresponding to its current priority.
1369 *
1370 * Called with the thread in transition, onproc or stopped state
1371 * and locked (transition implies locked) and at high spl.
1372 * Returns with the thread in TS_RUN state and still locked.
1373 */
1374 void
1375 setfrontdq(kthread_t *tp)
1376 {
1377     disp_t      *dp;
1378     dispq_t     *dq;
1379     cpu_t       *cp;
1380     pri_t       tpri;
1381     int         bound;
1382
1383     ASSERT(THREAD_LOCK_HELD(tp));
1384     ASSERT((tp->t_schedflag & TS_ALLSTART) == 0);
1385     ASSERT(!thread_on_queue(tp)); /* make sure tp isn't on a runq */
1386
1387     /*
1388     * If thread is "swapped" or on the swap queue don't
1389     * queue it, but wake sched.
1390     */
1391     if ((tp->t_schedflag & (TS_LOAD | TS_ON_SWAPQ)) != TS_LOAD) {
1392         disp_swapped_setrun(tp);
1393         return;
1394     }
1395
1396     if (tp->t_bound_cpu || tp->t_weakbound_cpu)
1397         bound = 1;
1398     else
1399         bound = 0;
1400
1401     tpri = DISP_PRIO(tp);
1402     if (ncpus == 1)
1403         cp = tp->t_cpu;
1404     else if (!bound) {
1405         if (tpri >= kqppri) {
1406             setkpdq(tp, SETKP_FRONT);
1407             return;
1408         }
1409         cp = tp->t_cpu;
1410         if (tp->t_cpupart == cp->cpu_part) {
1411             /*
1412             * We'll generally let this thread continue to run
1413             * where it last ran, but will consider migration if:
1414             * - The thread last ran outside its home lgroup.
1415             * - The thread last ran outside it's home lgroup.
1416             * - The CPU where it last ran is the target of an
1417             *   offline request (a thread_nomigrate() on the in

```

```

1417         * motion CPU relies on this when forcing a preempt).
1418         * - The thread isn't the highest priority thread where
1419         *   it last ran, and it is considered not likely to
1420         *   have significant cache warmth.
1421         */
1422         if (!LGRP_CONTAINS_CPU(tp->t_lpl->lpl_lgrp, cp) ||
1423             cp == cpu_inmotion ||
1424             (tpri < cp->cpu_disp->disp_maxrunpri &&
1425              !THREAD_HAS_CACHE_WARMTH(tp))) {
1426             cp = disp_lowpri_cpu(tp->t_cpu, tp, tpri);
1427         } else if (!LGRP_CONTAINS_CPU(tp->t_lpl->lpl_lgrp, cp)) ||
1428             (cp == cpu_inmotion) {
1429             cp = disp_lowpri_cpu(tp->t_cpu, tp->t_lpl, tpri,
1430                                 (tp == curthread) ? cp : NULL);
1431         } else if ((tpri < cp->cpu_disp->disp_maxrunpri) &&
1432                    (!THREAD_HAS_CACHE_WARMTH(tp))) {
1433             cp = disp_lowpri_cpu(tp->t_cpu, tp->t_lpl, tpri,
1434                                 NULL);
1435         } else {
1436             /*
1437              * Migrate to a cpu in the new partition.
1438              */
1439             cp = disp_lowpri_cpu(tp->t_cpupart->cp_cpulist,
1440                                 tp, tp->t_pri);
1441             tp->t_lpl, tp->t_pri, NULL);
1442         }
1443         ASSERT((cp->cpu_flags & CPU_QUIESCED) == 0);
1444     } else {
1445         /*
1446          * It is possible that t_weakbound_cpu != t_bound_cpu (for
1447          * a short time until weak binding that existed when the
1448          * strong binding was established has dropped) so we must
1449          * favour weak binding over strong.
1450          */
1451         cp = tp->t_weakbound_cpu ?
1452             tp->t_weakbound_cpu : tp->t_bound_cpu;
1453     }
1454
1455     /*
1456      * A thread that is ONPROC may be temporarily placed on the run queue
1457      * but then chosen to run again by disp. If the thread we're placing on
1458      * the queue is in TS_ONPROC state, don't set its t_waitrq until a
1459      * replacement process is actually scheduled in swtch(). In this
1460      * situation, curthread is the only thread that could be in the ONPROC
1461      * state.
1462      */
1463     if ((tp != curthread) && (tp->t_waitrq == 0)) {
1464         hrttime_t curtime;
1465
1466         curtime = gethrtime_unscaled();
1467         (void) cpu_update_pct(tp, curtime);
1468         tp->t_waitrq = curtime;
1469     } else {
1470         (void) cpu_update_pct(tp, gethrtime_unscaled());
1471     }
1472
1473     dp = cp->cpu_disp;
1474     disp_lock_enter_high(&dp->disp_lock);
1475
1476     TRACE_2(TR_FAC_DISP, TR_FRONTQ, "frontq:pri %d tid %p", tpri, tp);
1477     DTRACE_SCHD3(enqueue, kthread_t *, tp, disp_t *, dp, int, 1);
1478
1479 #ifndef NPROBE
1480     /* Kernel probe */
1481     if (tnf_tracing_active)

```

```

1474         tnf_thread_queue(tp, cp, tpri);
1475 #endif /* NPROBE */
1476
1477     ASSERT(tpri >= 0 && tpri < dp->disp_npri);
1478
1479     THREAD_RUN(tp, &dp->disp_lock); /* set TS_RUN state and lock */
1480     tp->t_disp_queue = dp;
1481
1482     dq = &dp->disp_q[tpri];
1483     dp->disp_nrunnable++;
1484     if (!bound)
1485         dp->disp_steal = 0;
1486     membar_enter();
1487
1488     if (dq->dq_sruncnt++ != 0) {
1489         ASSERT(dq->dq_last != NULL);
1490         tp->t_link = dq->dq_first;
1491         dq->dq_first = tp;
1492     } else {
1493         ASSERT(dq->dq_last == NULL);
1494         ASSERT(dq->dq_first == NULL);
1495         tp->t_link = NULL;
1496         dq->dq_first = dq->dq_last = tp;
1497         BT_SET(dp->disp_qactmap, tpri);
1498         if (tpri > dp->disp_maxrunpri) {
1499             dp->disp_maxrunpri = tpri;
1500             membar_enter();
1501             cpu_resched(cp, tpri);
1502         }
1503     }
1504
1505     if (!bound && tpri > dp->disp_max_unbound_pri) {
1506         if (tp == curthread && dp->disp_max_unbound_pri == -1 &&
1507             cp == CPU) {
1508             /*
1509              * If there are no other unbound threads on the
1510              * run queue, don't allow other CPUs to steal
1511              * this thread while we are in the middle of a
1512              * context switch. We may just switch to it
1513              * again right away. CPU_DISP_DONTSTEAL is cleared
1514              * in swtch and swtch_to.
1515              */
1516             cp->cpu_disp_flags |= CPU_DISP_DONTSTEAL;
1517         }
1518         dp->disp_max_unbound_pri = tpri;
1519     }
1520     (*disp_enq_thread)(cp, bound);
1521 }
1522
1523 /*
1524  * Put a high-priority unbound thread on the kp queue
1525  */
1526 static void
1527 setkpdq(kthread_t *tp, int borf)
1528 {
1529     dispq_t *dq;
1530     disp_t *dp;
1531     cpu_t *cp;
1532     pri_t tpri;
1533
1534     tpri = DISP_PRIO(tp);
1535
1536     dp = &tp->t_cpupart->cp_kp_queue;
1537     disp_lock_enter_high(&dp->disp_lock);
1538
1539     TRACE_2(TR_FAC_DISP, TR_FRONTQ, "frontq:pri %d tid %p", tpri, tp);

```

```

1541     ASSERT(tpri >= 0 && tpri < dp->disp_npri);
1542     DTRACE_SCHED3(enqueue, kthread_t *, tp, disp_t *, dp, int, borf);
1543     THREAD_RUN(tp, &dp->disp_lock);          /* set t_state to TS_RUN */
1544     tp->t_disp_queue = dp;
1545     dp->disp_nrunnable++;
1546     dq = &dp->disp_q[tpri];

1548     if (dq->dq_sruncnt++ != 0) {
1549         if (borf == SETKP_BACK) {
1550             ASSERT(dq->dq_first != NULL);
1551             tp->t_link = NULL;
1552             dq->dq_last->t_link = tp;
1553             dq->dq_last = tp;
1554         } else {
1555             ASSERT(dq->dq_last != NULL);
1556             tp->t_link = dq->dq_first;
1557             dq->dq_first = tp;
1558         }
1559     } else {
1560         if (borf == SETKP_BACK) {
1561             ASSERT(dq->dq_first == NULL);
1562             ASSERT(dq->dq_last == NULL);
1563             dq->dq_first = dq->dq_last = tp;
1564         } else {
1565             ASSERT(dq->dq_last == NULL);
1566             ASSERT(dq->dq_first == NULL);
1567             tp->t_link = NULL;
1568             dq->dq_first = dq->dq_last = tp;
1569         }
1570         BT_SET(dp->disp_qactmap, tpri);
1571         if (tpri > dp->disp_max_unbound_pri)
1572             dp->disp_max_unbound_pri = tpri;
1573         if (tpri > dp->disp_maxrunpri) {
1574             dp->disp_maxrunpri = tpri;
1575             membar_enter();
1576         }
1577     }

1579     cp = tp->t_cpu;
1580     if (tp->t_cpupart != cp->cpu_part) {
1581         /* migrate to a cpu in the new partition */
1582         cp = tp->t_cpupart->cp_cpulist;
1583     }
1584     cp = disp_lowpri_cpu(cp, tp, tp->t_pri);
1585     cp = disp_lowpri_cpu(cp, tp->t_lpl, tp->t_pri, NULL);
1586     disp_lock_enter_high(&cp->cpu_disp->disp_lock);
1587     ASSERT((cp->cpu_flags & CPU_QUIESCED) == 0);

1588 #ifndef NPROBE
1589     /* Kernel probe */
1590     if (tnf_tracing_active)
1591         tnf_thread_queue(tp, cp, tpri);
1592 #endif /* NPROBE */

1594     if (cp->cpu_chosen_level < tpri)
1595         cp->cpu_chosen_level = tpri;
1596     cpu_resched(cp, tpri);
1597     disp_lock_exit_high(&cp->cpu_disp->disp_lock);
1598     (*disp_enq_thread)(cp, 0);
1599 }

```

unchanged_portion_omitted

```

2556 /*
2557 * Return a score rating this CPU for running this thread: lower is better.
2558 * disp_lowpri_cpu - find CPU running the lowest priority thread.

```

```

2557 * The hint passed in is used as a starting point so we don't favor
2558 * CPU 0 or any other CPU. The caller should pass in the most recently
2559 * used CPU for the thread.
2560 *
2561 * If curthread is looking for a new CPU, then we ignore cpu_dispatch_pri for
2562 * curcpu (as that's our own priority).
2563 * The lgroup and priority are used to determine the best CPU to run on
2564 * in a NUMA machine. The lgroup specifies which CPUs are closest while
2565 * the thread priority will indicate whether the thread will actually run
2566 * there. To pick the best CPU, the CPUs inside and outside of the given
2567 * lgroup which are running the lowest priority threads are found. The
2568 * remote CPU is chosen only if the thread will not run locally on a CPU
2569 * within the lgroup, but will run on the remote CPU. If the thread
2570 * cannot immediately run on any CPU, the best local CPU will be chosen.
2571 *
2572 * If a cpu is the target of an offline request, then try to avoid it.
2573 * The lpl specified also identifies the cpu partition from which
2574 * disp_lowpri_cpu should select a CPU.
2575 *
2576 * Otherwise we'll use double the effective dispatcher priority for the CPU.
2577 * curcpu is used to indicate that disp_lowpri_cpu is being called on
2578 * behalf of the current thread. (curthread is looking for a new cpu)
2579 * In this case, cpu_dispatch_pri for this thread's cpu should be
2580 * ignored.
2581 *
2582 * We do this so ht_adjust_cpu_score() can increment the score if needed,
2583 * without ending up over-riding a dispatcher priority.
2584 */
2585 static pri_t
2586 cpu_score(cpu_t *cp, kthread_t *tp)
2587 {
2588     pri_t score;

2590     if (tp == curthread && cp == curthread->t_cpu)
2591         score = 2 * CPU_IDLE_PRI;
2592     else if (cp == cpu_inmotion)
2593         score = SHRT_MAX;
2594     else
2595         score = 2 * cp->cpu_dispatch_pri;

2597     if (2 * cp->cpu_disp->disp_maxrunpri > score)
2598         score = 2 * cp->cpu_disp->disp_maxrunpri;
2599     if (2 * cp->cpu_chosen_level > score)
2600         score = 2 * cp->cpu_chosen_level;

2602     return (ht_adjust_cpu_score(tp, cp, score));
2603 }

2604 /*
2605 * disp_lowpri_cpu - find a suitable CPU to run the given thread.
2606 * If a cpu is the target of an offline request then try to avoid it.
2607 *
2608 * We are looking for a CPU with an effective dispatch priority lower than the
2609 * thread's, so that the thread will run immediately rather than be enqueued.
2610 * For NUMA locality, we prefer "home" CPUs within the thread's ->t_lpl group.
2611 * If we don't find an available CPU there, we will expand our search to include
2612 * wider locality levels. (Note these groups are already divided by CPU
2613 * partition.)
2614 *
2615 * If the thread cannot immediately run on *any* CPU, we'll enqueue ourselves on
2616 * the best home CPU we found.
2617 *
2618 * The hint passed in is used as a starting point so we don't favor CPU 0 or any
2619 * other CPU. The caller should pass in the most recently used CPU for the
2620 * thread; it's of course possible that this CPU isn't in the home lgroup.
2621 */

```

```

2606 * This function must be called at either high SPL, or with preemption disabled,
2607 * so that the "hint" CPU cannot be removed from the online CPU list while we
2608 * are traversing it.
2580 * This function must be called at either high SPL, or with preemption
2581 * disabled, so that the "hint" CPU cannot be removed from the online
2582 * CPU list while we are traversing it.
2609 */
2610 cpu_t *
2611 disp_lowpri_cpu(cpu_t *hint, kthread_t *tp, pri_t tpri)
2585 disp_lowpri_cpu(cpu_t *hint, lpl_t *lpl, pri_t tpri, cpu_t *curcpu)
2612 {
2613     cpu_t     *bestcpu;
2614     cpu_t     *besthomecpu;
2615     cpu_t     *cp, *cpstart;

2591     pri_t     bestpri;
2592     pri_t     cpupri;

2617     klgrpset_t     done;
2595     klgrpset_t     cur_set;

2619     lpl_t         *lpl_iter, *lpl_leaf;
2598     int           i;

2600     /*
2601     * Scan for a CPU currently running the lowest priority thread.
2602     * Cannot get cpu_lock here because it is adaptive.
2603     * We do not require lock on CPU list.
2604     */
2621     ASSERT(hint != NULL);
2622     ASSERT(tp->t_lpl->lpl_ncpu > 0);
2606     ASSERT(lpl != NULL);
2607     ASSERT(lpl->lpl_ncpu > 0);

2609     /*
2610     * First examine local CPUs. Note that it's possible the hint CPU
2611     * passed in in remote to the specified home lgroup. If our priority
2612     * isn't sufficient enough such that we can run immediately at home,
2613     * then examine CPUs remote to our home lgroup.
2614     * We would like to give preference to CPUs closest to "home".
2615     * If we can't find a CPU where we'll run at a given level
2616     * of locality, we expand our search to include the next level.
2617     */
2624     bestcpu = besthomecpu = NULL;
2625     klgrpset_clear(done);
2620     /* start with lpl we were passed */

2627     lpl_iter = tp->t_lpl;
2622     lpl_iter = lpl;

2629     do {
2630         pri_t best = SHRT_MAX;
2631         klgrpset_t cur_set;

2626         bestpri = SHRT_MAX;
2633         klgrpset_clear(cur_set);

2635         for (int i = 0; i < lpl_iter->lpl_nrset; i++) {
2629             for (i = 0; i < lpl_iter->lpl_nrset; i++) {
2636                 lpl_leaf = lpl_iter->lpl_rset[i];
2637                 if (klgrpset_ismember(done, lpl_leaf->lpl_lgrpid))
2638                     continue;

2640                 klgrpset_add(cur_set, lpl_leaf->lpl_lgrpid);

2642                 if (hint->cpu_lpl == lpl_leaf)

```

```

2643     cp = cpstart = hint;
2644     else
2645     cp = cpstart = lpl_leaf->lpl_cpus;

2647     do {
2648         pri_t score = cpu_score(cp, tp);

2650         if (score < best) {
2651             best = score;
2642             if (cp == curcpu)
2643                 cpupri = -1;
2644             else if (cp == cpu_inmotion)
2645                 cpupri = SHRT_MAX;
2646             else
2647                 cpupri = cp->cpu_dispatch_pri;
2648             if (cp->cpu_disp->disp_maxrunpri > cpupri)
2649                 cpupri = cp->cpu_disp->disp_maxrunpri;
2650             if (cp->cpu_chosen_level > cpupri)
2651                 cpupri = cp->cpu_chosen_level;
2652             if (cpupri < bestpri) {
2653                 if (CPU_IDLING(cpupri)) {
2654                     ASSERT((cp->cpu_flags &
2655                         CPU_QUIESCED) == 0);
2656                     return (cp);
2657                 }
2652             bestcpu = cp;

2654             /* An idle CPU: we're done. */
2655             if (score / 2 == CPU_IDLE_PRI)
2656                 goto out;
2659             bestpri = cpupri;
2657         }
2658     } while ((cp = cp->cpu_next_lpl) != cpstart);
2659 }

2661 if (bestcpu != NULL && tpri > (best / 2))
2662     goto out;

2664 if (bestcpu && (tpri > bestpri)) {
2665     ASSERT((bestcpu->cpu_flags & CPU_QUIESCED) == 0);
2666     return (bestcpu);
2667 }
2664 if (besthomecpu == NULL)
2665     besthomecpu = bestcpu;

2667     /*
2668     * Add the lgrps we just considered to the "done" set
2669     */
2670     klgrpset_or(done, cur_set);

2672 } while ((lpl_iter = lpl_iter->lpl_parent) != NULL);

2674     /*
2675     * The specified priority isn't high enough to run immediately
2676     * anywhere, so just return the best CPU from the home lgroup.
2677     */
2678     bestcpu = besthomecpu;

2680 out:
2681     ASSERT((bestcpu->cpu_flags & CPU_QUIESCED) == 0);
2682     return (bestcpu);
2681     ASSERT((besthomecpu->cpu_flags & CPU_QUIESCED) == 0);
2682     return (besthomecpu);
2683 }

```

_____unchanged_portion_omitted_____

```
2697 /*ARGSUSED*/
2698 static void
2699 generic_enq_thread(cpu_t *cpu, int bound)
2700 {
2701 }

2703 cpu_t *
2704 disp_choose_best_cpu(void)
2705 {
2706     kthread_t *t = curthread;
2707     cpu_t *curcpu = CPU;

2709     ASSERT(t->t_preempt > 0);
2710     ASSERT(t->t_state == TS_ONPROC);
2711     ASSERT(t->t_schedflag & TS_VCPU);

2713     if (ht_should_run(t, curcpu))
2714         return (curcpu);

2716     return (disp_lowpri_cpu(curcpu, t, t->t_pri));
2717 }
_____unchanged_portion_omitted_____
```

new/usr/src/uts/common/disp/thread.c

1

```
*****
55705 Tue May 7 04:49:56 2019
new/usr/src/uts/common/disp/thread.c
OS-7125 Need mitigation of LITF (CVE-2018-3646)
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
*****
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) 2018 Joyent, Inc.
25  */

27 #include <sys/types.h>
28 #include <sys/param.h>
29 #include <sys/sysmacros.h>
30 #include <sys/signal.h>
31 #include <sys/stack.h>
32 #include <sys/pcb.h>
33 #include <sys/user.h>
34 #include <sys/system.h>
35 #include <sys/sysinfo.h>
36 #include <sys/errno.h>
37 #include <sys/cmn_err.h>
38 #include <sys/cred.h>
39 #include <sys/resource.h>
40 #include <sys/task.h>
41 #include <sys/project.h>
42 #include <sys/proc.h>
43 #include <sys/debug.h>
44 #include <sys/disp.h>
45 #include <sys/class.h>
46 #include <vm/seg_kmem.h>
47 #include <vm/seg_kp.h>
48 #include <sys/machlock.h>
49 #include <sys/kmem.h>
50 #include <sys/varargs.h>
51 #include <sys/turnstile.h>
52 #include <sys/poll.h>
53 #include <sys/vtrace.h>
54 #include <sys/callb.h>
55 #include <c2/audit.h>
56 #include <sys/tnf.h>
57 #include <sys/sobject.h>
58 #include <sys/cpupart.h>
59 #include <sys/pset.h>
```

new/usr/src/uts/common/disp/thread.c

2

```
60 #include <sys/door.h>
61 #include <sys/spl.h>
62 #include <sys/copyops.h>
63 #include <sys/rctl.h>
64 #include <sys/brand.h>
65 #include <sys/pool.h>
66 #include <sys/zone.h>
67 #include <sys/tsol/label.h>
68 #include <sys/tsol/tndb.h>
69 #include <sys/cpc_impl.h>
70 #include <sys/sdt.h>
71 #include <sys/reboot.h>
72 #include <sys/kdi.h>
73 #include <sys/schedctl.h>
74 #include <sys/waitq.h>
75 #include <sys/cpucaps.h>
76 #include <sys/kiconv.h>
77 #include <sys/ctype.h>
78 #include <sys/ht.h>

80 #ifndef STACK_GROWTH_DOWN
81 #error Stacks do not grow downward; 3b2 zombie attack detected!
82 #endif

84 struct kmem_cache *thread_cache; /* cache of free threads */
85 struct kmem_cache *lwp_cache; /* cache of free lwps */
86 struct kmem_cache *turnstile_cache; /* cache of free turnstiles */

88 /*
89  * allthreads is only for use by kmem_readers. All kernel loops can use
90  * the current thread as a start/end point.
91  */
92 kthread_t *allthreads = &t0; /* circular list of all threads */

94 static kcondvar_t reaper_cv; /* synchronization var */
95 kthread_t *thread_deathrow; /* circular list of reapable threads */
96 kthread_t *lwp_deathrow; /* circular list of reapable threads */
97 kmutex_t reaplock; /* protects lwp and thread deathrows */
98 int thread_reapcnt = 0; /* number of threads on deathrow */
99 int lwp_reapcnt = 0; /* number of lwps on deathrow */
100 int reaplimit = 16; /* delay reaping until reaplimit */

102 thread_free_lock_t *thread_free_lock;
103 /* protects tick thread from reaper */

105 extern int nthread;

107 /* System Scheduling classes. */
108 id_t syscid; /* system scheduling class ID */
109 id_t sysdccid = CLASS_UNUSED; /* reset when SDC loads */

111 void *segkp_thread; /* cookie for segkp pool */

113 int lwp_cache_sz = 32;
114 int t_cache_sz = 8;
115 static kt_did_t next_t_id = 1;

117 /* Default mode for thread binding to CPUs and processor sets */
118 int default_binding_mode = TB_ALLHARD;

120 /*
121  * Min/Max stack sizes for stack size parameters
122  */
123 #define MAX_STKSIZE (32 * DEFAULTSTKSZ)
124 #define MIN_STKSIZE DEFAULTSTKSZ
```

```

126 /*
127 * default_stksize overrides lwp_default_stksize if it is set.
128 */
129 int     default_stksize;
130 int     lwp_default_stksize;

132 static zone_key_t zone_thread_key;

134 unsigned int kmem_stackinfo;      /* stackinfo feature on-off */
135 kmem_stkinfo_t *kmem_stkinfo_log; /* stackinfo circular log */
136 static kmutex_t kmem_stkinfo_lock; /* protects kmem_stkinfo_log */

138 /*
139 * forward declarations for internal thread specific data (tsd)
140 */
141 static void *tsd_realloc(void *, size_t, size_t);

143 void thread_reaper(void);

145 /* forward declarations for stackinfo feature */
146 static void stkinfo_begin(kthread_t *);
147 static void stkinfo_end(kthread_t *);
148 static size_t stkinfo_percent(caddr_t, caddr_t, caddr_t);

150 /*ARGSUSED*/
151 static int
152 turnstile_constructor(void *buf, void *cdrarg, int kmflags)
153 {
154     bzero(buf, sizeof (turnstile_t));
155     return (0);
156 }
unchanged portion omitted

320 /*
321 * Create a thread.
322 *
323 * thread_create() blocks for memory if necessary.  It never fails.
324 *
325 * If stk is NULL, the thread is created at the base of the stack
326 * and cannot be swapped.
327 */
328 kthread_t *
329 thread_create(
330     caddr_t stk,
331     size_t stksize,
332     void (*proc)(),
333     void *arg,
334     size_t len,
335     proc_t *pp,
336     int state,
337     pri_t pri)
338 {
339     kthread_t *t;
340     extern struct classfuncs sys_classfuncs;
341     turnstile_t *ts;

343     /*
344     * Every thread keeps a turnstile around in case it needs to block.
345     * The only reason the turnstile is not simply part of the thread
346     * structure is that we may have to break the association whenever
347     * more than one thread blocks on a given synchronization object.
348     * From a memory-management standpoint, turnstiles are like the
349     * "attached mblks" that hang off dblks in the streams allocator.
350     */
351     ts = kmem_cache_alloc(turnstile_cache, KM_SLEEP);

```

```

353     if (stk == NULL) {
354         /*
355         * alloc both thread and stack in segkp chunk
356         */

358         if (stksize < default_stksize)
359             stksize = default_stksize;

361         if (stksize == default_stksize) {
362             stk = (caddr_t)segkp_cache_get(segkp_thread);
363         } else {
364             stksize = roundup(stksize, PAGESIZE);
365             stk = (caddr_t)segkp_get(segkp, stksize,
366                 (KPD_HASREDZONE | KPD_NO_ANON | KPD_LOCKED));
367         }

369         ASSERT(stk != NULL);

371         /*
372         * The machine-dependent mutex code may require that
373         * thread pointers (since they may be used for mutex owner
374         * fields) have certain alignment requirements.
375         * PTR24_ALIGN is the size of the alignment quanta.
376         * XXX - assumes stack grows toward low addresses.
377         */
378         if (stksize <= sizeof (kthread_t) + PTR24_ALIGN)
379             cmn_err(CE_PANIC, "thread_create: proposed stack size"
380                 " too small to hold thread.");
381 #ifdef STACK_GROWTH_DOWN
382         stksize -= SA(sizeof (kthread_t) + PTR24_ALIGN - 1);
383         stksize &= -PTR24_ALIGN; /* make thread aligned */
384         t = (kthread_t *) (stk + stksize);
385         bzero(t, sizeof (kthread_t));
386         if (audit_active)
387             audit_thread_create(t);
388         t->t_stk = stk + stksize;
389         t->t_stkbase = stk;
390 #else /* stack grows to larger addresses */
391         stksize -= SA(sizeof (kthread_t));
392         t = (kthread_t *) (stk);
393         bzero(t, sizeof (kthread_t));
394         t->t_stk = stk + sizeof (kthread_t);
395         t->t_stkbase = stk + stksize + sizeof (kthread_t);
396 #endif /* STACK_GROWTH_DOWN */
397         t->t_flag |= T_TALLOCSTK;
398         t->t_swap = stk;
399     } else {
400         t = kmem_cache_alloc(thread_cache, KM_SLEEP);
401         bzero(t, sizeof (kthread_t));
402         ASSERT(((uintptr_t)t & (PTR24_ALIGN - 1)) == 0);
403         if (audit_active)
404             audit_thread_create(t);
405         /*
406         * Initialize t_stk to the kernel stack pointer to use
407         * upon entry to the kernel
408         */
409 #ifdef STACK_GROWTH_DOWN
410         t->t_stk = stk + stksize;
411         t->t_stkbase = stk;
412 #else
413         t->t_stk = stk; /* 3b2-like */
414         t->t_stkbase = stk + stksize;
415 #endif /* STACK_GROWTH_DOWN */
416     }

418     if (kmem_stackinfo != 0) {

```

```

419         stkinfo_begin(t);
420     }

422     t->t_ts = ts;

424     /*
425      * p_cred could be NULL if it thread_create is called before cred_init
426      * is called in main.
427      */
428     mutex_enter(&pp->p_crlock);
429     if (pp->p_cred)
430         crhold(t->t_cred = pp->p_cred);
431     mutex_exit(&pp->p_crlock);
432     t->t_start = getthretime_sec();
433     t->t_startpc = proc;
434     t->t_procp = pp;
435     t->t_clfuncs = &sys_classfuncs.thread;
436     t->t_cid = syscid;
437     t->t_pri = pri;
438     t->t_stime = ddi_get_lbolt();
439     t->t_schedflag = TS_LOAD | TS_DONT_SWAP;
440     t->t_bind_cpu = PBIND_NONE;
441     t->t_bindflag = (uchar_t)default_binding_mode;
442     t->t_bind_pset = PS_NONE;
443     t->t_plockp = &pp->p_lock;
444     t->t_copyops = NULL;
445     t->t_taskq = NULL;
446     t->t_anttime = 0;
447     t->t_hatdepth = 0;

449     t->t_dtrace_vtime = 1; /* assure vtimestamp is always non-zero */

451     CPU_STATS_ADDQ(CPU, sys, nthreads, 1);
452 #ifndef NPROBE
453     /* Kernel probe */
454     tnf_thread_create(t);
455 #endif /* NPROBE */
456     LOCK_INIT_CLEAR(&t->t_lock);

458     /*
459      * Callers who give us a NULL proc must do their own
460      * stack initialization. e.g. lwp_create()
461      */
462     if (proc != NULL) {
463         t->t_stk = thread_stk_init(t->t_stk);
464         thread_load(t, proc, arg, len);
465     }

467     /*
468      * Put a hold on project0. If this thread is actually in a
469      * different project, then t_proj will be changed later in
470      * lwp_create(). All kernel-only threads must be in project 0.
471      */
472     t->t_proj = project_hold(proj0p);

474     lgrp_affinity_init(&t->t_lgrp_affinity);

476     mutex_enter(&pidlock);
477     nthread++;
478     t->t_did = next_t_id++;
479     t->t_prev = curthread->t_prev;
480     t->t_next = curthread;

482     /*
483      * Add the thread to the list of all threads, and initialize
484      * its t_cpu pointer. We need to block preemption since

```

```

485     * cpu_offline walks the thread list looking for threads
486     * with t_cpu pointing to the CPU being offlined. We want
487     * to make sure that the list is consistent and that if t_cpu
488     * is set, the thread is on the list.
489     */
490     kpreempt_disable();
491     curthread->t_prev->t_next = t;
492     curthread->t_prev = t;

494     /*
495      * Threads should never have a NULL t_cpu pointer so assign it
496      * here. If the thread is being created with state TS_RUN a
497      * better CPU may be chosen when it is placed on the run queue.
498      */
499     /* We need to keep kernel preemption disabled when setting all
500      * three fields to keep them in sync. Also, always create in
501      * the default partition since that's where kernel threads go
502      * (if this isn't a kernel thread, t_cpupart will be changed
503      * in lwp_create before setting the thread runnable).
504      */
505     t->t_cpupart = &cp_default;

507     /*
508      * For now, affiliate this thread with the root lgroup.
509      * Since the kernel does not (presently) allocate its memory
510      * in a locality aware fashion, the root is an appropriate home.
511      * If this thread is later associated with an lwp, it will have
512      * its lgroup re-assigned at that time.
513      */
514     lgrp_move_thread(t, &cp_default.cp_lgrploads[LGRP_ROOTID], 1);

516     /*
517      * Inherit the current cpu. If this cpu isn't part of the chosen
518      * lgroup, a new cpu will be chosen by cpu_choose when the thread
519      * is ready to run.
520      */
521     if (CPU->cpu_part == &cp_default)
522         t->t_cpu = CPU;
523     else
524         t->t_cpu = disp_lowpri_cpu(cp_default.cp_cpulist, t,
525                                 t->t_pri);
526     t->t_cpu = disp_lowpri_cpu(cp_default.cp_cpulist, t->t_lpl,
527                             t->t_pri, NULL);

527     t->t_disp_queue = t->t_cpu->cpu_disp;
528     kpreempt_enable();

530     /*
531      * Initialize thread state and the dispatcher lock pointer.
532      * Need to hold onto pidlock to block allthreads walkers until
533      * the state is set.
534      */
535     switch (state) {
536     case TS_RUN:
537         curthread->t_oldspl = splhigh(); /* get dispatcher spl */
538         THREAD_SET_STATE(t, TS_STOPPED, &transition_lock);
539         CL_SETRUN(t);
540         thread_unlock(t);
541         break;

543     case TS_ONPROC:
544         THREAD_ONPROC(t, t->t_cpu);
545         break;

547     case TS_FREE:
548         /*

```



```

549         * Free state will be used for intr threads.
550         * The interrupt routine must set the thread dispatcher
551         * lock pointer (t_lockp) if starting on a CPU
552         * other than the current one.
553         */
554         THREAD_FREEINTR(t, CPU);
555         break;

557     case TS_STOPPED:
558         THREAD_SET_STATE(t, TS_STOPPED, &stop_lock);
559         break;

561     default:
562         /* TS_SLEEP, TS_ZOMB or TS_TRANS */
563         cmn_err(CE_PANIC, "thread_create: invalid state %d", state);
564     }
565     mutex_exit(&pidlock);
566     return (t);
}

```

unchanged_portion_omitted

```

1309 /*
1310 * Unpin an interrupted thread.
1311 * When an interrupt occurs, the interrupt is handled on the stack
1312 * of an interrupt thread, taken from a pool linked to the CPU structure.
1313 *
1314 * When swtch() is switching away from an interrupt thread because it
1315 * blocked or was preempted, this routine is called to complete the
1316 * saving of the interrupted thread state, and returns the interrupted
1317 * thread pointer so it may be resumed.
1318 *
1319 * Called by swtch() only at high spl.
1320 */
1321 kthread_t *
1322 thread_unpin()
1323 {
1324     kthread_t     *t = curthread; /* current thread */
1325     kthread_t     *itp;           /* interrupted thread */
1326     int           i;              /* interrupt level */
1327     extern int     intr_passivate();

1329     ASSERT(t->t_intr != NULL);

1331     itp = t->t_intr;              /* interrupted thread */
1332     t->t_intr = NULL;            /* clear interrupt ptr */

1334     ht_end_intr();

1336     /*
1337     * Get state from interrupt thread for the one
1338     * it interrupted.
1339     */

1341     i = intr_passivate(t, itp);

1343     TRACE_5(TR_FAC_INTR, TR_INTR_PASSIVATE,
1344           "intr_passivate:level %d curthread %p (%T) ithread %p (%T)",
1345           i, t, t, itp, itp);

1347     /*
1348     * Dissociate the current thread from the interrupted thread's LWP.
1349     */
1350     t->t_lwp = NULL;

1352     /*
1353     * Interrupt handlers above the level that spinlocks block must
1354     * not block.

```

```

1355         */
1356     #if DEBUG
1357         if (i < 0 || i > LOCK_LEVEL)
1358             cmn_err(CE_PANIC, "thread_unpin: ipl out of range %x", i);
1359     #endif

1361     /*
1362     * Compute the CPU's base interrupt level based on the active
1363     * interrupts.
1364     */
1365     ASSERT(CPU->cpu_intr_actv & (1 << i));
1366     set_base_spl();

1368     return (itp);
1369 }

```

unchanged_portion_omitted

```

*****
54504 Tue May 7 04:49:56 2019
new/usr/src/uts/common/fs/zfs/zvol.c
OS-7125 Need mitigation of LITF (CVE-2018-3646)
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
*****
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) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
23 *
24 * Portions Copyright 2010 Robert Milkowski
25 *
26 * Copyright 2017 Nexenta Systems, Inc. All rights reserved.
27 * Copyright (c) 2012, 2017 by Delphix. All rights reserved.
28 * Copyright (c) 2013, Joyent, Inc. All rights reserved.
29 * Copyright (c) 2014 Integros [integros.com]
30 * Copyright (c) 2019, Joyent, Inc.
31 */
32 /*
33 * ZFS volume emulation driver.
34 *
35 * Makes a DMU object look like a volume of arbitrary size, up to 2^64 bytes.
36 * Volumes are accessed through the symbolic links named:
37 *
38 * /dev/zvol/dsk/<pool_name>/<dataset_name>
39 * /dev/zvol/rdisk/<pool_name>/<dataset_name>
40 *
41 * These links are created by the /dev filesystem (sdev_zvolops.c).
42 * Volumes are persistent through reboot. No user command needs to be
43 * run before opening and using a device.
44 */
46 #include <sys/types.h>
47 #include <sys/param.h>
48 #include <sys/errno.h>
49 #include <sys/uio.h>
50 #include <sys/buf.h>
51 #include <sys/modctl.h>
52 #include <sys/open.h>
53 #include <sys/kmem.h>
54 #include <sys/conf.h>
55 #include <sys/cmn_err.h>
56 #include <sys/stat.h>
57 #include <sys/zap.h>
58 #include <sys/spa.h>

```

```

59 #include <sys/spa_impl.h>
60 #include <sys/zio.h>
61 #include <sys/dmu_traverse.h>
62 #include <sys/dnode.h>
63 #include <sys/dsl_dataset.h>
64 #include <sys/dsl_prop.h>
65 #include <sys/dkio.h>
66 #include <sys/efi_partition.h>
67 #include <sys/byteorder.h>
68 #include <sys/pathname.h>
69 #include <sys/ddi.h>
70 #include <sys/sunddi.h>
71 #include <sys/crc32.h>
72 #include <sys/dirent.h>
73 #include <sys/policy.h>
74 #include <sys/fs/zfs.h>
75 #include <sys/zfs_ioctl.h>
76 #include <sys/mkdev.h>
77 #include <sys/zil.h>
78 #include <sys/refcount.h>
79 #include <sys/zfs_znode.h>
80 #include <sys/zfs_rlock.h>
81 #include <sys/vdev_disk.h>
82 #include <sys/vdev_impl.h>
83 #include <sys/vdev_raidz.h>
84 #include <sys/zvol.h>
85 #include <sys/dumphdr.h>
86 #include <sys/zil_impl.h>
87 #include <sys/dbuf.h>
88 #include <sys/dmu_tx.h>
89 #include <sys/zfeature.h>
90 #include <sys/zio_checksum.h>
91 #include <sys/zil_impl.h>
92 #include <sys/ht.h>
93 #include <sys/dkio_free_util.h>
94 #include <sys/zfs_rlock.h>
95
96 #include "zfs_namecheck.h"
97
98 void *zfsdev_state;
99 static char *zvol_tag = "zvol_tag";
100
101 #define ZVOL_DUMPSIZE "dumpsize"
102
103 /*
104 * This lock protects the zfsdev_state structure from being modified
105 * while it's being used, e.g. an open that comes in before a create
106 * finishes. It also protects temporary opens of the dataset so that,
107 * e.g., an open doesn't get a spurious EBUSY.
108 */
109 kmutex_t zfsdev_state_lock;
110 static uint32_t zvol_minors;
111
112 typedef struct zvol_extnt {
113     list_node_t ze_node;
114     dva_t ze_dva; /* dva associated with this extent */
115     uint64_t ze_nblks; /* number of blocks in extent */
116 } zvol_extnt_t;
117 #include "unchanged_portion_omitted"
118
119 int
120 zvol_strategy(buf_t *bp)
121 {
122     zfs_soft_state_t *zs = NULL;
123     zvol_state_t *zv;
124     uint64_t off, volsize;

```

```

1221     size_t resid;
1222     char *addr;
1223     objset_t *os;
1224     int error = 0;
1225     boolean_t doread = bp->b_flags & B_READ;
1226     boolean_t is_dumpified;
1227     boolean_t sync;

1229     if (getminor(bp->b_edev) == 0) {
1230         error = SET_ERROR(EINVAL);
1231     } else {
1232         zs = ddi_get_soft_state(zfsdev_state, getminor(bp->b_edev));
1233         if (zs == NULL)
1234             error = SET_ERROR(ENXIO);
1235         else if (zs->zss_type != ZSST_ZVOL)
1236             error = SET_ERROR(EINVAL);
1237     }

1239     if (error) {
1240         bioerror(bp, error);
1241         biodone(bp);
1242         return (0);
1243     }

1245     zv = zs->zss_data;

1247     if (!(bp->b_flags & B_READ) && (zv->zv_flags & ZVOL_RDONLY)) {
1248         bioerror(bp, EROFS);
1249         biodone(bp);
1250         return (0);
1251     }

1253     off = ldbtob(bp->b_blkno);
1254     volsize = zv->zv_volsize;

1256     os = zv->zv_objset;
1257     ASSERT(os != NULL);

1259     bp_mapin(bp);
1260     addr = bp->b_un.b_addr;
1261     resid = bp->b_bcount;

1263     if (resid > 0 && (off < 0 || off >= volsize)) {
1264         bioerror(bp, EIO);
1265         biodone(bp);
1266         return (0);
1267     }

1269     is_dumpified = zv->zv_flags & ZVOL_DUMPIFIED;
1270     sync = (!(bp->b_flags & B_ASYNC) &&
1271         !(zv->zv_flags & ZVOL_WCE) ||
1272         (zv->zv_objset->os_sync == ZFS_SYNC_ALWAYS)) &&
1273         !doread && !is_dumpified;

1275     ht_begin_unsafe();

1277     /*
1278     * There must be no buffer changes when doing a dmu_sync() because
1279     * we can't change the data whilst calculating the checksum.
1280     */
1281     locked_range_t *lr = rangelock_enter(&zv->zv_rangelock, off, resid,
1282         doread ? RL_READER : RL_WRITER);

1284     while (resid != 0 && off < volsize) {
1285         size_t size = MIN(resid, zvol_maxphys);
1286         if (is_dumpified) {

```

```

1287             size = MIN(size, P2END(off, zv->zv_volblocksize) - off);
1288             error = zvol_dumpio(zv, addr, off, size,
1289                 doread, B_FALSE);
1290         } else if (doread) {
1291             error = dmu_read(os, ZVOL_OBJ, off, size, addr,
1292                 DMU_READ_PREFETCH);
1293         } else {
1294             dmu_tx_t *tx = dmu_tx_create(os);
1295             dmu_tx_hold_write(tx, ZVOL_OBJ, off, size);
1296             error = dmu_tx_assign(tx, TXG_WAIT);
1297             if (error) {
1298                 dmu_tx_abort(tx);
1299             } else {
1300                 dmu_write(os, ZVOL_OBJ, off, size, addr, tx);
1301                 zvol_log_write(zv, tx, off, size, sync);
1302                 dmu_tx_commit(tx);
1303             }
1304         }
1305     }
1306     if (error) {
1307         /* convert checksum errors into IO errors */
1308         if (error == ECKSUM)
1309             error = SET_ERROR(EIO);
1310         break;
1311     }
1312     off += size;
1313     addr += size;
1314     resid -= size;
1315     rangelock_exit(lr);

1317     if ((bp->b_resid = resid) == bp->b_bcount)
1318         bioerror(bp, off > volsize ? EINVAL : error);

1320     if (sync)
1321         zil_commit(zv->zv_zilog, ZVOL_OBJ);
1322     biodone(bp);

1324     ht_end_unsafe();

1326     return (0);
1327 }

unchanged_portion_omitted

1379 /*ARGSUSED*/
1380 int
1381 zvol_read(dev_t dev, uio_t *uio, cred_t *cr)
1382 {
1383     minor_t minor = getminor(dev);
1384     zvol_state_t *zv;
1385     uint64_t volsize;
1386     int error = 0;

1388     zv = zfsdev_get_soft_state(minor, ZSST_ZVOL);
1389     if (zv == NULL)
1390         return (SET_ERROR(ENXIO));

1392     volsize = zv->zv_volsize;
1393     if (uio->uio_resid > 0 &&
1394         (uio->uio_loffset < 0 || uio->uio_loffset >= volsize))
1395         return (SET_ERROR(EIO));

1397     if (zv->zv_flags & ZVOL_DUMPIFIED) {
1398         error = physio(zvol_strategy, NULL, dev, B_READ,
1399             zvol_minphys, uio);
1400         return (error);
1401     }

```

```

1403     ht_begin_unsafe();
1405     locked_range_t *lr = rangelock_enter(&zv->zv_rangelock,
1406     uio->uio_loffset, uio->uio_resid, RL_READER);
1407     while (uio->uio_resid > 0 && uio->uio_loffset < volsize) {
1408     uint64_t bytes = MIN(uio->uio_resid, DMU_MAX_ACCESS >> 1);
1410         /* don't read past the end */
1411         if (bytes > volsize - uio->uio_loffset)
1412             bytes = volsize - uio->uio_loffset;
1414         error = dmu_read_uio(zv->zv_objset, ZVOL_OBJ, uio, bytes);
1415         if (error) {
1416             /* convert checksum errors into IO errors */
1417             if (error == ECKSUM)
1418                 error = SET_ERROR(EIO);
1419             break;
1420         }
1421     }
1422     rangelock_exit(lr);
1424     ht_end_unsafe();
1426     return (error);
1427 }
1429 /*ARGSUSED*/
1430 int
1431 zvol_write(dev_t dev, uio_t *uio, cred_t *cr)
1432 {
1433     minor_t minor = getminor(dev);
1434     zvol_state_t *zv;
1435     uint64_t volsize;
1436     int error = 0;
1437     boolean_t sync;
1439     zv = zfsdev_get_soft_state(minor, ZSST_ZVOL);
1440     if (zv == NULL)
1441         return (SET_ERROR(ENXIO));
1443     volsize = zv->zv_volsize;
1444     if (uio->uio_resid > 0 &&
1445     (uio->uio_loffset < 0 || uio->uio_loffset >= volsize))
1446         return (SET_ERROR(EIO));
1448     if (zv->zv_flags & ZVOL_DUMPIFIED) {
1449         error = physio(zvol_strategy, NULL, dev, B_WRITE,
1450             zvol_minphys, uio);
1451         return (error);
1452     }
1454     ht_begin_unsafe();
1456     sync = !(zv->zv_flags & ZVOL_WCE) ||
1457     (zv->zv_objset->os_sync == ZFS_SYNC_ALWAYS);
1459     locked_range_t *lr = rangelock_enter(&zv->zv_rangelock,
1460     uio->uio_loffset, uio->uio_resid, RL_WRITER);
1461     while (uio->uio_resid > 0 && uio->uio_loffset < volsize) {
1462         uint64_t bytes = MIN(uio->uio_resid, DMU_MAX_ACCESS >> 1);
1463         uint64_t off = uio->uio_loffset;
1464         dmu_tx_t *tx = dmu_tx_create(zv->zv_objset);
1466         if (bytes > volsize - off) /* don't write past the end */
1467             bytes = volsize - off;

```

```

1469         dmu_tx_hold_write(tx, ZVOL_OBJ, off, bytes);
1470         error = dmu_tx_assign(tx, TXG_WAIT);
1471         if (error) {
1472             dmu_tx_abort(tx);
1473             break;
1474         }
1475         error = dmu_write_uio_dnode(zv->zv_dn, uio, bytes, tx);
1476         if (error == 0)
1477             zvol_log_write(zv, tx, off, bytes, sync);
1478         dmu_tx_commit(tx);
1480         if (error)
1481             break;
1482     }
1483     rangelock_exit(lr);
1485     if (sync)
1486         zil_commit(zv->zv_zilog, ZVOL_OBJ);
1488     ht_end_unsafe();
1490     return (error);
1491 }
1492 unchanged_portion_omitted
1493
1494 /*
1495  * Dirtbag ioctls to support mkfs(1M) for UFS filesystems. See dkio(7I).
1496  * Also a dirtbag dkio ioctl for unmap/free-block functionality.
1497  */
1498 /*ARGSUSED*/
1499 int
1500 zvol_ioctl(dev_t dev, int cmd, intptr_t arg, int flag, cred_t *cr, int *rvalp)
1501 {
1502     zvol_state_t *zv;
1503     struct dk_callback *dkc;
1504     int error = 0;
1505     locked_range_t *lr;
1507     mutex_enter(&zfsdev_state_lock);
1509     zv = zfsdev_get_soft_state(getminor(dev), ZSST_ZVOL);
1511     if (zv == NULL) {
1512         mutex_exit(&zfsdev_state_lock);
1513         return (SET_ERROR(ENXIO));
1514     }
1515     ASSERT(zv->zv_total_opens > 0);
1517     switch (cmd) {
1519     case DKIOCINFO:
1520     {
1521         struct dk_cinfo dki;
1523         bzero(&dki, sizeof (dki));
1524         (void) strcpy(dki.dki_cname, "zvol");
1525         (void) strcpy(dki.dki_dname, "zvol");
1526         dki.dki_ctype = DKC_UNKNOWN;
1527         dki.dki_unit = getminor(dev);
1528         dki.dki_maxtransfer =
1529             1 << (SPA_OLD_MAXBLOCKSHIFT - zv->zv_min_bs);
1530         mutex_exit(&zfsdev_state_lock);
1531         if (ddi_copyout(&dki, (void *)arg, sizeof (dki), flag))
1532             error = SET_ERROR(EFAULT);
1533         return (error);

```

```

1686     }
1688     case DKIOCGMEDIAINFO:
1689     {
1690         struct dk_minfo dkm;
1692         bzero(&dkm, sizeof (dkm));
1693         dkm.dki_lbsize = 1U << zv->zv_min_bs;
1694         dkm.dki_capacity = zv->zv_volsize >> zv->zv_min_bs;
1695         dkm.dki_media_type = DK_UNKNOWN;
1696         mutex_exit(&zfsdev_state_lock);
1697         if (ddi_copyout(&dkm, (void *)arg, sizeof (dkm), flag))
1698             error = SET_ERROR(EFAULT);
1699         return (error);
1700     }
1702     case DKIOCGMEDIAINFOEXT:
1703     {
1704         struct dk_minfo_ext dkmext;
1706         bzero(&dkmext, sizeof (dkmext));
1707         dkmext.dki_lbsize = 1U << zv->zv_min_bs;
1708         dkmext.dki_pbsize = zv->zv_volblocks;
1709         dkmext.dki_capacity = zv->zv_volsize >> zv->zv_min_bs;
1710         dkmext.dki_media_type = DK_UNKNOWN;
1711         mutex_exit(&zfsdev_state_lock);
1712         if (ddi_copyout(&dkmext, (void *)arg, sizeof (dkmext), flag))
1713             error = SET_ERROR(EFAULT);
1714         return (error);
1715     }
1717     case DKIOCGTEFEI:
1718     {
1719         uint64_t vs = zv->zv_volsize;
1720         uint8_t bs = zv->zv_min_bs;
1722         mutex_exit(&zfsdev_state_lock);
1723         error = zvol_getefi((void *)arg, flag, vs, bs);
1724         return (error);
1725     }
1727     case DKIOCFLUSHWRITECACHE:
1728         dkc = (struct dk_callback *)arg;
1729         mutex_exit(&zfsdev_state_lock);
1731         ht_begin_unsafe();
1733         zil_commit(zv->zv_zilog, ZVOL_OBJ);
1734         if ((flag & FKIOCTL) && dkc != NULL && dkc->dkc_callback) {
1735             (*dkc->dkc_callback)(dkc->dkc_cookie, error);
1736             error = 0;
1737         }
1739         ht_end_unsafe();
1741         return (error);
1743     case DKIOCGETWCE:
1744     {
1745         int wce = (zv->zv_flags & ZVOL_WCE) ? 1 : 0;
1746         if (ddi_copyout(&wce, (void *)arg, sizeof (int),
1747             flag))
1748             error = SET_ERROR(EFAULT);
1749         break;
1750     }
1751     case DKIOCGSETWCE:

```

```

1752     {
1753         int wce;
1754         if (ddi_copyin((void *)arg, &wce, sizeof (int),
1755             flag)) {
1756             error = SET_ERROR(EFAULT);
1757             break;
1758         }
1759         if (wce) {
1760             zv->zv_flags |= ZVOL_WCE;
1761             mutex_exit(&zfsdev_state_lock);
1762         } else {
1763             zv->zv_flags &= ~ZVOL_WCE;
1764             mutex_exit(&zfsdev_state_lock);
1765             ht_begin_unsafe();
1766             zil_commit(zv->zv_zilog, ZVOL_OBJ);
1767             ht_end_unsafe();
1768         }
1769         return (0);
1770     }
1772     case DKIOCGGEOM:
1773     case DKIOCGVTOC:
1774         /*
1775          * commands using these (like prtvtoc) expect ENOTSUP
1776          * since we're emulating an EFI label
1777          */
1778         error = SET_ERROR(ENOTSUP);
1779         break;
1781     case DKIOCDUMPINIT:
1782         lr = rangelock_enter(&zv->zv_rangelock, 0, zv->zv_volsize,
1783             RL_WRITER);
1784         error = zvol_dumpify(zv);
1785         rangelock_exit(lr);
1786         break;
1788     case DKIOCDUMPFINI:
1789         if (!(zv->zv_flags & ZVOL_DUMPIFIED))
1790             break;
1791         lr = rangelock_enter(&zv->zv_rangelock, 0, zv->zv_volsize,
1792             RL_WRITER);
1793         error = zvol_dump_fini(zv);
1794         rangelock_exit(lr);
1795         break;
1797     case DKIOCFREE:
1798     {
1799         dkioic_free_list_t *dfl;
1800         dmu_tx_t *tx;
1802         if (!zvol_unmap_enabled)
1803             break;
1805         if (!(flag & FKIOCTL)) {
1806             error = dfl_copyin((void *)arg, &dfl, flag, KM_SLEEP);
1807             if (error != 0)
1808                 break;
1809         } else {
1810             dfl = (dkioic_free_list_t *)arg;
1811             ASSERT3U(dfl->dfl_num_exts, <=, DFL_COPYIN_MAX_EXTS);
1812             if (dfl->dfl_num_exts > DFL_COPYIN_MAX_EXTS) {
1813                 error = SET_ERROR(EINVAL);
1814                 break;
1815             }
1816         }

```

```

1818         mutex_exit(&zfsdev_state_lock);
1820     ht_begin_unsafe();
1822     for (int i = 0; i < dfl->dfl_num_exts; i++) {
1823         uint64_t start = dfl->dfl_exts[i].dfle_start,
1824             length = dfl->dfl_exts[i].dfle_length,
1825             end = start + length;
1827         /*
1828          * Apply Postel's Law to length-checking.  If they
1829          * overshoot, just blank out until the end, if there's
1830          * a need to blank out anything.
1831          */
1832         if (start >= zv->zv_volsize)
1833             continue; /* No need to do anything... */
1834         if (end > zv->zv_volsize) {
1835             end = DMU_OBJECT_END;
1836             length = end - start;
1837         }
1839         lr = rangelock_enter(&zv->zv_rangelock, start, length,
1840             RL_WRITER);
1841         tx = dmu_tx_create(zv->zv_objset);
1842         error = dmu_tx_assign(tx, TXG_WAIT);
1843         if (error != 0) {
1844             dmu_tx_abort(tx);
1845         } else {
1846             zvol_log_truncate(zv, tx, start, length,
1847                 B_TRUE);
1848             dmu_tx_commit(tx);
1849             error = dmu_free_long_range(zv->zv_objset,
1850                 ZVOL_OBJ, start, length);
1851         }
1853         rangelock_exit(lr);
1855         if (error != 0)
1856             break;
1857     }
1859     /*
1860     * If the write-cache is disabled, 'sync' property
1861     * is set to 'always', or if the caller is asking for
1862     * a synchronous free, commit this operation to the zil.
1863     * This will sync any previous uncommitted writes to the
1864     * zvol object.
1865     * Can be overridden by the zvol_unmap_sync_enabled tunable.
1866     */
1867     if ((error == 0) && zvol_unmap_sync_enabled &&
1868         (!(zv->zv_flags & ZVOL_WCE) ||
1869         (zv->zv_objset->os_sync == ZFS_SYNC_ALWAYS) ||
1870         (dfl->dfl_flags & DF_WAIT_SYNC))) {
1871         zil_commit(zv->zv_zilog, ZVOL_OBJ);
1872     }
1874     if (!(flag & FKIOCTL))
1875         dfl_free(dfl);
1877     ht_end_unsafe();
1879     return (error);
1880 }
1882 default:
1883     error = SET_ERROR(ENOTTY);

```

```

1884         break;
1886     }
1887     mutex_exit(&zfsdev_state_lock);
1888     return (error);
1889 }

```

unchanged_portion_omitted

```

*****
95359 Tue May 7 04:49:57 2019
new/usr/src/uts/common/os/cpu.c
OS-7125 Need mitigation of LITF (CVE-2018-3646)
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
*****
unchanged_portion_omitted_

388 /*
389  * Set affinity for a specified CPU.
390  *
391  * Specifying a cpu_id of CPU_CURRENT, allowed_only when setting affinity for
392  * curthread, will set affinity to the CPU on which the thread is currently
393  * running. For other cpu_id values, the caller must ensure that the
394  * referenced CPU remains valid, which can be done by holding cpu_lock across
395  * this call.
396  *
397  * CPU affinity is guaranteed after return of thread_affinity_set(). If a
398  * caller setting affinity to CPU_CURRENT requires that its thread not migrate
399  * CPUs prior to a successful return, it should take extra precautions (such as
400  * their own call to kpreempt_disable) to ensure that safety.
401  *
402  * CPU_BEST can be used to pick a "best" CPU to migrate to, including
403  * potentially the current CPU.
404  *
405  * A CPU affinity reference count is maintained by thread_affinity_set and
406  * thread_affinity_clear (incrementing and decrementing it, respectively),
407  * maintaining CPU affinity while the count is non-zero, and allowing regions
408  * of code which require affinity to be nested.
409  */
410 void
411 thread_affinity_set(kthread_id_t t, int cpu_id)
412 {
413     cpu_t *cp;

415     ASSERT(!(t == curthread && t->t_weakbound_cpu != NULL));

417     if (cpu_id == CPU_CURRENT) {
418         VERIFY3P(t, ==, curthread);
419         kpreempt_disable();
420         cp = CPU;
421     } else if (cpu_id == CPU_BEST) {
422         VERIFY3P(t, ==, curthread);
423         kpreempt_disable();
424         cp = disp_choose_best_cpu();
425     } else {
426         /*
427          * We should be asserting that cpu_lock is held here, but
428          * the NCA code doesn't acquire it. The following assert
429          * should be uncommented when the NCA code is fixed.
430          *
431          * ASSERT(MUTEX_HELD(&cpu_lock));
432          */
433         VERIFY((cpu_id >= 0) && (cpu_id < NCPU));
434         cp = cpu[cpu_id];

436         /* user must provide a good cpu_id */
437         VERIFY(cp != NULL);
438     }

440     /*
441      * If there is already a hard affinity requested, and this affinity
442      * conflicts with that, panic.
443      */
444     thread_lock(t);

```

```

445     if (t->t_affinitycnt > 0 && t->t_bound_cpu != cp) {
446         panic("affinity_set: setting %p but already bound to %p",
447             (void *)cp, (void *)t->t_bound_cpu);
448     }
449     t->t_affinitycnt++;
450     t->t_bound_cpu = cp;

452     /*
453      * Make sure we're running on the right CPU.
454      */
455     if (cp != t->t_cpu || t != curthread) {
456         ASSERT(cpu_id != CPU_CURRENT);
457         force_thread_migrate(t); /* drops thread lock */
458     } else {
459         thread_unlock(t);
460     }

462     if (cpu_id == CPU_CURRENT || cpu_id == CPU_BEST)
463         if (cpu_id == CPU_CURRENT) {
464             kpreempt_enable();
465         }

unchanged_portion_omitted_

1261 /*
1262  * Take the indicated CPU offline.
1263  */
1264 int
1265 cpu_offline(cpu_t *cp, int flags)
1266 {
1267     cpupart_t *pp;
1268     int error = 0;
1269     cpu_t *ncp;
1270     int intr_enable;
1271     int cyclic_off = 0;
1272     int callout_off = 0;
1273     int loop_count;
1274     int no_quiesce = 0;
1275     int (*bound_func)(struct cpu *, int);
1276     kthread_t *t;
1277     lpl_t *cpu_lpl;
1278     proc_t *p;
1279     int lgrp_diff_lpl;
1280     boolean_t unbind_all_threads = (flags & CPU_FORCED) != 0;

1282     ASSERT(MUTEX_HELD(&cpu_lock));

1284     /*
1285      * If we're going from faulted or spare to offline, just
1286      * clear these flags and update CPU state.
1287      */
1288     if (cp->cpu_flags & (CPU_FAULTED | CPU_SPARE)) {
1289         if (cp->cpu_flags & CPU_FAULTED) {
1290             cp->cpu_flags &= ~CPU_FAULTED;
1291             mp_cpu_faulted_exit(cp);
1292         }
1293         cp->cpu_flags &= ~CPU_SPARE;
1294         cpu_set_state(cp);
1295         return (0);
1296     }

1298     /*
1299      * Handle off-line request.
1300      */
1301     pp = cp->cpu_part;
1302     /*

```

```

1303     * Don't offline last online CPU in partition
1304     */
1305     if (ncpus_online <= 1 || pp->cp_ncpus <= 1 || cpu_intr_count(cp) < 2)
1306         return (EBUSY);
1307     /*
1308     * Unbind all soft-bound threads bound to our CPU and hard bound threads
1309     * if we were asked to.
1310     */
1311     error = cpu_unbind(cp->cpu_id, unbind_all_threads);
1312     if (error != 0)
1313         return (error);
1314     /*
1315     * We shouldn't be bound to this CPU ourselves.
1316     */
1317     if (curthread->t_bound_cpu == cp)
1318         return (EBUSY);
1319
1320     /*
1321     * Tell interested parties that this CPU is going offline.
1322     */
1323     CPU_NEW_GENERATION(cp);
1324     cpu_state_change_notify(cp->cpu_id, CPU_OFF);
1325
1326     /*
1327     * Tell the PG subsystem that the CPU is leaving the partition
1328     */
1329     pg_cpupart_out(cp, pp);
1330
1331     /*
1332     * Take the CPU out of interrupt participation so we won't find
1333     * bound kernel threads. If the architecture cannot completely
1334     * shut off interrupts on the CPU, don't quiesce it, but don't
1335     * run anything but interrupt thread... this is indicated by
1336     * the CPU_OFFLINE flag being on but the CPU_QUIESCE flag being
1337     * off.
1338     */
1339     intr_enable = cp->cpu_flags & CPU_ENABLE;
1340     if (intr_enable)
1341         no_quiesce = cpu_intr_disable(cp);
1342
1343     /*
1344     * Record that we are aiming to offline this cpu. This acts as
1345     * a barrier to further weak binding requests in thread_nomigrate
1346     * and also causes cpu_choose, disp_lowpri_cpu and setfrontdq to
1347     * lean away from this cpu. Further strong bindings are already
1348     * avoided since we hold cpu_lock. Since threads that are set
1349     * runnable around now and others coming off the target cpu are
1350     * directed away from the target, existing strong and weak bindings
1351     * (especially the latter) to the target cpu stand maximum chance of
1352     * being able to unbind during the short delay loop below (if other
1353     * unbound threads compete they may not see cpu in time to unbind
1354     * even if they would do so immediately.
1355     */
1356     cpu_inmotion = cp;
1357     membar_enter();
1358
1359     /*
1360     * Check for kernel threads (strong or weak) bound to that CPU.
1361     * Strongly bound threads may not unbind, and we'll have to return
1362     * EBUSY. Weakly bound threads should always disappear - we've
1363     * stopped more weak binding with cpu_inmotion and existing
1364     * bindings will drain imminently (they may not block). Nonetheless
1365     * we will wait for a fixed period for all bound threads to disappear.
1366     * Inactive interrupt threads are OK (they'll be in TS_FREE
1367     * state). If test finds some bound threads, wait a few ticks
1368     * to give short-lived threads (such as interrupts) chance to

```

```

1369     * complete. Note that if no_quiesce is set, i.e. this cpu
1370     * is required to service interrupts, then we take the route
1371     * that permits interrupt threads to be active (or bypassed).
1372     */
1373     bound_func = no_quiesce ? disp_bound_threads : disp_bound_anythreads;
1374
1375     again: for (loop_count = 0; (*bound_func)(cp, 0); loop_count++) {
1376         if (loop_count >= 5) {
1377             error = EBUSY; /* some threads still bound */
1378             break;
1379         }
1380
1381         /*
1382         * If some threads were assigned, give them
1383         * a chance to complete or move.
1384         */
1385         * This assumes that the clock_thread is not bound
1386         * to any CPU, because the clock_thread is needed to
1387         * do the delay(hz/100).
1388         */
1389         * Note: we still hold the cpu_lock while waiting for
1390         * the next clock tick. This is OK since it isn't
1391         * needed for anything else except processor_bind(2),
1392         * and system initialization. If we drop the lock,
1393         * we would risk another p_online disabling the last
1394         * processor.
1395         */
1396         delay(hz/100);
1397     }
1398
1399     if (error == 0 && callout_off == 0) {
1400         callout_cpu_offline(cp);
1401         callout_off = 1;
1402     }
1403
1404     if (error == 0 && cyclic_off == 0) {
1405         if (!cyclic_offline(cp)) {
1406             /*
1407             * We must have bound cyclics...
1408             */
1409             error = EBUSY;
1410             goto out;
1411         }
1412         cyclic_off = 1;
1413     }
1414
1415     /*
1416     * Call mp_cpu_stop() to perform any special operations
1417     * needed for this machine architecture to offline a CPU.
1418     */
1419     if (error == 0)
1420         error = mp_cpu_stop(cp); /* arch-dep hook */
1421
1422     /*
1423     * If that all worked, take the CPU offline and decrement
1424     * ncpus_online.
1425     */
1426     if (error == 0) {
1427         /*
1428         * Put all the cpus into a known safe place.
1429         * No mutexes can be entered while CPUs are paused.
1430         */
1431         pause_cpus(cp, NULL);
1432         /*
1433         * Repeat the operation, if necessary, to make sure that
1434         * all outstanding low-level interrupts run to completion

```



```

1435     * before we set the CPU QUIESCED flag. It's also possible
1436     * that a thread has weak bound to the cpu despite our raising
1437     * cpu_inmotion above since it may have loaded that
1438     * value before the barrier became visible (this would have
1439     * to be the thread that was on the target cpu at the time
1440     * we raised the barrier).
1441     */
1442     if ((!no_quiesce && cp->cpu_intr_actv != 0) ||
1443         (*bound_func)(cp, 1)) {
1444         start_cpus();
1445         (void) mp_cpu_start(cp);
1446         goto again;
1447     }
1448     ncp = cp->cpu_next_part;
1449     cpu_lpl = cp->cpu_lpl;
1450     ASSERT(cpu_lpl != NULL);
1451
1452     /*
1453     * Remove the CPU from the list of active CPUs.
1454     */
1455     cpu_remove_active(cp);
1456
1457     /*
1458     * Walk the active process list and look for threads
1459     * whose home lgroup needs to be updated, or
1460     * the last CPU they run on is the one being offlined now.
1461     */
1462
1463     ASSERT(curthread->t_cpu != cp);
1464     for (p = practive; p != NULL; p = p->p_next) {
1465
1466         t = p->p_tlist;
1467
1468         if (t == NULL)
1469             continue;
1470
1471         lgrp_diff_lpl = 0;
1472
1473         do {
1474             ASSERT(t->t_lpl != NULL);
1475             /*
1476             * Taking last CPU in lpl offline
1477             * Rehome thread if it is in this lpl
1478             * Otherwise, update the count of how many
1479             * threads are in this CPU's lgroup but have
1480             * a different lpl.
1481             */
1482             if (cpu_lpl->lpl_ncpu == 0) {
1483                 if (t->t_lpl == cpu_lpl)
1484                     lgrp_move_thread(t,
1485                                     lgrp_choose(t,
1486                                                 t->t_cpupart), 0);
1487                 else if (t->t_lpl->lpl_lgrp_id ==
1488                        cpu_lpl->lpl_lgrp_id)
1489                     lgrp_diff_lpl++;
1490             }
1491             ASSERT(t->t_lpl->lpl_ncpu > 0);
1492
1493             /*
1494             * Update CPU last ran on if it was this CPU
1495             */
1496             if (t->t_cpu == cp && t->t_bound_cpu != cp)
1497                 t->t_cpu = disp_lowpri_cpu(ncp, t,
1498                                           t->t_pri);
1499             t->t_cpu = disp_lowpri_cpu(ncp,
1500                                     t->t_cpu,

```

```

1493         t->t_lpl, t->t_pri, NULL);
1500         ASSERT(t->t_cpu != cp || t->t_bound_cpu == cp ||
1501             t->t_weakbound_cpu == cp);
1502
1503         t = t->t_forw;
1504     } while (t != p->p_tlist);
1505
1506     /*
1507     * Didn't find any threads in the same lgroup as this
1508     * CPU with a different lpl, so remove the lgroup from
1509     * the process lgroup bitmask.
1510     */
1511     if (lgrp_diff_lpl == 0)
1512         klgrpset_del(p->p_lgrpset, cpu_lpl->lpl_lgrp_id);
1513     }
1514
1515     /*
1516     * Walk thread list looking for threads that need to be
1517     * rehomed, since there are some threads that are not in
1518     * their process's p_tlist.
1519     */
1520
1521     t = curthread;
1522     do {
1523         ASSERT(t != NULL && t->t_lpl != NULL);
1524
1525         /*
1526         * Rehome threads with same lpl as this CPU when this
1527         * is the last CPU in the lpl.
1528         */
1529         if ((cpu_lpl->lpl_ncpu == 0) && (t->t_lpl == cpu_lpl))
1530             lgrp_move_thread(t,
1531                             lgrp_choose(t, t->t_cpupart), 1);
1532
1533         ASSERT(t->t_lpl->lpl_ncpu > 0);
1534
1535         /*
1536         * Update CPU last ran on if it was this CPU
1537         */
1538         if (t->t_cpu == cp && t->t_bound_cpu != cp)
1539             t->t_cpu = disp_lowpri_cpu(ncp, t, t->t_pri);
1540
1541         if (t->t_cpu == cp && t->t_bound_cpu != cp) {
1542             t->t_cpu = disp_lowpri_cpu(ncp,
1543                                     t->t_cpu,
1544                                     t->t_pri, NULL);
1545         }
1546         ASSERT(t->t_cpu != cp || t->t_bound_cpu == cp ||
1547             t->t_weakbound_cpu == cp);
1548         t = t->t_next;
1549     } while (t != curthread);
1550     ASSERT((cp->cpu_flags & (CPU_FAULTED | CPU_SPARE)) == 0);
1551     cp->cpu_flags |= CPU_OFFLINE;
1552     disp_cpu_inactive(cp);
1553     if (!no_quiesce)
1554         cp->cpu_flags |= CPU QUIESCED;
1555     ncpus_online--;
1556     cpu_set_state(cp);
1557     cpu_inmotion = NULL;
1558     start_cpus();
1559     cpu_stats_kstat_destroy(cp);
1560     cpu_delete_intrstat(cp);
1561     lgrp_kstat_destroy(cp);

```

```
1561     }
1563 out:
1564     cpu_inmotion = NULL;
1566     /*
1567     * If we failed, re-enable interrupts.
1568     * Do this even if cpu_intr_disable returned an error, because
1569     * it may have partially disabled interrupts.
1570     */
1571     if (error && intr_enable)
1572         cpu_intr_enable(cp);
1574     /*
1575     * If we failed, but managed to offline the cyclic subsystem on this
1576     * CPU, bring it back online.
1577     */
1578     if (error && cyclic_off)
1579         cyclic_online(cp);
1581     /*
1582     * If we failed, but managed to offline callouts on this CPU,
1583     * bring it back online.
1584     */
1585     if (error && callout_off)
1586         callout_cpu_online(cp);
1588     /*
1589     * If we failed, tell the PG subsystem that the CPU is back
1590     */
1591     pg_cpupart_in(cp, pp);
1593     /*
1594     * If we failed, we need to notify everyone that this CPU is back on.
1595     */
1596     if (error != 0) {
1597         CPU_NEW_GENERATION(cp);
1598         cpu_state_change_notify(cp->cpu_id, CPU_ON);
1599         cpu_state_change_notify(cp->cpu_id, CPU_INTR_ON);
1600     }
1602     return (error);
1603 }
_____unchanged_portion_omitted_____
```

```

*****
119462 Tue May 7 04:49:57 2019
new/usr/src/uts/common/os/lgrp.c
OS-7125 Need mitigation of L1TF (CVE-2018-3646)
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
*****
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 2009 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 * Copyright 2018 Joyent, Inc.
25 */

27 /*
28 * Basic NUMA support in terms of locality groups
29 *
30 * Solaris needs to know which CPUs, memory, etc. are near each other to
31 * provide good performance on NUMA machines by optimizing for locality.
32 * In order to do this, a new abstraction called a "locality group (lgroup)"
33 * has been introduced to keep track of which CPU-like and memory-like hardware
34 * resources are close to each other. Currently, latency is the only measure
35 * used to determine how to group hardware resources into lgroups, but this
36 * does not limit the groupings to be based solely on latency. Other factors
37 * may be used to determine the groupings in the future.
38 *
39 * Lgroups are organized into a hierarchy or topology that represents the
40 * latency topology of the machine. There is always at least a root lgroup in
41 * the system. It represents all the hardware resources in the machine at a
42 * latency big enough that any hardware resource can at least access any other
43 * hardware resource within that latency. A Uniform Memory Access (UMA)
44 * machine is represented with one lgroup (the root). In contrast, a NUMA
45 * machine is represented at least by the root lgroup and some number of leaf
46 * lgroups where the leaf lgroups contain the hardware resources within the
47 * least latency of each other and the root lgroup still contains all the
48 * resources in the machine. Some number of intermediate lgroups may exist
49 * which represent more levels of locality than just the local latency of the
50 * leaf lgroups and the system latency of the root lgroup. Non-leaf lgroups
51 * (eg. root and intermediate lgroups) contain the next nearest resources to
52 * its children lgroups. Thus, the lgroup hierarchy from a given leaf lgroup
53 * to the root lgroup shows the hardware resources from closest to farthest
54 * from the leaf lgroup such that each successive ancestor lgroup contains
55 * the next nearest resources at the next level of locality from the previous.
56 *
57 * The kernel uses the lgroup abstraction to know how to allocate resources
58 * near a given process/thread. At fork() and lwp/thread_create() time, a
59 * "home" lgroup is chosen for a thread. This is done by picking the lgroup

```

```

60 * with the lowest load average. Binding to a processor or processor set will
61 * change the home lgroup for a thread. The scheduler has been modified to try
62 * to dispatch a thread on a CPU in its home lgroup. Physical memory
63 * allocation is lgroup aware too, so memory will be allocated from the current
64 * thread's home lgroup if possible. If the desired resources are not
65 * available, the kernel traverses the lgroup hierarchy going to the parent
66 * lgroup to find resources at the next level of locality until it reaches the
67 * root lgroup.
68 */

70 #include <sys/lgrp.h>
71 #include <sys/lgrp_user.h>
72 #include <sys/types.h>
73 #include <sys/mman.h>
74 #include <sys/param.h>
75 #include <sys/var.h>
76 #include <sys/thread.h>
77 #include <sys/cpuvar.h>
78 #include <sys/cpupart.h>
79 #include <sys/kmem.h>
80 #include <vm/seg.h>
81 #include <vm/seg_kmem.h>
82 #include <vm/seg_spt.h>
83 #include <vm/seg_vn.h>
84 #include <vm/as.h>
85 #include <sys/atomic.h>
86 #include <sys/system.h>
87 #include <sys/errno.h>
88 #include <sys/cmn_err.h>
89 #include <sys/kstat.h>
90 #include <sys/sysmacros.h>
91 #include <sys/pg.h>
92 #include <sys/promif.h>
93 #include <sys/sdt.h>
94 #include <sys/ht.h>

96 lgrp_gen_t lgrp_gen = 0; /* generation of lgroup hierarchy */
97 lgrp_t *lgrp_table[NLGRPS_MAX]; /* table of all initialized lgrp_t structs */
98 /* indexed by lgrp_id */
99 int nlgrps; /* number of lgroups in machine */
100 int lgrp_alloc_hint = -1; /* hint for where to try to allocate next */
101 int lgrp_alloc_max = 0; /* max lgroup ID allocated so far */

103 /*
104 * Kstat data for lgroups.
105 *
106 * Actual kstat data is collected in lgrp_stats array.
107 * The lgrp_kstat_data array of named kstats is used to extract data from
108 * lgrp_stats and present it to kstat framework. It is protected from parallel
109 * modifications by lgrp_kstat_mutex. This may cause some contention when
110 * several kstat commands run in parallel but this is not the
111 * performance-critical path.
112 */
113 extern struct lgrp_stats lgrp_stats[]; /* table of per-lgrp stats */

115 /*
116 * Declare kstat names statically for enums as defined in the header file.
117 */
118 LGRP_KSTAT_NAMES;

120 static void lgrp_kstat_init(void);
121 static int lgrp_kstat_extract(kstat_t *, int);
122 static void lgrp_kstat_reset(lgrp_id_t);

124 static struct kstat_named lgrp_kstat_data[LGRP_NUM_STATS];
125 static kmutex_t lgrp_kstat_mutex;

```

```

128 /*
129 * max number of lgroups supported by the platform
130 */
131 int      nlgrpsmax = 0;

133 /*
134 * The root lgroup. Represents the set of resources at the system wide
135 * level of locality.
136 */
137 lgrp_t      *lgrp_root = NULL;

139 /*
140 * During system bootstrap cp_default does not contain the list of lgrp load
141 * averages (cp_lgrploads). The list is allocated after the first CPU is brought
142 * on-line when cp_default is initialized by cpupart_initialize_default().
143 * Configuring CPU0 may create a two-level topology with root and one leaf node
144 * containing CPU0. This topology is initially constructed in a special
145 * statically allocated 2-element lpl list lpl_bootstrap_list and later cloned
146 * to cp_default when cp_default is initialized. The lpl_bootstrap_list is used
147 * for all lpl operations until cp_default is fully constructed.
148 *
149 * The lpl_bootstrap_list is maintained by the code in lgrp.c. Every other
150 * consumer who needs default lpl should use lpl_bootstrap which is a pointer to
151 * the first element of lpl_bootstrap_list.
152 *
153 * CPUs that are added to the system, but have not yet been assigned to an
154 * lgrp will use lpl_bootstrap as a default lpl. This is necessary because
155 * on some architectures (x86) it's possible for the slave CPU startup thread
156 * to enter the dispatcher or allocate memory before calling lgrp_cpu_init().
157 */
158 #define LPL_BOOTSTRAP_SIZE 2
159 static lpl_t      lpl_bootstrap_list[LPL_BOOTSTRAP_SIZE];
160 lpl_t      *lpl_bootstrap;
161 static lpl_t      *lpl_bootstrap_rset[LPL_BOOTSTRAP_SIZE];
162 static int      lpl_bootstrap_id2rset[LPL_BOOTSTRAP_SIZE];

164 /*
165 * If cp still references the bootstrap lpl, it has not yet been added to
166 * an lgrp. lgrp_mem_choose() uses this macro to detect the case where
167 * a thread is trying to allocate memory close to a CPU that has no lgrp.
168 */
169 #define LGRP_CPU_HAS_NO_LGRP(cp)      ((cp)->cpu_lpl == lpl_bootstrap)

171 static lgrp_t      lroot;

173 /*
174 * Size, in bytes, beyond which random memory allocation policy is applied
175 * to non-shared memory. Default is the maximum size, so random memory
176 * allocation won't be used for non-shared memory by default.
177 */
178 size_t      lgrp_privm_random_thresh = (size_t)(-1);

180 /* the maximum effect that a single thread can have on it's lgroup's load */
181 #define LGRP_LOADAVG_MAX_EFFECT(ncpu) \
182      ((lgrp_loadavg_max_effect) / (ncpu))
183 uint32_t      lgrp_loadavg_max_effect = LGRP_LOADAVG_THREAD_MAX;

186 /*
187 * Size, in bytes, beyond which random memory allocation policy is applied to
188 * shared memory. Default is 8MB (2 ISM pages).
189 */
190 size_t      lgrp_shm_random_thresh = 8*1024*1024;

```

```

192 /*
193 * Whether to do processor set aware memory allocation by default
194 */
195 int      lgrp_mem_pset_aware = 0;

197 /*
198 * Set the default memory allocation policy for root lgroup
199 */
200 lgrp_mem_policy_t      lgrp_mem_policy_root = LGRP_MEM_POLICY_RANDOM;

202 /*
203 * Set the default memory allocation policy. For most platforms,
204 * next touch is sufficient, but some platforms may wish to override
205 * this.
206 */
207 lgrp_mem_policy_t      lgrp_mem_default_policy = LGRP_MEM_POLICY_NEXT;

210 /*
211 * lgroup CPU event handlers
212 */
213 static void      lgrp_cpu_init(struct cpu *);
214 static void      lgrp_cpu_fini(struct cpu *, lgrp_id_t);
215 static lgrp_t      *lgrp_cpu_to_lgrp(struct cpu *);

217 /*
218 * lgroup memory event handlers
219 */
220 static void      lgrp_mem_init(int, lgrp_handle_t, boolean_t);
221 static void      lgrp_mem_fini(int, lgrp_handle_t, boolean_t);
222 static void      lgrp_mem_rename(int, lgrp_handle_t, lgrp_handle_t);

224 /*
225 * lgroup CPU partition event handlers
226 */
227 static void      lgrp_part_add_cpu(struct cpu *, lgrp_id_t);
228 static void      lgrp_part_del_cpu(struct cpu *);

230 /*
231 * lgroup framework initialization
232 */
233 static void      lgrp_main_init(void);
234 static void      lgrp_main_mp_init(void);
235 static void      lgrp_root_init(void);
236 static void      lgrp_setup(void);

238 /*
239 * lpl topology
240 */
241 static void      lpl_init(lpl_t *, lpl_t *, lgrp_t *);
242 static void      lpl_clear(lpl_t *);
243 static void      lpl_leaf_insert(lpl_t *, struct cpupart *);
244 static void      lpl_leaf_remove(lpl_t *, struct cpupart *);
245 static void      lpl_rset_add(lpl_t *, lpl_t *);
246 static void      lpl_rset_del(lpl_t *, lpl_t *);
247 static int      lpl_rset_contains(lpl_t *, lpl_t *);
248 static void      lpl_cpu_adjcnt(lpl_act_t, struct cpu *);
249 static void      lpl_child_update(lpl_t *, struct cpupart *);
250 static int      lpl_pick(lpl_t *, lpl_t *);
251 static void      lpl_verify_wrapper(struct cpupart *);

253 /*
254 * defines for lpl topology verifier return codes
255 */

257 #define LPL_TOPO_CORRECT      0

```

```
258 #define LPL_TOPO_PART_HAS_NO_LPL          -1
259 #define LPL_TOPO_CPUS_NOT_EMPTY          -2
260 #define LPL_TOPO_LGRP_MISMATCH           -3
261 #define LPL_TOPO_MISSING_PARENT         -4
262 #define LPL_TOPO_PARENT_MISMATCH        -5
263 #define LPL_TOPO_BAD_CPUCNT             -6
264 #define LPL_TOPO_RSET_MISMATCH          -7
265 #define LPL_TOPO_LPL_ORPHANED           -8
266 #define LPL_TOPO_LPL_BAD_NCPU           -9
267 #define LPL_TOPO_RSET_MSSNG_LF          -10
268 #define LPL_TOPO_CPU_HAS_BAD_LPL        -11
269 #define LPL_TOPO_NONLEAF_HAS_CPUS       -12
270 #define LPL_TOPO_LGRP_NOT_LEAF          -13
271 #define LPL_TOPO_BAD_RSETCNT            -14

273 /*
274  * Return whether lgroup optimizations should be enabled on this system
275  */
276 int
277 lgrp_optimizations(void)
278 {
279     /*
280      * System must have more than 2 lgroups to enable lgroup optimizations
281      *
282      * XXX This assumes that a 2 lgroup system has an empty root lgroup
283      * with one child lgroup containing all the resources. A 2 lgroup
284      * system with a root lgroup directly containing CPUs or memory might
285      * need lgroup optimizations with its child lgroup, but there
286      * isn't such a machine for now....
287      */
288     if (nlgrps > 2)
289         return (1);
291     return (0);
292 }
293 unchanged_portion_omitted

516 /*
517  * Finish lgrp initialization after all CPUS are brought on-line.
518  * This routine is called after start_other_cpus().
519  */
520 static void
521 lgrp_main_mp_init(void)
522 {
523     klggrpset_t changed;

525     ht_init();

527     /*
528      * Update lgroup topology (if necessary)
529      */
530     klggrpset_clear(changed);
531     (void) lgrp_topo_update(lgrp_table, lgrp_alloc_max + 1, &changed);
532     lgrp_topo_initialized = 1;
533 }
534 unchanged_portion_omitted
```

```

*****
30390 Tue May 7 04:49:57 2019
new/usr/src/uts/common/sys/cpuvar.h
OS-7125 Need mitigation of L1TF (CVE-2018-3646)
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
*****
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) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
24  * Copyright (c) 2012 by Delphix. All rights reserved.
25  * Copyright 2014 Igor Kozhukhov <ikozhukhov@gmail.com>.
26  * Copyright 2018 Joyent, Inc.
27  * Copyright 2017 RackTop Systems.
28  */

30 #ifndef _SYS_CPUVAR_H
31 #define _SYS_CPUVAR_H

33 #include <sys/thread.h>
34 #include <sys/sysinfo.h> /* has cpu_stat_t definition */
35 #include <sys/disp.h>
36 #include <sys/processor.h>
37 #include <sys/kcpc.h> /* has kcpc_ctx_t definition */

39 #include <sys/loadavg.h>
40 #if defined(_KERNEL) || defined(_KMEMUSER) && defined(_MACHDEP)
41 #include <sys/machcpuvar.h>
42 #endif

44 #include <sys/types.h>
45 #include <sys/file.h>
46 #include <sys/bitmap.h>
47 #include <sys/rwlock.h>
48 #include <sys/msacct.h>
49 #if defined(__GNUC__) && defined(_ASM_INLINES) && defined(_KERNEL) && \
50     (defined(__i386) || defined(__amd64))
51 #include <asm/cpuvar.h>
52 #endif

54 #ifdef __cplusplus
55 extern "C" {
56 #endif

58 struct queue_set_s;

```

```

60 #define CPU_CACHE_COHERENCE_SIZE 64

62 /*
63  * For fast event tracing.
64  */
65 struct ftrace_record;
66 typedef struct ftrace_data {
67     int ftd_state; /* ftrace flags */
68     kmutex_t ftd_unused; /* ftrace buffer lock, unused */
69     struct ftrace_record *ftd_cur; /* current record */
70     struct ftrace_record *ftd_first; /* first record */
71     struct ftrace_record *ftd_last; /* last record */
72 } ftrace_data_t;
unchanged_portion_omitted

526 #define CPuset_ATOMIC_DEL(set, cpu) atomic_and_ulong(&(set), ~CPuset(cpu))
527 #define CPuset_ATOMIC_ADD(set, cpu) atomic_or_ulong(&(set), CPuset(cpu))

529 #define CPuset_ATOMIC_XADD(set, cpu, result) \
530     { result = atomic_set_long_excl(&(set), (cpu)); }

532 #define CPuset_ATOMIC_XDEL(set, cpu, result) \
533     { result = atomic_clear_long_excl(&(set), (cpu)); }

535 #else /* CPuset_WORDS <= 0 */

537 #error NCPuset is undefined or invalid

539 #endif /* CPuset_WORDS */

541 extern cpuset_t cpuset_inuse;

543 #endif /* _MACHDEP */
544 #endif /* _KERNEL || _KMEMUSER || _BOOT */

546 #define CPU_CPR_OFFLINE 0x0
547 #define CPU_CPR_ONLINE 0x1
548 #define CPU_CPR_IS_OFFLINE(cpu) ((cpu)->cpu_cpr_flags & CPU_CPR_ONLINE) == 0
549 #define CPU_CPR_IS_ONLINE(cpu) ((cpu)->cpu_cpr_flags & CPU_CPR_ONLINE)
550 #define CPU_SET_CPR_FLAGS(cpu, flag) ((cpu)->cpu_cpr_flags |= flag)

552 #if defined(_KERNEL) || defined(_KMEMUSER)

554 extern struct cpu *cpu[]; /* indexed by CPU number */
555 extern struct cpu **cpu_seq; /* indexed by sequential CPU id */
556 extern cpu_t *cpu_list; /* list of CPUs */
557 extern cpu_t *cpu_active; /* list of active CPUs */
558 extern int ncpus; /* number of CPUs present */
559 extern int ncpus_online; /* number of CPUs not quiesced */
560 extern int max_ncpus; /* max present before ncpus is known */
561 extern int boot_max_ncpus; /* like max_ncpus but for real */
562 extern int boot_ncpus; /* # cpus present @ boot */
563 extern processorid_t max_cpuid; /* maximum CPU number */
564 extern struct cpu *cpu_inmotion; /* offline or partition move target */
565 extern cpu_t *clock_cpu_list;
566 extern processorid_t max_cpu_seqid_ever; /* maximum seqid ever given */

568 #if defined(__i386) || defined(__amd64)
569 extern struct cpu *curcpup(void);
570 #define CPU (curcpup()) /* Pointer to current CPU */
571 #else
572 #define CPU (curthread->t_cpu) /* Pointer to current CPU */
573 #endif

575 /*
576  * CPU_CURRENT indicates to thread_affinity_set() to use whatever curthread's

```

```
577 * current CPU is; holding cpu_lock is not required.
575 * CPU_CURRENT indicates to thread_affinity_set to use CPU->cpu_id
576 * as the target and to grab cpu_lock instead of requiring the caller
577 * to grab it.
578 */
579 #define CPU_CURRENT      -3

581 /*
582 * CPU_BEST can be used by thread_affinity_set() callers to set affinity to a
583 * good CPU (in particular, an ht_acquire()-friendly choice); holding cpu_lock
584 * is not required.
585 */
586 #define CPU_BEST        -4

588 /*
589 * Per-CPU statistics
590 *
591 * cpu_stats_t contains numerous system and VM-related statistics, in the form
592 * of gauges or monotonically-increasing event occurrence counts.
593 */

595 #define CPU_STATS_ENTER_K()      kpreempt_disable()
596 #define CPU_STATS_EXIT_K()      kpreempt_enable()

598 #define CPU_STATS_ADD_K(class, stat, amount) \
599     { kpreempt_disable(); /* keep from switching CPUs */ \
600       CPU_STATS_ADDQ(CPU, class, stat, amount); \
601       kpreempt_enable(); \
602     }

604 #define CPU_STATS_ADDQ(cp, class, stat, amount) { \
605     extern void __dtrace_probe__cpu_##class##info_##stat(uint_t, \
606     uint64_t *, cpu_t *); \
607     uint64_t *stataddr = &((cp)->cpu_stats.class.stat); \
608     __dtrace_probe__cpu_##class##info_##stat((amount), \
609     stataddr, cp); \
610     *(stataddr) += (amount); \
611 }

_____unchanged_portion_omitted_
```

```

*****
6027 Tue May 7 04:49:58 2019
new/usr/src/uts/common/sys/disp.h
OS-7125 Need mitigation of LITF (CVE-2018-3646)
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
*****
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 2007 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 *
25 * Copyright 2013 Nexenta Systems, Inc. All rights reserved.
26 *
27 * Copyright 2018 Joyent, Inc.
28 */

30 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
31 /*      All Rights Reserved      */

34 #ifndef _SYS_DISP_H
35 #define _SYS_DISP_H

37 #include <sys/priocntl.h>
38 #include <sys/thread.h>
39 #include <sys/class.h>

41 #ifdef __cplusplus
42 extern "C" {
43 #endif

45 /*
46  * The following is the format of a dispatcher queue entry.
47  */
48 typedef struct dispq {
49     kthread_t    *dq_first;    /* first thread on queue or NULL */
50     kthread_t    *dq_last;    /* last thread on queue or NULL */
51     int          dq_srunct;    /* number of loaded, runnable */
52                                     /* threads on queue */
53 } dispq_t;
unchanged portion omitted

84 #if defined(_KERNEL) || defined(_FAKE_KERNEL)

86 #define MAXCLSYSPRI    99
87 #define MINCLSYSPRI    60

```

```

90 /*
91  * Global scheduling variables.
92  * - See sys/cpuvar.h for CPU-local variables.
93  */
94 extern int      nswapped;    /* number of swapped threads */
95                                     /* nswapped protected by swap_lock */

97 extern pri_t    minclsyspri; /* minimum level of any system class */
98 extern pri_t    maxclsyspri; /* maximum level of any system class */
99 extern pri_t    intr_pri;   /* interrupt thread priority base level */

101 #endif /* _KERNEL || _FAKE_KERNEL */
102 #if defined(_KERNEL)

104 /*
105  * Minimum amount of time that a thread can remain runnable before it can
106  * be stolen by another CPU (in nanoseconds).
107  */
108 extern hrtime_t nosteal_nsec;

110 /*
111  * Kernel preemption occurs if a higher-priority thread is runnable with
112  * a priority at or above kpreemptpri.
113  *
114  * So that other processors can watch for such threads, a separate
115  * dispatch queue with unbound work above kpreemptpri is maintained.
116  * This is part of the CPU partition structure (cpupart_t).
117  */
118 extern pri_t    kpreemptpri; /* level above which preemption takes place */

120 extern void      disp_kp_alloc(disp_t *, pri_t); /* allocate kp queue */
121 extern void      disp_kp_free(disp_t *);        /* free kp queue */

123 /*
124  * Macro for use by scheduling classes to decide whether the thread is about
125  * to be scheduled or not. This returns the maximum run priority.
126  */
127 #define DISP_MAXRUNPRI(t)    ((t)->t_disp_queue->disp_maxrunpri)

129 /*
130  * Platform callbacks for various dispatcher operations
131  *
132  * idle_cpu() is invoked when a cpu goes idle, and has nothing to do.
133  * disp_eng_thread() is invoked when a thread is placed on a run queue.
134  */
135 extern void      (*idle_cpu)();
136 extern void      (*disp_eng_thread)(struct cpu *, int);

139 extern int      dispdeq(kthread_t *);
140 extern void      dispinit(void);
141 extern void      disp_add(sclass_t *);
142 extern int      intr_active(struct cpu *, int);
143 extern int      servicing_interrupt(void);
144 extern void      preempt(void);
145 extern void      setbackdq(kthread_t *);
146 extern void      setfrontdq(kthread_t *);
147 extern void      swtch(void);
148 extern void      swtch_to(kthread_t *);
149 extern void      swtch_from_zombie(void);
150                                     __NORETURN;
151 extern void      dq_sruninc(kthread_t *);
152 extern void      dq_srundec(kthread_t *);
153 extern void      cpu_rechoose(kthread_t *);

```



```
154 extern void      cpu_surrender(kthread_t *);
155 extern void      kpreempt(int);
156 extern struct cpu *disp_lowpri_cpu(struct cpu *, kthread_t *, pri_t);
154 extern struct cpu *disp_lowpri_cpu(struct cpu *, struct lgrp_ld *, pri_t,
155 struct cpu *);
157 extern int      disp_bound_threads(struct cpu *, int);
158 extern int      disp_bound_anythreads(struct cpu *, int);
159 extern int      disp_bound_partition(struct cpu *, int);
160 extern void      disp_cpu_init(struct cpu *);
161 extern void      disp_cpu_fini(struct cpu *);
162 extern void      disp_cpu_inactive(struct cpu *);
163 extern void      disp_adjust_unbound_pri(kthread_t *);
164 extern void      resume(kthread_t *);
165 extern void      resume_from_intr(kthread_t *);
166 extern void      resume_from_zombie(kthread_t *)
167 __NORETURN;
168 extern void      disp_swapped_enq(kthread_t *);
169 extern int      disp_anywork(void);

171 extern struct cpu *disp_choose_best_cpu(void);

173 #define KPREEMPT_SYNC      (-1)
174 #define kpreempt_disable() \
175 { \
176     curthread->t_preempt++; \
177     ASSERT(curthread->t_preempt >= 1); \
178 }
179 #define kpreempt_enable() \
180 { \
181     ASSERT(curthread->t_preempt >= 1); \
182     if (--curthread->t_preempt == 0 && \
183         CPU->cpu_kprunrun) \
184         kpreempt(KPREEMPT_SYNC); \
185 }

187 #endif /* _KERNEL */

189 #define CPU_IDLE_PRI (-1)

191 #ifdef __cplusplus
192 }
    unchanged_portion_omitted_
```

```

*****
27314 Tue May 7 04:49:58 2019
new/usr/src/uts/common/sys/thread.h
OS-7125 Need mitigation of LITF (CVE-2018-3646)
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
*****
unchanged_portion_omitted

100 typedef struct _kthread *kthread_id_t;

102 struct turnstile;
103 struct panic_trap_info;
104 struct upimutex;
105 struct kproject;
106 struct on_trap_data;
107 struct waitq;
108 struct _kcpc_ctx;
109 struct _kcpc_set;

111 /* Definition for kernel thread identifier type */
112 typedef uint64_t kt_did_t;

114 typedef struct _kthread {
115     struct _kthread *t_link; /* dispq, sleepq, and free queue link */

117     caddr_t t_stk; /* base of stack (kernel sp value to use) */
118     void (*t_startpc)(void); /* PC where thread started */
119     struct cpu *t_bound_cpu; /* cpu bound to, or NULL if not bound */
120     short t_affinitycnt; /* nesting level of kernel affinity-setting */
121     short t_bind_cpu; /* user-specified CPU binding (-1 if none) */
122     ushort_t t_flag; /* modified only by current thread */
123     ushort_t t_proc_flag; /* modified holding tproc(t)->p_lock */
124     ushort_t t_schedflag; /* modified holding thread_lock(t) */
125     volatile char t_preempt; /* don't preempt thread if set */
126     volatile char t_preempt_lk;
127     uint_t t_state; /* thread state (protected by thread_lock) */
128     pri_t t_pri; /* assigned thread priority */
129     pri_t t_epri; /* inherited thread priority */
130     pri_t t_cpri; /* thread scheduling class priority */
131     char t_writer; /* sleeping in lwp_rwlock_lock(RW_WRITE_LOCK) */
132     uchar_t t_bindflag; /* CPU and pset binding type */
133     label_t t_pcb; /* pcb, save area when switching */
134     lwpchan_t t_lwpchan; /* reason for blocking */
135 #define t_wchan0 t_lwpchan.lc_wchan0
136 #define t_wchan t_lwpchan.lc_wchan
137     struct _sobj_ops *t_sobj_ops;
138     id_t t_cid; /* scheduling class id */
139     struct thread_ops *t_clfuncs; /* scheduling class ops vector */
140     void *t_cldata; /* per scheduling class specific data */
141     ctxop_t *t_ctx; /* thread context */
142     uintptr_t t_lofault; /* ret pc for failed page faults */
143     label_t *t_onfault; /* on_fault() setjmp buf */
144     struct on_trap_data *t_ontrap; /* on_trap() protection data */
145     caddr_t t_swap; /* the bottom of the stack, if from segkp */
146     lock_t t_lock; /* used to resume() a thread */
147     uint8_t t_lockstat; /* set while thread is in lockstat code */
148     uint8_t t_pil; /* interrupt thread PIL */
149     disp_lock_t t_pi_lock; /* lock protecting t_prioinv list */
150     char t_nomigrate; /* do not migrate if set */
151     struct cpu *t_cpu; /* CPU that thread last ran on */
152     struct cpu *t_weakbound_cpu; /* cpu weakly bound to */
153     struct lgrp_ld *t_lpl; /* load average for home lgroup */
154     void *t_lgrp_reserv[2]; /* reserved for future */
155     struct _kthread *t_intr; /* interrupted (pinned) thread */
156     uint64_t t_intr_start; /* timestamp when time slice began */

```

```

157     kt_did_t t_did; /* thread id for kernel debuggers */
158     caddr_t t_tnf_tdp; /* Trace facility data pointer */
159     struct _kcpc_ctx *t_cpc_ctx; /* performance counter context */
160     struct _kcpc_set *t_cpc_set; /* set this thread has bound */

162     /*
163     * non swappable part of the lwp state.
164     */
165     id_t t_tid; /* lwp's id */
166     id_t t_waitfor; /* target lwp id in lwp_wait() */
167     struct sigqueue *t_sigqueue; /* queue of siginfo structs */
168     k_sigset_t t_sig; /* signals pending to this process */
169     k_sigset_t t_extsig; /* signals sent from another contract */
170     k_sigset_t t_hold; /* hold signal bit mask */
171     k_sigset_t t_sigwait; /* sigtimedwait/sigfd accepting these */
172     struct _kthread *t_forw; /* process's forward thread link */
173     struct _kthread *t_back; /* process's backward thread link */
174     struct _kthread *t_thlink; /* tid (lwpid) lookup hash link */
175     klpw_t *t_lwp; /* thread's lwp pointer */
176     struct proc *t_procp; /* proc pointer */
177     struct t_audit_data *t_audit_data; /* per thread audit data */
178     struct _kthread *t_next; /* doubly linked list of all threads */
179     struct _kthread *t_prev;
180     ushort_t t_whystop; /* reason for stopping */
181     ushort_t t_whatstop; /* more detailed reason */
182     int t_dslot; /* index in proc's thread directory */
183     struct pollstate *t_pollstate; /* state used during poll(2) */
184     struct pollcache *t_pollcache; /* to pass a pcache ptr by /dev/poll */
185     struct cred *t_cred; /* pointer to current cred */
186     time_t t_start; /* start time, seconds since epoch */
187     clock_t t_lbolt; /* lbolt at last clock_tick() */
188     hrtime_t t_stoptime; /* timestamp at stop() */
189     uint_t t_pctcpu; /* %cpu at last clock_tick(), binary */
190     /* point at right of high-order bit */
191     short t_sysnum; /* system call number */
192     kcondvar_t t_delay_cv;
193     kmutex_t t_delay_lock;

195     /*
196     * Pointer to the dispatcher lock protecting t_state and state-related
197     * flags. This pointer can change during waits on the lock, so
198     * it should be grabbed only by thread_lock().
199     */
200     disp_lock_t *t_lockp; /* pointer to the dispatcher lock */
201     ushort_t t_oldspl; /* spl level before dispatcher locked */
202     volatile char t_pre_sys; /* pre-syscall work needed */
203     lock_t t_lock_flush; /* for lock_mutex_flush() impl */
204     struct _disp *t_disp_queue; /* run queue for chosen CPU */
205     clock_t t_disp_time; /* last time this thread was running */
206     uint_t t_kpri_req; /* kernel priority required */

208     /*
209     * Post-syscall / post-trap flags.
210     * No lock is required to set these.
211     * These must be cleared only by the thread itself.
212     */
213     * t_astflag indicates that some post-trap processing is required,
214     * possibly a signal or a preemption. The thread will not
215     * return to user with this set.
216     * t_post_sys indicates that some unusually post-system call
217     * handling is required, such as an error or tracing.
218     * t_sig_check indicates that some condition in ISSIG() must be
219     * checked, but doesn't prevent returning to user.
220     * t_post_sys_ast is a way of checking whether any of these three
221     * flags are set.
222     */

```

```

223     union __tu {
224         struct __ts {
225             volatile char   _t_astflag;    /* AST requested */
226             volatile char   _t_sig_check;  /* ISSIG required */
227             volatile char   _t_post_sys;   /* post_syscall req */
228             volatile char   _t_trapret;    /* call CL_TRAPRET */
229         } _ts;
230         volatile int        _t_post_sys_ast; /* OR of these flags */
231     } _tu;
232 #define t_astflag      _tu._ts._t_astflag
233 #define t_sig_check   _tu._ts._t_sig_check
234 #define t_post_sys    _tu._ts._t_post_sys
235 #define t_trapret     _tu._ts._t_trapret
236 #define t_post_sys_ast _tu._t_post_sys_ast
237
238 /*
239  * Real time microstate profiling.
240  */
241 /* possible 4-byte filler */
242 hrtime_t t_waitrq; /* timestamp for run queue wait time */
243 int t_mstate; /* current microstate */
244 struct rprof {
245     int rp_anystate; /* set if any state non-zero */
246     uint_t rp_state[NMSTATES]; /* mstate profiling counts */
247 } *t_rprof;
248
249 /*
250  * There is a turnstile inserted into the list below for
251  * every priority inverted synchronization object that
252  * this thread holds.
253  */
254
255 struct turnstile *t_prioinv;
256
257 /*
258  * Pointer to the turnstile attached to the synchronization
259  * object where this thread is blocked.
260  */
261
262 struct turnstile *t_ts;
263
264 /*
265  * kernel thread specific data
266  * Borrowed from userland implementation of POSIX tsd
267  */
268 struct tsd_thread {
269     struct tsd_thread *ts_next; /* threads with TSD */
270     struct tsd_thread *ts_prev; /* threads with TSD */
271     uint_t ts_nkeys; /* entries in value array */
272     void **ts_value; /* array of value/key */
273 } *t_tsd;
274
275 clock_t t_stime; /* time stamp used by the swapper */
276 struct door_data *t_door; /* door invocation data */
277 kmutex_t *t_plockp; /* pointer to process's p_lock */
278
279 struct sc_shared *t_schedctl; /* scheduler activations shared data */
280 uintptr_t t_sc_uaddr; /* user-level address of shared data */
281
282 struct cpupart *t_cpupart; /* partition containing thread */
283 int t_bind_pset; /* processor set binding */
284
285 struct copyops *t_copyops; /* copy in/out ops vector */
286
287 caddr_t t_stkbase; /* base of the the stack */
288 struct page *t_red_pp; /* if non-NULL, redzone is mapped */

```

```

290     afd_t t_activefd; /* active file descriptor table */
291
292     struct _kthread *t_priforw; /* sleepq per-priority sublist */
293     struct _kthread *t_priback;
294
295     struct sleepq *t_sleepq; /* sleep queue thread is waiting on */
296     struct panic_trap_info *t_panic_trap; /* saved data from fatal trap */
297     int *t_lgrp_affinity; /* lgroup affinity */
298     struct upimutex *t_upimutex; /* list of upimutexes owned by thread */
299     uint32_t t_nupinest; /* number of nested held upi mutexes */
300     struct kproject *t_proj; /* project containing this thread */
301     uint8_t t_unpark; /* modified holding t_delay_lock */
302     uint8_t t_release; /* lwp_release() waked up the thread */
303     uint8_t t_hatdepth; /* depth of recursive hat_memloads */
304     uint8_t t_xpvcntr; /* see xen_block_migrate() */
305     kcondvar_t t_joincv; /* cv used to wait for thread exit */
306     void *t_taskq; /* for threads belonging to taskq */
307     hrtime_t t_anttime; /* most recent time anticipatory load */
308     /* was added to an lgroup's load */
309     /* on this thread's behalf */
310     char *t_pdmsg; /* privilege debugging message */
311
312     uint_t t_predcache; /* DTrace predicate cache */
313     hrtime_t t_dtrace_vtime; /* DTrace virtual time */
314     hrtime_t t_dtrace_start; /* DTrace slice start time */
315
316     uint8_t t_dtrace_stop; /* indicates a DTrace-desired stop */
317     uint8_t t_dtrace_sig; /* signal sent via DTrace's raise() */
318
319     union __tdu {
320         struct __tds {
321             uint8_t t_dtrace_on; /* hit a fasttrap tracepoint */
322             uint8_t t_dtrace_step; /* about to return to kernel */
323             uint8_t t_dtrace_ret; /* handling a return probe */
324             uint8_t t_dtrace_ast; /* saved ast flag */
325 #ifdef __amd64
326             uint8_t t_dtrace_reg; /* modified register */
327 #endif
328         } _tds;
329         ulong_t t_dtrace_ft; /* bitwise or of these flags */
330     } _tdu;
331 #define t_dtrace_ft _tdu._tds.t_dtrace_ft
332 #define t_dtrace_on _tdu._tds.t_dtrace_on
333 #define t_dtrace_step _tdu._tds.t_dtrace_step
334 #define t_dtrace_ret _tdu._tds.t_dtrace_ret
335 #define t_dtrace_ast _tdu._tds.t_dtrace_ast
336 #ifdef __amd64
337 #define t_dtrace_reg _tdu._tds.t_dtrace_reg
338 #endif
339
340     uintptr_t t_dtrace_pc; /* DTrace saved pc from fasttrap */
341     uintptr_t t_dtrace_npc; /* DTrace next pc from fasttrap */
342     uintptr_t t_dtrace_scrpc; /* DTrace per-thread scratch location */
343     uintptr_t t_dtrace_astpc; /* DTrace return sequence location */
344 #ifdef __amd64
345     uint64_t t_dtrace_regv; /* DTrace saved reg from fasttrap */
346     uint64_t t_useracc; /* SMAP state saved across swtch() */
347 #endif
348     hrtime_t t_hrtime; /* high-res last time on cpu */
349     kmutex_t t_ctx_lock; /* protects t_ctx in removectx() */
350     struct waitq *t_waitq; /* wait queue */
351     kmutex_t t_wait_mutex; /* used in CV wait functions */
352
353     char *t_name; /* thread name */

```

```

355     uint64_t      t_unsafe;    /* unsafe to run with HT VCPU thread */
356 } kthread_t;

358 /*
359 * Thread flag (t_flag) definitions.
360 * These flags must be changed only for the current thread,
361 * and not during preemption code, since the code being
362 * preempted could be modifying the flags.
363 *
364 * For the most part these flags do not need locking.
365 * The following flags will only be changed while the thread_lock is held,
366 * to give assurance that they are consistent with t_state:
367 *
368 *     T_WAKEABLE
369 #define T_INTR_THREAD 0x0001 /* thread is an interrupt thread */
370 #define T_WAKEABLE    0x0002 /* thread is blocked, signals enabled */
371 #define T_TOMASK      0x0004 /* use lwp_sigoldmask on return from signal */
372 #define T_TALLOCSK   0x0008 /* thread structure allocated from stk */
373 #define T_FORKALL     0x0010 /* thread was cloned by forkall() */
374 #define T_WOULDBLOCK 0x0020 /* for lockfs */
375 #define T_DONTBLOCK  0x0040 /* for lockfs */
376 #define T_DONTPEND   0x0080 /* for lockfs */
377 #define T_SYS_PROF   0x0100 /* profiling on for duration of system call */
378 #define T_WAITCVSEM  0x0200 /* waiting for a lwp_cv or lwp_sema on sleepq */
379 #define T_WATCHPT    0x0400 /* thread undergoing a watchpoint emulation */
380 #define T_PANIC      0x0800 /* thread initiated a system panic */
381 #define T_LWPREUSE    0x1000 /* stack and LWP can be reused */
382 #define T_CAPTURING   0x2000 /* thread is in page capture logic */
383 #define T_VFPARENT    0x4000 /* thread is vfork parent, must call vfwait */
384 #define T_DONTDTRACE 0x8000 /* disable DTrace probes */

386 /*
387 * Flags in t_proc_flag.
388 * These flags must be modified only when holding the p_lock
389 * for the associated process.
390 */
391 #define TP_DAEMON      0x0001 /* this is an LWP_DAEMON lwp */
392 #define TP_HOLDLWP    0x0002 /* hold thread's lwp */
393 #define TP_TWAIT      0x0004 /* wait to be freed by lwp_wait() */
394 #define TP_LWPEXIT    0x0008 /* lwp has exited */
395 #define TP_PRSTOP     0x0010 /* thread is being stopped via /proc */
396 #define TP_CHKPT      0x0020 /* thread is being stopped via CPR checkpoint */
397 #define TP_EXITLWP    0x0040 /* terminate this lwp */
398 #define TP_PRVSTOP    0x0080 /* thread is virtually stopped via /proc */
399 #define TP_MSACCT     0x0100 /* collect micro-state accounting information */
400 #define TP_STOPPING   0x0200 /* thread is executing stop() */
401 #define TP_WATCHPT    0x0400 /* process has watchpoints in effect */
402 #define TP_PAUSE      0x0800 /* process is being stopped via pauselwps() */
403 #define TP_CHANGEBIND 0x1000 /* thread has a new cpu/cpart binding */
404 #define TP_ZTHREAD    0x2000 /* this is a kernel thread for a zone */
405 #define TP_WATCHSTOP  0x4000 /* thread is stopping via holdwatch() */

407 /*
408 * Thread scheduler flag (t_schedflag) definitions.
409 * The thread must be locked via thread_lock() or equiv. to change these.
410 */
411 #define TS_LOAD        0x0001 /* thread is in memory */
412 #define TS_DONT_SWAP  0x0002 /* thread/lwp should not be swapped */
413 #define TS_SWAPENQ    0x0004 /* swap thread when it reaches a safe point */
414 #define TS_ON_SWAPQ   0x0008 /* thread is on the swap queue */
415 #define TS_SIGNALLED  0x0010 /* thread was awakened by cv_signal() */
416 #define TS_PROJWAITQ  0x0020 /* thread is on its project's waitq */
417 #define TS_ZONEWAITQ  0x0040 /* thread is on its zone's waitq */
418 #define TS_VCPU       0x0080 /* thread will enter guest context */
419 #define TS_CSTART     0x0100 /* setrun() by continuelwps() */
420 #define TS_UNPAUSE    0x0200 /* setrun() by unpauselwps() */

```

```

421 #define TS_XSTART      0x0400 /* setrun() by SIGCONT */
422 #define TS_PSTART     0x0800 /* setrun() by /proc */
423 #define TS_RESUME     0x1000 /* setrun() by CPR resume process */
424 #define TS_CREATE     0x2000 /* setrun() by syslwp_create() */
425 #define TS_RUNQMATCH  0x4000 /* exact run queue balancing by setbackdq() */
426 #define TS_ALLSTART   \
427     (TS_CSTART|TS_UNPAUSE|TS_XSTART|TS_PSTART|TS_RESUME|TS_CREATE)
428 #define TS_ANYWAITQ   (TS_PROJWAITQ|TS_ZONEWAITQ)

430 /*
431 * Thread binding types
432 */
433 #define TB_ALLHARD    0
434 #define TB_CPU_SOFT  0x01 /* soft binding to CPU */
435 #define TB_PSET_SOFT 0x02 /* soft binding to pset */

437 #define TB_CPU_SOFT_SET(t) ((t)->t_bindflag |= TB_CPU_SOFT)
438 #define TB_CPU_HARD_SET(t) ((t)->t_bindflag &= ~TB_CPU_SOFT)
439 #define TB_PSET_SOFT_SET(t) ((t)->t_bindflag |= TB_PSET_SOFT)
440 #define TB_PSET_HARD_SET(t) ((t)->t_bindflag &= ~TB_PSET_SOFT)
441 #define TB_CPU_IS_SOFT(t) ((t)->t_bindflag & TB_CPU_SOFT)
442 #define TB_CPU_IS_HARD(t) (!TB_CPU_IS_SOFT(t))
443 #define TB_PSET_IS_SOFT(t) ((t)->t_bindflag & TB_PSET_SOFT)

445 /*
446 * No locking needed for AST field.
447 */
448 #define aston(t) ((t)->t_astflag = 1)
449 #define astoff(t) ((t)->t_astflag = 0)

451 /* True if thread is stopped on an event of interest */
452 #define ISTOPPED(t) ((t)->t_state == TS_STOPPED && \
453     !((t)->t_schedflag & TS_PSTART))

455 /* True if thread is asleep and wakeable */
456 #define ISWAKEABLE(t) ((t)->t_state == TS_SLEEP && \
457     ((t)->t_flag & T_WAKEABLE))

459 /* True if thread is on the wait queue */
460 #define ISWAITING(t) ((t)->t_state == TS_WAIT)

462 /* similar to ISTOPPED except the event of interest is CPR */
463 #define CPR_ISTOPPED(t) ((t)->t_state == TS_STOPPED && \
464     !((t)->t_schedflag & TS_RESUME))

466 /*
467 * True if thread is virtually stopped (is or was asleep in
468 * one of the lwp_*() system calls and marked to stop by /proc.)
469 */
470 #define VSTOPPED(t) ((t)->t_proc_flag & TP_PRVSTOP)

472 /* similar to VSTOPPED except the point of interest is CPR */
473 #define CPR_VSTOPPED(t) \
474     ((t)->t_state == TS_SLEEP && \
475     (t)->t_wchan0 != NULL && \
476     ((t)->t_flag & T_WAKEABLE) && \
477     ((t)->t_proc_flag & TP_CHKPT))

479 /* True if thread has been stopped by hold*() or was created stopped */
480 #define SUSPENDED(t) ((t)->t_state == TS_STOPPED && \
481     ((t)->t_schedflag & (TS_CSTART|TS_UNPAUSE)) != (TS_CSTART|TS_UNPAUSE))

483 /* True if thread possesses an inherited priority */
484 #define INHERITED(t) ((t)->t_epri != 0)

486 /* The dispatch priority of a thread */

```

```

487 #define DISP_PRIO(t) ((t)->t_epri > (t)->t_pri ? (t)->t_epri : (t)->t_pri)
489 /* The assigned priority of a thread */
490 #define ASSIGNED_PRIO(t) ((t)->t_pri)

492 /*
493  * Macros to determine whether a thread can be swapped.
494  * If t_lock is held, the thread is either on a processor or being swapped.
495  */
496 #define SWAP_OK(t) (!LOCK_HELD(&(t)->t_lock))

498 /*
499  * proctot(x)
500  * convert a proc pointer to a thread pointer. this only works with
501  * procs that have only one lwp.
502  */
503 #define proctolwp(x)
504 /* convert a proc pointer to a lwp pointer. this only works with
505  * procs that have only one lwp.
506  */
507 #define ttolwp(x)
508 /* convert a thread pointer to its lwp pointer.
509  */
510 #define ttoproc(x)
511 /* convert a thread pointer to its proc pointer.
512  */
513 #define ttoproj(x)
514 /* convert a thread pointer to its project pointer.
515  */
516 #define ttozone(x)
517 /* convert a thread pointer to its zone pointer.
518  */
519 #define lwptot(x)
520 /* convert a lwp pointer to its thread pointer.
521  */
522 #define lwptoproc(x)
523 /* convert a lwp to its proc pointer.
524  */
525 #define proctot(x) ((x)->p_tlist)
526 #define proctolwp(x) ((x)->p_tlist->t_lwp)
527 #define ttolwp(x) ((x)->t_lwp)
528 #define ttoproc(x) ((x)->t_procp)
529 #define ttoproj(x) ((x)->t_proj)
530 #define ttozone(x) ((x)->t_procp->p_zone)
531 #define lwptot(x) ((x)->lwp_thread)
532 #define lwptoproc(x) ((x)->lwp_procp)

534 #define t_pc t_pcb.val[0]
535 #define t_sp t_pcb.val[1]

537 #ifdef _KERNEL

539 extern kthread_t *threadp(void); /* inline, returns thread pointer */
540 #define curthread (threadp()) /* current thread pointer */
541 #define curproc (ttoproc(curthread)) /* current process pointer */
542 #define curproj (ttoproj(curthread)) /* current project pointer */
543 #define curzone (curproc->p_zone) /* current zone pointer */

545 extern struct _kthread t0; /* the scheduler thread */
546 extern kmutex_t pidlock; /* global process lock */

548 /*
549  * thread_free_lock is used by the tick accounting thread to keep a thread
550  * from being freed while it is being examined.
551  */
552  * Thread structures are 32-byte aligned structures. That is why we use the

```

```

553  * following formula.
554  */
555 #define THREAD_FREE_BITS 10
556 #define THREAD_FREE_NUM (1 << THREAD_FREE_BITS)
557 #define THREAD_FREE_MASK (THREAD_FREE_NUM - 1)
558 #define THREAD_FREE_1 PTR24_LSB
559 #define THREAD_FREE_2 (PTR24_LSB + THREAD_FREE_BITS)
560 #define THREAD_FREE_SHIFT(t) \
561 ((ulong_t)(t) >> THREAD_FREE_1) ^ ((ulong_t)(t) >> THREAD_FREE_2)
562 #define THREAD_FREE_HASH(t) (THREAD_FREE_SHIFT(t) & THREAD_FREE_MASK)

564 typedef struct thread_free_lock {
565     kmutex_t tf_lock;
566     uchar_t tf_pad[64 - sizeof(kmutex_t)];
567 } thread_free_lock_t;
_____unchanged_portion_omitted_____

```

```

*****
6324 Tue May 7 04:49:58 2019
new/usr/src/uts/i86pc/Makefile.files
OS-7125 Need mitigation of LITF (CVE-2018-3646)
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
*****
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 #
23 # Copyright (c) 1992, 2010, Oracle and/or its affiliates. All rights reserved.
24 #
25 # Copyright (c) 2010, Intel Corporation.
26 # Copyright 2018 Joyent, Inc.
27 # Copyright 2019 OmniOS Community Edition (OmniOSce) Association.
28 #
29 # This Makefile defines file modules in the directory uts/i86pc
30 # and its children. These are the source files which are i86pc
31 # "implementation architecture" dependent.
32 #
33 #
34 #
35 # object lists
36 #
37 CORE_OBJ += \
38 acpi_stubs.o \
39 biosdisk.o \
40 bios_call.o \
41 cbe.o \
42 cmi.o \
43 cmi_hw.o \
44 cms.o \
45 comm_page.o \
46 confunix.o \
47 cpu_idle.o \
48 cpuid.o \
49 cpuid_subr.o \
50 cpupm.o \
51 cpupm_mach.o \
52 cpupm_amd.o \
53 cpupm_intel.o \
54 cpupm_throttle.o \
55 cpu_acpi.o \
56 dis_tables.o \
57 ddi_impl.o \
58 dtrace_subr.o \
59 dvma.o \

```

```

60 fpu_subr.o \
61 fakebop.o \
62 fastboot.o \
63 fb_swtpch.o \
64 graphics.o \
65 hardclk.o \
66 hat_i86.o \
67 hat_kdi.o \
68 hma_fpu.o \
69 hment.o \
70 hold_page.o \
71 hrtimers.o \
72 ht.o \
73 htable.o \
74 hypercall.o \
75 hypersubr.o \
76 i86_mmu.o \
77 ibft.o \
78 instr_size.o \
79 intr.o \
80 kboot_mmu.o \
81 kdi_idt.o \
82 kdi_idthdl.o \
83 kdi_asm.o \
84 lgrpplat.o \
85 mach_kdi.o \
86 mach_sysconfig.o \
87 machdep.o \
88 mem_config.o \
89 mem_config_stubs.o \
90 mem_config_arch.o \
91 memlist_new.o \
92 memnode.o \
93 microcode.o \
94 microfind.o \
95 mlsetup.o \
96 mp_call.o \
97 mp_implfuncs.o \
98 mp_machdep.o \
99 mp_pc.o \
100 mp_startup.o \
101 memscrub.o \
102 mpcore.o \
103 notes.o \
104 pci_bios.o \
105 pci_cfgacc.o \
106 pci_cfgacc_x86.o \
107 pci_cfgspace.o \
108 pci_mech1.o \
109 pci_mech1_amd.o \
110 pci_mech2.o \
111 pci_neptune.o \
112 pci_orion.o \
113 pmem.o \
114 ppage.o \
115 pwrnow.o \
116 speedstep.o \
117 ssp.o \
118 startup.o \
119 timestamp.o \
120 todpc_subr.o \
121 trap.o \
122 turbo.o \
123 vm_machdep.o \
124 xpv_platform.o \
125 x_call.o \

```

```

127 #
128 #   Add the SMBIOS subsystem object files directly to the list of objects
129 #   built into unix itself; this is all common code except for smb_dev.c.
130 #
131 CORE_OBJS += $(SMBIOS_OBJS)

133 #
134 # These get compiled twice:
135 # - once in the dboot (direct boot) identity mapped code
136 # - once for use during early startup in unix
137 #
138 BOOT_DRIVER_OBJS = \
139   boot_console.o \
140   boot_keyboard.o \
141   boot_keyboard_table.o \
142   boot_vga.o \
143   boot_fb.o \
144   boot_mmu.o \
145   dboot_multiboot2.o \
146   $(FONT_OBJS)

148 CORE_OBJS += $(BOOT_DRIVER_OBJS)

150 #
151 #   locore.o is special. It must be the first file relocated so that it
152 #   it is relocated just where its name implies.
153 #
154 SPECIAL_OBJS_32 += \
155   locore.o \
156   fast_trap_asm.o \
157   interrupt.o \
158   syscall_asm.o

160 SPECIAL_OBJS_64 += \
161   locore.o \
162   fast_trap_asm.o \
163   interrupt.o \
164   syscall_asm_amd64.o \
165   kpti_trampoline.o

167 SPECIAL_OBJS += $(SPECIAL_OBJS_$(CLASS))

169 #
170 # Objects that get compiled into the identity mapped PT_LOAD section of unix
171 # to handle the earliest part of booting.
172 #
173 DBOOT_OBJS_32 =

175 DBOOT_OBJS_64 += dboot_elfload.o

177 DBOOT_OBJS += \
178   dboot_asm.o \
179   dboot_grub.o \
180   dboot_printf.o \
181   dboot_startkern.o \
182   memcpy.o \
183   memset.o \
184   muldiv.o \
185   shal.o \
186   string.o \
187   $(BOOT_DRIVER_OBJS) \
188   $(DBOOT_OBJS_$(CLASS))

190 #
191 #   driver and misc modules

```

```

192 #
193 GFX_PRIVATE_OBJS += gfx_private.o gfxp_pci.o gfxp_segmap.o \
194   gfxp_devmap.o gfxp_vgatext.o gfxp_vm.o vgasubr.o \
195   gfxp_fb.o gfxp_bitmap.o
196 PIPE_OBJS += pipe_drv.o pipe_pm.o
197 IOAT_OBJS += ioat.o ioat_rs.o ioat_ioctl.o ioat_chan.o
198 ISANEXUS_OBJS += isa.o dma_engine.o i8237A.o
199 PCIE_MISC_OBJS += pcie_acpi.o pciehpc_acpi.o pcie_x86.o
200 PCI_E_NEXUS_OBJS += npe.o npe_misc.o
201 PCI_E_NEXUS_OBJS += pci_common.o pci_kstats.o pci_tools.o
202 PCINEXUS_OBJS += pci.o pci_common.o pci_kstats.o pci_tools.o
203 PCPLUSMP_OBJS += apic.o apic_regops.o psm_common.o apic_introp.o \
204   mp_platform_common.o mp_platform_misc.o \
205   hpet_acpi.o apic_common.o apic_timer.o
206 APIX_OBJS += apix.o apic_regops.o psm_common.o apix_intr.o apix_utils.o \
207   apix_irm.o mp_platform_common.o hpet_acpi.o apic_common.o \
208   apic_timer.o apix_regops.o

211 ACPI_DRV_OBJS += acpi_drv.o acpi_video.o
212 ACPINEX_OBJS += acpinex_drv.o acpinex_event.o

214 CPUDRV_OBJS += \
215   cpudrv.o \
216   cpudrv_mach.o

218 PPM_OBJS += ppm_subr.o ppm.o ppm_plat.o

220 ACPIPPM_OBJS += acpippm.o acpisleep.o
221 ACPIDEV_OBJS += acpidev_drv.o \
222   acpidev_scope.o acpidev_device.o \
223   acpidev_container.o \
224   acpidev_cpu.o \
225   acpidev_dr.o \
226   acpidev_memory.o \
227   acpidev_pci.o \
228   acpidev_resource.o \
229   acpidev_usbport.o \
230   acpidev_util.o

232 DRMACH ACPI_OBJS += drmach_acpi.o dr_util.o drmach_err.o

234 DR_OBJS += dr.o dr_cpu.o dr_err.o dr_io.o dr_mem_acpi.o dr_quiesce.o dr_util.o

236 ROOTNEX_OBJS += rootnex.o immu.o immu_dmar.o immu_dvma.o \
237   immu_intrmap.o immu_qinv.o immu_regs.o

239 TZMON_OBJS += tzmon.o
240 UPPC_OBJS += uppc.o psm_common.o
241 XSVC_OBJS += xsvc.o
242 AMD_IOMMU_OBJS += amd_iommu.o amd_iommu_impl.o amd_iommu_acpi.o \
243   amd_iommu_cmd.o amd_iommu_log.o amd_iommu_page_tables.o

245 #
246 #   Build up defines and paths.
247 #
248 ALL_DEFS += -Di86pc
249 INC_PATH += -I$(UTSBASE)/i86pc -I$(SRC)/common
250 INC_PATH += -I$(UTSBASE)/i86pxpv -I$(UTSBASE)/common/xen

252 #
253 # Since the assym files are derived, the dependencies must be explicit for
254 # all files including this file. (This is only actually required in the
255 # instance when the .nse_depinfo file does not exist.)
256 #

```

```
258 ASSYM_DEPS      +=      \
259      copy.o      \
260      desctbls_asm.o  \
261      ddi_i86_asm.o  \
262      exception.o   \
263      fast_trap_asm.o \
264      float.o       \
265      i86_subr.o    \
266      interrupt.o   \
267      lock_prim.o   \
268      locore.o      \
269      mpcore.o      \
270      sseblk.o      \
271      swtch.o       \
272      syscall_asm.o \
273      syscall_asm_amd64.o \
274      kpti_trampoline.o \
275      cpr_wakecode.o

277 CPR_IMPL_OBJS   = cpr_impl.o   cpr_wakecode.o

279 $(KDI_ASSYM_DEPS:%=$(OBJDIR)/%):      $(DSFDIR)/$(OBJDIR)/kdi_assym.h
```



```

*****
25174 Tue May 7 04:49:58 2019
new/usr/src/uts/i86pc/io/apix/apix_intr.c
OS-7125 Need mitigation of LITF (CVE-2018-3646)
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
*****
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) 2010, Oracle and/or its affiliates. All rights reserved.
24  * Copyright 2018 Western Digital Corporation. All rights reserved.
25  * Copyright 2018 Joyent, Inc.
26 */

28 #include <sys/cpuvar.h>
29 #include <sys/cpu_event.h>
30 #include <sys/param.h>
31 #include <sys/cmn_err.h>
32 #include <sys/t_lock.h>
33 #include <sys/kmem.h>
34 #include <sys/machlock.h>
35 #include <sys/system.h>
36 #include <sys/archsystem.h>
37 #include <sys/atomic.h>
38 #include <sys/sdt.h>
39 #include <sys/processor.h>
40 #include <sys/time.h>
41 #include <sys/psm.h>
42 #include <sys/smp_impldefs.h>
43 #include <sys/cram.h>
44 #include <sys/apic.h>
45 #include <sys/pit.h>
46 #include <sys/ddi.h>
47 #include <sys/sunddi.h>
48 #include <sys/ddi_impldefs.h>
49 #include <sys/pci.h>
50 #include <sys/promif.h>
51 #include <sys/x86_archext.h>
52 #include <sys/cpc_impl.h>
53 #include <sys/uadmin.h>
54 #include <sys/panic.h>
55 #include <sys/debug.h>
56 #include <sys/trap.h>
57 #include <sys/machsystem.h>
58 #include <sys/sysmacros.h>
59 #include <sys/rm_platter.h>

```

```

60 #include <sys/privregs.h>
61 #include <sys/note.h>
62 #include <sys/pci_intr_lib.h>
63 #include <sys/spl.h>
64 #include <sys/clock.h>
65 #include <sys/dditypes.h>
66 #include <sys/sunddi.h>
67 #include <sys/x_call.h>
68 #include <sys/reboot.h>
69 #include <vm/hat_i86.h>
70 #include <sys/stack.h>
71 #include <sys/apix.h>
72 #include <sys/ht.h>

74 static void apix_post_hardint(int);

76 /*
77  * Insert an vector into the tail of the interrupt pending list
78  */
79 static __inline__ void
80 apix_insert_pending_av(apix_impl_t *apixp, struct autovec *avp, int ipl)
81 {
82     struct autovec **head = apixp->x_intr_head;
83     struct autovec **tail = apixp->x_intr_tail;

85     avp->av_ipl_link = NULL;
86     if (tail[ipl] == NULL) {
87         head[ipl] = tail[ipl] = avp;
88         return;
89     }

91     tail[ipl]->av_ipl_link = avp;
92     tail[ipl] = avp;
93 }

    unchanged portion omitted

229 static caddr_t
230 apix_do_softint_prolog(struct cpu *cpu, uint_t pil, uint_t oldpil,
231     caddr_t stackptr)
232 {
233     kthread_t *t, *volatile it;
234     struct machcpu *mcpu = &cpu->cpu_m;
235     hrttime_t now;

237     UNREFERENCED_1PARAMETER(oldpil);
238     ASSERT(pil > mcpu->mcpu_pri && pil > cpu->cpu_base_spl);

240     atomic_and_32((uint32_t *)&mcpu->mcpu_softinfo.st_pending, ~(1 << pil));

242     mcpu->mcpu_pri = pil;

244     now = tsc_read();

246     /*
247      * Get set to run interrupt thread.
248      * There should always be an interrupt thread since we
249      * allocate one for each level on the CPU.
250      */
251     it = cpu->cpu_intr_thread;
252     ASSERT(it != NULL);
253     cpu->cpu_intr_thread = it->t_link;

255     /* t_intr_start could be zero due to cpu_intr_swthc_enter. */
256     t = cpu->cpu_thread;
257     if ((t->t_flag & T_INTR_THREAD) && t->t_intr_start != 0) {
258         hrttime_t intrtime = now - t->t_intr_start;

```

```

259         mcpu->intrstat[pil][0] += intrtime;
260         cpu->cpu_intracct[cpu->cpu_mstate] += intrtime;
261         t->t_intr_start = 0;
262     }

264     /*
265     * Note that the code in kpcpc_overflow_intr -relies- on the
266     * ordering of events here - in particular that t->t_lwp of
267     * the interrupt thread is set to the pinned thread *before*
268     * curthread is changed.
269     */
270     it->t_lwp = t->t_lwp;
271     it->t_state = TS_ONPROC;

273     /*
274     * Push interrupted thread onto list from new thread.
275     * Set the new thread as the current one.
276     * Set interrupted thread's T_SP because if it is the idle thread,
277     * resume() may use that stack between threads.
278     */

280     ASSERT(SA((uintptr_t)stackptr) == (uintptr_t)stackptr);
281     t->t_sp = (uintptr_t)stackptr;

283     it->t_intr = t;
284     cpu->cpu_thread = it;
285     ht_begin_intr(pil);

287     /*
288     * Set bit for this pil in CPU's interrupt active bitmask.
289     */
290     ASSERT((cpu->cpu_intr_actv & (1 << pil)) == 0);
291     cpu->cpu_intr_actv |= (1 << pil);

293     /*
294     * Initialize thread priority level from intr_pri
295     */
296     it->t_pil = (uchar_t)pil;
297     it->t_pri = (pri_t)pil + intr_pri;
298     it->t_intr_start = now;

300     return (it->t_stk);
301 }

303 static void
304 apix_do_softint_epilog(struct cpu *cpu, uint_t oldpil)
305 {
306     struct machcpu *mcpu = &cpu->cpu_m;
307     kthread_t *t, *it;
308     uint_t pil, basespl;
309     hrtime_t intrtime;
310     hrtime_t now = tsc_read();

312     it = cpu->cpu_thread;
313     pil = it->t_pil;

315     cpu->cpu_stats.sys.intr[pil - 1]++;

317     ASSERT(cpu->cpu_intr_actv & (1 << pil));
318     cpu->cpu_intr_actv &= ~(1 << pil);

320     intrtime = now - it->t_intr_start;
321     mcpu->intrstat[pil][0] += intrtime;
322     cpu->cpu_intracct[cpu->cpu_mstate] += intrtime;

324     /*

```

```

325     * If there is still an interrupted thread underneath this one
326     * then the interrupt was never blocked and the return is
327     * fairly simple. Otherwise it isn't.
328     */
329     if ((t = it->t_intr) == NULL) {
330         /*
331         * Put thread back on the interrupt thread list.
332         * This was an interrupt thread, so set CPU's base SPL.
333         */
334         set_base_spl();
335         /* mcpu->mcpu_pri = cpu->cpu_base_spl; */

337         /*
338         * If there are pending interrupts, send a softint to
339         * re-enter apix_do_interrupt() and get them processed.
340         */
341         if (apixs[cpu->cpu_id]->x_intr_pending)
342             siron();

344         it->t_state = TS_FREE;
345         it->t_link = cpu->cpu_intr_thread;
346         cpu->cpu_intr_thread = it;
347         (void) splhigh();
348         sti();
349         swtch();
350         /*NOTREACHED*/
351         panic("dosoftint_epilog: swtch returned");
352     }
353     it->t_link = cpu->cpu_intr_thread;
354     cpu->cpu_intr_thread = it;
355     it->t_state = TS_FREE;
356     ht_end_intr();
357     cpu->cpu_thread = t;

359     if (t->t_flag & T_INTR_THREAD)
360         t->t_intr_start = now;
361     basespl = cpu->cpu_base_spl;
362     pil = MAX(oldpil, basespl);
363     mcpu->mcpu_pri = pil;
364 }

    unchanged portion omitted

415 static int
416 apix_hilevel_intr_prolog(struct cpu *cpu, uint_t pil, uint_t oldpil,
417     struct regs *rp)
418 {
419     struct machcpu *mcpu = &cpu->cpu_m;
420     hrtime_t intrtime;
421     hrtime_t now = tsc_read();
422     apix_impl_t *apixp = apixs[cpu->cpu_id];
423     uint_t mask;

425     ASSERT(pil > mcpu->mcpu_pri && pil > cpu->cpu_base_spl);

427     if (pil == CBE_HIGH_PIL) { /* 14 */
428         cpu->cpu_profile_pil = oldpil;
429         if (USERMODE(rp->r_cs)) {
430             cpu->cpu_profile_pc = 0;
431             cpu->cpu_profile_upc = rp->r_pc;
432             cpu->cpu_cpcprofile_pc = 0;
433             cpu->cpu_cpcprofile_upc = rp->r_pc;
434         } else {
435             cpu->cpu_profile_pc = rp->r_pc;
436             cpu->cpu_profile_upc = 0;
437             cpu->cpu_cpcprofile_pc = rp->r_pc;
438             cpu->cpu_cpcprofile_upc = 0;

```

```

439     }
440 }

442 mcpu->mcpu_pri = pil;

444 mask = cpu->cpu_intr_actv & CPU_INTR_ACTV_HIGH_LEVEL_MASK;
445 if (mask != 0) {
446     int nestpil;

448     /*
449      * We have interrupted another high-level interrupt.
450      * Load starting timestamp, compute interval, update
451      * cumulative counter.
452      */
453     nestpil = bsrw_insn((uint16_t)mask);
454     intrtime = now -
455         mcpu->pil_high_start[nestpil - (LOCK_LEVEL + 1)];
456     mcpu->intrstat[nestpil][0] += intrtime;
457     cpu->cpu_intracct[cpu->cpu_mstate] += intrtime;
458 } else {
459     kthread_t *t = cpu->cpu_thread;

461     /*
462      * See if we are interrupting a low-level interrupt thread.
463      * If so, account for its time slice only if its time stamp
464      * is non-zero.
465      */
466     if ((t->t_flag & T_INTR_THREAD) != 0 && t->t_intr_start != 0) {
467         intrtime = now - t->t_intr_start;
468         mcpu->intrstat[t->t_pil][0] += intrtime;
469         cpu->cpu_intracct[cpu->cpu_mstate] += intrtime;
470         t->t_intr_start = 0;
471     }
472 }

474 ht_begin_intr(pil);

476 /* store starting timestamp in CPU structure for this IPL */
477 mcpu->pil_high_start[pil - (LOCK_LEVEL + 1)] = now;

479 if (pil == 15) {
480     /*
481      * To support reentrant level 15 interrupts, we maintain a
482      * recursion count in the top half of cpu_intr_actv. Only
483      * when this count hits zero do we clear the PIL 15 bit from
484      * the lower half of cpu_intr_actv.
485      */
486     uint16_t *refcntp = (uint16_t *)&cpu->cpu_intr_actv + 1;
487     (*refcntp)++;
488 }

490 cpu->cpu_intr_actv |= (1 << pil);
491 /* clear pending ipl level bit */
492 apixp->x_intr_pending &= ~(1 << pil);

494 return (mask);
495 }

497 static int
498 apix_hilevel_intr_epilog(struct cpu *cpu, uint_t oldpil)
499 {
500     struct machcpu *mcpu = &cpu->cpu_m;
501     uint_t mask, pil;
502     hrtime_t intrtime;
503     hrtime_t now = tsc_read();

```

```

505     pil = mcpu->mcpu_pri;
506     cpu->cpu_stats.sys.intr[pil - 1]++;

508 ASSERT(cpu->cpu_intr_actv & (1 << pil));

510 if (pil == 15) {
511     /*
512      * To support reentrant level 15 interrupts, we maintain a
513      * recursion count in the top half of cpu_intr_actv. Only
514      * when this count hits zero do we clear the PIL 15 bit from
515      * the lower half of cpu_intr_actv.
516      */
517     uint16_t *refcntp = (uint16_t *)&cpu->cpu_intr_actv + 1;

519     ASSERT(*refcntp > 0);

521     if (--(*refcntp) == 0)
522         cpu->cpu_intr_actv &= ~(1 << pil);
523 } else {
524     cpu->cpu_intr_actv &= ~(1 << pil);
525 }

527 ASSERT(mcpu->pil_high_start[pil - (LOCK_LEVEL + 1)] != 0);

529 intrtime = now - mcpu->pil_high_start[pil - (LOCK_LEVEL + 1)];
530 mcpu->intrstat[pil][0] += intrtime;
531 cpu->cpu_intracct[cpu->cpu_mstate] += intrtime;

533 /*
534  * Check for lower-pil nested high-level interrupt beneath
535  * current one. If so, place a starting timestamp in its
536  * pil_high_start entry.
537  */
538 mask = cpu->cpu_intr_actv & CPU_INTR_ACTV_HIGH_LEVEL_MASK;
539 if (mask != 0) {
540     int nestpil;

542     /*
543      * find PIL of nested interrupt
544      */
545     nestpil = bsrw_insn((uint16_t)mask);
546     ASSERT(nestpil < pil);
547     mcpu->pil_high_start[nestpil - (LOCK_LEVEL + 1)] = now;
548     /*
549      * (Another high-level interrupt is active below this one,
550      * so there is no need to check for an interrupt
551      * thread. That will be done by the lowest priority
552      * high-level interrupt active.)
553     */
554 } else {
555     /*
556      * Check to see if there is a low-level interrupt active.
557      * If so, place a starting timestamp in the thread
558      * structure.
559      */
560     kthread_t *t = cpu->cpu_thread;

562     if (t->t_flag & T_INTR_THREAD)
563         t->t_intr_start = now;
564 }

566 ht_end_intr();

568 mcpu->mcpu_pri = oldpil;
569 if (pil < CBE_HIGH_PIL)
570     (void) (*setlvx)(oldpil, 0);

```

```

572     return (mask);
573 }
    unchanged_portion_omitted

616 /*
617  * Get an interrupt thread and switch to it. It's called from do_interrupt().
618  * The IF flag is cleared and thus all maskable interrupts are blocked at
619  * the time of calling.
620  */
621 static caddr_t
622 apix_intr_thread_prolog(struct cpu *cpu, uint_t pil, caddr_t stackptr)
623 {
624     apix_impl_t *apixp = apixs[cpu->cpu_id];
625     struct machcpu *mcpu = &cpu->cpu_m;
626     hrtime_t now = tsc_read();
627     kthread_t *t, *volatile it;

629     ASSERT(pil > mcpu->mcpu_pri && pil > cpu->cpu_base_spl);

631     apixp->x_intr_pending &= ~(1 << pil);
632     ASSERT((cpu->cpu_intr_actv & (1 << pil)) == 0);
633     cpu->cpu_intr_actv |= (1 << pil);
634     mcpu->mcpu_pri = pil;

636     /*
637      * Get set to run interrupt thread.
638      * There should always be an interrupt thread since we
639      * allocate one for each level on the CPU.
640      */
641     /* t_intr_start could be zero due to cpu_intr_swch_enter. */
642     t = cpu->cpu_thread;
643     if ((t->t_flag & T_INTR_THREAD) && t->t_intr_start != 0) {
644         hrtime_t intrtime = now - t->t_intr_start;
645         mcpu->intrstat[pil][0] += intrtime;
646         cpu->cpu_intracct[cpu->cpu_mstate] += intrtime;
647         t->t_intr_start = 0;
648     }

650     /*
651      * Push interrupted thread onto list from new thread.
652      * Set the new thread as the current one.
653      * Set interrupted thread's T_SP because if it is the idle thread,
654      * resume() may use that stack between threads.
655      */

657     ASSERT(SA((uintptr_t)stackptr) == (uintptr_t)stackptr);

659     t->t_sp = (uintptr_t)stackptr; /* mark stack in curthread for resume */

661     /*
662      * Note that the code in kpcp_overflow_intr -relies- on the
663      * ordering of events here - in particular that t->t_lwp of
664      * the interrupt thread is set to the pinned thread *before*
665      * curthread is changed.
666      */
667     it = cpu->cpu_intr_thread;
668     cpu->cpu_intr_thread = it->t_link;
669     it->t_intr = t;
670     it->t_lwp = t->t_lwp;

672     /*
673      * (threads on the interrupt thread free list could have state
674      * preset to TS_ONPROC, but it helps in debugging if
675      * they're TS_FREE.)
676      */

```

```

677     it->t_state = TS_ONPROC;

679     cpu->cpu_thread = it;
680     ht_begin_intr(pil);

682     /*
683      * Initialize thread priority level from intr_pri
684      */
685     it->t_pil = (uchar_t)pil;
686     it->t_pri = (pri_t)pil + intr_pri;
687     it->t_intr_start = now;

689     return (it->t_stk);
690 }

692 static void
693 apix_intr_thread_epilog(struct cpu *cpu, uint_t oldpil)
694 {
695     struct machcpu *mcpu = &cpu->cpu_m;
696     kthread_t *t, *it = cpu->cpu_thread;
697     uint_t pil, basespl;
698     hrtime_t intrtime;
699     hrtime_t now = tsc_read();

701     pil = it->t_pil;
702     cpu->cpu_stats.sys.intr[pil - 1]++;

704     ASSERT(cpu->cpu_intr_actv & (1 << pil));
705     cpu->cpu_intr_actv &= ~(1 << pil);

707     ASSERT(it->t_intr_start != 0);
708     intrtime = now - it->t_intr_start;
709     mcpu->intrstat[pil][0] += intrtime;
710     cpu->cpu_intracct[cpu->cpu_mstate] += intrtime;

712     /*
713      * If there is still an interrupted thread underneath this one
714      * then the interrupt was never blocked and the return is
715      * fairly simple. Otherwise it isn't.
716      */
717     if ((t = it->t_intr) == NULL) {
718         /*
719          * The interrupted thread is no longer pinned underneath
720          * the interrupt thread. This means the interrupt must
721          * have blocked, and the interrupted thread has been
722          * unpinned, and has probably been running around the
723          * system for a while.
724          *
725          * Since there is no longer a thread under this one, put
726          * this interrupt thread back on the CPU's free list and
727          * resume the idle thread which will dispatch the next
728          * thread to run.
729          */
730         cpu->cpu_stats.sys.intrblk++;

732         /*
733          * Put thread back on the interrupt thread list.
734          * This was an interrupt thread, so set CPU's base SPL.
735          */
736         set_base_spl();
737         basespl = cpu->cpu_base_spl;
738         mcpu->mcpu_pri = basespl;
739         (*setlvlx)(basespl, 0);

741         /*
742          * If there are pending interrupts, send a softint to

```

```
743     * re-enter apix_do_interrupt() and get them processed.
744     */
745     if (apixs[cpu->cpu_id]->x_intr_pending)
746         siron();

748     it->t_state = TS_FREE;
749     /*
750     * Return interrupt thread to pool
751     */
752     it->t_link = cpu->cpu_intr_thread;
753     cpu->cpu_intr_thread = it;

755     (void) splhigh();
756     sti();
757     swtch();
758     /*NOTREACHED*/
759     panic("dosoftint_epilog: swtch returned");
760 }

762 /*
763 * Return interrupt thread to the pool
764 */
765 it->t_link = cpu->cpu_intr_thread;
766 cpu->cpu_intr_thread = it;
767 it->t_state = TS_FREE;

769 ht_end_intr();
770 cpu->cpu_thread = t;

772 if (t->t_flag & T_INTR_THREAD)
773     t->t_intr_start = now;
774 basespl = cpu->cpu_base_spl;
775 mcpu->mcpu_pri = MAX(oldpil, basespl);
776 (*setlvlx)(mcpu->mcpu_pri, 0);
777 }
unchanged_portion_omitted
```

```

*****
48773 Tue May 7 04:49:59 2019
new/usr/src/uts/i86pc/io/apix/apix_utils.c
OS-7125 Need mitigation of L1TF (CVE-2018-3646)
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
*****
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) 2010, Oracle and/or its affiliates. All rights reserved.
24 */
25 /*
26  * Copyright (c) 2010, Intel Corporation.
27  * All rights reserved.
28 */
29 /*
30  * Copyright 2013 Nexenta Systems, Inc. All rights reserved.
31  * Copyright 2013 Pluribus Networks, Inc.
32  * Copyright 2018 Joyent, Inc.
33 */

35 #include <sys/processor.h>
36 #include <sys/time.h>
37 #include <sys/psm.h>
38 #include <sys/smp_impldefs.h>
39 #include <sys/cram.h>
40 #include <sys/acpi/acpi.h>
41 #include <sys/acpica.h>
42 #include <sys/psm_common.h>
43 #include <sys/pit.h>
44 #include <sys/ddi.h>
45 #include <sys/sunddi.h>
46 #include <sys/ddi_impldefs.h>
47 #include <sys/pci.h>
48 #include <sys/promif.h>
49 #include <sys/x86_archext.h>
50 #include <sys/cpc_impl.h>
51 #include <sys/uadmin.h>
52 #include <sys/panic.h>
53 #include <sys/debug.h>
54 #include <sys/archsystem.h>
55 #include <sys/trap.h>
56 #include <sys/machsystem.h>
57 #include <sys/sysmacros.h>
58 #include <sys/cpuvar.h>
59 #include <sys/rm_platter.h>

```

```

60 #include <sys/privregs.h>
61 #include <sys/note.h>
62 #include <sys/pci_intr_lib.h>
63 #include <sys/spl.h>
64 #include <sys/clock.h>
65 #include <sys/dditypes.h>
66 #include <sys/sunddi.h>
67 #include <sys/x_call.h>
68 #include <sys/reboot.h>
69 #include <sys/apix.h>
70 #include <sys/ht.h>

72 static int apix_get_avail_vector_oncpu(uint32_t, int, int);
73 static apix_vector_t *apix_init_vector(processorid_t, uchar_t);
74 static void apix_cleanup_vector(apix_vector_t *);
75 static void apix_insert_av(apix_vector_t *, void *, avfunc, caddr_t, caddr_t,
76     uint64_t *, int, dev_info_t *);
77 static void apix_remove_av(apix_vector_t *, struct autovec *);
78 static void apix_clear_dev_map(dev_info_t *, int, int);
79 static boolean_t apix_is_cpu_enabled(processorid_t);
80 static void apix_wait_till_seen(processorid_t, int);

82 #define GET_INTR_INUM(ihdlp) \
83     (((ihdlp) != NULL) ? ((ddi_intr_handle_impl_t *) (ihdlp))->ih_inum : 0)

85 apix_rebind_info_t apix_rebindinfo = {0, 0, 0, NULL, 0, NULL};

87 /*
88  * Allocate IPI
89  *
90  * Return vector number or 0 on error
91 */
92 uchar_t
93 apix_alloc_ipi(int ipl)
94 {
95     apix_vector_t *vecp;
96     uchar_t vector;
97     int cpun;
98     int nproc;

100     APIX_ENTER_CPU_LOCK(0);

102     vector = apix_get_avail_vector_oncpu(0, APIX_IPI_MIN, APIX_IPI_MAX);
103     if (vector == 0) {
104         APIX_LEAVE_CPU_LOCK(0);
105         cmn_err(CE_WARN, "apix: no available IPI\n");
106         apic_error |= APIC_ERR_GET_IPIVECT_FAIL;
107         return (0);
108     }

110     nproc = max(apic_nproc, apic_max_nproc);
111     for (cpun = 0; cpun < nproc; cpun++) {
112         vecp = xv_vector(cpun, vector);
113         if (vecp == NULL) {
114             vecp = kmem_zalloc(sizeof (apix_vector_t), KM_NOSLEEP);
115             if (vecp == NULL) {
116                 cmn_err(CE_WARN, "apix: No memory for ipi");
117                 goto fail;
118             }
119             xv_vector(cpun, vector) = vecp;
120         }
121         vecp->v_state = APIX_STATE_ALLOCATED;
122         vecp->v_type = APIX_TYPE_IPI;
123         vecp->v_cpuid = vecp->v_bound_cpuid = cpun;
124         vecp->v_vector = vector;
125         vecp->v_pri = ipl;

```

```

126     }
127     APIX_LEAVE_CPU_LOCK(0);
128     return (vector);

130 fail:
131     while (--cpun >= 0)
132         apix_cleanup_vector(xv_vector(cpun, vector));
133     APIX_LEAVE_CPU_LOCK(0);
134     return (0);
135 }
unchanged portion omitted

772 /*
773  * Operations on avintr
774  */

776 #define INIT_AUTOVEC(p, intr_id, f, arg1, arg2, ticksp, ipl, dip) \
777 do { \
778     (p)->av_intr_id = intr_id; \
779     (p)->av_vector = f; \
780     (p)->av_intarg1 = arg1; \
781     (p)->av_intarg2 = arg2; \
782     (p)->av_ticksp = ticksp; \
783     (p)->av_prilevel = ipl; \
784     (p)->av_dip = dip; \
785     (p)->av_flags = 0; \
786     _NOTE(CONSTCOND)} while (0)

788 /*
789  * Insert an interrupt service routine into chain by its priority from
790  * high to low
791  */
792 static void
793 apix_insert_av(apix_vector_t *vecp, void *intr_id, avfunc f, caddr_t arg1,
794               caddr_t arg2, uint64_t *ticksp, int ipl, dev_info_t *dip)
795 {
796     struct autovec *p, *prep, *mem;

798     APIC_VERBOSE(INTR, (CE_CONT, "apix_insert_av: dip %p, vector 0x%x, "
799                       "cpu %d\n", (void *)dip, vecp->v_vector, vecp->v_cpuid));

801     mem = kmem_zalloc(sizeof (struct autovec), KM_SLEEP);
802     INIT_AUTOVEC(mem, intr_id, f, arg1, arg2, ticksp, ipl, dip);
803     if (vecp->v_type == APIX_TYPE_FIXED && apic_level_intr[vecp->v_inum])
804         mem->av_flags |= AV_PENTRY_LEVEL;

806     vecp->v_share++;
807     vecp->v_pri = (ipl > vecp->v_pri) ? ipl : vecp->v_pri;

809     ht_intr_alloc_pil(vecp->v_pri);

811     if (vecp->v_autovec == NULL) { /* Nothing on list - put it at head */
812         vecp->v_autovec = mem;
813         return;
814     }

816     if (DDI_INTR_IS_MSI_OR_MSIX(vecp->v_type)) { /* MSI/X */
817         ASSERT(vecp->v_share == 1); /* No sharing for MSI/X */

819         INIT_AUTOVEC(vecp->v_autovec, intr_id, f, arg1, arg2, ticksp,
820                     ipl, dip);
821         prep = vecp->v_autovec->av_link;
822         vecp->v_autovec->av_link = NULL;

824         /* Free the following autovec chain */
825         while (prep != NULL) {

```

```

826         ASSERT(prepare->av_vector == NULL);

828         p = prep;
829         prep = prep->av_link;
830         kmem_free(p, sizeof (struct autovec));
831     }

833     kmem_free(mem, sizeof (struct autovec));
834     return;
835 }

837     /* find where it goes in list */
838     prep = NULL;
839     for (p = vecp->v_autovec; p != NULL; p = p->av_link) {
840         if (p->av_vector && p->av_prilevel <= ipl)
841             break;
842         prep = p;
843     }
844     if (prep != NULL) {
845         if (prep->av_vector == NULL) { /* freed struct available */
846             INIT_AUTOVEC(prepare, intr_id, f, arg1, arg2,
847                           ticksp, ipl, dip);
848             prep->av_flags = mem->av_flags;
849             kmem_free(mem, sizeof (struct autovec));
850             return;
851         }

853         mem->av_link = prep->av_link;
854         prep->av_link = mem;
855     } else {
856         /* insert new intpt at beginning of chain */
857         mem->av_link = vecp->v_autovec;
858         vecp->v_autovec = mem;
859     }
860 }
unchanged portion omitted

```

```

*****
33081 Tue May 7 04:49:59 2019
new/usr/src/uts/i86pc/io/pcplusmp/apic.c
OS-7125 Need mitigation of LITF (CVE-2018-3646)
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
*****
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) 1993, 2010, Oracle and/or its affiliates. All rights reserved.
24 */
25 /*
26  * Copyright (c) 2010, Intel Corporation.
27  * All rights reserved.
28  * Copyright 2018 Joyent, Inc.
29 */

31 /*
32  * To understand how the pcplusmp module interacts with the interrupt subsystem
33  * read the theory statement in uts/i86pc/os/intr.c.
34 */

36 /*
37  * PSMI 1.1 extensions are supported only in 2.6 and later versions.
38  * PSMI 1.2 extensions are supported only in 2.7 and later versions.
39  * PSMI 1.3 and 1.4 extensions are supported in Solaris 10.
40  * PSMI 1.5 extensions are supported in Solaris Nevada.
41  * PSMI 1.6 extensions are supported in Solaris Nevada.
42  * PSMI 1.7 extensions are supported in Solaris Nevada.
43 */
44 #define PSMI_1_7

46 #include <sys/processor.h>
47 #include <sys/time.h>
48 #include <sys/psm.h>
49 #include <sys/smp_impldefs.h>
50 #include <sys/cram.h>
51 #include <sys/acpi/acpi.h>
52 #include <sys/acpica.h>
53 #include <sys/psm_common.h>
54 #include <sys/apic.h>
55 #include <sys/pit.h>
56 #include <sys/ddi.h>
57 #include <sys/sunddi.h>
58 #include <sys/ddi_impldefs.h>
59 #include <sys/pci.h>

```

```

60 #include <sys/promif.h>
61 #include <sys/x86_archext.h>
62 #include <sys/cpc_impl.h>
63 #include <sys/uadmin.h>
64 #include <sys/panic.h>
65 #include <sys/debug.h>
66 #include <sys/archsystem.h>
67 #include <sys/trap.h>
68 #include <sys/machsystem.h>
69 #include <sys/sysmacros.h>
70 #include <sys/cpuvar.h>
71 #include <sys/rm_platter.h>
72 #include <sys/privregs.h>
73 #include <sys/note.h>
74 #include <sys/pci_intr_lib.h>
75 #include <sys/spl.h>
76 #include <sys/clock.h>
77 #include <sys/cyclic.h>
78 #include <sys/dditypes.h>
79 #include <sys/sunddi.h>
80 #include <sys/x_call.h>
81 #include <sys/reboot.h>
82 #include <sys/hpet.h>
83 #include <sys/apic_common.h>
84 #include <sys/apic_timer.h>
85 #include <sys/ht.h>

87 /*
88  * Local Function Prototypes
89 */
90 static void apic_init_intr(void);

92 /*
93  * standard MP entries
94 */
95 static int apic_probe(void);
96 static int apic_getclkirq(int ipl);
97 static void apic_init(void);
98 static void apic_picinit(void);
99 static int apic_post_cpu_start(void);
100 static int apic_intr_enter(int ipl, int *vect);
101 static void apic_setspl(int ipl);
102 static int apic_addspl(int ipl, int vector, int min_ipl, int max_ipl);
103 static int apic_delspl(int ipl, int vector, int min_ipl, int max_ipl);
104 static int apic_disable_intr(processorid_t cpun);
105 static void apic_enable_intr(processorid_t cpun);
106 static int apic_get_ipivect(int ipl, int type);
107 static void apic_post_cyclic_setup(void *arg);

109 #define UCHAR_MAX UINT8_MAX

111 /*
112  * The following vector assignments influence the value of iptopri and
113  * vectortoipl. Note that vectors 0 - 0xf are not used. We can program
114  * idle to 0 and IPL 0 to 0xf to differentiate idle in case
115  * we care to do so in future. Note some IPLs which are rarely used
116  * will share the vector ranges and heavily used IPLs (5 and 6) have
117  * a wide range.
118  *
119  * This array is used to initialize apic_ipls[] (in apic_init()).
120  *
121  * IPL Vector range. as passed to intr_enter
122  * 0 none.
123  * 1,2,3 0x20-0x2f 0x0-0xf
124  * 4 0x30-0x3f 0x10-0x1f
125  * 5 0x40-0x5f 0x20-0x3f

```



```

126 *      6      0x60-0x7f      0x40-0x5f
127 *      7,8,9    0x80-0x8f      0x60-0x6f
128 *      10     0x90-0x9f      0x70-0x7f
129 *      11     0xa0-0xaf      0x80-0x8f
130 *      ...     ...
131 *      15     0xe0-0xef      0xc0-0xcf
132 *      15     0xf0-0xff      0xd0-0xdf
133 */
134 uchar_t apic_vectortoipl[APIC_AVAIL_VECTOR / APIC_VECTOR_PER_IPL] = {
135     3, 4, 5, 5, 6, 6, 9, 10, 11, 12, 13, 14, 15, 15
136 };
    unchanged portion omitted

278 void
279 apic_init(void)
280 {
281     int i;
282     int j = 1;

284     psm_get_ioapicid = apic_get_ioapicid;
285     psm_get_localapicid = apic_get_localapicid;
286     psm_xlate_vector_by_irq = apic_xlate_vector_by_irq;

288     apic_ipltopri[0] = APIC_VECTOR_PER_IPL; /* leave 0 for idle */
289     for (i = 0; i < (APIC_AVAIL_VECTOR / APIC_VECTOR_PER_IPL); i++) {
290         if ((i < ((APIC_AVAIL_VECTOR / APIC_VECTOR_PER_IPL) - 1)) &&
291             (apic_vectortoipl[i + 1] == apic_vectortoipl[i])) &&
292             /* get to highest vector at the same ipl */
293             continue;
294         for (; j <= apic_vectortoipl[i]; j++) {
295             apic_ipltopri[j] = (i << APIC_IPL_SHIFT) +
296                 APIC_BASE_VECT;
297         }
298     }
299     for (; j < MAXIPL + 1; j++)
300         /* fill up any empty ipltopri slots */
301         apic_ipltopri[j] = (i << APIC_IPL_SHIFT) + APIC_BASE_VECT;
302     apic_init_common();

304     /*
305      * For pcplusmp, we'll keep things simple and always disable this.
306      */
307     ht_intr_alloc_pil(XC_CPUPOKE_PIL);

309     apic_pir_vect = apic_get_ipivect(XC_CPUPOKE_PIL, -1);

311 #if !defined(__amd64)
312     if (cpuid_have_cr8access(CPU))
313         apic_have_32bit_cr8 = 1;
314 #endif
315 }
    unchanged portion omitted

```

```

*****
197256 Tue May 7 04:49:59 2019
new/usr/src/uts/i86pc/os/cpuid.c
OS-7125 Need mitigation of L1TF (CVE-2018-3646)
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
*****
_____unchanged_portion_omitted_____

2122 static void
2123 spec_lld_flush_noop(void)
2124 {
2125 }

2127 static void
2128 spec_lld_flush_msr(void)
2129 {
2130     wrmsr(MSR_IA32_FLUSH_CMD, IA32_FLUSH_CMD_L1D);
2131 }

2133 void (*spec_lld_flush)(void) = spec_lld_flush_noop;

2135 static void
2136 cpuid_scan_security(cpu_t *cpu, uchar_t *featureset)
2137 {
2138     struct cpuid_info *cpi = cpu->cpu_m.mcpu_cpi;

2140     if (cpi->cpi_vendor == X86_VENDOR_AMD &&
2141         cpi->cpi_xmaxeax >= CPUID_LEAF_EXT_8) {
2142         if (cpi->cpi_extd[8].cp_ebx & CPUID_AMD_EBX_IBPB)
2143             add_x86_feature(featureset, X86FSET_IBPB);
2144         if (cpi->cpi_extd[8].cp_ebx & CPUID_AMD_EBX_IBRS)
2145             add_x86_feature(featureset, X86FSET_IBRS);
2146         if (cpi->cpi_extd[8].cp_ebx & CPUID_AMD_EBX_STIBP)
2147             add_x86_feature(featureset, X86FSET_STIBP);
2148         if (cpi->cpi_extd[8].cp_ebx & CPUID_AMD_EBX_IBRS_ALL)
2149             add_x86_feature(featureset, X86FSET_IBRS_ALL);
2150         if (cpi->cpi_extd[8].cp_ebx & CPUID_AMD_EBX_STIBP_ALL)
2151             add_x86_feature(featureset, X86FSET_STIBP_ALL);
2152         if (cpi->cpi_extd[8].cp_ebx & CPUID_AMD_EBX_PREFER_IBRS)
2153             add_x86_feature(featureset, X86FSET_RSBA);
2154         if (cpi->cpi_extd[8].cp_ebx & CPUID_AMD_EBX_SSB)
2155             add_x86_feature(featureset, X86FSET_SSB);
2156         if (cpi->cpi_extd[8].cp_ebx & CPUID_AMD_EBX_VIRT_SSB)
2157             add_x86_feature(featureset, X86FSET_SSB_VIRT);
2158         if (cpi->cpi_extd[8].cp_ebx & CPUID_AMD_EBX_SSB_NO)
2159             add_x86_feature(featureset, X86FSET_SSB_NO);
2160     } else if (cpi->cpi_vendor == X86_VENDOR_Intel &&
2161                cpi->cpi_maxeax >= 7) {
2162         struct cpuid_regs *ecp;
2163         ecp = &cpi->cpi_std[7];

2165         if (ecp->cp_edx & CPUID_INTC_EDX_7_0_SPEC_CTRL) {
2166             add_x86_feature(featureset, X86FSET_IBRS);
2167             add_x86_feature(featureset, X86FSET_IBPB);
2168         }

2170         if (ecp->cp_edx & CPUID_INTC_EDX_7_0_STIBP) {
2171             add_x86_feature(featureset, X86FSET_STIBP);
2172         }

2174         /*
2175          * Don't read the arch caps MSR on xpv where we lack the
2176          * on_trap().
2177          */
2178 #ifndef __xpv

```

```

2179         if (ecp->cp_edx & CPUID_INTC_EDX_7_0_ARCH_CAPS) {
2180             on_trap_data_t otd;

2182             /*
2183              * Be paranoid and assume we'll get a #GP.
2184              */
2185             if (!on_trap(&otd, OT_DATA_ACCESS)) {
2186                 uint64_t reg;

2188                 reg = rdmsr(MSR_IA32_ARCH_CAPABILITIES);
2189                 if (reg & IA32_ARCH_CAP_RDCL_NO) {
2190                     add_x86_feature(featureset,
2191                                     X86FSET_RDCL_NO);
2192                 }
2193                 if (reg & IA32_ARCH_CAP_IBRS_ALL) {
2194                     add_x86_feature(featureset,
2195                                     X86FSET_IBRS_ALL);
2196                 }
2197                 if (reg & IA32_ARCH_CAP_RSBA) {
2198                     add_x86_feature(featureset,
2199                                     X86FSET_RSBA);
2200                 }
2201                 if (reg & IA32_ARCH_CAP_SKIP_L1DFL_VMENTRY) {
2202                     add_x86_feature(featureset,
2203                                     X86FSET_L1D_VM_NO);
2204                 }
2205                 if (reg & IA32_ARCH_CAP_SSB_NO) {
2206                     add_x86_feature(featureset,
2207                                     X86FSET_SSB_NO);
2208                 }
2209             }
2210             no_trap();
2211         }
2212 #endif /* !__xpv */

2214         if (ecp->cp_edx & CPUID_INTC_EDX_7_0_SSB)
2215             add_x86_feature(featureset, X86FSET_SSB);

2217         if (ecp->cp_edx & CPUID_INTC_EDX_7_0_FLUSH_CMD)
2218             add_x86_feature(featureset, X86FSET_FLUSH_CMD);
2219     }

2221     if (cpu->cpu_id != 0)
2222         return;

2224     /*
2225      * We're the boot CPU, so let's figure out our L1TF status.
2226      * First, if this is a RDCL_NO CPU, then we are not vulnerable: we don't
2227      * need to exclude with ht_acquire(), and we don't need to flush.
2228      */
2229     if (is_x86_feature(featureset, X86FSET_RDCL_NO)) {
2230         extern int ht_exclusion;
2231         ht_exclusion = 0;
2232         spec_lld_flush = spec_lld_flush_noop;
2233         membar_producer();
2234         return;
2235     }

2238     /*
2239      * If HT is enabled, we will need HT exclusion, as well as the flush on
2240      * VM entry. If HT isn't enabled, we still need at least the flush for
2241      * the L1TF sequential case.
2242      * However, if X86FSET_L1D_VM_NO is set, we're most likely running
2243      * inside a VM ourselves, and we don't need the flush.

```

```
2245     *
2246     * If we don't have the FLUSH_CMD available at all, we'd better just
2247     * hope HT is disabled.
2248     */
2249     if (is_x86_feature(featureset, X86FSET_FLUSH_CMD) &&
2250         !is_x86_feature(featureset, X86FSET_L1D_VM_NO)) {
2251         spec_l1d_flush = spec_l1d_flush_msr;
2252     } else {
2253         spec_l1d_flush = spec_l1d_flush_noop;
2254     }
2255     membar_producer();
2256 }
2257 }
unchanged_portion_omitted
```

```

*****
15578 Tue May 7 04:50:00 2019
new/usr/src/uts/i86pc/os/ht.c
OS-7125 Need mitigation of L1TF (CVE-2018-3646)
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
*****
1 /*
2  * This file and its contents are supplied under the terms of the
3  * Common Development and Distribution License ("CDDL"), version 1.0.
4  * You may only use this file in accordance with the terms of version
5  * 1.0 of the CDDL.
6  *
7  * A full copy of the text of the CDDL should have accompanied this
8  * source. A copy of the CDDL is also available via the Internet at
9  * http://www.illumos.org/license/CDDL.
10 */

12 /*
13  * Copyright 2018 Joyent, Inc.
14 */

16 /*
17  * HT exclusion: prevent a sibling in a hyper-threaded core from running in VMX
18  * non-root guest mode, when certain threads are running on the other sibling.
19  * This avoids speculation-based information leaks such as L1TF being available
20  * to the untrusted guest. The stance we take is that threads from the same
21  * zone as the guest VCPU thread are considered safe to run alongside, but all
22  * other threads (except the idle thread), and all interrupts, are unsafe. Note
23  * that due to the implementation here, there are significant sections of e.g.
24  * the dispatcher code that can run concurrently with a guest, until the thread
25  * reaches ht_mark(). This code assumes there are only two HT threads per core.
26  *
27  * The entry points are as follows:
28  *
29  * ht_mark_as_vcpu()
30  *
31  * All threads that enter guest mode (i.e. VCPU threads) need to call this at
32  * least once, which sets TS_VCPU in ->t_schedflag.
33  *
34  * ht_mark()
35  *
36  * A new ->cpu_thread is now curthread (although interrupt threads have their
37  * own separate handling). After preventing any interrupts, we will take our
38  * own CPU's spinlock and update our own state in mcpu_ht.
39  *
40  * If our sibling is poisoned (i.e. in guest mode or the little bit of code
41  * around it), and we're not compatible (that is, same zone ID, or the idle
42  * thread), then we need to ht_kick() that sibling. ht_kick() itself waits for
43  * the sibling to call ht_release(), and it will not re-enter guest mode until
44  * allowed.
45  *
46  * Note that we ignore the fact a process can change its zone ID: poisoning
47  * threads never do so, and we can ignore the other cases.
48  *
49  * ht_acquire()
50  *
51  * We are a VCPU thread about to start guest execution. Interrupts are
52  * disabled. We must have already run ht_mark() to be in this code, so there's
53  * no need to take our *own* spinlock in order to mark ourselves as CM_POISONED.
54  * Instead, we take our sibling's lock to also mark ourselves as poisoned in the
55  * sibling cpu_ht_t. This is so ht_mark() will only ever need to look at its
56  * local mcpu_ht.
57  *
58  * We'll loop here for up to ht_acquire_wait_time microseconds; this is mainly
59  * to wait out any sibling interrupt: many of them will complete quicker than

```

```

60  * this.
61  *
62  * Finally, if we succeeded in acquiring the core, we'll flush the L1 cache as
63  * mitigation against L1TF: no incompatible thread will now be able to populate
64  * the L1 cache until *we* ht_release().
65  *
66  * ht_release()
67  *
68  * Simply unpoison ourselves similarly to ht_acquire(); ht_kick() will wait for
69  * this to happen if needed.
70  *
71  * ht_begin_intr()
72  *
73  * In an interrupt prolog. We're either a hilevel interrupt, or a pinning
74  * interrupt. In both cases, we mark our interrupt depth, and potentially
75  * ht_kick(). This enforces exclusion, but doesn't otherwise modify ->ch_state:
76  * we want the dispatcher code to essentially ignore interrupts.
77  *
78  * ht_end_intr()
79  *
80  * In an interrupt epilogue *or* thread_unpin(). In the first case, we never
81  * slept, and we can simply decrement our counter. In the second case, we're an
82  * interrupt thread about to sleep: we'll still just decrement our counter, and
83  * henceforth treat the thread as a normal thread when it next gets scheduled,
84  * until it finally gets to its epilogue.
85  *
86  * ht_mark_unsafe() / ht_mark_safe()
87  *
88  * Mark the current thread as temporarily unsafe (guests should not be executing
89  * while a sibling is marked unsafe). This can be used for a thread that's
90  * otherwise considered safe, if it needs to handle potentially sensitive data.
91  * Right now, this means certain I/O handling operations that reach down into
92  * the networking and ZFS sub-systems.
93  *
94  * ht_should_run(thread, cpu)
95  *
96  * This is used by the dispatcher when making scheduling decisions: if the
97  * sibling is compatible with the given thread, we return B_TRUE. This is
98  * essentially trying to guess if any subsequent ht_acquire() will fail, by
99  * peeking at the sibling CPU's state. The peek is racy, but if we get things
100 * wrong, the "only" consequence is that ht_acquire() may lose.
101 *
102 * ht_adjust_cpu_score()
103 *
104 * Used when scoring other CPUs in disp_lowpri_cpu(). If we shouldn't run here,
105 * we'll add a small penalty to the score. This also makes sure a VCPU thread
106 * migration behaves properly.
107 */

109 #include <sys/archsystem.h>
110 #include <sys/disp.h>
111 #include <sys/cmt.h>
112 #include <sys/system.h>
113 #include <sys/cpu.h>
114 #include <sys/var.h>
115 #include <sys/xc_levels.h>
116 #include <sys/cmn_err.h>
117 #include <sys/sysmacros.h>
118 #include <sys/x86_archext.h>

120 #define CS_SHIFT (8)
121 #define CS_MASK ((1 << CS_SHIFT) - 1)
122 #define CS_MARK(s) ((s) & CS_MASK)
123 #define CS_ZONE(s) ((s) >> CS_SHIFT)
124 #define CS_MK(s, z) ((s) | (z << CS_SHIFT))

```

```

126 typedef enum ch_mark {
127     CM_IDLE = 0, /* running CPU idle thread */
128     CM_THREAD, /* running general non-VCPU thread */
129     CM_UNSAFE, /* running ->t_unsafe thread */
130     CM_VCPU, /* running VCPU thread */
131     CM_POISONED /* running in guest */
132 } ch_mark_t;

134 /* Double-check our false-sharing padding. */
135 CTASSERT(offsetof(cpu_ht_t, ch_sib) == 64);
136 CTASSERT(CM_IDLE == 0);
137 CTASSERT(CM_POISONED < (1 << CS_SHIFT));
138 CTASSERT(CM_POISONED > CM_VCPU);
139 CTASSERT(CM_VCPU > CM_UNSAFE);

141 static uint_t empty_pil = XC_CPUPUKE_PIL;

143 /*
144  * If disabled, no HT exclusion is performed, and system is potentially
145  * vulnerable to L1TF if hyper-threading is enabled, and we don't have the "not
146  * vulnerable" CPUID bit.
147  */
148 int ht_exclusion = 1;

150 /*
151  * How long ht_acquire() will spin trying to acquire the core, in micro-seconds.
152  * This is enough time to wait out a significant proportion of interrupts.
153  */
154 clock_t ht_acquire_wait_time = 64;

156 static cpu_t *
157 ht_find_sibling(cpu_t *cp)
158 {
159     for (uint_t i = 0; i < GROUP_SIZE(&cp->cpu_pg->cmt_pgs); i++) {
160         pg_cmt_t *pg = GROUP_ACCESS(&cp->cpu_pg->cmt_pgs, i);
161         group_t *cg = &pg->cmt_pg.pghw_pg.pg_cpus;

163         if (pg->cmt_pg.pghw_hw != PGHW_IPIPE)
164             continue;

166         if (GROUP_SIZE(cg) == 1)
167             break;

169         VERIFY3U(GROUP_SIZE(cg), ==, 2);

171         if (GROUP_ACCESS(cg, 0) != cp)
172             return (GROUP_ACCESS(cg, 0));

174         VERIFY3P(GROUP_ACCESS(cg, 1), !=, cp);

176         return (GROUP_ACCESS(cg, 1));
177     }

179     return (NULL);
180 }

182 /*
183  * Initialize HT links. We have to be careful here not to race with
184  * ht_begin/end_intr(), which also complicates trying to do this initialization
185  * from a cross-call; hence the slightly odd approach below.
186  */
187 void
188 ht_init(void)
189 {
190     cpu_t *scp = CPU;
191     cpu_t *cp = scp;

```

```

192     ulong_t flags;

194     if (!ht_exclusion)
195         return;

197     mutex_enter(&cpu_lock);

199     do {
200         thread_affinity_set(curthread, cp->cpu_id);
201         flags = intr_clear();

203         cp->cpu_m.mcpu_ht.ch_intr_depth = 0;
204         cp->cpu_m.mcpu_ht.ch_state = CS_MK(CM_THREAD, GLOBAL_ZONEID);
205         cp->cpu_m.mcpu_ht.ch_sibstate = CS_MK(CM_THREAD, GLOBAL_ZONEID);
206         ASSERT3P(cp->cpu_m.mcpu_ht.ch_sib, ==, NULL);
207         cp->cpu_m.mcpu_ht.ch_sib = ht_find_sibling(cp);

209         intr_restore(flags);
210         thread_affinity_clear(curthread);
211     } while ((cp = cp->cpu_next_onln) != scp);

213     mutex_exit(&cpu_lock);
214 }

216 /*
217  * We're adding an interrupt handler of some kind at the given PIL. If this
218  * happens to be the same PIL as XC_CPUPUKE_PIL, then we need to disable our
219  * pil_needs_kick() optimization, as there is now potentially an unsafe
220  * interrupt handler at that PIL. This typically won't occur, so we're not that
221  * careful about what's actually getting added, which CPU it's on, or if it gets
222  * removed. This also presumes that softints can't cover our empty_pil.
223  */
224 void
225 ht_intr_alloc_pil(uint_t pil)
226 {
227     ASSERT(pil <= PIL_MAX);

229     if (empty_pil == pil)
230         empty_pil = PIL_MAX + 1;
231 }

233 /*
234  * If our sibling is also a VCPU thread from a different zone, we need one of
235  * them to give up, otherwise they will just battle each other for exclusion
236  * until they exhaust their quantum.
237  *
238  * We arbitrate between them by dispatch priority: clearly, a higher-priority
239  * thread deserves to win the acquisition. However, under CPU load, it'll be
240  * very common to see both threads with ->t_pri == 1. If so, we'll break the
241  * tie by cpu_id (which is hopefully arbitrary enough).
242  *
243  * If we lose, the VMM code will take this as a hint to call
244  * thread_affinity_set(CPU_BEST), which will likely migrate the VCPU thread
245  * somewhere else.
246  *
247  * Note that all of this state examination is racy, as we don't own any locks
248  * here.
249  */
250 static boolean_t
251 yield_to_vcpu(cpu_t *sib, zoneid_t zoneid)
252 {
253     cpu_ht_t *sibht = &sib->cpu_m.mcpu_ht;
254     uint64_t sibstate = sibht->ch_state;

256     /*
257      * If we're likely just waiting for an interrupt, don't yield.

```

```

258     */
259     if (sibht->ch_intr_depth != 0)
260         return (B_FALSE);

262     /*
263     * We're only interested in VCPUs from a different zone.
264     */
265     if (CS_MARK(sibstate) < CM_VCPU || CS_ZONE(sibstate) == zoneid)
266         return (B_FALSE);

268     if (curthread->t_pri < sib->cpu_dispatch_pri)
269         return (B_TRUE);

271     if (curthread->t_pri == sib->cpu_dispatch_pri &&
272         CPU->cpu_id < sib->cpu_id)
273         return (B_TRUE);

275     return (B_FALSE);
276 }

278 static inline boolean_t
279 sibling_compatible(cpu_ht_t *sibht, zoneid_t zoneid)
280 {
281     uint64_t sibstate = sibht->ch_state;

283     if (sibht->ch_intr_depth != 0)
284         return (B_FALSE);

286     if (CS_MARK(sibstate) == CM_UNSAFE)
287         return (B_FALSE);

289     if (CS_MARK(sibstate) == CM_IDLE)
290         return (B_TRUE);

292     return (CS_ZONE(sibstate) == zoneid);
293 }

295 int
296 ht_acquire(void)
297 {
298     clock_t wait = ht_acquire_wait_time;
299     cpu_ht_t *ht = &CPU->cpu_m.mcpu_ht;
300     zoneid_t zoneid = getzoneid();
301     cpu_ht_t *sibht;
302     int ret = 0;

304     ASSERT(!interrupts_enabled());

306     if (ht->ch_sib == NULL) {
307         /* For the "sequential" L1TF case. */
308         spec_llid_flush();
309         return (1);
310     }

312     sibht = &ht->ch_sib->cpu_m.mcpu_ht;

314     /* A VCPU thread should never change zone. */
315     ASSERT3U(CS_ZONE(ht->ch_state), ==, zoneid);
316     ASSERT3U(CS_MARK(ht->ch_state), ==, CM_VCPU);
317     ASSERT3U(zoneid, !=, GLOBAL_ZONEID);
318     ASSERT3U(curthread->t_preempt, >=, 1);
319     ASSERT(curthread->t_schedflag & TS_VCPU);

321     while (ret == 0 && wait > 0) {
323         if (yield_to_vcpu(ht->ch_sib, zoneid)) {

```

```

324         ret = -1;
325         break;
326     }

328     if (sibling_compatible(sibht, zoneid)) {
329         lock_set(&sibht->ch_lock);

331         if (sibling_compatible(sibht, zoneid)) {
332             ht->ch_state = CS_MK(CM_POISONED, zoneid);
333             sibht->ch_sibstate = CS_MK(CM_POISONED, zoneid);
334             membar_enter();
335             ret = 1;
336         }

338         lock_clear(&sibht->ch_lock);
339     } else {
340         drv_usecwait(10);
341         wait -= 10;
342     }
343 }

345 DTRACE_PROBE4(ht_acquire, int, ret, uint64_t, sibht->ch_state,
346               uint64_t, sibht->ch_intr_depth, clock_t, wait);

348     if (ret == 1)
349         spec_llid_flush();

351     return (ret);
352 }

354 void
355 ht_release(void)
356 {
357     cpu_ht_t *ht = &CPU->cpu_m.mcpu_ht;
358     zoneid_t zoneid = getzoneid();
359     cpu_ht_t *sibht;

361     ASSERT(!interrupts_enabled());

363     if (ht->ch_sib == NULL)
364         return;

366     ASSERT3U(zoneid, !=, GLOBAL_ZONEID);
367     ASSERT3U(CS_ZONE(ht->ch_state), ==, zoneid);
368     ASSERT3U(CS_MARK(ht->ch_state), ==, CM_POISONED);
369     ASSERT3U(curthread->t_preempt, >=, 1);

371     sibht = &ht->ch_sib->cpu_m.mcpu_ht;

373     lock_set(&sibht->ch_lock);

375     ht->ch_state = CS_MK(CM_VCPU, zoneid);
376     sibht->ch_sibstate = CS_MK(CM_VCPU, zoneid);
377     membar_producer();

379     lock_clear(&sibht->ch_lock);
380 }

382 static void
383 ht_kick(cpu_ht_t *ht, zoneid_t zoneid)
384 {
385     uint64_t sibstate;

387     ASSERT(LOCK_HELD(&ht->ch_lock));
388     ASSERT(!interrupts_enabled());

```

```

390     poke_cpu(ht->ch_sib->cpu_id);
392     membar_consumer();
393     sibstate = ht->ch_sibstate;
395     if (CS_MARK(sibstate) != CM_POISONED || CS_ZONE(sibstate) == zoneid)
396         return;
398     lock_clear(&ht->ch_lock);
400     /*
401     * Spin until we can see the sibling has been kicked out or is otherwise
402     * OK.
403     */
404     for (;;) {
405         membar_consumer();
406         sibstate = ht->ch_sibstate;
408         if (CS_MARK(sibstate) != CM_POISONED ||
409             CS_ZONE(sibstate) == zoneid)
410             break;
412         SMT_PAUSE();
413     }
415     lock_set(&ht->ch_lock);
416 }
418 static boolean_t
419 pil_needs_kick(uint_t pil)
420 {
421     return (pil != empty_pil);
422 }
424 void
425 ht_begin_intr(uint_t pil)
426 {
427     ulong_t flags;
428     cpu_ht_t *ht;
430     ASSERT(pil <= PIL_MAX);
432     flags = intr_clear();
433     ht = &CPU->cpu_m.mcpu_ht;
435     if (ht->ch_sib == NULL) {
436         intr_restore(flags);
437         return;
438     }
440     if (atomic_inc_64_nv(&ht->ch_intr_depth) == 1 && pil_needs_kick(pil)) {
441         lock_set(&ht->ch_lock);
443         membar_consumer();
445         if (CS_MARK(ht->ch_sibstate) == CM_POISONED)
446             ht_kick(ht, GLOBAL_ZONEID);
448         lock_clear(&ht->ch_lock);
449     }
451     intr_restore(flags);
452 }
454 void
455 ht_end_intr(void)

```

```

456 {
457     ulong_t flags;
458     cpu_ht_t *ht;
460     flags = intr_clear();
461     ht = &CPU->cpu_m.mcpu_ht;
463     if (ht->ch_sib == NULL) {
464         intr_restore(flags);
465         return;
466     }
468     ASSERT3U(ht->ch_intr_depth, >, 0);
469     atomic_dec_64(&ht->ch_intr_depth);
471     intr_restore(flags);
472 }
474 static inline boolean_t
475 ht_need_kick(cpu_ht_t *ht, zoneid_t zoneid)
476 {
477     membar_consumer();
479     if (CS_MARK(ht->ch_sibstate) != CM_POISONED)
480         return (B_FALSE);
482     if (CS_MARK(ht->ch_state) == CM_UNSAFE)
483         return (B_TRUE);
485     return (CS_ZONE(ht->ch_sibstate) != zoneid);
486 }
488 void
489 ht_mark(void)
490 {
491     zoneid_t zoneid = getzoneid();
492     kthread_t *t = curthread;
493     ulong_t flags;
494     cpu_ht_t *ht;
495     cpu_t *cp;
497     flags = intr_clear();
499     cp = CPU;
500     ht = &cp->cpu_m.mcpu_ht;
502     if (ht->ch_sib == NULL) {
503         intr_restore(flags);
504         return;
505     }
507     lock_set(&ht->ch_lock);
509     /*
510     * If we were a nested interrupt and went through the resume_from_intr()
511     * path, we can now be resuming to a pinning interrupt thread; in which
512     * case, skip marking, until we later resume to a "real" thread.
513     */
514     if (ht->ch_intr_depth > 0) {
515         ASSERT3P(t->t_intr, !=, NULL);
517         if (ht_need_kick(ht, zoneid))
518             ht_kick(ht, zoneid);
519         goto out;
520     }

```

```

522     if (t == t->t_cpu->cpu_idle_thread) {
523         ASSERT3U(zoneid, ==, GLOBAL_ZONEID);
524         ht->ch_state = CS_MK(CM_IDLE, zoneid);
525     } else {
526         uint64_t state = CM_THREAD;

528         if (t->t_unsafe)
529             state = CM_UNSAFE;
530         else if (t->t_schedflag & TS_VCPU)
531             state = CM_VCPU;

533         ht->ch_state = CS_MK(state, zoneid);

535         if (ht_need_kick(ht, zoneid))
536             ht_kick(ht, zoneid);
537     }

539 out:
540     membar_producer();
541     lock_clear(&ht->ch_lock);
542     intr_restore(flags);
543 }

545 void
546 ht_begin_unsafe(void)
547 {
548     curthread->t_unsafe++;
549     ht_mark();
550 }

552 void
553 ht_end_unsafe(void)
554 {
555     ASSERT3U(curthread->t_unsafe, >, 0);
556     curthread->t_unsafe--;
557     ht_mark();
558 }

560 void
561 ht_mark_as_vcpu(void)
562 {
563     thread_lock(curthread);
564     curthread->t_schedflag |= TS_VCPU;
565     ht_mark();
566     thread_unlock(curthread);
567 }

569 boolean_t
570 ht_should_run(kthread_t *t, cpu_t *cp)
571 {
572     uint64_t sibstate;
573     cpu_t *sib;

575     if (t == t->t_cpu->cpu_idle_thread)
576         return (B_TRUE);

578     if ((sib = cp->cpu_m.mcpu_ht.ch_sib) == NULL)
579         return (B_TRUE);

581     sibstate = sib->cpu_m.mcpu_ht.ch_state;

583     if ((t->t_schedflag & TS_VCPU) {
584         if (CS_MARK(sibstate) == CM_IDLE)
585             return (B_TRUE);
586         if (CS_MARK(sibstate) == CM_UNSAFE)
587             return (B_FALSE);

```

```

588         return (CS_ZONE(sibstate) == ttozone(t)->zone_id);
589     }

591     if (CS_MARK(sibstate) < CM_VCPU)
592         return (B_TRUE);

594     return (CS_ZONE(sibstate) == ttozone(t)->zone_id);
595 }

597 pri_t
598 ht_adjust_cpu_score(kthread_t *t, struct cpu *cp, pri_t score)
599 {
600     if (ht_should_run(t, cp))
601         return (score);

603     /*
604      * If we're a VCPU thread scoring our current CPU, we are most likely
605      * asking to be rescheduled elsewhere after losing ht_acquire(). In
606      * this case, the current CPU is not a good choice, most likely, and we
607      * should go elsewhere.
608      */
609     if ((t->t_schedflag & TS_VCPU) && cp == t->t_cpu && score < 0)
610         return ((v.v_maxsyspri + 1) * 2);

612     return (score + 1);
613 }

```



```

*****
57826 Tue May 7 04:50:00 2019
new/usr/src/uts/i86pc/os/intr.c
OS-7125 Need mitigation of L1TF (CVE-2018-3646)
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
*****
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) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
24  * Copyright (c) 2018 Joyent, Inc. All rights reserved.
25 */

27 /*
28  * To understand the present state of interrupt handling on i86pc, we must
29  * first consider the history of interrupt controllers and our way of handling
30  * interrupts.
31  *
32  * History of Interrupt Controllers on i86pc
33  * -----
34  *
35  * Intel 8259 and 8259A
36  *
37  * The first interrupt controller that attained widespread use on i86pc was
38  * the Intel 8259(A) Programmable Interrupt Controller that first saw use with
39  * the 8086. It took up to 8 interrupt sources and combined them into one
40  * output wire. Up to 8 8259s could be slaved together providing up to 64 IRQs.
41  * With the switch to the 8259A, level mode interrupts became possible. For a
42  * long time on i86pc the 8259A was the only way to handle interrupts and it
43  * had its own set of quirks. The 8259A and its corresponding interval timer
44  * the 8254 are programmed using outb and inb instructions.
45  *
46  * Intel Advanced Programmable Interrupt Controller (APIC)
47  *
48  * Starting around the time of the introduction of the P6 family
49  * microarchitecture (i686) Intel introduced a new interrupt controller.
50  * Instead of having the series of slaved 8259A devices, Intel opted to outfit
51  * each processor with a Local APIC (lapic) and to outfit the system with at
52  * least one, but potentially more, I/O APICs (ioapic). The lapics and ioapics
53  * initially communicated over a dedicated bus, but this has since been
54  * replaced. Each physical core and even hyperthread currently contains its
55  * own local apic, which is not shared. There are a few exceptions for
56  * hyperthreads, but that does not usually concern us.
57  *
58  * Instead of talking directly to 8259 for status, sending End Of Interrupt
59  * (EOI), etc. a microprocessor now communicates directly to the lapic. This

```

```

60 * also allows for each microprocessor to be able to have independent controls.
61 * The programming method is different from the 8259. Consumers map the lapic
62 * registers into uncacheable memory to read and manipulate the state.
63 *
64 * The number of addressable interrupt vectors was increased to 256. However
65 * vectors 0-31 are reserved for the processor exception handling, leaving the
66 * remaining vectors for general use. In addition to hardware generated
67 * interrupts, the lapic provides a way for generating inter-processor
68 * interrupts (IPI) which are the basis for CPU cross calls and CPU pokes.
69 *
70 * AMD ended up implementing the Intel APIC architecture in lieu of their work
71 * with Cyrix.
72 *
73 * Intel x2apic
74 *
75 * The x2apic is an extension to the lapic which started showing up around the
76 * same time as the Sandy Bridge chipsets. It provides a new programming mode
77 * as well as new features. The goal of the x2apic is to solve a few problems
78 * with the previous generation of lapic and the x2apic is backwards compatible
79 * with the previous programming and model. The only downsides to using the
80 * backwards compatibility is that you are not able to take advantage of the new
81 * x2apic features.
82 *
83 * o The APIC ID is increased from an 8-bit value to a 32-bit value. This
84 * increases the maximum number of addressable physical processors beyond
85 * 256. This new ID is assembled in a similar manner as the information that
86 * is obtainable by the extended cpuid topology leaves.
87 *
88 * o A new means of generating IPIs was introduced.
89 *
90 * o Instead of memory mapping the registers, the x2apic only allows for
91 * programming it through a series of wrmsrs. This has important semantic
92 * side effects. Recall that the registers were previously all mapped to
93 * uncacheable memory which meant that all operations to the local apic were
94 * serializing instructions. With the switch to using wrmsrs this has been
95 * relaxed and these operations can no longer be assumed to be serializing
96 * instructions.
97 *
98 * Note for the rest of this we are only going to concern ourselves with the
99 * apic and x2apic which practically all of i86pc has been using now for
100 * quite some time.
101 *
102 * Interrupt Priority Levels
103 * -----
104 *
105 * On i86pc systems there are a total of fifteen interrupt priority levels
106 * (ipls) which range from 1-15. Level 0 is for normal processing and
107 * non-interrupt processing. To manipulate these values the family of spl
108 * functions (which date back to UNIX on the PDP-11) are used. Specifically,
109 * splr() to raise the priority level and splx() to lower it. One should not
110 * generally call setspl() directly.
111 *
112 * Both i86pc and the supported SPARC platforms honor the same conventions for
113 * the meaning behind these IPLs. The most important IPL is the platform's
114 * LOCK_LEVEL (0xa on i86pc). If a thread is above LOCK_LEVEL it _must_ not
115 * sleep on any synchronization object. The only allowed synchronization
116 * primitive is a mutex that has been specifically initialized to be a spin
117 * lock (see mutex_init(9F)). Another important level is DISP_LEVEL (0xb on
118 * i86pc). You must be at DISP_LEVEL if you want to control the dispatcher.
119 * The XC_HI_PIL is the highest level (0xf) and is used during cross-calls.
120 *
121 * Each interrupt that is registered in the system fires at a specific IPL.
122 * Generally most interrupts fire below LOCK_LEVEL.
123 *
124 * PSM Drivers
125 * -----

```

```

126 *
127 * We currently have three sets of PSM (platform specific module) drivers
128 * available. uppc, pcplusmp, and apix. uppc (uni-processor PC) is the original
129 * driver that interacts with the 8259A and 8254. In general, it is not used
130 * anymore given the prevalence of the apic.
131 *
132 * The system prefers to use the apix driver over the pcplusmp driver. The apix
133 * driver requires HW support for an x2apic. If there is no x2apic HW, apix
134 * will not be used. In general we prefer using the apix driver over the
135 * pcplusmp driver because it gives us much more flexibility with respect to
136 * interrupts. In the apix driver each local apic has its own independent set
137 * of interrupts, whereas the pcplusmp driver only has a single global set of
138 * interrupts. This is why pcplusmp only supports a finite number of interrupts
139 * per IPL -- generally 16, often less. The apix driver supports using either
140 * the x2apic or the local apic programming modes. The programming mode does not
141 * change the number of interrupts available, just the number of processors
142 * that we can address. For the apix driver, the x2apic mode is enabled if the
143 * system supports interrupt re-mapping, otherwise the module manages the
144 * x2apic in local mode.
145 *
146 * When there is no x2apic present, we default back to the pcplusmp PSM driver.
147 * In general, this is not problematic unless you have more than 256
148 * processors in the machine or you do not have enough interrupts available.
149 *
150 * Controlling Interrupt Generation on i86pc
151 * -----
152 *
153 * There are two different ways to manipulate which interrupts will be
154 * generated on i86pc. Each offers different degrees of control.
155 *
156 * The first is through the flags register (eflags and rflags on i386 and amd64
157 * respectively). The IF bit determines whether or not interrupts are enabled
158 * or disabled. This is manipulated in one of several ways. The most common way
159 * is through the cli and sti instructions. These clear the IF flag and set it,
160 * respectively, for the current processor. The other common way is through the
161 * use of the intr_clear and intr_restore functions.
162 *
163 * Assuming interrupts are not blocked by the IF flag, then the second form is
164 * through the Processor-Priority Register (PPR). The PPR is used to determine
165 * whether or not a pending interrupt should be delivered. If the ipl of the
166 * new interrupt is higher than the current value in the PPR, then the lapic
167 * will either deliver it immediately (if interrupts are not in progress) or it
168 * will deliver it once the current interrupt processing has issued an EOI. The
169 * highest unmasked interrupt will be the one delivered.
170 *
171 * The PPR register is based upon the max of the following two registers in the
172 * lapic, the TPR register (also known as CR8 on amd64) that can be used to
173 * mask interrupt levels, and the current vector. Because the pcplusmp module
174 * always sets TPR appropriately early in the do_interrupt path, we can usually
175 * just think that the PPR is the TPR. The pcplusmp module also issues an EOI
176 * once it has set the TPR, so higher priority interrupts can come in while
177 * we're servicing a lower priority interrupt.
178 *
179 * Handling Interrupts
180 * -----
181 *
182 * Interrupts can be broken down into three categories based on priority and
183 * source:
184 *
185 *   o High level interrupts
186 *   o Low level hardware interrupts
187 *   o Low level software interrupts
188 *
189 * High Level Interrupts
190 *
191 * High level interrupts encompasses both hardware-sourced and software-sourced

```

```

192 * interrupts. Examples of high level hardware interrupts include the serial
193 * console. High level software-sourced interrupts are still delivered through
194 * the local apic through IPis. This is primarily cross calls.
195 *
196 * When a high level interrupt comes in, we will raise the SPL and then pin the
197 * current lwp to the processor. We will use its lwp, but our own interrupt
198 * stack and process the high level interrupt in-situ. These handlers are
199 * designed to be very short in nature and cannot go to sleep, only block on a
200 * spin lock. If the interrupt has a lot of work to do, it must generate a
201 * low-priority software interrupt that will be processed later.
202 *
203 * Low level hardware interrupts
204 *
205 * Low level hardware interrupts start off like their high-level cousins. The
206 * current CPU contains a number of kernel threads (kthread_t) that can be used
207 * to process low level interrupts. These are shared between both low level
208 * hardware and software interrupts. Note that while we run with our
209 * kthread_t, we borrow the pinned threads lwp_t until such a time as we hit a
210 * synchronization object. If we hit one and need to sleep, then the scheduler
211 * will instead create the rest of what we need.
212 *
213 * Low level software interrupts
214 *
215 * Low level software interrupts are handled in a similar way as hardware
216 * interrupts, but the notification vector is different. Each CPU has a bitmask
217 * of pending software interrupts. We can notify a CPU to process software
218 * interrupts through a specific trap vector as well as through several
219 * checks that are performed throughout the code. These checks will look at
220 * processing software interrupts as we lower our spl.
221 *
222 * We attempt to process the highest pending software interrupt that we can
223 * which is greater than our current IPL. If none currently exist, then we move
224 * on. We process a software interrupt in a similar fashion to a hardware
225 * interrupt.
226 *
227 * Traditional Interrupt Flow
228 * -----
229 *
230 * The following diagram tracks the flow of the traditional uppc and pcplusmp
231 * interrupt handlers. The apix driver has its own version of do_interrupt().
232 * We come into the interrupt handler with all interrupts masked by the IF
233 * flag. This is because we set up the handler using an interrupt-gate, which
234 * is defined architecturally to have cleared the IF flag for us.
235 *
236 * +-----+ +-----+ +-----+
237 * |_interrupt() |---->| do_interrupt() |---->| *setlvl() |
238 * +-----+ +-----+ +-----+
239 *
240 *
241 *
242 *
243 *
244 *
245 *
246 *
247 *
248 *
249 *
250 *
251 *
252 *
253 *
254 *
255 *
256 *
257 *

```

```

+-----+ +-----+ +-----+
|_interrupt() |---->| do_interrupt() |---->| *setlvl() |
+-----+ +-----+ +-----+

```

```

+-----+ +-----+
|intr_thread_ | <-----+
|prolog()     |
+-----+

```

```

+-----+
|switch_sp_  |
|and_call()  |
+-----+

```

```

+-----+
|dispatch_h |
|hilevel()  |
+-----+

```

```

+-----+ +-----+
|switch_sp_ |
|and_call() |
+-----+

```

```

+-----+
|dispatch_  |
|hardint()  |
+-----+

```

```

258 * +-----+
259 * |         |
260 * +-----+ +-----+ +-----+
261 * |  sti  |->| av_dispatch_autovect |->| cli  |
262 * +-----+ +-----+ +-----+
263 *
264 *
265 *
266 *
267 *
268 *
269 *
270 * | intr_thread_ |
271 * |  epilog()  |
272 * +-----+
273 *
274 *
275 *
276 *
277 *
278 *
279 *
280 *
281 *
282 *
283 *
284 *
285 *
286 *
287 *
288 *
289 *
290 *
291 *
292 *
293 *
294 *
295 *
296 *
297 *
298 *
299 *
300 *
301 *
302 *
303 *
304 *
305 *
306 *
307 *
308 *
309 *
310 *
311 *
312 *
313 *
314 *
315 *
316 *
317 *
318 *
319 *
320 *
321 *
322 *
323 *

```

```

311 *      Calls made on Interrupt Stacks and Epilogue routines
312 *
313 * We use the switch_sp_and_call() assembly routine to switch our sp to the
314 * interrupt stacks and then call the appropriate dispatch function. In the
315 * case of interrupts which may block, softints and hardints, we always ensure
316 * that we are still on the interrupt thread when we call the epilog routine.
317 * This is not just important, it's necessary. If the interrupt thread blocked,
318 * we won't return from our switch_sp_and_call() function and instead we'll go
319 * through and set ourselves up to swtch() directly.
320 *
321 * New Interrupt Flow
322 * +-----+
323 *

```

```

324 * The apix module has its own interrupt path. This is done for various
325 * reasons. The first is that rather than having global interrupt vectors, we
326 * now have per-cpu vectors.
327 *
328 * The other substantial change is that the apix design does not use the TPR to
329 * mask interrupts below the current level. In fact, except for one special
330 * case, it does not use the TPR at all. Instead, it only uses the IF flag
331 * (cli/sti) to either block all interrupts or allow any interrupts to come in.
332 * The design is such that when interrupts are allowed to come in, if we are
333 * currently servicing a higher priority interrupt, the new interrupt is treated
334 * as pending and serviced later. Specifically, in the pcplumsp module's
335 * apix_intr_enter() the code masks interrupts at or below the current
336 * IPL using the TPR before sending EOI, whereas the apix module's
337 * apix_intr_enter() simply sends EOI.
338 *
339 * The one special case where the apix code uses the TPR is when it calls
340 * through the apix_reg_ops function pointer apix_write_task_reg in
341 * apix_init_intr() to initially mask all levels and then finally to enable all
342 * levels.
343 *
344 * Recall that we come into the interrupt handler with all interrupts masked
345 * by the IF flag. This is because we set up the handler using an
346 * interrupt-gate which is defined architecturally to have cleared the IF flag
347 * for us.
348 *
349 *
350 * +-----+ +-----+
351 * | _interrupt() |--->| apix_do_interrupt() |
352 * +-----+ +-----+
353 *
354 *
355 *
356 *
357 *
358 *
359 *
360 *
361 *
362 *
363 *
364 *
365 *
366 *
367 *
368 *
369 *
370 *
371 *
372 *
373 *
374 *
375 *
376 *
377 *
378 *
379 *
380 *
381 *
382 *
383 *
384 *
385 *
386 *
387 *
388 *
389 *

```

```

378 *
379 *
380 *
381 *
382 *
383 *
384 *
385 *
386 *
387 *
388 *
389 *

```

```

390 *
391 *
392 *
393 *
394 *
395 *
396 *
397 *
398 *
399 *
400 *
401 *
402 *
403 *
404 *
405 *
406 *
407 *
408 *
409 *
410 *
411 *
412 *
413 *
414 *
415 *
416 *
417 *
418 *
419 *
420 *
421 *
422 *
423 *
424 *
425 *
426 *
427 *
428 *
429 *
430 *
431 *
432 *
433 *
434 *
435 *
436 *
437 *
438 */

```

```

440 #include <sys/cpuvar.h>
441 #include <sys/cpu_event.h>
442 #include <sys/regset.h>
443 #include <sys/psw.h>
444 #include <sys/types.h>
445 #include <sys/thread.h>
446 #include <sys/system.h>
447 #include <sys/segments.h>
448 #include <sys/pcb.h>
449 #include <sys/trap.h>
450 #include <sys/ftrace.h>
451 #include <sys/traptrace.h>
452 #include <sys/clock.h>
453 #include <sys/panic.h>
454 #include <sys/disp.h>
455 #include <vm/seg_kp.h>

```

```

456 #include <sys/stack.h>
457 #include <sys/sysmacros.h>
458 #include <sys/cmn_err.h>
459 #include <sys/kstat.h>
460 #include <sys/smp_impldefs.h>
461 #include <sys/pool_pset.h>
462 #include <sys/zone.h>
463 #include <sys/bitmap.h>
464 #include <sys/archsystem.h>
465 #include <sys/machsystem.h>
466 #include <sys/ontrap.h>
467 #include <sys/x86_archext.h>
468 #include <sys/promif.h>
469 #include <sys/ht.h>
470 #include <vm/hat_i86.h>
471 #if defined(__xpv)
472 #include <sys/hypervisor.h>
473 #endif

```

```

475 /* If these fail, then the padding numbers in machcpuvar.h are wrong. */
476 #if !defined(__xpv)
477 #define MCOFF(member) \
478     (offsetof(cpu_t, cpu_m) + offsetof(struct machcpu, member))
479 CTASSERT(MCOFF(mcpu_pad) == MACHCPU_SIZE);
480 CTASSERT(MCOFF(mcpu_pad2) == MMU_PAGESIZE);
481 CTASSERT((MCOFF(mcpu_kpti) & 0xF) == 0);
474 #if defined(__amd64) && !defined(__xpv)
475 /* If this fails, then the padding numbers in machcpuvar.h are wrong. */
476 CTASSERT((offsetof(cpu_t, cpu_m) + offsetof(struct machcpu, mcpu_pad)) <
477     MMU_PAGESIZE);
478 CTASSERT((offsetof(cpu_t, cpu_m) + offsetof(struct machcpu, mcpu_kpti)) >=
479     MMU_PAGESIZE);
480 CTASSERT((offsetof(cpu_t, cpu_m) + offsetof(struct machcpu, mcpu_kpti_dbg)) <
481     2 * MMU_PAGESIZE);
482 CTASSERT((offsetof(cpu_t, cpu_m) + offsetof(struct machcpu, mcpu_pad2)) <
483     2 * MMU_PAGESIZE);
484 CTASSERT(((sizeof (struct kpti_frame)) & 0xF) == 0);
485 CTASSERT(((offsetof(cpu_t, cpu_m) +
486     offsetof(struct machcpu, mcpu_kpti_dbg)) & 0xF) == 0);
483 CTASSERT((offsetof(struct kpti_frame, kf_tr_rsp) & 0xF) == 0);
484 CTASSERT(MCOFF(mcpu_pad3) < 2 * MMU_PAGESIZE);
485 #endif

```

```

487 #if defined(__xpv) && defined(DEBUG)

```

```

489 /*
490 * This panic message is intended as an aid to interrupt debugging.
491 *
492 * The associated assertion tests the condition of enabling
493 * events when events are already enabled. The implication
494 * being that whatever code the programmer thought was
495 * protected by having events disabled until the second
496 * enable happened really wasn't protected at all ..
497 */

```

```

499 int ststipanic = 1; /* controls the debug panic check */
500 const char *stistmsg = "stisti";
501 ulong_t laststi[NCPU];

```

```

503 /*
504 * This variable tracks the last place events were disabled on each cpu
505 * it assists in debugging when asserts that interrupts are enabled trip.
506 */
507 ulong_t lastcli[NCPU];

```

```

509 #endif

```

```

511 void do_interrupt(struct regs *rp, trap_trace_rec_t *ttp);

513 void (*do_interrupt_common)(struct regs *, trap_trace_rec_t *) = do_interrupt;
514 uintptr_t (*get_intr_handler)(int, short) = NULL;

516 /*
517 * Set cpu's base SPL level to the highest active interrupt level
518 */
519 void
520 set_base_spl(void)
521 {
522     struct cpu *cpu = CPU;
523     uint16_t active = (uint16_t)cpu->cpu_intr_actv;

525     cpu->cpu_base_spl = active == 0 ? 0 : bsrw_insn(active);
526 }

528 /*
529 * Do all the work necessary to set up the cpu and thread structures
530 * to dispatch a high-level interrupt.
531 *
532 * Returns 0 if we're -not- already on the high-level interrupt stack,
533 * (and *must* switch to it), non-zero if we are already on that stack.
534 *
535 * Called with interrupts masked.
536 * The 'pil' is already set to the appropriate level for rp->r_trapno.
537 */
538 static int
539 hilevel_intr_prolog(struct cpu *cpu, uint_t pil, uint_t oldpil, struct regs *rp)
540 {
541     struct machcpu *mcpu = &cpu->cpu_m;
542     uint_t mask;
543     hrtime_t intrtime;
544     hrtime_t now = tsc_read();

546     ASSERT(pil > LOCK_LEVEL);

548     if (pil == CBE_HIGH_PIL) {
549         cpu->cpu_profile_pil = oldpil;
550         if (USERMODE(rp->r_cs)) {
551             cpu->cpu_profile_pc = 0;
552             cpu->cpu_profile_upc = rp->r_pc;
553             cpu->cpu_cpcprofile_pc = 0;
554             cpu->cpu_cpcprofile_upc = rp->r_pc;
555         } else {
556             cpu->cpu_profile_pc = rp->r_pc;
557             cpu->cpu_profile_upc = 0;
558             cpu->cpu_cpcprofile_pc = rp->r_pc;
559             cpu->cpu_cpcprofile_upc = 0;
560         }
561     }

563     mask = cpu->cpu_intr_actv & CPU_INTR_ACTV_HIGH_LEVEL_MASK;
564     if (mask != 0) {
565         int nestpil;

567         /*
568          * We have interrupted another high-level interrupt.
569          * Load starting timestamp, compute interval, update
570          * cumulative counter.
571          */
572         nestpil = bsrw_insn((uint16_t)mask);
573         ASSERT(nestpil < pil);
574         intrtime = now -
575             mcpu->pil_high_start[nestpil - (LOCK_LEVEL + 1)];

```

```

576         mcpu->intrstat[nestpil][0] += intrtime;
577         cpu->cpu_intracct[cpu->cpu_mstate] += intrtime;
578         /*
579          * Another high-level interrupt is active below this one, so
580          * there is no need to check for an interrupt thread. That
581          * will be done by the lowest priority high-level interrupt
582          * active.
583          */
584     } else {
585         kthread_t *t = cpu->cpu_thread;

587         /*
588          * See if we are interrupting a low-level interrupt thread.
589          * If so, account for its time slice only if its time stamp
590          * is non-zero.
591          */
592         if ((t->t_flag & T_INTR_THREAD) != 0 && t->t_intr_start != 0) {
593             intrtime = now - t->t_intr_start;
594             mcpu->intrstat[t->t_pil][0] += intrtime;
595             cpu->cpu_intracct[cpu->cpu_mstate] += intrtime;
596             t->t_intr_start = 0;
597         }
598     }

600     ht_begin_intr(pil);

602     /*
603      * Store starting timestamp in CPU structure for this PIL.
604      */
605     mcpu->pil_high_start[pil - (LOCK_LEVEL + 1)] = now;

607     ASSERT((cpu->cpu_intr_actv & (1 << pil)) == 0);

609     if (pil == 15) {
610         /*
611          * To support reentrant level 15 interrupts, we maintain a
612          * recursion count in the top half of cpu_intr_actv. Only
613          * when this count hits zero do we clear the PIL 15 bit from
614          * the lower half of cpu_intr_actv.
615          */
616         uint16_t *refcntp = (uint16_t *)&cpu->cpu_intr_actv + 1;
617         (*refcntp)++;
618     }

620     mask = cpu->cpu_intr_actv;

622     cpu->cpu_intr_actv |= (1 << pil);

624     return (mask & CPU_INTR_ACTV_HIGH_LEVEL_MASK);
625 }

627 /*
628 * Does most of the work of returning from a high level interrupt.
629 *
630 * Returns 0 if there are no more high level interrupts (in which
631 * case we must switch back to the interrupted thread stack) or
632 * non-zero if there are more (in which case we should stay on it).
633 *
634 * Called with interrupts masked
635 */
636 static int
637 hilevel_intr_epilog(struct cpu *cpu, uint_t pil, uint_t oldpil, uint_t vecnum)
638 {
639     struct machcpu *mcpu = &cpu->cpu_m;
640     uint_t mask;
641     hrtime_t intrtime;

```

```

642     hrttime_t now = tsc_read();
644     ASSERT(mcpu->mcpu_pri == pil);
646     cpu->cpu_stats.sys.intr[pil - 1]++;
648     ASSERT(cpu->cpu_intr_actv & (1 << pil));
650     if (pil == 15) {
651         /*
652          * To support reentrant level 15 interrupts, we maintain a
653          * recursion count in the top half of cpu_intr_actv. Only
654          * when this count hits zero do we clear the PIL 15 bit from
655          * the lower half of cpu_intr_actv.
656          */
657         uint16_t *refcntp = (uint16_t *)&cpu->cpu_intr_actv + 1;
659         ASSERT(*refcntp > 0);
661         if (--(*refcntp) == 0)
662             cpu->cpu_intr_actv &= ~(1 << pil);
663     } else {
664         cpu->cpu_intr_actv &= ~(1 << pil);
665     }
667     ASSERT(mcpu->pil_high_start[pil - (LOCK_LEVEL + 1)] != 0);
669     intrtime = now - mcpu->pil_high_start[pil - (LOCK_LEVEL + 1)];
670     mcpu->intrstat[pil][0] += intrtime;
671     cpu->cpu_intracct[cpu->cpu_mstate] += intrtime;
673     /*
674     * Check for lower-pil nested high-level interrupt beneath
675     * current one. If so, place a starting timestamp in its
676     * pil_high_start entry.
677     */
678     mask = cpu->cpu_intr_actv & CPU_INTR_ACTV_HIGH_LEVEL_MASK;
679     if (mask != 0) {
680         int nestpil;
682         /*
683         * find PIL of nested interrupt
684         */
685         nestpil = bsrw_insn((uint16_t)mask);
686         ASSERT(nestpil < pil);
687         mcpu->pil_high_start[nestpil - (LOCK_LEVEL + 1)] = now;
688         /*
689         * (Another high-level interrupt is active below this one,
690         * so there is no need to check for an interrupt
691         * thread. That will be done by the lowest priority
692         * high-level interrupt active.)
693         */
694     } else {
695         /*
696         * Check to see if there is a low-level interrupt active.
697         * If so, place a starting timestamp in the thread
698         * structure.
699         */
700         kthread_t *t = cpu->cpu_thread;
702         if (t->t_flag & T_INTR_THREAD)
703             t->t_intr_start = now;
704     }
706     ht_end_intr();

```

```

708     mcpu->mcpu_pri = oldpil;
709     (void) (*setlvlx)(oldpil, vecnum);
711     return (cpu->cpu_intr_actv & CPU_INTR_ACTV_HIGH_LEVEL_MASK);
712 }
714 /*
715 * Set up the cpu, thread and interrupt thread structures for
716 * executing an interrupt thread. The new stack pointer of the
717 * interrupt thread (which *must* be switched to) is returned.
718 */
719 static caddr_t
720 intr_thread_prolog(struct cpu *cpu, caddr_t stackptr, uint_t pil)
721 {
722     struct machcpu *mcpu = &cpu->cpu_m;
723     kthread_t *t, *volatile it;
724     hrttime_t now = tsc_read();
726     ASSERT(pil > 0);
727     ASSERT((cpu->cpu_intr_actv & (1 << pil)) == 0);
728     cpu->cpu_intr_actv |= (1 << pil);
730     /*
731     * Get set to run an interrupt thread.
732     * There should always be an interrupt thread, since we
733     * allocate one for each level on each CPU.
734     *
735     * t_intr_start could be zero due to cpu_intr_swch_enter.
736     */
737     t = cpu->cpu_thread;
738     if ((t->t_flag & T_INTR_THREAD) && t->t_intr_start != 0) {
739         hrttime_t intrtime = now - t->t_intr_start;
740         mcpu->intrstat[t->t_pil][0] += intrtime;
741         cpu->cpu_intracct[cpu->cpu_mstate] += intrtime;
742         t->t_intr_start = 0;
743     }
745     ASSERT(SA((uintptr_t)stackptr) == (uintptr_t)stackptr);
747     t->t_sp = (uintptr_t)stackptr; /* mark stack in curthread for resume */
749     /*
750     * unlink the interrupt thread off the cpu
751     *
752     * Note that the code in kpcp_overflow_intr -relies- on the
753     * ordering of events here - in particular that t->t_lwp of
754     * the interrupt thread is set to the pinned thread *before*
755     * curthread is changed.
756     */
757     it = cpu->cpu_intr_thread;
758     cpu->cpu_intr_thread = it->t_link;
759     it->t_intr = t;
760     it->t_lwp = t->t_lwp;
762     /*
763     * (threads on the interrupt thread free list could have state
764     * preset to TS_ONPROC, but it helps in debugging if
765     * they're TS_FREE.)
766     */
767     it->t_state = TS_ONPROC;
769     cpu->cpu_thread = it; /* new curthread on this cpu */
770     ht_begin_intr(pil);
772     it->t_pil = (uchar_t)pil;
773     it->t_pri = intr_pri + (pri_t)pil;

```

```

774         it->t_intr_start = now;
775     }
776     return (it->t_stk);
777 }

780 #ifdef DEBUG
781 int intr_thread_cnt;
782 #endif

784 /*
785  * Called with interrupts disabled
786  */
787 static void
788 intr_thread_epilog(struct cpu *cpu, uint_t vec, uint_t oldpil)
789 {
790     struct machcpu *mcpu = &cpu->cpu_m;
791     kthread_t *t;
792     kthread_t *it = cpu->cpu_thread;    /* curthread */
793     uint_t pil, basespl;
794     hrtime_t intrtime;
795     hrtime_t now = tsc_read();

797     pil = it->t_pil;
798     cpu->cpu_stats.sys.intr[pil - 1]++;

800     ASSERT(it->t_intr_start != 0);
801     intrtime = now - it->t_intr_start;
802     mcpu->intrstat[pil][0] += intrtime;
803     cpu->cpu_intracct[cpu->cpu_mstate] += intrtime;

805     ASSERT(cpu->cpu_intr_actv & (1 << pil));
806     cpu->cpu_intr_actv &= ~(1 << pil);

808     /*
809     * If there is still an interrupted thread underneath this one
810     * then the interrupt was never blocked and the return is
811     * fairly simple.  Otherwise it isn't.
812     */
813     if ((t = it->t_intr) == NULL) {
814         /*
815          * The interrupted thread is no longer pinned underneath
816          * the interrupt thread.  This means the interrupt must
817          * have blocked, and the interrupted thread has been
818          * unpinned, and has probably been running around the
819          * system for a while.
820          *
821          * Since there is no longer a thread under this one, put
822          * this interrupt thread back on the CPU's free list and
823          * resume the idle thread which will dispatch the next
824          * thread to run.
825          */
826     #ifdef DEBUG
827         intr_thread_cnt++;
828     #endif
829     cpu->cpu_stats.sys.intrblk++;
830     /*
831     * Set CPU's base SPL based on active interrupts bitmask
832     */
833     set_base_spl();
834     basespl = cpu->cpu_base_spl;
835     mcpu->mcpu_pri = basespl;
836     (*setlvlx)(basespl, vec);
837     (void) splhigh();
838     sti();
839     it->t_state = TS_FREE;

```

```

840         /*
841          * Return interrupt thread to pool
842          */
843         it->t_link = cpu->cpu_intr_thread;
844         cpu->cpu_intr_thread = it;
845         swtch();
846         panic("intr_thread_epilog: swtch returned");
847         /*NOTREACHED*/
848     }

850     /*
851     * Return interrupt thread to the pool
852     */
853     it->t_link = cpu->cpu_intr_thread;
854     cpu->cpu_intr_thread = it;
855     it->t_state = TS_FREE;

857     basespl = cpu->cpu_base_spl;
858     pil = MAX(oldpil, basespl);
859     mcpu->mcpu_pri = pil;
860     (*setlvlx)(pil, vec);
861     t->t_intr_start = now;
862     ht_end_intr();
863     cpu->cpu_thread = t;
864 }

    unchanged_portion_omitted

956 static caddr_t
957 dosoftint_prolog(
958     struct cpu *cpu,
959     caddr_t stackptr,
960     uint32_t st_pending,
961     uint_t oldpil)
962 {
963     kthread_t *t, *volatile it;
964     struct machcpu *mcpu = &cpu->cpu_m;
965     uint_t pil;
966     hrtime_t now;

968 top:
969     ASSERT(st_pending == mcpu->mcpu_softinfo.st_pending);

971     pil = bsrw_insn((uint16_t)st_pending);
972     if (pil <= oldpil || pil <= cpu->cpu_base_spl)
973         return (0);

975     /*
976     * XX64 Sigh.
977     *
978     * This is a transliteration of the i386 assembler code for
979     * soft interrupts.  One question is "why does this need
980     * to be atomic?"  One possible race is -other- processors
981     * posting soft interrupts to us in set_pending() i.e. the
982     * CPU might get preempted just after the address computation,
983     * but just before the atomic transaction, so another CPU would
984     * actually set the original CPU's st_pending bit.  However,
985     * it looks like it would be simpler to disable preemption there.
986     * Are there other races for which preemption control doesn't work?
987     *
988     * The i386 assembler version -also- checks to see if the bit
989     * being cleared was actually set; if it wasn't, it rechecks
990     * for more.  This seems a bit strange, as the only code that
991     * ever clears the bit is -this- code running with interrupts
992     * disabled on -this- CPU.  This code would probably be cheaper:
993     *
994     * atomic_and_32((uint32_t *)&mcpu->mcpu_softinfo.st_pending,

```

```

995     * ~(1 << pil));
996     *
997     * and t->t_preempt--/++ around set_pending() even cheaper,
998     * but at this point, correctness is critical, so we slavishly
999     * emulate the i386 port.
1000    */
1001    if (atomic_btr32((uint32_t *)
1002        &mcpu->mcpu_software.st_pending, pil) == 0) {
1003        st_pending = mcpu->mcpu_software.st_pending;
1004        goto top;
1005    }
1007    mcpu->mcpu_pri = pil;
1008    (*setspl)(pil);
1010    now = tsc_read();
1012    /*
1013     * Get set to run interrupt thread.
1014     * There should always be an interrupt thread since we
1015     * allocate one for each level on the CPU.
1016     */
1017    it = cpu->cpu_intr_thread;
1018    cpu->cpu_intr_thread = it->t_link;
1020    /* t_intr_start could be zero due to cpu_intr_swch_enter. */
1021    t = cpu->cpu_thread;
1022    if ((t->t_flag & T_INTR_THREAD) && t->t_intr_start != 0) {
1023        hrtime_t intrtime = now - t->t_intr_start;
1024        mcpu->intrstat[pil][0] += intrtime;
1025        cpu->cpu_intracct[cpu->cpu_mstate] += intrtime;
1026        t->t_intr_start = 0;
1027    }
1029    /*
1030     * Note that the code in kpcpc_overflow_intr -relies- on the
1031     * ordering of events here - in particular that t->t_lwp of
1032     * the interrupt thread is set to the pinned thread *before*
1033     * curthread is changed.
1034     */
1035    it->t_lwp = t->t_lwp;
1036    it->t_state = TS_ONPROC;
1038    /*
1039     * Push interrupted thread onto list from new thread.
1040     * Set the new thread as the current one.
1041     * Set interrupted thread's T_SP because if it is the idle thread,
1042     * resume() may use that stack between threads.
1043     */
1045    ASSERT(SA((uintptr_t)stackptr) == (uintptr_t)stackptr);
1046    t->t_sp = (uintptr_t)stackptr;
1048    it->t_intr = t;
1049    cpu->cpu_thread = it;
1050    ht_begin_intr(pil);
1052    /*
1053     * Set bit for this pil in CPU's interrupt active bitmask.
1054     */
1055    ASSERT((cpu->cpu_intr_actv & (1 << pil)) == 0);
1056    cpu->cpu_intr_actv |= (1 << pil);
1058    /*
1059     * Initialize thread priority level from intr_pri
1060     */

```

```

1061     it->t_pil = (uchar_t)pil;
1062     it->t_pri = (pri_t)pil + intr_pri;
1063     it->t_intr_start = now;
1065     return (it->t_stk);
1066 }
1068 static void
1069 dosoftint_epilog(struct cpu *cpu, uint_t oldpil)
1070 {
1071     struct machcpu *mcpu = &cpu->cpu_m;
1072     kthread_t *t, *it;
1073     uint_t pil, basespl;
1074     hrtime_t intrtime;
1075     hrtime_t now = tsc_read();
1077     it = cpu->cpu_thread;
1078     pil = it->t_pil;
1080     cpu->cpu_stats.sys.intr[pil - 1]++;
1082     ASSERT(cpu->cpu_intr_actv & (1 << pil));
1083     cpu->cpu_intr_actv &= ~(1 << pil);
1084     intrtime = now - it->t_intr_start;
1085     mcpu->intrstat[pil][0] += intrtime;
1086     cpu->cpu_intracct[cpu->cpu_mstate] += intrtime;
1088     /*
1089     * If there is still an interrupted thread underneath this one
1090     * then the interrupt was never blocked and the return is
1091     * fairly simple. Otherwise it isn't.
1092     */
1093     if ((t = it->t_intr) == NULL) {
1094         /*
1095          * Put thread back on the interrupt thread list.
1096          * This was an interrupt thread, so set CPU's base SPL.
1097          */
1098         set_base_spl();
1099         it->t_state = TS_FREE;
1100         it->t_link = cpu->cpu_intr_thread;
1101         cpu->cpu_intr_thread = it;
1102         (void) splhigh();
1103         sti();
1104         swch();
1105         /*NOTREACHED*/
1106         panic("dosoftint_epilog: swch returned");
1107     }
1108     it->t_link = cpu->cpu_intr_thread;
1109     cpu->cpu_intr_thread = it;
1110     it->t_state = TS_FREE;
1111     ht_end_intr();
1112     cpu->cpu_thread = t;
1114     if (t->t_flag & T_INTR_THREAD)
1115         t->t_intr_start = now;
1116     basespl = cpu->cpu_base_spl;
1117     pil = MAX(oldpil, basespl);
1118     mcpu->mcpu_pri = pil;
1119     (*setspl)(pil);
1120 }

```

unchanged_portion_omitted

new/usr/src/uts/i86pc/sys/Makefile

1

```
*****
2008 Tue May 7 04:50:00 2019
new/usr/src/uts/i86pc/sys/Makefile
OS-7125 Need mitigation of LITF (CVE-2018-3646)
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
*****
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 2009 Sun Microsystems, Inc. All rights reserved.
23 # Use is subject to license terms.
24 # Copyright 2018 Joyent, Inc.
25 #
26 # uts/i86pc/sys/Makefile
27 #
28 UTBASE = ../../

30 #
31 # include global definitions
32 #
33 include ../Makefile.i86pc

35 #
36 # Override defaults.
37 #
38 FILEMODE = 644

40 HDRS= \
41     acpidev.h      \
42     amd_iommu.h   \
43     asm_misc.h    \
44     clock.h       \
45     cram.h        \
46     ddi_subrdefs.h \
47     debug_info.h  \
48     fastboot.h   \
49     ht.h          \
50     mach_mmu.h    \
51     machclock.h  \
52     machcpuvar.h \
53     machparam.h  \
54     machprivregs.h \
55     machsystem.h \
56     machthread.h \
57     memnode.h    \
58     pc_mmu.h     \
59     psm.h        \
```

new/usr/src/uts/i86pc/sys/Makefile

2

```
60     psm_defs.h    \
61     psm_modctl.h  \
62     psm_types.h   \
63     rm_platter.h  \
64     smp_impldefs.h \
65     sbd_ioctl.h   \
66     vm_machparam.h \
67     x_call.h      \
68     xc_levels.h   \
69     xsvc.h

71 ROOTHDRS=      $(HDRS:%=$(USR_PSM_ISYS_DIR)/%)

73 ROOTDIR=       $(ROOT)/usr/share/src
74 ROOTDIRS=      $(ROOTDIR)/uts $(ROOTDIR)/uts/$(PLATFORM)

76 ROOTLINK=      $(ROOTDIR)/uts/$(PLATFORM)/sys
77 LINKDEST=      ../../../../platform/$(PLATFORM)/include/sys

79 CHECKHDRS=     $(HDRS:%.h=%.check)

81 .KEEP_STATE:

83 .PARALLEL:     $(CHECKHDRS) $(ROOTHDRS)

85 install_h:     $(ROOTDIRS) .WAIT $(ROOTHDRS) $(ROOTLINK)

87 check:         $(CHECKHDRS)

89 $(ROOTDIRS):
90     $(INS.dir)

92 $(ROOTLINK):   $(ROOTDIRS)
93     -$(RM) -r $@; $(SYMLINK) $(LINKDEST) $@

95 FRC:

97 include ../../Makefile.targ
```

new/usr/src/uts/i86pc/sys/ht.h

1

1094 Tue May 7 04:50:00 2019

new/usr/src/uts/i86pc/sys/ht.h

OS-7125 Need mitigation of LITF (CVE-2018-3646)

Reviewed by: Robert Mustacchi <rm@joyent.com>

Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>

```
1 /*
2  * This file and its contents are supplied under the terms of the
3  * Common Development and Distribution License ("CDDL"), version 1.0.
4  * You may only use this file in accordance with the terms of version
5  * 1.0 of the CDDL.
6  *
7  * A full copy of the text of the CDDL should have accompanied this
8  * source. A copy of the CDDL is also available via the Internet at
9  * http://www.illumos.org/license/CDDL.
10 */

12 /*
13  * Copyright 2018 Joyent, Inc.
14 */

16 #ifndef _SYS_HT_H
17 #define _SYS_HT_H

19 #include <sys/types.h>
20 #include <sys/thread.h>

22 #ifdef __cplusplus
23 extern "C" {
24 #endif

26 struct cpu;

28 extern void ht_init(void);
29 extern void ht_intr_alloc_pil(uint_t);

31 extern int ht_acquire(void);
32 extern void ht_release(void);
33 extern void ht_mark(void);
34 extern void ht_begin_unsafe(void);
35 extern void ht_end_unsafe(void);
36 extern void ht_begin_intr(uint_t);
37 extern void ht_end_intr(void);
38 extern void ht_mark_as_vcpu(void);

40 extern boolean_t ht_should_run(kthread_t *, struct cpu *);
41 extern pri_t ht_adjust_cpu_score(kthread_t *, struct cpu *, pri_t);

43 #ifdef __cplusplus
44 }
45 #endif

47 #endif /* _SYS_HT_H */
```

```

*****
7797 Tue May 7 04:50:00 2019
new/usr/src/uts/i86pc/sys/machcpuvar.h
OS-7125 Need mitigation of L1TF (CVE-2018-3646)
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
*****
unchanged_portion_omitted_

137 typedef struct cpu_ht {
138     lock_t ch_lock;
139     char ch_pad[56];
140     struct cpu *ch_sib;
141     volatile uint64_t ch_intr_depth;
142     volatile uint64_t ch_state;
143     volatile uint64_t ch_sibstate;
144 } cpu_ht_t;

146 /*
147  * This first value, MACHCPU_SIZE is the size of all the members in the cpu_t
148  * AND struct machcpu, before we get to the mcpu_pad and the kpti area.
149  * The KPTI is used to contain per-CPU data that is visible in both sets of
150  * page-tables, and hence must be page-aligned and page-sized. See
151  * hat_pcp_setup().
152  *
153  * There are CTASSERTs in os/intr.c that verify this all works out.
154  * There is a CTASSERT in os/intr.c that checks these numbers.
155  */
155 #define MACHCPU_SIZE    (1568 + 688)
156 #define MACHCPU_SIZE    (572 + 1584)
156 #define MACHCPU_PAD     (MMU_PAGESIZE - MACHCPU_SIZE)
157 #define MACHCPU_PAD2    (MMU_PAGESIZE - 16 - 3 * sizeof (struct kpti_frame))

159 struct machcpu {
160     /*
161      * x_call fields - used for interprocessor cross calls
162      */
163     struct xc_msg *xc_msgbox;
164     struct xc_msg *xc_free;
165     xc_data_t xc_data;
166     uint32_t xc_wait_cnt;
167     volatile uint32_t xc_work_cnt;

169     int mcpu_nodeid; /* node-id */
170     int mcpu_pri; /* CPU priority */

172     struct hat *mcpu_current_hat; /* cpu's current hat */

174     struct hat_cpu_info *mcpu_hat_info;

176     volatile ulong_t mcpu_tlb_info;

178     /* i86 hardware table addresses that cannot be shared */

180     user_desc_t *mcpu_gdt; /* GDT */
181     gate_desc_t *mcpu_idt; /* current IDT */

183     tss_t *mcpu_tss; /* TSS */
184     void *mcpu_ldt;
185     size_t mcpu_ldt_len;

187     kmutex_t mcpu_ppaddr_mutex;
188     caddr_t mcpu_caddr1; /* per cpu CADDR1 */
189     caddr_t mcpu_caddr2; /* per cpu CADDR2 */
190     uint64_t mcpu_caddr1pte;
191     uint64_t mcpu_caddr2pte;

```

```

193     struct softint mcpu_softinfo;
194     uint64_t pil_high_start[HIGH_LEVELS];
195     uint64_t intrstat[PIL_MAX + 1][2];

197     struct cpuid_info *mcpu_cpi;

199 #if defined(__amd64)
200     greg_t mcpu_rtmp_rsp; /* syscall: temporary %rsp stash */
201     greg_t mcpu_rtmp_r15; /* syscall: temporary %r15 stash */
202 #endif

204     struct vcpu_info *mcpu_vcpu_info;
205     uint64_t mcpu_gdtpa; /* hypervisor: GDT physical address */

207     uint16_t mcpu_intr_pending; /* hypervisor: pending intrprt levels */
208     uint16_t mcpu_ec_mbox; /* hypervisor: evtchn_dev mailbox */
209     struct xen_evt_data *mcpu_evt_pending; /* hypervisor: pending events */

211     volatile uint32_t *mcpu_mwait; /* MONITOR/MWAIT buffer */
212     void (*mcpu_idle_cpu)(void); /* idle function */
213     uint16_t mcpu_idle_type; /* CPU next idle type */
214     uint16_t max_cstates; /* supported max cstates */

216     struct cpu_ucode_info *mcpu_ucode_info;

218     void *mcpu_pm_mach_state;
219     struct cmi_hdl *mcpu_cmi_hdl;
220     void *mcpu_mach_ctx_ptr;

222     /*
223      * A stamp that is unique per processor and changes
224      * whenever an interrupt happens. Useful for detecting
225      * if a section of code gets interrupted.
226      * The high order 16 bits will hold the cpu->cpu_id.
227      * The low order bits will be incremented on every interrupt.
228      */
229     volatile uint32_t mcpu_istamp;

231     cpu_ht_t mcpu_ht;

233     char mcpu_pad[MACHCPU_PAD];

235     /* This is the start of the page */
236     char mcpu_pad2[MACHCPU_PAD2];
237     struct kpti_frame mcpu_kpti;
238     struct kpti_frame mcpu_kptiflt;
239     struct kpti_frame mcpu_kpti_dbg;
240     char mcpu_pad3[16];
241 };
unchanged_portion_omitted_

```

```

*****
5626 Tue May 7 04:50:01 2019
new/usr/src/uts/i86xpv/Makefile.files
OS-7125 Need mitigation of LITF (CVE-2018-3646)
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
*****
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 #
23 # Copyright (c) 2007, 2010, Oracle and/or its affiliates. All rights reserved.
24 #
25 # Copyright 2018 Joyent, Inc.
26 #
27 #
28 # This Makefile defines file modules in the directory uts/i86xpv
29 # and its children. These are the source files which are i86xpv
30 # "implementation architecture" dependent.
31 #
32 #
33 #
34 # object lists
35 #
36 CORE_OBJJS += \
37     acpi_stubs.o \
38     balloon.o \
39     biosdisk.o \
40     cbe.o \
41     cmi.o \
42     cmi_hw.o \
43     cms.o \
44     confunix.o \
45     cpuid.o \
46     cpuid_subr.o \
47     cpupm.o \
48     cpupm_mach.o \
49     dis_tables.o \
50     ddi_impl.o \
51     dtrace_subr.o \
52     dvma.o \
53     fakebop.o \
54     fpu_subr.o \
55     fastboot.o \
56     fb_swtd.o \
57     graphics.o \
58     hardclk.o \
59     hat_i86.o \

```

```

60     hat_kdi.o \
61     hment.o \
62     hold_page.o \
63     hrtimers.o \
64     ht.o \
65     htable.o \
66     i86_mmu.o \
67     ibft.o \
68     instr_size.o \
69     intr.o \
70     kboot_mmu.o \
71     kdi_idt.o \
72     kdi_idthdl.o \
73     kdi_asm.o \
74     lgrpplat.o \
75     mach_kdi.o \
76     mach_sysconfig.o \
77     machdep.o \
78     mem_config_stubs.o \
79     memnode.o \
80     microcode.o \
81     mlsetup.o \
82     mp_call.o \
83     mp_implfuncs.o \
84     mp_machdep.o \
85     mp_startup.o \
86     memscrub.o \
87     notes.o \
88     pci_bios.o \
89     pci_cfgacc.o \
90     pci_cfgacc_x86.o \
91     pci_cfgspace.o \
92     pci_mech1.o \
93     pci_mech2.o \
94     pci_neptune.o \
95     pci_orion.o \
96     pmem.o \
97     ppage.o \
98     startup.o \
99     ssp.o \
100     xpv_timestamp.o \
101     todpc_subr.o \
102     trap.o \
103     vm_machdep.o \
104     x_call.o \
105 #
106 #
107 # Add the SMBIOS subsystem object files directly to the list of objects
108 # built into unix itself; this is all common code except for smb_dev.c.
109 #
110 CORE_OBJJS += $(SMBIOS_OBJJS)
111 #
112 #
113 # These get compiled twice:
114 # - once in the dboot (direct boot) identity mapped code
115 # - once for use during early startup in unix
116 #
117 BOOT_DRIVER_OBJJS = \
118     boot_console.o \
119     boot_keyboard.o \
120     boot_keyboard_table.o \
121     boot_mmu.o \
122     boot_vga.o \
123     boot_fb.o \
124     boot_xconsole.o \
125     dboot_multiboot2.o \

```

```

126     $(FONT_OBJS)
128 CORE_OBJS += $(BOOT_DRIVER_OBJS)

130 #
131 # Extra XEN files separated out for now.
132 #
133 CORE_OBJS += \
134     cpr_driver.o           \
135     evtchn.o              \
136     gnttab.o              \
137     hypercall.o          \
138     hyperevent.o         \
139     hypersubr.o          \
140     mp_xen.o              \
141     panic_asm.o          \
142     xenguest.o           \
143     xenbus_client.o      \
144     xenbus_comms.o       \
145     xenbus_probe.o       \
146     xenbus_xs.o          \
147     xen_machdep.o        \
148     xen_mmu.o            \
149     xpvtod.o             \
150     xvdi.o

152 #
153 #     locore.o is special. It must be the first file relocated so that it
154 #     it is relocated just where its name implies.
155 #
156 SPECIAL_OBJS_32 += \
157     locore.o             \
158     fast_trap_asm.o     \
159     interrupt.o         \
160     syscall_asm.o

162 SPECIAL_OBJS_64 += \
163     locore.o             \
164     fast_trap_asm.o     \
165     interrupt.o         \
166     syscall_asm_amd64.o \
167     kpti_trampoline.o

169 SPECIAL_OBJS += $(SPECIAL_OBJS_$(CLASS))

171 #
172 # object files used to boot into full kernel
173 #
174 DBOOT_OBJS_32 = muldiv.o

176 DBOOT_OBJS_64 =

178 DBOOT_OBJS += \
179     dboot_asm.o         \
180     dboot_printf.o     \
181     dboot_startkern.o  \
182     dboot_xen.o        \
183     hypercall.o        \
184     hypersubr.o        \
185     memcpy.o           \
186     memset.o           \
187     string.o           \
188     $(BOOT_DRIVER_OBJS) \
189     $(DBOOT_OBJS_$(CLASS))

191 #

```

```

192 #     driver & misc modules
193 #
194 BALLOON_OBJS += balloon_drv.o
195 DOMCAPS_OBJS += domcaps.o
196 EVTCHN_OBJS += evtchn_dev.o
197 GFX_PRIVATE_OBJS += gfx_private.o gfxp_pci.o gfxp_segmap.o \
198     gfxp_devmap.o gfxp_vgatext.o gfxp_vm.o vgasubr.o \
199     gfxp_fb.o gfxp_bitmap.o
200 IOAT_OBJS += ioat.o ioat_rs.o ioat_ioctl.o ioat_chan.o
201 ISANEXUS_OBJS += isa.o dma_engine.o i8237A.o
202 PCI_E_NEXUS_OBJS += npe.o npe_misc.o
203 PCI_E_NEXUS_OBJS += pci_common.o pci_kstats.o pci_tools.o
204 PCINEXUS_OBJS += pci.o pci_common.o pci_kstats.o pci_tools.o
205 PRIVCMD_OBJS += seg_mf.o privcmd.o privcmd_hcall.o
206 ROOTNEX_OBJS += rootnex.o
207 XPVTOD_OBJS += xpvtod.o
208 XPV_AUTOCONFIG_OBJS += xpvtod.o
209 XPV_PSM_OBJS += xpvtod.o mp_platform_common.o mp_platform_xpv.o \
210     apic_regops.o psm_common.o xpvtod.o
211 XPV_UPPC_OBJS += xpvtod.o psm_common.o
212 XENBUS_OBJS += xenbus_dev.o
213 XENCONS_OBJS += xencons.o
214 XPVD_OBJS += xpvd.o
215 XPVTAP_OBJS += xpvtod.o blk_common.o seg_mf.o
216 XNB_OBJS += xnb.o
217 XNBE_OBJS += xnbe.o
218 XNBO_OBJS += xnbo.o
219 XNBU_OBJS += xnbu.o
220 XNF_OBJS += xnf.o
221 XSVC_OBJS += xsvc.o
222 XDF_OBJS += xdf.o
223 XDB_OBJS += xdb.o
224 XDT_OBJS += xdt.o

226 #
227 #     Build up defines and paths.
228 #
229 INC_PATH += -I$(UTSBASE)/i86xpv -I$(UTSBASE)/i86pc -I$(SRC)/common \
230     -I$(UTSBASE)/common/xen

232 #
233 # Since the assym files are derived, the dependencies must be explicit for
234 # all files including this file. (This is only actually required in the
235 # instance when the .nse_depinfo file does not exist.) It may seem that
236 # the lint targets should also have a similar dependency, but they don't
237 # since only C headers are included when #defined(__lint) is true.
238 #

240 ASSYM_DEPS += \
241     copy.o              \
242     desctbls_asm.o     \
243     ddi_i86_asm.o      \
244     exception.o        \
245     fast_trap_asm.o    \
246     float.o            \
247     hyperevent.o       \
248     i86_subr.o         \
249     kdi_asm.o          \
250     interrupt.o        \
251     lock_prim.o        \
252     locore.o           \
253     panic_asm.o        \
254     sseblk.o           \
255     swtch.o            \
256     syscall_asm.o      \
257     syscall_asm_amd64.o

```

new/usr/src/uts/i86xpv/Makefile.files

5

259 \$(KDI_ASSYM_DEPS:%=\$(OBJSDIR)/%): \$(DSFDIR)/\$(OBJSDIR)/kdi_assym.h

new/usr/src/uts/intel/ia32/ml/copy.s

1

```
*****
68145 Tue May 7 04:50:01 2019
new/usr/src/uts/intel/ia32/ml/copy.s
OS-7125 Need mitigation of L1TF (CVE-2018-3646)
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
*****
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 2009 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */
25
26 /*
27 * Copyright (c) 2009, Intel Corporation
28 * All rights reserved.
29 */
30
31 /*      Copyright (c) 1990, 1991 UNIX System Laboratories, Inc.      */
32 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989, 1990 AT&T      */
33 /*      All Rights Reserved                                          */
34
35 /*      Copyright (c) 1987, 1988 Microsoft Corporation              */
36 /*      All Rights Reserved                                          */
37
38 /*
39 * Copyright (c) 2018 Joyent, Inc.
40 * Copyright 2016 Joyent, Inc.
41 */
42 #include <sys/errno.h>
43 #include <sys/asm_linkage.h>
44
45 #if defined(__lint)
46 #include <sys/types.h>
47 #include <sys/system.h>
48 #else /* __lint */
49 #include "assym.h"
50 #endif /* __lint */
51
52 #define KCOPY_MIN_SIZE 128 /* Must be >= 16 bytes */
53 #define XCOPY_MIN_SIZE 128 /* Must be >= 16 bytes */
54 /*
55 * Non-temoprpal access (NTA) alignment requirement
56 */
57 #define NTA_ALIGN_SIZE 4 /* Must be at least 4-byte aligned */
58 #define NTA_ALIGN_MASK _CONST(NTA_ALIGN_SIZE-1)
```

new/usr/src/uts/intel/ia32/ml/copy.s

2

```
59 #define COUNT_ALIGN_SIZE 16 /* Must be at least 16-byte aligned */
60 #define COUNT_ALIGN_MASK _CONST(COUNT_ALIGN_SIZE-1)
61
62 /*
63 * With the introduction of Broadwell, Intel has introduced supervisor mode
64 * access protection -- SMAP. SMAP forces the kernel to set certain bits to
65 * enable access of user pages (AC in rflags, defines as PS_ACHK in
66 * <sys/psw.h>). One of the challenges is that the implementation of many of the
67 * userland copy routines directly use the kernel ones. For example, copyin and
68 * copyout simply go and jump to the do_copy_fault label and traditionally let
69 * those deal with the return for them. In fact, changing that is a can of frame
70 * pointers.
71 *
72 * Rules and Constraints:
73 *
74 * 1. For anything that's not in copy.s, we have it do explicit calls to the
75 * smap related code. It usually is in a position where it is able to. This is
76 * restricted to the following three places: DTrace, resume() in swtch.s and
77 * on_fault/no_fault. If you want to add it somewhere else, we should be
78 * thinking twice.
79 *
80 * 2. We try to toggle this at the smallest window possible. This means that if
81 * we take a fault, need to try to use a copyin() or copyout(), or any
82 * other function, we will always leave with SMAP enabled (the kernel cannot
83 * access user pages).
84 *
85 * 3. None of the *_noerr() or ucopy/uzero routines should toggle SMAP. They are
86 * explicitly only allowed to be called while in an on_fault()/no_fault() handle
87 * which already takes care of ensuring that SMAP is enabled and disabled. Note
88 * this means that when under an on_fault()/no_fault() handler, one must not
89 * call the non-*_noeer() routines.
90 *
91 * 4. The first thing we should do after coming out of an lofault handler is to
92 * make sure that we call smap_enable again to ensure that we are safely
93 * protected, as more often than not, we will have disabled smap to get there.
94 *
95 * 5. The SMAP functions, smap_enable and smap_disable may not touch any
96 * registers beyond those done by the call and ret. These routines may be called
97 * from arbitrary contexts in copy.s where we have slightly more special ABIs in
98 * place.
99 *
100 * 6. For any inline user of SMAP, the appropriate SMAP_ENABLE_INSTR and
101 * SMAP_DISABLE_INSTR macro should be used (except for smap_enable() and
102 * smap_disable()). If the number of these is changed, you must update the
103 * constants SMAP_ENABLE_COUNT and SMAP_DISABLE_COUNT below.
104 *
105 * 7. Note, at this time SMAP is not implemented for the 32-bit kernel. There is
106 * no known technical reason preventing it from being enabled.
107 *
108 * 8. Generally this .s file is processed by a K&R style cpp. This means that it
109 * really has a lot of feelings about whitespace. In particular, if you have a
110 * macro FOO with the arguments FOO(1, 3), the second argument is in fact ' 3'.
111 *
112 * 9. The smap_enable and smap_disable functions should not generally be called.
113 * They exist such that DTrace and on_trap() may use them, that's it.
114 *
115 * 10. In general, the kernel has its own value for rflags that gets used. This
116 * is maintained in a few different places which vary based on how the thread
117 * comes into existence and whether it's a user thread. In general, when the
118 * kernel takes a trap, it always will set ourselves to a known set of flags,
119 * mainly as part of ENABLE_INTR_FLAGS and F_OFF and F_ON. These ensure that
120 * PS_ACHK is cleared for us. In addition, when using the sysenter instruction,
121 * we mask off PS_ACHK off via the AMD_SFMASK MSR. See init_cpu_syscall() for
122 * where that gets masked off.
123 */
```

```

125 /*
126 * The optimal 64-bit bcopy and kcopy for modern x86 processors uses
127 * "rep movq" for large sizes. Performance data shows that many calls to
128 * bcopy/kcopy/bzero/kzero operate on small buffers. For best performance for
129 * these small sizes unrolled code is used. For medium sizes loops writing
130 * 64-bytes per loop are used. Transition points were determined experimentally.
131 */
132 #define BZERO_USE_REP    (1024)
133 #define BCOPY_DFLT_REP  (128)
134 #define BCOPY_NHM_REP   (768)

136 /*
137 * Copy a block of storage, returning an error code if 'from' or
138 * 'to' takes a kernel pagefault which cannot be resolved.
139 * Returns errno value on pagefault error, 0 if all ok
140 */

142 /*
143 * I'm sorry about these macros, but copy.s is unsurprisingly sensitive to
144 * additional call instructions.
145 */
146 #if defined(__amd64)
147 #define SMAP_DISABLE_COUNT    16
148 #define SMAP_ENABLE_COUNT    26
149 #elif defined(__i386)
150 #define SMAP_DISABLE_COUNT    0
151 #define SMAP_ENABLE_COUNT    0
152 #endif

154 #define SMAP_DISABLE_INSTR(ITER) \
155     .globl _smap_disable_patch_/**/ITER; \
156     _smap_disable_patch_/**/ITER/**/;; \
157     nop; nop; nop;

159 #define SMAP_ENABLE_INSTR(ITER) \
160     .globl _smap_enable_patch_/**/ITER; \
161     _smap_enable_patch_/**/ITER/**/;; \
162     nop; nop; nop;

164 #if defined(__lint)

166 /* ARGSUSED */
167 int
168 kcopy(const void *from, void *to, size_t count)
169 { return (0); }

171 #else /* __lint */

173     .globl kernelbase
174     .globl postbootkernelbase

176 #if defined(__amd64)

178     ENTRY(kcopy)
179     pushq %rbp
180     movq %rsp, %rbp
181 #ifdef DEBUG
182     cmpq postbootkernelbase(%rip), %rdi /* %rdi = from */
183     jb 0f
184     cmpq postbootkernelbase(%rip), %rsi /* %rsi = to */
185     jnb 1f
186 0:    leaq .kcopy_panic_msg(%rip), %rdi
187     xorl %eax, %eax
188     call panic
189 1:
190 #endif

```

```

191 /*
192 * pass lofault value as 4th argument to do_copy_fault
193 */
194 leaq _kcopy_copyerr(%rip), %rcx
195 movq %gs:CPU_THREAD, %r9 /* %r9 = thread addr */

197 do_copy_fault:
198     movq T_LOFAULT(%r9), %r11 /* save the current lofault */
199     movq %rcx, T_LOFAULT(%r9) /* new lofault */
200     call bcopy_altentry
201     xorl %eax, %eax /* return 0 (success) */
202     SMAP_ENABLE_INSTR(0)

204 /*
205 * A fault during do_copy_fault is indicated through an errno value
206 * in %rax and we iretq from the trap handler to here.
207 */
208 _kcopy_copyerr:
209     movq %r11, T_LOFAULT(%r9) /* restore original lofault */
210     leave
211     ret
212     SET_SIZE(kcopy)
    _____
    unchanged_portion_omitted_

434 #undef ARG_FROM
435 #undef ARG_TO
436 #undef ARG_COUNT

438 #endif /* __i386 */
439 #endif /* __lint */

441 #if defined(__lint)

443 /* ARGSUSED */
444 void
445 bcopy(const void *from, void *to, size_t count)
446 {}

448 #else /* __lint */

450 #if defined(__amd64)

452     ENTRY(bcopy)
453 #ifdef DEBUG
454     orq %rdx, %rdx /* %rdx = count */
455     jz 1f
456     cmpq postbootkernelbase(%rip), %rdi /* %rdi = from */
457     jb 0f
458     cmpq postbootkernelbase(%rip), %rsi /* %rsi = to */
459     jnb 1f
460 0:    leaq .bcopy_panic_msg(%rip), %rdi
461     jmp call_panic /* setup stack and call panic */
462 1:
463 #endif
464 /*
465 * bcopy_altentry() is called from kcopy, i.e., do_copy_fault.
466 * kcopy assumes that bcopy doesn't touch %r9 and %r11. If bcopy
467 * uses these registers in future they must be saved and restored.
468 */
469     ALTENTRY(bcopy_altentry)
470 do_copy:
471 #define L(s) .bcopy/**/s
472     cmpq $0x50, %rdx /* 80 */
473     jae bcopy_ck_size

475 /*

```



```

476     * Performance data shows many caller's copy small buffers. So for
477     * best perf for these sizes unrolled code is used. Store data without
478     * worrying about alignment.
479     */
480     leaq   L(fwdPxQx)(%rip), %r10
481     addq   %rdx, %rdi
482     addq   %rdx, %rsi
483     movslq (%r10,%rdx,4), %rcx
484     leaq   (%rcx,%r10,1), %r10
485     jmpq   *%r10

487     .p2align 4
488 L(fwdPxQx):
489     .int   L(P0Q0)-L(fwdPxQx) /* 0 */
490     .int   L(P1Q0)-L(fwdPxQx)
491     .int   L(P2Q0)-L(fwdPxQx)
492     .int   L(P3Q0)-L(fwdPxQx)
493     .int   L(P4Q0)-L(fwdPxQx)
494     .int   L(P5Q0)-L(fwdPxQx)
495     .int   L(P6Q0)-L(fwdPxQx)
496     .int   L(P7Q0)-L(fwdPxQx)

498     .int   L(P0Q1)-L(fwdPxQx) /* 8 */
499     .int   L(P1Q1)-L(fwdPxQx)
500     .int   L(P2Q1)-L(fwdPxQx)
501     .int   L(P3Q1)-L(fwdPxQx)
502     .int   L(P4Q1)-L(fwdPxQx)
503     .int   L(P5Q1)-L(fwdPxQx)
504     .int   L(P6Q1)-L(fwdPxQx)
505     .int   L(P7Q1)-L(fwdPxQx)

507     .int   L(P0Q2)-L(fwdPxQx) /* 16 */
508     .int   L(P1Q2)-L(fwdPxQx)
509     .int   L(P2Q2)-L(fwdPxQx)
510     .int   L(P3Q2)-L(fwdPxQx)
511     .int   L(P4Q2)-L(fwdPxQx)
512     .int   L(P5Q2)-L(fwdPxQx)
513     .int   L(P6Q2)-L(fwdPxQx)
514     .int   L(P7Q2)-L(fwdPxQx)

516     .int   L(P0Q3)-L(fwdPxQx) /* 24 */
517     .int   L(P1Q3)-L(fwdPxQx)
518     .int   L(P2Q3)-L(fwdPxQx)
519     .int   L(P3Q3)-L(fwdPxQx)
520     .int   L(P4Q3)-L(fwdPxQx)
521     .int   L(P5Q3)-L(fwdPxQx)
522     .int   L(P6Q3)-L(fwdPxQx)
523     .int   L(P7Q3)-L(fwdPxQx)

525     .int   L(P0Q4)-L(fwdPxQx) /* 32 */
526     .int   L(P1Q4)-L(fwdPxQx)
527     .int   L(P2Q4)-L(fwdPxQx)
528     .int   L(P3Q4)-L(fwdPxQx)
529     .int   L(P4Q4)-L(fwdPxQx)
530     .int   L(P5Q4)-L(fwdPxQx)
531     .int   L(P6Q4)-L(fwdPxQx)
532     .int   L(P7Q4)-L(fwdPxQx)

534     .int   L(P0Q5)-L(fwdPxQx) /* 40 */
535     .int   L(P1Q5)-L(fwdPxQx)
536     .int   L(P2Q5)-L(fwdPxQx)
537     .int   L(P3Q5)-L(fwdPxQx)
538     .int   L(P4Q5)-L(fwdPxQx)
539     .int   L(P5Q5)-L(fwdPxQx)
540     .int   L(P6Q5)-L(fwdPxQx)
541     .int   L(P7Q5)-L(fwdPxQx)

```

```

543     .int   L(P0Q6)-L(fwdPxQx) /* 48 */
544     .int   L(P1Q6)-L(fwdPxQx)
545     .int   L(P2Q6)-L(fwdPxQx)
546     .int   L(P3Q6)-L(fwdPxQx)
547     .int   L(P4Q6)-L(fwdPxQx)
548     .int   L(P5Q6)-L(fwdPxQx)
549     .int   L(P6Q6)-L(fwdPxQx)
550     .int   L(P7Q6)-L(fwdPxQx)

552     .int   L(P0Q7)-L(fwdPxQx) /* 56 */
553     .int   L(P1Q7)-L(fwdPxQx)
554     .int   L(P2Q7)-L(fwdPxQx)
555     .int   L(P3Q7)-L(fwdPxQx)
556     .int   L(P4Q7)-L(fwdPxQx)
557     .int   L(P5Q7)-L(fwdPxQx)
558     .int   L(P6Q7)-L(fwdPxQx)
559     .int   L(P7Q7)-L(fwdPxQx)

561     .int   L(P0Q8)-L(fwdPxQx) /* 64 */
562     .int   L(P1Q8)-L(fwdPxQx)
563     .int   L(P2Q8)-L(fwdPxQx)
564     .int   L(P3Q8)-L(fwdPxQx)
565     .int   L(P4Q8)-L(fwdPxQx)
566     .int   L(P5Q8)-L(fwdPxQx)
567     .int   L(P6Q8)-L(fwdPxQx)
568     .int   L(P7Q8)-L(fwdPxQx)

570     .int   L(P0Q9)-L(fwdPxQx) /* 72 */
571     .int   L(P1Q9)-L(fwdPxQx)
572     .int   L(P2Q9)-L(fwdPxQx)
573     .int   L(P3Q9)-L(fwdPxQx)
574     .int   L(P4Q9)-L(fwdPxQx)
575     .int   L(P5Q9)-L(fwdPxQx)
576     .int   L(P6Q9)-L(fwdPxQx)
577     .int   L(P7Q9)-L(fwdPxQx) /* 79 */

579     .p2align 4
580 L(P0Q9):
581     mov    -0x48(%rdi), %rcx
582     mov    %rcx, -0x48(%rsi)
583 L(P0Q8):
584     mov    -0x40(%rdi), %r10
585     mov    %r10, -0x40(%rsi)
586 L(P0Q7):
587     mov    -0x38(%rdi), %r8
588     mov    %r8, -0x38(%rsi)
589 L(P0Q6):
590     mov    -0x30(%rdi), %rcx
591     mov    %rcx, -0x30(%rsi)
592 L(P0Q5):
593     mov    -0x28(%rdi), %r10
594     mov    %r10, -0x28(%rsi)
595 L(P0Q4):
596     mov    -0x20(%rdi), %r8
597     mov    %r8, -0x20(%rsi)
598 L(P0Q3):
599     mov    -0x18(%rdi), %rcx
600     mov    %rcx, -0x18(%rsi)
601 L(P0Q2):
602     mov    -0x10(%rdi), %r10
603     mov    %r10, -0x10(%rsi)
604 L(P0Q1):
605     mov    -0x8(%rdi), %r8
606     mov    %r8, -0x8(%rsi)
607 L(P0Q0):

```

```

608         ret
610         .p2align 4
611 L(P1Q9):
612     mov     -0x49(%rdi), %r8
613     mov     %r8, -0x49(%rsi)
614 L(P1Q8):
615     mov     -0x41(%rdi), %rcx
616     mov     %rcx, -0x41(%rsi)
617 L(P1Q7):
618     mov     -0x39(%rdi), %r10
619     mov     %r10, -0x39(%rsi)
620 L(P1Q6):
621     mov     -0x31(%rdi), %r8
622     mov     %r8, -0x31(%rsi)
623 L(P1Q5):
624     mov     -0x29(%rdi), %rcx
625     mov     %rcx, -0x29(%rsi)
626 L(P1Q4):
627     mov     -0x21(%rdi), %r10
628     mov     %r10, -0x21(%rsi)
629 L(P1Q3):
630     mov     -0x19(%rdi), %r8
631     mov     %r8, -0x19(%rsi)
632 L(P1Q2):
633     mov     -0x11(%rdi), %rcx
634     mov     %rcx, -0x11(%rsi)
635 L(P1Q1):
636     mov     -0x9(%rdi), %r10
637     mov     %r10, -0x9(%rsi)
638 L(P1Q0):
639     movzwbq -0x1(%rdi), %r8
640     mov     %r8b, -0x1(%rsi)
641     ret
643         .p2align 4
644 L(P2Q9):
645     mov     -0x4a(%rdi), %r8
646     mov     %r8, -0x4a(%rsi)
647 L(P2Q8):
648     mov     -0x42(%rdi), %rcx
649     mov     %rcx, -0x42(%rsi)
650 L(P2Q7):
651     mov     -0x3a(%rdi), %r10
652     mov     %r10, -0x3a(%rsi)
653 L(P2Q6):
654     mov     -0x32(%rdi), %r8
655     mov     %r8, -0x32(%rsi)
656 L(P2Q5):
657     mov     -0x2a(%rdi), %rcx
658     mov     %rcx, -0x2a(%rsi)
659 L(P2Q4):
660     mov     -0x22(%rdi), %r10
661     mov     %r10, -0x22(%rsi)
662 L(P2Q3):
663     mov     -0x1a(%rdi), %r8
664     mov     %r8, -0x1a(%rsi)
665 L(P2Q2):
666     mov     -0x12(%rdi), %rcx
667     mov     %rcx, -0x12(%rsi)
668 L(P2Q1):
669     mov     -0xa(%rdi), %r10
670     mov     %r10, -0xa(%rsi)
671 L(P2Q0):
672     movzwbq -0x2(%rdi), %r8
673     mov     %r8w, -0x2(%rsi)

```

```

674         ret
676         .p2align 4
677 L(P3Q9):
678     mov     -0x4b(%rdi), %r8
679     mov     %r8, -0x4b(%rsi)
680 L(P3Q8):
681     mov     -0x43(%rdi), %rcx
682     mov     %rcx, -0x43(%rsi)
683 L(P3Q7):
684     mov     -0x3b(%rdi), %r10
685     mov     %r10, -0x3b(%rsi)
686 L(P3Q6):
687     mov     -0x33(%rdi), %r8
688     mov     %r8, -0x33(%rsi)
689 L(P3Q5):
690     mov     -0x2b(%rdi), %rcx
691     mov     %rcx, -0x2b(%rsi)
692 L(P3Q4):
693     mov     -0x23(%rdi), %r10
694     mov     %r10, -0x23(%rsi)
695 L(P3Q3):
696     mov     -0x1b(%rdi), %r8
697     mov     %r8, -0x1b(%rsi)
698 L(P3Q2):
699     mov     -0x13(%rdi), %rcx
700     mov     %rcx, -0x13(%rsi)
701 L(P3Q1):
702     mov     -0xb(%rdi), %r10
703     mov     %r10, -0xb(%rsi)
704     /*
705     * These trailing loads/stores have to do all their loads 1st,
706     * then do the stores.
707     */
708 L(P3Q0):
709     movzwbq -0x3(%rdi), %r8
710     movzwbq -0x1(%rdi), %r10
711     mov     %r8w, -0x3(%rsi)
712     mov     %r10b, -0x1(%rsi)
713     ret
715         .p2align 4
716 L(P4Q9):
717     mov     -0x4c(%rdi), %r8
718     mov     %r8, -0x4c(%rsi)
719 L(P4Q8):
720     mov     -0x44(%rdi), %rcx
721     mov     %rcx, -0x44(%rsi)
722 L(P4Q7):
723     mov     -0x3c(%rdi), %r10
724     mov     %r10, -0x3c(%rsi)
725 L(P4Q6):
726     mov     -0x34(%rdi), %r8
727     mov     %r8, -0x34(%rsi)
728 L(P4Q5):
729     mov     -0x2c(%rdi), %rcx
730     mov     %rcx, -0x2c(%rsi)
731 L(P4Q4):
732     mov     -0x24(%rdi), %r10
733     mov     %r10, -0x24(%rsi)
734 L(P4Q3):
735     mov     -0x1c(%rdi), %r8
736     mov     %r8, -0x1c(%rsi)
737 L(P4Q2):
738     mov     -0x14(%rdi), %rcx
739     mov     %rcx, -0x14(%rsi)

```

```

740 L(P4Q1):
741     mov     -0xc(%rdi), %r10
742     mov     %r10, -0xc(%rsi)
743 L(P4Q0):
744     mov     -0x4(%rdi), %r8d
745     mov     %r8d, -0x4(%rsi)
746     ret

748     .p2align 4
749 L(P5Q9):
750     mov     -0x4d(%rdi), %r8
751     mov     %r8, -0x4d(%rsi)
752 L(P5Q8):
753     mov     -0x45(%rdi), %rcx
754     mov     %rcx, -0x45(%rsi)
755 L(P5Q7):
756     mov     -0x3d(%rdi), %r10
757     mov     %r10, -0x3d(%rsi)
758 L(P5Q6):
759     mov     -0x35(%rdi), %r8
760     mov     %r8, -0x35(%rsi)
761 L(P5Q5):
762     mov     -0x2d(%rdi), %rcx
763     mov     %rcx, -0x2d(%rsi)
764 L(P5Q4):
765     mov     -0x25(%rdi), %r10
766     mov     %r10, -0x25(%rsi)
767 L(P5Q3):
768     mov     -0x1d(%rdi), %r8
769     mov     %r8, -0x1d(%rsi)
770 L(P5Q2):
771     mov     -0x15(%rdi), %rcx
772     mov     %rcx, -0x15(%rsi)
773 L(P5Q1):
774     mov     -0xd(%rdi), %r10
775     mov     %r10, -0xd(%rsi)
776 L(P5Q0):
777     mov     -0x5(%rdi), %r8d
778     movzwbq -0x1(%rdi), %r10
779     mov     %r8d, -0x5(%rsi)
780     mov     %r10b, -0x1(%rsi)
781     ret

783     .p2align 4
784 L(P6Q9):
785     mov     -0x4e(%rdi), %r8
786     mov     %r8, -0x4e(%rsi)
787 L(P6Q8):
788     mov     -0x46(%rdi), %rcx
789     mov     %rcx, -0x46(%rsi)
790 L(P6Q7):
791     mov     -0x3e(%rdi), %r10
792     mov     %r10, -0x3e(%rsi)
793 L(P6Q6):
794     mov     -0x36(%rdi), %r8
795     mov     %r8, -0x36(%rsi)
796 L(P6Q5):
797     mov     -0x2e(%rdi), %rcx
798     mov     %rcx, -0x2e(%rsi)
799 L(P6Q4):
800     mov     -0x26(%rdi), %r10
801     mov     %r10, -0x26(%rsi)
802 L(P6Q3):
803     mov     -0x1e(%rdi), %r8
804     mov     %r8, -0x1e(%rsi)
805 L(P6Q2):

```

```

806     mov     -0x16(%rdi), %rcx
807     mov     %rcx, -0x16(%rsi)
808 L(P6Q1):
809     mov     -0xe(%rdi), %r10
810     mov     %r10, -0xe(%rsi)
811 L(P6Q0):
812     mov     -0x6(%rdi), %r8d
813     movzwbq -0x2(%rdi), %r10
814     mov     %r8d, -0x6(%rsi)
815     mov     %r10w, -0x2(%rsi)
816     ret

818     .p2align 4
819 L(P7Q9):
820     mov     -0x4f(%rdi), %r8
821     mov     %r8, -0x4f(%rsi)
822 L(P7Q8):
823     mov     -0x47(%rdi), %rcx
824     mov     %rcx, -0x47(%rsi)
825 L(P7Q7):
826     mov     -0x3f(%rdi), %r10
827     mov     %r10, -0x3f(%rsi)
828 L(P7Q6):
829     mov     -0x37(%rdi), %r8
830     mov     %r8, -0x37(%rsi)
831 L(P7Q5):
832     mov     -0x2f(%rdi), %rcx
833     mov     %rcx, -0x2f(%rsi)
834 L(P7Q4):
835     mov     -0x27(%rdi), %r10
836     mov     %r10, -0x27(%rsi)
837 L(P7Q3):
838     mov     -0x1f(%rdi), %r8
839     mov     %r8, -0x1f(%rsi)
840 L(P7Q2):
841     mov     -0x17(%rdi), %rcx
842     mov     %rcx, -0x17(%rsi)
843 L(P7Q1):
844     mov     -0xf(%rdi), %r10
845     mov     %r10, -0xf(%rsi)
846 L(P7Q0):
847     mov     -0x7(%rdi), %r8d
848     movzwbq -0x3(%rdi), %r10
849     movzwbq -0x1(%rdi), %rcx
850     mov     %r8d, -0x7(%rsi)
851     mov     %r10w, -0x3(%rsi)
852     mov     %cl, -0x1(%rsi)
853     ret

855     /*
856     * For large sizes rep smovq is fastest.
857     * Transition point determined experimentally as measured on
858     * Intel Xeon processors (incl. Nehalem and previous generations) and
859     * AMD Opteron. The transition value is patched at boot time to avoid
860     * memory reference hit.
861     */
862     .globl bcopy_patch_start
863 bcopy_patch_start:
864     cmpq    $BCOPY_NHM_REP, %rdx
865     .globl bcopy_patch_end
866 bcopy_patch_end:

868     .p2align 4
869     ALTENTRY(bcopy_ck_size)

869     .globl bcopy_ck_size

```

```

870 bcopy_ck_size:
871     cmpq    $BCOPY_DFLT_REP, %rdx
872     jae    L(use_rep)

874     /*
875     * Align to a 8-byte boundary. Avoids penalties from unaligned stores
876     * as well as from stores spanning cachelines.
877     */
878     test   $0x7, %rsi
879     jz    L(aligned_loop)
880     test   $0x1, %rsi
881     jz    2f
882     movzbq (%rdi), %r8
883     dec   %rdx
884     inc   %rdi
885     mov   %r8b, (%rsi)
886     inc   %rsi
887 2:
888     test   $0x2, %rsi
889     jz    4f
890     movzwb (%rdi), %r8
891     sub   $0x2, %rdx
892     add   $0x2, %rdi
893     mov   %r8w, (%rsi)
894     add   $0x2, %rsi
895 4:
896     test   $0x4, %rsi
897     jz    L(aligned_loop)
898     mov   (%rdi), %r8d
899     sub   $0x4, %rdx
900     add   $0x4, %rdi
901     mov   %r8d, (%rsi)
902     add   $0x4, %rsi

904     /*
905     * Copy 64-bytes per loop
906     */
907     .p2align 4
908 L(aligned_loop):
909     mov   (%rdi), %r8
910     mov   0x8(%rdi), %r10
911     lea  -0x40(%rdx), %rdx
912     mov   %r8, (%rsi)
913     mov   %r10, 0x8(%rsi)
914     mov   0x10(%rdi), %rcx
915     mov   0x18(%rdi), %r8
916     mov   %rcx, 0x10(%rsi)
917     mov   %r8, 0x18(%rsi)

919     cmp   $0x40, %rdx
920     mov   0x20(%rdi), %r10
921     mov   0x28(%rdi), %rcx
922     mov   %r10, 0x20(%rsi)
923     mov   %rcx, 0x28(%rsi)
924     mov   0x30(%rdi), %r8
925     mov   0x38(%rdi), %r10
926     lea  0x40(%rdi), %rdi
927     mov   %r8, 0x30(%rsi)
928     mov   %r10, 0x38(%rsi)
929     lea  0x40(%rsi), %rsi
930     jae  L(aligned_loop)

932     /*
933     * Copy remaining bytes (0-63)
934     */
935 L(do_remainder):

```

```

936     leaq   L(fwdPxQx)(%rip), %r10
937     addq   %rdx, %rdi
938     addq   %rdx, %rsi
939     movslq (%r10,%rdx,4), %rcx
940     leaq   (%rcx,%r10,1), %r10
941     jmpq   *%r10

943     /*
944     * Use rep smovq. Clear remainder via unrolled code
945     */
946     .p2align 4
947 L(use_rep):
948     xchgq  %rdi, %rsi           /* %rsi = source, %rdi = destination */
949     movq   %rdx, %rcx         /* %rcx = count */
950     shrq   $3, %rcx          /* 8-byte word count */
951     rep
952     smovq

954     xchgq  %rsi, %rdi         /* %rdi = src, %rsi = destination */
955     andq   $7, %rdx          /* remainder */
956     jnz   L(do_remainder)
957     ret
958 #undef L
959 SET_SIZE(bcopy_ck_size)

961 #ifdef DEBUG
962     /*
963     * Setup frame on the run-time stack. The end of the input argument
964     * area must be aligned on a 16 byte boundary. The stack pointer %rsp,
965     * always points to the end of the latest allocated stack frame.
966     * panic(const char *format, ...) is a varargs function. When a
967     * function taking variable arguments is called, %rax must be set
968     * to eight times the number of floating point parameters passed
969     * to the function in SSE registers.
970     */
971 call_panic:
972     pushq  %rbp               /* align stack properly */
973     movq   %rsp, %rbp
974     xorl   %eax, %eax         /* no variable arguments */
975     call   panic              /* %rdi = format string */
976 #endif
977     SET_SIZE(bcopy_alentry)

```

unchanged_portion_omitted

new/usr/src/uts/intel/ia32/ml/swtch.s

1

```
*****
14241 Tue May 7 04:50:01 2019
new/usr/src/uts/intel/ia32/ml/swtch.s
OS-7125 Need mitigation of L1TF (CVE-2018-3646)
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
*****
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 2007 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */
25
26 /*
27 * Copyright 2019 Joyent, Inc.
28 * Copyright (c) 2018 Joyent, Inc.
29 */
30 /*
31 * Process switching routines.
32 */
33
34 #if defined(__lint)
35 #include <sys/thread.h>
36 #include <sys/system.h>
37 #include <sys/time.h>
38 #else /* __lint */
39 #include "assym.h"
40 #endif /* __lint */
41
42 #include <sys/asm_linkage.h>
43 #include <sys/asm_misc.h>
44 #include <sys/regset.h>
45 #include <sys/privregs.h>
46 #include <sys/stack.h>
47 #include <sys/segments.h>
48 #include <sys/psw.h>
49
50 #include "assym.h"
51
52 /*
53  * resume(thread_id_t t);
54  *
55  * a thread can only run on one processor at a time. there
56  * exists a window on MPs where the current thread on one
57  * processor is capable of being dispatched by another processor.
58  * some overlap between outgoing and incoming threads can happen
59  */
```

new/usr/src/uts/intel/ia32/ml/swtch.s

2

```
51 * when they are the same thread. in this case where the threads
52 * are the same, resume() on one processor will spin on the incoming
53 * thread until resume() on the other processor has finished with
54 * the outgoing thread.
55 *
56 * The MMU context changes when the resuming thread resides in a different
57 * process. Kernel threads are known by resume to reside in process 0.
58 * The MMU context, therefore, only changes when resuming a thread in
59 * a process different from curproc.
60 *
61 * resume_from_intr() is called when the thread being resumed was not
62 * passivated by resume (e.g. was interrupted). This means that the
63 * resume lock is already held and that a restore context is not needed.
64 * Also, the MMU context is not changed on the resume in this case.
65 *
66 * resume_from_zombie() is the same as resume except the calling thread
67 * is a zombie and must be put on the deathrow list after the CPU is
68 * off the stack.
69 */
70
71 #if !defined(__lint)
72 #error LWP_PCB_FPU MUST be defined as 0 for code in swtch.s to work
73 #endif /* LWP_PCB_FPU != 0 */
74
75 #endif /* !__lint */
76
77 #if defined(__amd64)
78
79 /*
80  * Save non-volatile regs other than %rsp (%rbx, %rbp, and %r12 - %r15)
81  *
82  * The stack frame must be created before the save of %rsp so that tracebacks
83  * of swtch()-ed-out processes show the process as having last called swtch().
84  */
85 #define SAVE_REGS(thread_t, retaddr) \
86     movq   %rbp, T_RBP(thread_t); \
87     movq   %rbx, T_RBX(thread_t); \
88     movq   %r12, T_R12(thread_t); \
89     movq   %r13, T_R13(thread_t); \
90     movq   %r14, T_R14(thread_t); \
91     movq   %r15, T_R15(thread_t); \
92     pushq  %rbp; \
93     movq   %rsp, %rbp; \
94     movq   %rsp, T_SP(thread_t); \
95     movq   retaddr, T_PC(thread_t); \
96     movq   %rdi, %r12; \
97     call   __dtrace_probe__sched_off__cpu
98
99 /*
100  * Restore non-volatile regs other than %rsp (%rbx, %rbp, and %r12 - %r15)
101  *
102  * We load up %rsp from the label_t as part of the context switch, so
103  * we don't repeat that here.
104  *
105  * We don't do a 'leave,' because reloading %rsp/%rbp from the label_t
106  * already has the effect of putting the stack back the way it was when
107  * we came in.
108  */
109 #define RESTORE_REGS(scratch_reg) \
110     movq   %gs:CPU_THREAD, scratch_reg; \
111     movq   T_RBP(scratch_reg), %rbp; \
112     movq   T_RBX(scratch_reg), %rbx; \
113     movq   T_R12(scratch_reg), %r12; \
114     movq   T_R13(scratch_reg), %r13; \
```

```

111     movq    T_R14(scratch_reg), %r14;    \
112     movq    T_R15(scratch_reg), %r15;    \

114 /*
115 * Get pointer to a thread's hat structure
116 */
117 #define GET_THREAD_HATP(hatp, thread_t, scratch_reg) \
118     movq    T_PROCP(thread_t), hatp;    \
119     movq    P_AS(hatp), scratch_reg;    \
120     movq    A_HAT(scratch_reg), hatp

122 #define TSC_READ() \
123     call    tsc_read; \
124     movq    %rax, %r14;

126 /*
127 * If we are resuming an interrupt thread, store a timestamp in the thread
128 * structure. If an interrupt occurs between tsc_read() and its subsequent
129 * store, the timestamp will be stale by the time it is stored. We can detect
130 * this by doing a compare-and-swap on the thread's timestamp, since any
131 * interrupt occurring in this window will put a new timestamp in the thread's
132 * t_intr_start field.
133 */
134 #define STORE_INTR_START(thread_t) \
135     testw   $T_INTR_THREAD, T_FLAGS(thread_t); \
136     jz     1f; \
137 0: \
138     TSC_READ(); \
139     movq    T_INTR_START(thread_t), %rax; \
140     cmpxchgq %r14, T_INTR_START(thread_t); \
141     jnz    0b; \
142 1:

156 #elif defined (__i386)

158 /*
159 * Save non-volatile registers (%ebp, %esi, %edi and %ebx)
160 *
161 * The stack frame must be created before the save of %esp so that tracebacks
162 * of swtch()ed-out processes show the process as having last called swtch().
163 */
164 #define SAVE_REGS(thread_t, retaddr) \
165     movl    %ebp, T_EBP(thread_t); \
166     movl    %ebx, T_EBX(thread_t); \
167     movl    %esi, T_ESI(thread_t); \
168     movl    %edi, T_EDI(thread_t); \
169     pushl   %ebp; \
170     movl    %esp, %ebp; \
171     movl    %esp, T_SP(thread_t); \
172     movl    retaddr, T_PC(thread_t); \
173     movl    8(%ebp), %edi; \
174     pushl   %edi; \
175     call    __dtrace_probe__sched_off__cpu; \
176     addl    $CLONGSIZE, %esp

178 /*
179 * Restore non-volatile registers (%ebp, %esi, %edi and %ebx)
180 *
181 * We don't do a 'leave,' because reloading %rsp/%rbp from the label_t
182 * already has the effect of putting the stack back the way it was when
183 * we came in.
184 */
185 #define RESTORE_REGS(scratch_reg) \
186     movl    %gs:CPU_THREAD, scratch_reg; \
187     movl    T_EBP(scratch_reg), %ebp; \
188     movl    T_EBX(scratch_reg), %ebx;

```

```

189     movl    T_ESI(scratch_reg), %esi;    \
190     movl    T_EDI(scratch_reg), %edi

192 /*
193 * Get pointer to a thread's hat structure
194 */
195 #define GET_THREAD_HATP(hatp, thread_t, scratch_reg) \
196     movl    T_PROCP(thread_t), hatp;    \
197     movl    P_AS(hatp), scratch_reg;    \
198     movl    A_HAT(scratch_reg), hatp

200 /*
201 * If we are resuming an interrupt thread, store a timestamp in the thread
202 * structure. If an interrupt occurs between tsc_read() and its subsequent
203 * store, the timestamp will be stale by the time it is stored. We can detect
204 * this by doing a compare-and-swap on the thread's timestamp, since any
205 * interrupt occurring in this window will put a new timestamp in the thread's
206 * t_intr_start field.
207 */
208 #define STORE_INTR_START(thread_t) \
209     testw   $T_INTR_THREAD, T_FLAGS(thread_t); \
210     jz     1f; \
211     pushl   %ecx; \
212 0: \
213     pushl   T_INTR_START(thread_t); \
214     pushl   T_INTR_START+4(thread_t); \
215     call    tsc_read; \
216     movl    %eax, %ebx; \
217     movl    %edx, %ecx; \
218     popl    %edx; \
219     popl    %eax; \
220     cmpxchg8b T_INTR_START(thread_t); \
221     jnz    0b; \
222     popl    %ecx; \
223 1:

225 #endif /* __amd64 */

227 #if defined(__lint)

229 /* ARGSUSED */
230 void
231 resume(kthread_t *t)
232 {}

234 #else /* __lint */

236 #if defined(__amd64)

144     .global kpti_enable

146     ENTRY(resume)
147     movq    %gs:CPU_THREAD, %rax
148     leaq   resume_return(%rip), %r11

150     /*
151     * Deal with SMAP here. A thread may be switched out at any point while
152     * it is executing. The thread could be under on_fault() or it could be
153     * pre-empted while performing a copy interruption. If this happens and
154     * we're not in the context of an interrupt which happens to handle
155     * saving and restoring rflags correctly, we may lose our SMAP related
156     * state.
157     *
158     * To handle this, as part of being switched out, we first save whether
159     * or not userland access is allowed ($PS_ACHK in rflags) and store that
160     * in t_useracc on the kthread_t and unconditionally enable SMAP to

```

```

161     * protect the system.
162     *
163     * Later, when the thread finishes resuming, we potentially disable smap
164     * if PS_ACHK was present in rflags. See uts/intel/ia32/ml/copy.s for
165     * more information on rflags and SMAP.
166     */
167     pushfq
168     popq   %rsi
169     andq   $PS_ACHK, %rsi
170     movq   %rsi, T_USERACC(%rax)
171     call   smap_enable

173     /*
174     * Save non-volatile registers, and set return address for current
175     * thread to resume_return.
176     *
177     * %r12 = t (new thread) when done
178     */
179     SAVE_REGS(%rax, %r11)

182     LOADCPU(%r15)
183     movq   CPU_THREAD(%r15), %r13      /* %r15 = CPU */
184                                         /* %r13 = curthread */

185     /*
186     * Call savectx if thread has installed context ops.
187     *
188     * Note that if we have floating point context, the save op
189     * (either fpsave_begin or fpxsave_begin) will issue the
190     * async save instruction (fnsave or fxsave respectively)
191     * that we wait for below.
192     */
193     cmpq   $0, T_CTX(%r13)             /* should current thread savectx? */
194     je     .nosavectx                 /* skip call when zero */

196     movq   %r13, %rdi                 /* arg = thread pointer */
197     call   savectx                    /* call ctx ops */
198 .nosavectx:

200     /*
201     * Call savepctx if process has installed context ops.
202     */
203     movq   T_PROCP(%r13), %r14        /* %r14 = proc */
204     cmpq   $0, P_PCTX(%r14)          /* should current thread savectx? */
205     je     .nosavepctx                /* skip call when zero */

207     movq   %r14, %rdi                 /* arg = proc pointer */
208     call   savepctx                   /* call ctx ops */
209 .nosavepctx:

211     /*
212     * Temporarily switch to the idle thread's stack
213     */
214     movq   CPU_IDLE_THREAD(%r15), %rax /* idle thread pointer */

216     /*
217     * Set the idle thread as the current thread
218     */
219     movq   T_SP(%rax), %rsp           /* It is safe to set rsp */
220     movq   %rax, CPU_THREAD(%r15)

222     /*
223     * Switch in the hat context for the new thread
224     */
225     /*
226     GET_THREAD_HATP(%rdi, %r12, %r11)

```

```

227     call   hat_switch

229     /*
230     * Clear and unlock previous thread's t_lock
231     * to allow it to be dispatched by another processor.
232     */
233     movb   $0, T_LOCK(%r13)

235     /*
236     * IMPORTANT: Registers at this point must be:
237     *           %r12 = new thread
238     *
239     * Here we are in the idle thread, have dropped the old thread.
240     */
241     ALTENTRY(_resume_from_idle)
242     /*
243     * spin until dispatched thread's mutex has
244     * been unlocked. this mutex is unlocked when
245     * it becomes safe for the thread to run.
246     */
247 .lock_thread_mutex:
248     lock
249     btsl   $0, T_LOCK(%r12)           /* attempt to lock new thread's mutex */
250     jnc   .thread_mutex_locked      /* got it */

252 .spin_thread_mutex:
253     pause
254     cmpb   $0, T_LOCK(%r12)           /* check mutex status */
255     jz     .lock_thread_mutex        /* clear, retry lock */
256     jmp   .spin_thread_mutex        /* still locked, spin... */

258 .thread_mutex_locked:
259     /*
260     * Fix CPU structure to indicate new running thread.
261     * Set pointer in new thread to the CPU structure.
262     */
263     LOADCPU(%r13)                     /* load current CPU pointer */
264     cmpq   %r13, T_CPU(%r12)
265     je     .setup_cpu

267     /* cp->cpu_stats.sys.cpumigrate++ */
268     incq   CPU_STATS_SYS_CPUMIGRATE(%r13)
269     movq   %r13, T_CPU(%r12)          /* set new thread's CPU pointer */

271 .setup_cpu:
272     /*
273     * Setup rsp0 (kernel stack) in TSS to curthread's saved regs
274     * structure. If this thread doesn't have a regs structure above
275     * the stack -- that is, if lwp_stk_init() was never called for the
276     * thread -- this will set rsp0 to the wrong value, but it's harmless
277     * as it's a kernel thread, and it won't actually attempt to implicitly
278     * use the rsp0 via a privilege change.
279     *
280     * Note that when we have KPTI enabled on amd64, we never use this
281     * value at all (since all the interrupts have an IST set).
282     */
283     movq   CPU_TSS(%r13), %r14
284     #if !defined(__xpv)
285     cmpq   $1, kpti_enable
286     jne   1f
287     leaq   CPU_KPTI_TR_RSP(%r13), %rax
288     jmp   2f
289 1:
290     movq   T_STACK(%r12), %rax
291     addq   $REGSIZE+MINFRAME, %rax /* to the bottom of thread stack */
292 2:

```

```

293     movq    %rax, TSS_RSP0(%r14)
294 #else
295     movq    T_STACK(%r12), %rax
296     addq    $REGSIZE+MINFRAME, %rax /* to the bottom of thread stack */
297     movl    $KDS_SEL, %edi
298     movq    %rax, %rsi
299     call   HYPERVISOR_stack_switch
300 #endif /* __xpv */

302     movq    %r12, CPU_THREAD(%r13) /* set CPU's thread pointer */
303     mfence /* synchronize with mutex_exit() */
304     xorl    %ebp, %ebp /* make $<threadlist behave better */
305     movq    T_LWP(%r12), %rax /* set associated lwp to */
306     movq    %rax, CPU_LWP(%r13) /* CPU's lwp ptr */

308     movq    T_SP(%r12), %rsp /* switch to outgoing thread's stack */
309     movq    T_PC(%r12), %r13 /* saved return addr */

311 /*
312  * Call restorectx if context ops have been installed.
313  */
314     cmpq    $0, T_CTX(%r12) /* should resumed thread restorectx? */
315     jz     .norestorectx /* skip call when zero */
316     movq    %r12, %rdi /* arg = thread pointer */
317     call   restorectx /* call ctx ops */
318 .norestorectx:

320 /*
321  * Call restoreptx if context ops have been installed for the proc.
322  */
323     movq    T_PROCP(%r12), %rcx
324     cmpq    $0, P_PCTX(%rcx)
325     jz     .norestoreptx
326     movq    %rcx, %rdi
327     call   restoreptx
328 .norestoreptx:

330     STORE_INTR_START(%r12)

332 /*
333  * If we came into swtch with the ability to access userland pages, go
334  * ahead and restore that fact by disabling SMAP. Clear the indicator
335  * flag out of paranoia.
336  */
337     movq    T_USERACC(%r12), %rax /* should we disable smap? */
338     cmpq    $0, %rax /* skip call when zero */
339     jz     .nosmap
340     xorq    %rax, %rax
341     movq    %rax, T_USERACC(%r12)
342     call   smap_disable
343 .nosmap:

345     call   ht_mark

347 /*
348  * Restore non-volatile registers, then have spl0 return to the
349  * resuming thread's PC after first setting the priority as low as
350  * possible and blocking all interrupt threads that may be active.
351  */
352     movq    %r13, %rax /* save return address */
353     RESTORE_REGS(%r11)
354     pushq   %rax /* push return address for spl0() */
355     call   __dtrace_probe__sched_on_cpu
356     jmp    spl0

358 resume_return:

```

```

359 /*
360  * Remove stack frame created in SAVE_REGS()
361  */
362     addq    $CLONGSIZE, %rsp
363     ret
364     SET_SIZE(_resume_from_idle)
    unchanged_portion_omitted

459 #elif defined (__i386)

461     ENTRY(resume)
462     movl    %gs:CPU_THREAD, %eax
463     movl    $resume_return, %ecx

465 /*
466  * Save non-volatile registers, and set return address for current
467  * thread to resume_return.
468  *
469  * %edi = t (new thread) when done.
470  */
471     SAVE_REGS(%eax, %ecx)

473     LOADCPU(%ebx) /* %ebx = CPU */
474     movl    CPU_THREAD(%ebx), %esi /* %esi = curthread */

476 #ifdef DEBUG
477     call   assert_ints_enabled /* panics if we are cli'd */
478 #endif

479 /*
480  * Call savectx if thread has installed context ops.
481  *
482  * Note that if we have floating point context, the save op
483  * (either fpsave_begin or fpxsave_begin) will issue the
484  * async save instruction (fnsave or fxsave respectively)
485  * that we wait for below.
486  */
487     movl    T_CTX(%esi), %eax /* should current thread savectx? */
488     testl   %eax, %eax
489     jz     .nosavectx /* skip call when zero */
490     pushl   %esi /* arg = thread pointer */
491     call   savectx /* call ctx ops */
492     addl    $4, %esp /* restore stack pointer */
493 .nosavectx:

495 /*
496  * Call saveptx if process has installed context ops.
497  */
498     movl    T_PROCP(%esi), %eax /* %eax = proc */
499     cmpl   $0, P_PCTX(%eax) /* should current thread saveptx? */
500     je     .nosaveptx /* skip call when zero */
501     pushl   %eax /* arg = proc pointer */
502     call   saveptx /* call ctx ops */
503     addl    $4, %esp
504 .nosaveptx:

506 /*
507  * Temporarily switch to the idle thread's stack
508  */
509     movl    CPU_IDLE_THREAD(%ebx), %eax /* idle thread pointer */

511 /*
512  * Set the idle thread as the current thread
513  */
514     movl    T_SP(%eax), %esp /* It is safe to set esp */
515     movl    %eax, CPU_THREAD(%ebx)

```



```

517 /* switch in the hat context for the new thread */
518 GET_THREAD_HATP(%ecx, %edi, %ecx)
519 pushl %ecx
520 call hat_switch
521 addl $4, %esp

523 /*
524 * Clear and unlock previous thread's t_lock
525 * to allow it to be dispatched by another processor.
526 */
527 movb $0, T_LOCK(%esi)

529 /*
530 * IMPORTANT: Registers at this point must be:
531 * %edi = new thread
532 *
533 * Here we are in the idle thread, have dropped the old thread.
534 */
535 ALTENTRY(_resume_from_idle)
536 /*
537 * spin until dispatched thread's mutex has
538 * been unlocked. this mutex is unlocked when
539 * it becomes safe for the thread to run.
540 */
541 .L4:
542 lock
543 btsl $0, T_LOCK(%edi) /* lock new thread's mutex */
544 jc .L4_2 /* lock did not succeed */

546 /*
547 * Fix CPU structure to indicate new running thread.
548 * Set pointer in new thread to the CPU structure.
549 */
550 LOADCPU(%esi) /* load current CPU pointer */
551 movl T_STACK(%edi), %eax /* here to use v pipeline of */
552 /* Pentium. Used few lines below */
553 cmpl %esi, T_CPU(%edi)
554 jne .L5_2
555 .L5_1:
556 /*
557 * Setup esp0 (kernel stack) in TSS to curthread's stack.
558 * (Note: Since we don't have saved 'regs' structure for all
559 * the threads we can't easily determine if we need to
560 * change esp0. So, we simply change the esp0 to bottom
561 * of the thread stack and it will work for all cases.)
562 */
563 movl CPU_TSS(%esi), %ecx
564 addl $REGSIZE+MINFRAME, %eax /* to the bottom of thread stack */
565 #if !defined(__xpv)
566 movl %eax, TSS_ESP0(%ecx)
567 #else
568 pushl %eax
569 pushl $KDS_SEL
570 call HYPERVISOR_stack_switch
571 addl $8, %esp
572 #endif /* __xpv */

574 movl %edi, CPU_THREAD(%esi) /* set CPU's thread pointer */
575 mfence /* synchronize with mutex_exit() */
576 xorl %ebp, %ebp /* make $<threadlist behave better */
577 movl T_LWP(%edi), %eax /* set associated lwp to */
578 movl %eax, CPU_LWP(%esi) /* CPU's lwp ptr */

580 movl T_SP(%edi), %esp /* switch to outgoing thread's stack */
581 movl T_PC(%edi), %esi /* saved return addr */

```

```

583 /*
584 * Call restorectx if context ops have been installed.
585 */
586 movl T_CTX(%edi), %eax /* should resumed thread restorectx? */
587 testl %eax, %eax
588 jz .norestorectx /* skip call when zero */
589 pushl %edi /* arg = thread pointer */
590 call restorectx /* call ctx ops */
591 addl $4, %esp /* restore stack pointer */
592 .norestorectx:

594 /*
595 * Call restorepctx if context ops have been installed for the proc.
596 */
597 movl T_PROCP(%edi), %eax
598 cmpl $0, P_PCTX(%eax)
599 je .norestorepctx
600 pushl %eax /* arg = proc pointer */
601 call restorepctx
602 addl $4, %esp /* restore stack pointer */
603 .norestorepctx:

605 STORE_INTR_START(%edi)

607 /*
608 * Restore non-volatile registers, then have spl0 return to the
609 * resuming thread's PC after first setting the priority as low as
610 * possible and blocking all interrupt threads that may be active.
611 */
612 movl %esi, %eax /* save return address */
613 RESTORE_REGS(%ecx)
614 pushl %eax /* push return address for spl0() */
615 call __dtrace_probe__sched_on_cpu
616 jmp spl0

618 resume_return:
619 /*
620 * Remove stack frame created in SAVE_REGS()
621 */
622 addl $CLONGSIZE, %esp
623 ret

625 .L4_2:
626 pause
627 cmpb $0, T_LOCK(%edi)
628 je .L4
629 jmp .L4_2

631 .L5_2:
632 /* cp->cpu_stats.sys.cpumigrate++ */
633 addl $1, CPU_STATS_SYS_CPUMIGRATE(%esi)
634 adcl $0, CPU_STATS_SYS_CPUMIGRATE+4(%esi)
635 movl %esi, T_CPU(%edi) /* set new thread's CPU pointer */
636 jmp .L5_1

638 SET_SIZE(_resume_from_idle)
639 SET_SIZE(resume)

641 #endif /* __amd64 */
642 #endif /* __lint */

644 #if defined(__lint)

646 /* ARGSUSED */
647 void
648 resume_from_zombie(kthread_t *t)

```

```

649 {}
651 #else /* __lint */
653 #if defined(__amd64)
654
667 ENTRY(resume_from_zombie)
668 movq   %gs:CPU_THREAD, %rax
669 leaq   resume_from_zombie_return(%rip), %r11
670
671 /*
672  * Save non-volatile registers, and set return address for current
673  * thread to resume_from_zombie_return.
674  *
675  * %r12 = t (new thread) when done
676  */
677 SAVE_REGS(%rax, %r11)
678
679 movq   %gs:CPU_THREAD, %r13 /* %r13 = curthread */
680
681 /* clean up the fp unit. It might be left enabled */
682
683 #if defined(__xpv) /* XXPV XXtclayton */
684 /*
685  * Remove this after bringup.
686  * (Too many #gp's for an instrumented hypervisor.)
687  */
688 STTS(%rax)
689 #else
690 movq   %cr0, %rax
691 testq  $CR0_TS, %rax
692 jnz    .zfpu_disabled /* if TS already set, nothing to do */
693 fninit /* init fpu & discard pending error */
694 orq   $CR0_TS, %rax
695 movq  %rax, %cr0
696 .zfpu_disabled:
697
698 #endif /* __xpv */
699
700 /*
701  * Temporarily switch to the idle thread's stack so that the zombie
702  * thread's stack can be reclaimed by the reaper.
703  */
704 movq   %gs:CPU_IDLE_THREAD, %rax /* idle thread pointer */
705 movq   T_SP(%rax), %rsp /* get onto idle thread stack */
706
707 /*
708  * Sigh. If the idle thread has never run thread_start()
709  * then t_sp is mis-aligned by thread_load().
710  */
711 andq   $_BITNOT(STACK_ALIGN-1), %rsp
712
713 /*
714  * Set the idle thread as the current thread.
715  */
716 movq   %rax, %gs:CPU_THREAD
717
718 /* switch in the hat context for the new thread */
719 GET_THREAD_HATP(%rdi, %r12, %r11)
720 call   hat_switch
721
722 /*
723  * Put the zombie on death-row.
724  */
725 movq   %r13, %rdi
726 call   reapq_add

```

```

428 jmp    _resume_from_idle /* finish job of resume */
429
430 resume_from_zombie_return:
431 RESTORE_REGS(%r11) /* restore non-volatile registers */
432 call   __dtrace_probe__sched_on_cpu
433
434 /*
435  * Remove stack frame created in SAVE_REGS()
436  */
437 addq   $CLONGSIZE, %rsp
438 ret
439 SET_SIZE(resume_from_zombie)
440
441 #elif defined(__i386)
442
443 ENTRY(resume_from_zombie)
444 movl   %gs:CPU_THREAD, %eax
445 movl   $resume_from_zombie_return, %ecx
446
447 /*
448  * Save non-volatile registers, and set return address for current
449  * thread to resume_from_zombie_return.
450  *
451  * %edi = t (new thread) when done.
452  */
453 SAVE_REGS(%eax, %ecx)
454
455 #ifdef DEBUG
456 call   assert_ints_enabled /* panics if we are cli'd */
457 #endif
458 movl   %gs:CPU_THREAD, %esi /* %esi = curthread */
459
460 /* clean up the fp unit. It might be left enabled */
461
462 movl   %cr0, %eax
463 testl  $CR0_TS, %eax
464 jnz    .zfpu_disabled /* if TS already set, nothing to do */
465 fninit /* init fpu & discard pending error */
466 orl   $CR0_TS, %eax
467 movl   %eax, %cr0
468 .zfpu_disabled:
469
470 /*
471  * Temporarily switch to the idle thread's stack so that the zombie
472  * thread's stack can be reclaimed by the reaper.
473  */
474 movl   %gs:CPU_IDLE_THREAD, %eax /* idle thread pointer */
475 movl   T_SP(%eax), %esp /* get onto idle thread stack */
476
477 /*
478  * Set the idle thread as the current thread.
479  */
480 movl   %eax, %gs:CPU_THREAD
481
482 /*
483  * switch in the hat context for the new thread
484  */
485 GET_THREAD_HATP(%ecx, %edi, %ecx)
486 pushl  %ecx
487 call   hat_switch
488 addl   $4, %esp
489
490 /*
491  * Put the zombie on death-row.
492  */

```

```

781     pushl   %esi
782     call    reapq_add
783     addl    $4, %esp
784     jmp     _resume_from_idle      /* finish job of resume */

786 resume_from_zombie_return:
787     RESTORE_REGS(%ecx)           /* restore non-volatile registers */
788     call    __dtrace_probe__sched_on__cpu

790     /*
791     * Remove stack frame created in SAVE_REGS()
792     */
793     addl    $CLONGSIZE, %esp
794     ret
795     SET_SIZE(resume_from_zombie)

797 #endif /* __amd64 */
798 #endif /* __lint */

800 #if defined(__lint)

802 /* ARGSUSED */
803 void
804 resume_from_intr(kthread_t *t)
805 {}

807 #else /* __lint */

809 #if defined(__amd64)

441     ENTRY(resume_from_intr)
442     movq    %gs:CPU_THREAD, %rax
443     leaq    resume_from_intr_return(%rip), %r11

445     /*
446     * Save non-volatile registers, and set return address for current
447     * thread to resume_from_intr_return.
448     *
449     * %r12 = t (new thread) when done
450     */
451     SAVE_REGS(%rax, %r11)

453     movq    %gs:CPU_THREAD, %r13    /* %r13 = curthread */
454     movq    %r12, %gs:CPU_THREAD    /* set CPU's thread pointer */
455     mfence                                /* synchronize with mutex_exit() */
456     movq    T_SP(%r12), %rsp        /* restore resuming thread's sp */
457     xorl    %ebp, %ebp              /* make $<threadlist behave better */

459     /*
460     * Unlock outgoing thread's mutex dispatched by another processor.
461     */
462     xorl    %eax, %eax
463     xchgb   %al, T_LOCK(%r13)

465     STORE_INTR_START(%r12)

467     call    ht_mark

469     /*
470     * Restore non-volatile registers, then have spl0 return to the
471     * resuming thread's PC after first setting the priority as low as
472     * possible and blocking all interrupt threads that may be active.
473     */
474     movq    T_PC(%r12), %rax        /* saved return addr */
475     RESTORE_REGS(%r11);
476     pushq   %rax                    /* push return address for spl0() */

```

```

477     call    __dtrace_probe__sched_on__cpu
478     jmp     spl0

480 resume_from_intr_return:
481     /*
482     * Remove stack frame created in SAVE_REGS()
483     */
484     addq    $CLONGSIZE, %rsp
485     ret
486     SET_SIZE(resume_from_intr)

856 #elif defined(__i386)

858     ENTRY(resume_from_intr)
859     movl    %gs:CPU_THREAD, %eax
860     movl    $resume_from_intr_return, %ecx

862     /*
863     * Save non-volatile registers, and set return address for current
864     * thread to resume_return.
865     *
866     * %edi = t (new thread) when done.
867     */
868     SAVE_REGS(%eax, %ecx)

870 #ifdef DEBUG
871     call    assert_ints_enabled    /* panics if we are cli'd */
872 #endif

873     movl    %gs:CPU_THREAD, %esi    /* %esi = curthread */
874     movl    %edi, %gs:CPU_THREAD    /* set CPU's thread pointer */
875     mfence                                /* synchronize with mutex_exit() */
876     movl    T_SP(%edi), %esp        /* restore resuming thread's sp */
877     xorl    %ebp, %ebp              /* make $<threadlist behave better */

879     /*
880     * Unlock outgoing thread's mutex dispatched by another processor.
881     */
882     xorl    %eax, %eax
883     xchgb   %al, T_LOCK(%esi)

885     STORE_INTR_START(%edi)

887     /*
888     * Restore non-volatile registers, then have spl0 return to the
889     * resuming thread's PC after first setting the priority as low as
890     * possible and blocking all interrupt threads that may be active.
891     */
892     movl    T_PC(%edi), %eax        /* saved return addr */
893     RESTORE_REGS(%ecx)
894     pushl   %eax                    /* push return address for spl0() */
895     call    __dtrace_probe__sched_on__cpu
896     jmp     spl0

898 resume_from_intr_return:
899     /*
900     * Remove stack frame created in SAVE_REGS()
901     */
902     addl    $CLONGSIZE, %esp
903     ret
904     SET_SIZE(resume_from_intr)

906 #endif /* __amd64 */
907 #endif /* __lint */

909 #if defined(__lint)

```

```
911 void
912 thread_start(void)
913 {}

915 #else /* __lint */

917 #if defined(__amd64)

488     ENTRY(thread_start)
489     popq   %rax           /* start() */
490     popq   %rdi           /* arg */
491     popq   %rsi           /* len */
492     movq   %rsp, %rbp
493     call   *%rax
494     call   thread_exit    /* destroy thread if it returns. */
495     /*NOTREACHED*/
496     SET_SIZE(thread_start)

929 #elif defined(__i386)

931     ENTRY(thread_start)
932     popl   %eax
933     movl   %esp, %ebp
934     addl   $8, %ebp
935     call   *%eax
936     addl   $8, %esp
937     call   thread_exit    /* destroy thread if it returns. */
938     /*NOTREACHED*/
939     SET_SIZE(thread_start)

941 #endif /* __i386 */

943 #endif /* __lint */
```

new/usr/src/uts/intel/sys/x86_archext.h

1

```
*****
41704 Tue May 7 04:50:02 2019
new/usr/src/uts/intel/sys/x86_archext.h
OS-7125 Need mitigation of L1TF (CVE-2018-3646)
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
*****
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) 1995, 2010, Oracle and/or its affiliates. All rights reserved.
23 * Copyright (c) 2011 by Delphix. All rights reserved.
24 */
25 /*
26 * Copyright (c) 2010, Intel Corporation.
27 * All rights reserved.
28 */
29 /*
30 * Copyright 2019, Joyent, Inc.
31 * Copyright 2012 Jens Elkner <jel+illumos@cs.uni-magdeburg.de>
32 * Copyright 2012 Hans Rosenfeld <rosenfeld@grumpf.hope-2000.org>
33 * Copyright 2014 Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
34 * Copyright 2018 Nexenta Systems, Inc.
35 */
37 #ifndef _SYS_X86_ARCHEXT_H
38 #define _SYS_X86_ARCHEXT_H
39
40 #if !defined(_ASM)
41 #include <sys/regset.h>
42 #include <sys/processor.h>
43 #include <vm/seg_enum.h>
44 #include <vm/page.h>
45 #endif /* _ASM */
46
47 #ifdef __cplusplus
48 extern "C" {
49 #endif
50
51 /*
52  * cpuid instruction feature flags in %edx (standard function 1)
53  */
54
55 #define CPUID_INTC_EDX_FPU 0x00000001 /* x87 fpu present */
56 #define CPUID_INTC_EDX_VME 0x00000002 /* virtual-8086 extension */
57 #define CPUID_INTC_EDX_DE 0x00000004 /* debugging extensions */
58 #define CPUID_INTC_EDX_PSE 0x00000008 /* page size extension */
59 #define CPUID_INTC_EDX_TSC 0x00000010 /* time stamp counter */
```

new/usr/src/uts/intel/sys/x86_archext.h

2

```
60 #define CPUID_INTC_EDX_MSR 0x00000020 /* rdmsr and wrmsr */
61 #define CPUID_INTC_EDX_PAE 0x00000040 /* physical addr extension */
62 #define CPUID_INTC_EDX_MCE 0x00000080 /* machine check exception */
63 #define CPUID_INTC_EDX_CX8 0x00000100 /* cmpxchg8b instruction */
64 #define CPUID_INTC_EDX_APIC 0x00000200 /* local APIC */
65
66 #define CPUID_INTC_EDX_SEP 0x00000800 /* 0x400 - reserved */
67 #define CPUID_INTC_EDX_MTRR 0x00001000 /* sysenter and sysexit */
68 #define CPUID_INTC_EDX_PGE 0x00002000 /* memory type range reg */
69 #define CPUID_INTC_EDX_MCA 0x00004000 /* page global enable */
70 #define CPUID_INTC_EDX_CMOV 0x00008000 /* machine check arch */
71 #define CPUID_INTC_EDX_PAT 0x00010000 /* conditional move insns */
72 #define CPUID_INTC_EDX_PSE36 0x00020000 /* page attribute table */
73 #define CPUID_INTC_EDX_PSN 0x00040000 /* 36-bit pagesize extension */
74 #define CPUID_INTC_EDX_CLFSH 0x00080000 /* processor serial number */
75
76 #define CPUID_INTC_EDX_DS 0x00200000 /* cflush instruction */
77 #define CPUID_INTC_EDX ACPI 0x00400000 /* 0x100000 - reserved */
78 #define CPUID_INTC_EDX_MMX 0x00800000 /* debug store exists */
79 #define CPUID_INTC_EDX_FXSR 0x01000000 /* monitoring + clock ctrl */
80 #define CPUID_INTC_EDX_SSE 0x02000000 /* MMX instructions */
81 #define CPUID_INTC_EDX_SSE2 0x04000000 /* fxsave and fxrstor */
82 #define CPUID_INTC_EDX_SS 0x08000000 /* streaming SIMD extensions */
83 #define CPUID_INTC_EDX_HTT 0x10000000 /* SSE extensions */
84 #define CPUID_INTC_EDX_TM 0x20000000 /* self-snoop */
85 #define CPUID_INTC_EDX_IA64 0x40000000 /* Hyper Thread Technology */
86 #define CPUID_INTC_EDX_PBE 0x80000000 /* thermal monitoring */
87
88 /*
89  * cpuid instruction feature flags in %ecx (standard function 1)
90  */
91
92 #define CPUID_INTC_ECX_SSE3 0x00000001 /* Yet more SSE extensions */
93 #define CPUID_INTC_ECX_PCLMULQDQ 0x00000002 /* PCLMULQDQ insn */
94 #define CPUID_INTC_ECX_DTES64 0x00000004 /* 64-bit DS area */
95 #define CPUID_INTC_ECX_MON 0x00000008 /* MONITOR/MWAIT */
96 #define CPUID_INTC_ECX_DSCPL 0x00000010 /* CPL-qualified debug store */
97 #define CPUID_INTC_ECX_VMX 0x00000020 /* Hardware VM extensions */
98 #define CPUID_INTC_ECX_SMX 0x00000040 /* Secure mode extensions */
99 #define CPUID_INTC_ECX_EST 0x00000080 /* enhanced SpeedStep */
100 #define CPUID_INTC_ECX_TM2 0x00000100 /* thermal monitoring */
101 #define CPUID_INTC_ECX_SSSE3 0x00000200 /* Supplemental SSE3 insns */
102 #define CPUID_INTC_ECX_CID 0x00000400 /* L1 context ID */
103
104 #define CPUID_INTC_ECX_FMA 0x00001000 /* 0x00000800 - reserved */
105 #define CPUID_INTC_ECX_CX16 0x00002000 /* Fused Multiply Add */
106 #define CPUID_INTC_ECX_ETPRD 0x00004000 /* cmpxchg16 */
107 #define CPUID_INTC_ECX_PDCM 0x00008000 /* extended task pri messages */
108
109 #define CPUID_INTC_ECX_PCID 0x00020000 /* Perf/Debug Capability MSR */
110 #define CPUID_INTC_ECX_DCA 0x00040000 /* 0x00010000 - reserved */
111 #define CPUID_INTC_ECX_SSE4_1 0x00080000 /* process-context ids */
112 #define CPUID_INTC_ECX_SSE4_2 0x00100000 /* direct cache access */
113 #define CPUID_INTC_ECX_X2APIC 0x00200000 /* SSE4.1 insns */
114 #define CPUID_INTC_ECX_MOVBE 0x00400000 /* SSE4.2 insns */
115 #define CPUID_INTC_ECX_POPCNT 0x00800000 /* x2APIC */
116 #define CPUID_INTC_ECX_TSCDL 0x01000000 /* MOVBE insn */
117 #define CPUID_INTC_ECX_AES 0x02000000 /* POPCNT insn */
118 #define CPUID_INTC_ECX_XSAVE 0x04000000 /* Deadline TSC */
119 #define CPUID_INTC_ECX_OSXSAVE 0x08000000 /* AES insns */
120 #define CPUID_INTC_ECX_AVX 0x10000000 /* XSAVE/XRESTOR insns */
121 #define CPUID_INTC_ECX_F16C 0x20000000 /* OS supports XSAVE insns */
122 #define CPUID_INTC_ECX_RDRAND 0x40000000 /* AVX supported */
123 #define CPUID_INTC_ECX_HV 0x80000000 /* F16C supported */
124
125 /*
126  * Hypervisor */
```

```

126 * cpuid instruction feature flags in %edx (extended function 0x80000001)
127 */

129 #define CPUID_AMD_EDX_FPU      0x00000001    /* x87 fpu present */
130 #define CPUID_AMD_EDX_VME      0x00000002    /* virtual-8086 extension */
131 #define CPUID_AMD_EDX_DE       0x00000004    /* debugging extensions */
132 #define CPUID_AMD_EDX_PSE      0x00000008    /* page size extensions */
133 #define CPUID_AMD_EDX_TSC      0x00000010    /* time stamp counter */
134 #define CPUID_AMD_EDX_MSR      0x00000020    /* rdmsr and wrmsr */
135 #define CPUID_AMD_EDX_PAE      0x00000040    /* physical addr extension */
136 #define CPUID_AMD_EDX_MCE      0x00000080    /* machine check exception */
137 #define CPUID_AMD_EDX_CX8      0x00000100    /* cmpxchg8b instruction */
138 #define CPUID_AMD_EDX_APIC      0x00000200    /* local APIC */
139
140 #define CPUID_AMD_EDX_SYSC      0x00000800    /* AMD: syscall and sysret */
141 #define CPUID_AMD_EDX_MTRR      0x00001000    /* memory type and range reg */
142 #define CPUID_AMD_EDX_PGE      0x00002000    /* page global enable */
143 #define CPUID_AMD_EDX_MCA      0x00004000    /* machine check arch */
144 #define CPUID_AMD_EDX_CMV      0x00008000    /* conditional move insns */
145 #define CPUID_AMD_EDX_PAT      0x00010000    /* K7: page attribute table */
146 #define CPUID_AMD_EDX_FCMOV     0x00010000    /* FCMOVcc etc. */
147 #define CPUID_AMD_EDX_PSE36     0x00020000    /* 36-bit pagesize extension */
148 /* 0x00040000 - reserved */
149 /* 0x00080000 - reserved */
150 #define CPUID_AMD_EDX_NX        0x00100000    /* AMD: no-execute page prot */
151 /* 0x00200000 - reserved */
152 #define CPUID_AMD_EDX_MMXamd    0x00400000    /* AMD: MMX extensions */
153 #define CPUID_AMD_EDX_MMX      0x00800000    /* MMX instructions */
154 #define CPUID_AMD_EDX_FXSR      0x01000000    /* fxsave and fxrstor */
155 #define CPUID_AMD_EDX_FFXSR     0x02000000    /* fast fxsave/fxrstor */
156 #define CPUID_AMD_EDX_1GPG      0x04000000    /* 1GB page */
157 #define CPUID_AMD_EDX_TSCP      0x08000000    /* rdtscl instruction */
158 /* 0x10000000 - reserved */
159 #define CPUID_AMD_EDX_LM        0x20000000    /* AMD: long mode */
160 #define CPUID_AMD_EDX_3DNowx    0x40000000    /* AMD: extensions to 3DNow! */
161 #define CPUID_AMD_EDX_3DNow     0x80000000    /* AMD: 3DNow! instructions */

163 #define CPUID_AMD_ECX_AHF64     0x00000001    /* LAHF and SAHF in long mode */
164 #define CPUID_AMD_ECX_CMP_LGCV  0x00000002    /* AMD: multicore chip */
165 #define CPUID_AMD_ECX_SVM       0x00000004    /* AMD: secure VM */
166 #define CPUID_AMD_ECX_EAS       0x00000008    /* extended apic space */
167 #define CPUID_AMD_ECX_CR8D      0x00000010    /* AMD: 32-bit mov %cr8 */
168 #define CPUID_AMD_ECX_LZCNT     0x00000020    /* AMD: LZCNT insn */
169 #define CPUID_AMD_ECX_SSE4A     0x00000040    /* AMD: SSE4A insns */
170 #define CPUID_AMD_ECX_MAS       0x00000080    /* AMD: MisAlignSse mmode */
171 #define CPUID_AMD_ECX_3DNP      0x00000100    /* AMD: 3DNowPrefetch */
172 #define CPUID_AMD_ECX_OSVW      0x00000200    /* AMD: OSVW */
173 #define CPUID_AMD_ECX_IBS       0x00000400    /* AMD: IBS */
174 #define CPUID_AMD_ECX_XOP       0x00000800    /* AMD: Extended Operation */
175 #define CPUID_AMD_ECX_SKINIT    0x00001000    /* AMD: SKINIT */
176 #define CPUID_AMD_ECX_WDT       0x00002000    /* AMD: WDT */
177 /* 0x00004000 - reserved */
178 #define CPUID_AMD_ECX_LWP       0x00008000    /* AMD: Lightweight profiling */
179 #define CPUID_AMD_ECX_FMA4      0x00010000    /* AMD: 4-operand FMA support */
180 /* 0x00020000 - reserved */
181 /* 0x00040000 - reserved */
182 #define CPUID_AMD_ECX_NIDMSR    0x00080000    /* AMD: Node ID MSR */
183 /* 0x00100000 - reserved */
184 #define CPUID_AMD_ECX_TBM       0x00200000    /* AMD: trailing bit manips. */
185 #define CPUID_AMD_ECX_TOPOEXT   0x00400000    /* AMD: Topology Extensions */
186 #define CPUID_AMD_ECX_PCEC      0x00800000    /* AMD: Core ext perf counter */
187 #define CPUID_AMD_ECX_PCENB     0x01000000    /* AMD: NB ext perf counter */
188 /* 0x02000000 - reserved */
189 #define CPUID_AMD_ECX_DBKP      0x40000000    /* AMD: Data breakpoint */
190 #define CPUID_AMD_ECX_PERFCTSC  0x08000000    /* AMD: TSC Perf Counter */
191 #define CPUID_AMD_ECX_PERFL3    0x10000000    /* AMD: L3 Perf Counter */

```

```

192 #define CPUID_AMD_ECX_MONITORX 0x20000000    /* AMD: clzero */
193 /* 0x40000000 - reserved */
194 /* 0x80000000 - reserved */

196 /*
197 * AMD uses %ebx for some of their features (extended function 0x80000008).
198 */
199 #define CPUID_AMD_EBX_CLZERO    0x00000001    /* AMD: CLZERO instr */
200 #define CPUID_AMD_EBX_IRCMSR    0x00000002    /* AMD: Ret. instrs MSR */
201 #define CPUID_AMD_EBX_ERR_PTR_ZERO 0x00000004    /* AMD: FP Err. Ptr. Zero */
202 #define CPUID_AMD_EBX_IBPB      0x00000100    /* AMD: IBPB */
203 #define CPUID_AMD_EBX_IBRS      0x00000400    /* AMD: IBRS */
204 #define CPUID_AMD_EBX_STIBP     0x00000800    /* AMD: STIBP */
205 #define CPUID_AMD_EBX_IBRS_ALL  0x00001000    /* AMD: Enhanced IBRS */
206 #define CPUID_AMD_EBX_STIBP_ALL 0x00002000    /* AMD: STIBP ALL */
207 #define CPUID_AMD_EBX_PREFER_IBRS 0x00004000    /* AMD: Don't retpoline */
208 #define CPUID_AMD_EBX_SSB      0x00100000    /* AMD: SSB */
209 #define CPUID_AMD_EBX_VIRT_SSB  0x00200000    /* AMD: VIRT SSB */
210 #define CPUID_AMD_EBX_SSB_NO    0x00400000    /* AMD: SSB Fixed */

212 /*
213 * Intel now seems to have claimed part of the "extended" function
214 * space that we previously for non-Intel implementors to use.
215 * More excitingly still, they've claimed bit 20 to mean LAHF/SAHF
216 * is available in long mode i.e. what AMD indicate using bit 0.
217 * On the other hand, everything else is labelled as reserved.
218 */
219 #define CPUID_INTC_ECX_AHF64     0x00100000    /* LAHF and SAHF in long mode */

221 /*
222 * Intel also uses cpuid leaf 7 to have additional instructions and features.
223 * Like some other leaves, but unlike the current ones we care about, it
224 * requires us to specify both a leaf in %eax and a sub-leaf in %ecx. To deal
225 * with the potential use of additional sub-leaves in the future, we now
226 * specifically label the EBX features with their leaf and sub-leaf.
227 */
228 #define CPUID_INTC_EBX_7_0_FSGSBASE 0x00000001    /* FSGSBASE */
229 #define CPUID_INTC_EBX_7_0_TSC_ADJ 0x00000002    /* TSC adjust MSR */
230 #define CPUID_INTC_EBX_7_0_SGX    0x00000004    /* SGX */
231 #define CPUID_INTC_EBX_7_0_BMI1   0x00000008    /* BMI1 instrs */
232 #define CPUID_INTC_EBX_7_0_HLE    0x00000010    /* HLE */
233 #define CPUID_INTC_EBX_7_0_AVX2   0x00000020    /* AVX2 supported */
234 /* Bit 6 is reserved */
235 #define CPUID_INTC_EBX_7_0_SMEP   0x00000080    /* SMEP in CR4 */
236 #define CPUID_INTC_EBX_7_0_BMI2   0x00000100    /* BMI2 instrs */
237 #define CPUID_INTC_EBX_7_0_ENH_REP_MOV 0x00000200    /* Enhanced REP MOVSB */
238 #define CPUID_INTC_EBX_7_0_INVPCID 0x00000400    /* invpcid instr */
239 #define CPUID_INTC_EBX_7_0_RTM    0x00000800    /* RTM instrs */
240 #define CPUID_INTC_EBX_7_0_PQM    0x00001000    /* QoS Monitoring */
241 #define CPUID_INTC_EBX_7_0_DEP_CS_DS 0x00002000    /* Deprecates CS/DS */
242 #define CPUID_INTC_EBX_7_0_MPX    0x00004000    /* Mem. Prot. Ext. */
243 #define CPUID_INTC_EBX_7_0_PQE    0x00080000    /* QoS Enforcement */
244 #define CPUID_INTC_EBX_7_0_AVX512F 0x00010000    /* AVX512 foundation */
245 #define CPUID_INTC_EBX_7_0_AVX512DQ 0x00020000    /* AVX512DQ */
246 #define CPUID_INTC_EBX_7_0_RDSEED 0x00040000    /* RDSEED instr */
247 #define CPUID_INTC_EBX_7_0_ADX    0x00080000    /* ADX instrs */
248 #define CPUID_INTC_EBX_7_0_SMAP    0x00100000    /* SMAP in CR 4 */
249 #define CPUID_INTC_EBX_7_0_AVX512IFMA 0x00200000    /* AVX512IFMA */
250 /* Bit 22 is reserved */
251 #define CPUID_INTC_EBX_7_0_CLFLUSHOPT 0x00800000    /* CLFLUSHOPT */
252 #define CPUID_INTC_EBX_7_0_CLWB   0x01000000    /* CLWB */
253 #define CPUID_INTC_EBX_7_0_PTRACE 0x02000000    /* Processor Trace */
254 #define CPUID_INTC_EBX_7_0_AVX512PF 0x04000000    /* AVX512PF */
255 #define CPUID_INTC_EBX_7_0_AVX512ER 0x08000000    /* AVX512ER */
256 #define CPUID_INTC_EBX_7_0_AVX512CD 0x10000000    /* AVX512CD */
257 #define CPUID_INTC_EBX_7_0_SHA    0x20000000    /* SHA extensions */

```

```

258 #define CPUID_INTC_EBX_7_0_AVX512BW 0x40000000 /* AVX512BW */
259 #define CPUID_INTC_EBX_7_0_AVX512VL 0x80000000 /* AVX512VL */

261 #define CPUID_INTC_EBX_7_0_ALL_AVX512 \
262 (CPUID_INTC_EBX_7_0_AVX512F | CPUID_INTC_EBX_7_0_AVX512DQ | \
263 CPUID_INTC_EBX_7_0_AVX512IFMA | CPUID_INTC_EBX_7_0_AVX512PF | \
264 CPUID_INTC_EBX_7_0_AVX512ER | CPUID_INTC_EBX_7_0_AVX512CD | \
265 CPUID_INTC_EBX_7_0_AVX512BW | CPUID_INTC_EBX_7_0_AVX512VL)

267 #define CPUID_INTC_ECX_7_0_PREFETCHWT1 0x00000001 /* PREFETCHWT1 */
268 #define CPUID_INTC_ECX_7_0_AVX512VBMI 0x00000002 /* AVX512VBMI */
269 #define CPUID_INTC_ECX_7_0_UMIP 0x00000004 /* UMIP */
270 #define CPUID_INTC_ECX_7_0_PKU 0x00000008 /* umode prot. keys */
271 #define CPUID_INTC_ECX_7_0_OSPKE 0x00000010 /* OSPKE */
272 #define CPUID_INTC_ECX_7_0_WAITPKG 0x00000020 /* WAITPKG */
273 #define CPUID_INTC_ECX_7_0_AVX512VBMI2 0x00000040 /* AVX512 VBMI2 */
274 /* bit 7 is reserved */
275 #define CPUID_INTC_ECX_7_0_GFNI 0x00000100 /* GFNI */
276 #define CPUID_INTC_ECX_7_0_VAES 0x00000200 /* VAES */
277 #define CPUID_INTC_ECX_7_0_VPCLMULQDQ 0x00000400 /* VPCLMULQDQ */
278 #define CPUID_INTC_ECX_7_0_AVX512VNNI 0x00000800 /* AVX512 VNNI */
279 #define CPUID_INTC_ECX_7_0_AVX512BITALG 0x00001000 /* AVX512 BITALG */
280 /* bit 13 is reserved */
281 #define CPUID_INTC_ECX_7_0_AVX512VPOPCDQ 0x00004000 /* AVX512 VPOPCNTDQ */
282 /* bits 15-16 are reserved */
283 /* bits 17-21 are the value of MAWAU */
284 #define CPUID_INTC_ECX_7_0_RDPID 0x00400000 /* RPID, IA32_TSC_AUX */
285 /* bits 23-24 are reserved */
286 #define CPUID_INTC_ECX_7_0_CLDEMOTTE 0x02000000 /* Cache line demote */
287 /* bit 26 is reserved */
288 #define CPUID_INTC_ECX_7_0_MOVDIRI 0x08000000 /* MOVDIRI insn */
289 #define CPUID_INTC_ECX_7_0_MOVDIRI64B 0x10000000 /* MOVDIRI64B insn */
290 /* bit 29 is reserved */
291 #define CPUID_INTC_ECX_7_0_SGXLC 0x40000000 /* SGX Launch config */
292 /* bit 31 is reserved */

294 /*
295 * While CPUID_INTC_ECX_7_0_GFNI, CPUID_INTC_ECX_7_0_VAES, and
296 * CPUID_INTC_ECX_7_0_VPCLMULQDQ all have AVX512 components, they are still
297 * valid when AVX512 is not. However, the following flags all are only valid
298 * when AVX512 is present.
299 */
300 #define CPUID_INTC_ECX_7_0_ALL_AVX512 \
301 (CPUID_INTC_ECX_7_0_AVX512VBMI | CPUID_INTC_ECX_7_0_AVX512VNNI | \
302 CPUID_INTC_ECX_7_0_AVX512BITALG | CPUID_INTC_ECX_7_0_AVX512VPOPCDQ)

304 /* bits 0-1 are reserved */
305 #define CPUID_INTC_EDX_7_0_AVX5124NNIW 0x00000004 /* AVX512 4NNIW */
306 #define CPUID_INTC_EDX_7_0_AVX5124FMAPS 0x00000008 /* AVX512 4FMAPS */
307 #define CPUID_INTC_EDX_7_0_FSREPMOV 0x00000010 /* fast short rep mov */
308 /* bits 5-17 are reserved */
309 #define CPUID_INTC_EDX_7_0_PCONFIG 0x00040000 /* PCONFIG */
310 /* bits 19-26 are reserved */
311 #define CPUID_INTC_EDX_7_0_SPEC_CTRL 0x04000000 /* Spec, IBPB, IBRS */
312 #define CPUID_INTC_EDX_7_0_STIBP 0x08000000 /* STIBP */
313 #define CPUID_INTC_EDX_7_0_FLUSH_CMD 0x10000000 /* IA32_FLUSH_CMD */
314 #define CPUID_INTC_EDX_7_0_ARCH_CAPS 0x20000000 /* IA32_ARCH_CAPS */
315 #define CPUID_INTC_EDX_7_0_SSSD 0x80000000 /* SSSD */

317 #define CPUID_INTC_EDX_7_0_ALL_AVX512 \
318 (CPUID_INTC_EDX_7_0_AVX5124NNIW | CPUID_INTC_EDX_7_0_AVX5124FMAPS)

320 /*
321 * Intel also uses cpuid leaf 0xd to report additional instructions and features
322 * when the sub-leaf in %ecx == 1. We label these using the same convention as
323 * with leaf 7.

```

```

324 /*
325 #define CPUID_INTC_EAX_D_1_XSAVEOPT 0x00000001 /* xsaveopt inst. */
326 #define CPUID_INTC_EAX_D_1_XSAVEC 0x00000002 /* xsavec inst. */
327 #define CPUID_INTC_EAX_D_1_XSAVES 0x00000008 /* xsaves inst. */

329 #define REG_PAT 0x277
330 #define REG_TSC 0x10 /* timestamp counter */
331 #define REG_APIC_BASE_MSR 0x1b
332 #define REG_X2APIC_BASE_MSR 0x800 /* The MSR address offset of x2APIC */

334 #if !defined(__xpv)
335 /*
336 * AMD C1E
337 */
338 #define MSR_AMD_INT_PENDING_CMP_HALT 0xC0010055
339 #define AMD_ACTONCMPHALT_SHIFT 27
340 #define AMD_ACTONCMPHALT_MASK 3
341 #endif

343 #define MSR_DEBUGCTL 0x1d9

345 #define DEBUGCTL_LBR 0x01
346 #define DEBUGCTL_BTF 0x02

348 /* Intel P6, AMD */
349 #define MSR_LBR_FROM 0x1db
350 #define MSR_LBR_TO 0x1dc
351 #define MSR_LEX_FROM 0x1dd
352 #define MSR_LEX_TO 0x1de

354 /* Intel P4 (pre-Prescott, non P4 M) */
355 #define MSR_P4_LBSTK_TOS 0x1da
356 #define MSR_P4_LBSTK_0 0x1db
357 #define MSR_P4_LBSTK_1 0x1dc
358 #define MSR_P4_LBSTK_2 0x1dd
359 #define MSR_P4_LBSTK_3 0x1de

361 /* Intel Pentium M */
362 #define MSR_P6M_LBSTK_TOS 0x1c9
363 #define MSR_P6M_LBSTK_0 0x040
364 #define MSR_P6M_LBSTK_1 0x041
365 #define MSR_P6M_LBSTK_2 0x042
366 #define MSR_P6M_LBSTK_3 0x043
367 #define MSR_P6M_LBSTK_4 0x044
368 #define MSR_P6M_LBSTK_5 0x045
369 #define MSR_P6M_LBSTK_6 0x046
370 #define MSR_P6M_LBSTK_7 0x047

372 /* Intel P4 (Prescott) */
373 #define MSR_PR4_LBSTK_TOS 0x1da
374 #define MSR_PR4_LBSTK_FROM_0 0x680
375 #define MSR_PR4_LBSTK_FROM_1 0x681
376 #define MSR_PR4_LBSTK_FROM_2 0x682
377 #define MSR_PR4_LBSTK_FROM_3 0x683
378 #define MSR_PR4_LBSTK_FROM_4 0x684
379 #define MSR_PR4_LBSTK_FROM_5 0x685
380 #define MSR_PR4_LBSTK_FROM_6 0x686
381 #define MSR_PR4_LBSTK_FROM_7 0x687
382 #define MSR_PR4_LBSTK_FROM_8 0x688
383 #define MSR_PR4_LBSTK_FROM_9 0x689
384 #define MSR_PR4_LBSTK_FROM_10 0x68a
385 #define MSR_PR4_LBSTK_FROM_11 0x68b
386 #define MSR_PR4_LBSTK_FROM_12 0x68c
387 #define MSR_PR4_LBSTK_FROM_13 0x68d
388 #define MSR_PR4_LBSTK_FROM_14 0x68e
389 #define MSR_PR4_LBSTK_FROM_15 0x68f

```

```

390 #define MSR_PRP4_LBSTK_TO_0    0x6c0
391 #define MSR_PRP4_LBSTK_TO_1    0x6c1
392 #define MSR_PRP4_LBSTK_TO_2    0x6c2
393 #define MSR_PRP4_LBSTK_TO_3    0x6c3
394 #define MSR_PRP4_LBSTK_TO_4    0x6c4
395 #define MSR_PRP4_LBSTK_TO_5    0x6c5
396 #define MSR_PRP4_LBSTK_TO_6    0x6c6
397 #define MSR_PRP4_LBSTK_TO_7    0x6c7
398 #define MSR_PRP4_LBSTK_TO_8    0x6c8
399 #define MSR_PRP4_LBSTK_TO_9    0x6c9
400 #define MSR_PRP4_LBSTK_TO_10   0x6ca
401 #define MSR_PRP4_LBSTK_TO_11   0x6cb
402 #define MSR_PRP4_LBSTK_TO_12   0x6cc
403 #define MSR_PRP4_LBSTK_TO_13   0x6cd
404 #define MSR_PRP4_LBSTK_TO_14   0x6ce
405 #define MSR_PRP4_LBSTK_TO_15   0x6cf

407 /*
408  * General Xeon based MSRs
409  */
410 #define MSR_PPIN_CTL            0x04e
411 #define MSR_PPIN                0x04f
412 #define MSR_PLATFORM_INFO       0x0ce

414 #define MSR_PLATFORM_INFO_PPIN  (1 << 23)
415 #define MSR_PPIN_CTL_MASK      0x03
416 #define MSR_PPIN_CTL_LOCKED    0x01
417 #define MSR_PPIN_CTL_ENABLED   0x02

419 /*
420  * Intel IA32_ARCH_CAPABILITIES MSR.
421  */
422 #define MSR_IA32_ARCH_CAPABILITIES    0x10a
423 #define IA32_ARCH_CAP_RDCL_NO         0x0001
424 #define IA32_ARCH_CAP_IBRS_ALL        0x0002
425 #define IA32_ARCH_CAP_RSBA             0x0004
426 #define IA32_ARCH_CAP_SKIP_L1DFL_VMENTRY 0x0008
427 #define IA32_ARCH_CAP_SSB_NO          0x0010

429 /*
430  * Intel Speculation related MSRs
431  */
432 #define MSR_IA32_SPEC_CTRL            0x48
433 #define IA32_SPEC_CTRL_IBRS           0x01
434 #define IA32_SPEC_CTRL_STIBP          0x02
435 #define IA32_SPEC_CTRL_SSB            0x04

437 #define MSR_IA32_PRED_CMD              0x49
438 #define IA32_PRED_CMD_IBPB            0x01

440 #define MSR_IA32_FLUSH_CMD             0x10b
441 #define IA32_FLUSH_CMD_L1D            0x01

443 #define MCI_CTL_VALUE                   0xffffffff

445 #define MTRR_TYPE_UC                     0
446 #define MTRR_TYPE_WC                     1
447 #define MTRR_TYPE_WT                     4
448 #define MTRR_TYPE_WP                     5
449 #define MTRR_TYPE_WB                     6
450 #define MTRR_TYPE_UC_                    7

452 /*
453  * For Solaris we set up the page attribute table in the following way:
454  * PAT0 Write-Back
455  * PAT1 Write-Through

```

```

456 * PAT2 Uncacheable-
457 * PAT3 Uncacheable
458 * PAT4 Write-Back
459 * PAT5 Write-Through
460 * PAT6 Write-Combine
461 * PAT7 Uncacheable
462 * The only difference from h/w default is entry 6.
463 */
464 #define PAT_DEFAULT_ATTRIBUTE          \
465     ((uint64_t)MTRR_TYPE_WB |        \
466     ((uint64_t)MTRR_TYPE_WT << 8) | \
467     ((uint64_t)MTRR_TYPE_UC_ << 16) | \
468     ((uint64_t)MTRR_TYPE_UC << 24) | \
469     ((uint64_t)MTRR_TYPE_WB << 32) | \
470     ((uint64_t)MTRR_TYPE_WT << 40) | \
471     ((uint64_t)MTRR_TYPE_WC << 48) | \
472     ((uint64_t)MTRR_TYPE_UC << 56))

474 #define X86FSET_LARGEPAGE            0
475 #define X86FSET_TSC                   1
476 #define X86FSET_MSR                   2
477 #define X86FSET_MTRR                  3
478 #define X86FSET_PGE                    4
479 #define X86FSET_DE                     5
480 #define X86FSET_CMOV                   6
481 #define X86FSET_MMX                     7
482 #define X86FSET_MCA                     8
483 #define X86FSET_PAE                     9
484 #define X86FSET_CX8                    10
485 #define X86FSET_PAT                     11
486 #define X86FSET_SEP                     12
487 #define X86FSET_SSE                     13
488 #define X86FSET_SSE2                    14
489 #define X86FSET_HTTP                     15
490 #define X86FSET_ASYS                     16
491 #define X86FSET_NX                       17
492 #define X86FSET_SSE3                     18
493 #define X86FSET_CX16                     19
494 #define X86FSET_CMP                      20
495 #define X86FSET_TSCP                     21
496 #define X86FSET_MWAIT                    22
497 #define X86FSET_SSE4A                    23
498 #define X86FSET_CPUID                     24
499 #define X86FSET_SSSE3                     25
500 #define X86FSET_SSE4_1                    26
501 #define X86FSET_SSE4_2                    27
502 #define X86FSET_1GPG                      28
503 #define X86FSET_CLFSH                     29
504 #define X86FSET_64                        30
505 #define X86FSET_AES                       31
506 #define X86FSET_PCLMULQDQ                 32
507 #define X86FSET_XSAVE                     33
508 #define X86FSET_AVX                       34
509 #define X86FSET_VMX                       35
510 #define X86FSET_SVM                       36
511 #define X86FSET_TOPOEXT                    37
512 #define X86FSET_F16C                      38
513 #define X86FSET_RDRAND                    39
514 #define X86FSET_X2APIC                    40
515 #define X86FSET_AVX2                      41
516 #define X86FSET_BMI1                      42
517 #define X86FSET_BMI2                      43
518 #define X86FSET_FMA                       44
519 #define X86FSET_SMEP                      45
520 #define X86FSET_SMAP                      46
521 #define X86FSET_ADX                       47

```



```

522 #define X86FSET_RDSEED      48
523 #define X86FSET_MPX        49
524 #define X86FSET_AVX512F    50
525 #define X86FSET_AVX512DQ   51
526 #define X86FSET_AVX512PF   52
527 #define X86FSET_AVX512ER   53
528 #define X86FSET_AVX512CD   54
529 #define X86FSET_AVX512BW   55
530 #define X86FSET_AVX512VL   56
531 #define X86FSET_AVX512FMA  57
532 #define X86FSET_AVX512VBMI 58
533 #define X86FSET_AVX512VPOPCDQ 59
534 #define X86FSET_AVX512NNIW 60
535 #define X86FSET_AVX512FMAPS 61
536 #define X86FSET_XSAVEOPT   62
537 #define X86FSET_XSAVEC     63
538 #define X86FSET_XSAVES     64
539 #define X86FSET_SHA        65
540 #define X86FSET_UMIP       66
541 #define X86FSET_PKU        67
542 #define X86FSET_OSPKE     68
543 #define X86FSET_PCID      69
544 #define X86FSET_INVPCID   70
545 #define X86FSET_IBRS      71
546 #define X86FSET_IBPB     72
547 #define X86FSET_STIBP    73
548 #define X86FSET_SSBD     74
549 #define X86FSET_SSBD_VIRT 75
550 #define X86FSET_RDCL_NO   76
551 #define X86FSET_IBRS_ALL  77
552 #define X86FSET_RSBA     78
553 #define X86FSET_SSB_NO   79
554 #define X86FSET_STIBP_ALL 80
555 #define X86FSET_FLUSH_CMD 81
556 #define X86FSET_L1D_VM_NO 82
557 #define X86FSET_FSGSBASE 83
558 #define X86FSET_CLFLUSHOPT 84
559 #define X86FSET_CLWB     85
560 #define X86FSET_MONITORX  86
561 #define X86FSET_CLZERO   87
562 #define X86FSET_XOP      88
563 #define X86FSET_FMA4     89
564 #define X86FSET_TBM     90
565 #define X86FSET_AVX512VNNI 91

567 /*
568  * Intel Deep C-State invariant TSC in leaf 0x80000007.
569  */
570 #define CPUID_TSC_CSTATE_INVARIANCE    (0x100)

572 /*
573  * Intel Deep C-state always-running local APIC timer
574  */
575 #define CPUID_CSTATE_ARAT              (0x4)

577 /*
578  * Intel ENERGY_PERF_BIAS MSR indicated by feature bit CPUID.6.ECX[3].
579  */
580 #define CPUID_EPB_SUPPORT              (1 << 3)

582 /*
583  * Intel TSC deadline timer
584  */
585 #define CPUID_DEADLINE_TSC            (1 << 24)

587 /*

```

```

588  * x86_type is a legacy concept; this is supplanted
589  * for most purposes by x86_featureset; modern CPUs
590  * should be X86_TYPE_OTHER
591  */
592 #define X86_TYPE_OTHER        0
593 #define X86_TYPE_486         1
594 #define X86_TYPE_P5         2
595 #define X86_TYPE_P6         3
596 #define X86_TYPE_CYRIX_486  4
597 #define X86_TYPE_CYRIX_6x86L 5
598 #define X86_TYPE_CYRIX_6x86 6
599 #define X86_TYPE_CYRIX_GXm  7
600 #define X86_TYPE_CYRIX_6x86MX 8
601 #define X86_TYPE_CYRIX_MediaGX 9
602 #define X86_TYPE_CYRIX_MII  10
603 #define X86_TYPE_VIA_CYRIX_III 11
604 #define X86_TYPE_P4         12

606 /*
607  * x86_vendor allows us to select between
608  * implementation features and helps guide
609  * the interpretation of the cpuid instruction.
610  */
611 #define X86_VENDOR_Intel      0
612 #define X86_VENDORSTR_Intel  "GenuineIntel"

614 #define X86_VENDOR_IntelClone 1

616 #define X86_VENDOR_AMD       2
617 #define X86_VENDORSTR_AMD   "AuthenticAMD"

619 #define X86_VENDOR_Cyrix     3
620 #define X86_VENDORSTR_CYRIX "CyrixInstead"

622 #define X86_VENDOR_UMC       4
623 #define X86_VENDORSTR_UMC   "UMC UMC UMC "

625 #define X86_VENDOR_NexGen    5
626 #define X86_VENDORSTR_NexGen "NexGenDriven"

628 #define X86_VENDOR_Centaur   6
629 #define X86_VENDORSTR_Centaur "CentaurHauls"

631 #define X86_VENDOR_Rise      7
632 #define X86_VENDORSTR_Rise   "RiseRiseRise"

634 #define X86_VENDOR_SiS      8
635 #define X86_VENDORSTR_SiS   "SiS SiS SiS "

637 #define X86_VENDOR_TM        9
638 #define X86_VENDORSTR_TM    "GenuineTMx86"

640 #define X86_VENDOR_NSC      10
641 #define X86_VENDORSTR_NSC   "Geode by NSC"

643 /*
644  * Vendor string max len + \0
645  */
646 #define X86_VENDOR_STRLEN    13

648 /*
649  * Some vendor/family/model/stepping ranges are commonly grouped under
650  * a single identifying banner by the vendor. The following encode
651  * that "revision" in a uint32_t with the 8 most significant bits
652  * identifying the vendor with X86_VENDOR_*, the next 8 identifying the
653  * family, and the remaining 16 typically forming a bitmask of revisions

```

```

654 * within that family with more significant bits indicating "later" revisions.
655 */

657 #define _X86_CHIPREV_VENDOR_MASK      0xff000000u
658 #define _X86_CHIPREV_VENDOR_SHIFT    24
659 #define _X86_CHIPREV_FAMILY_MASK     0x00ff0000u
660 #define _X86_CHIPREV_FAMILY_SHIFT    16
661 #define _X86_CHIPREV_REV_MASK        0x0000ffffu

663 #define _X86_CHIPREV_VENDOR(x) \
664 ((x) & _X86_CHIPREV_VENDOR_MASK) >> _X86_CHIPREV_VENDOR_SHIFT)
665 #define _X86_CHIPREV_FAMILY(x) \
666 ((x) & _X86_CHIPREV_FAMILY_MASK) >> _X86_CHIPREV_FAMILY_SHIFT)
667 #define _X86_CHIPREV_REV(x) \
668 ((x) & _X86_CHIPREV_REV_MASK)

670 /* True if x matches in vendor and family and if x matches the given rev mask */
671 #define X86_CHIPREV_MATCH(x, mask) \
672 (_X86_CHIPREV_VENDOR(x) == _X86_CHIPREV_VENDOR(mask) && \
673  _X86_CHIPREV_FAMILY(x) == _X86_CHIPREV_FAMILY(mask) && \
674  ((_X86_CHIPREV_REV(x) & _X86_CHIPREV_REV(mask)) != 0))

676 /* True if x matches in vendor and family, and rev is at least minx */
677 #define X86_CHIPREV_ATLEAST(x, minx) \
678 (_X86_CHIPREV_VENDOR(x) == _X86_CHIPREV_VENDOR(minx) && \
679  _X86_CHIPREV_FAMILY(x) == _X86_CHIPREV_FAMILY(minx) && \
680  _X86_CHIPREV_REV(x) >= _X86_CHIPREV_REV(minx))

682 #define _X86_CHIPREV_MKREV(vendor, family, rev) \
683 ((uint32_t)(vendor) << _X86_CHIPREV_VENDOR_SHIFT | \
684  (family) << _X86_CHIPREV_FAMILY_SHIFT | (rev))

686 /* True if x matches in vendor, and family is at least minx */
687 #define X86_CHIPFAM_ATLEAST(x, minx) \
688 (_X86_CHIPREV_VENDOR(x) == _X86_CHIPREV_VENDOR(minx) && \
689  _X86_CHIPREV_FAMILY(x) >= _X86_CHIPREV_FAMILY(minx))

691 /* Revision default */
692 #define X86_CHIPREV_UNKNOWN      0x0

694 /*
695 * Definitions for AMD Family 0xf. Minor revisions C0 and CG are
696 * sufficiently different that we will distinguish them; in all other
697 * case we will identify the major revision.
698 */
699 #define X86_CHIPREV_AMD_F_REV_B _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0xf, 0x0001)
700 #define X86_CHIPREV_AMD_F_REV_C0 _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0xf, 0x0002)
701 #define X86_CHIPREV_AMD_F_REV_CG _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0xf, 0x0004)
702 #define X86_CHIPREV_AMD_F_REV_D _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0xf, 0x0008)
703 #define X86_CHIPREV_AMD_F_REV_E _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0xf, 0x0010)
704 #define X86_CHIPREV_AMD_F_REV_F _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0xf, 0x0020)
705 #define X86_CHIPREV_AMD_F_REV_G _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0xf, 0x0040)

707 /*
708 * Definitions for AMD Family 0x10. Rev A was Engineering Samples only.
709 */
710 #define X86_CHIPREV_AMD_10_REV_A \
711  _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x10, 0x0001)
712 #define X86_CHIPREV_AMD_10_REV_B \
713  _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x10, 0x0002)
714 #define X86_CHIPREV_AMD_10_REV_C2 \
715  _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x10, 0x0004)
716 #define X86_CHIPREV_AMD_10_REV_C3 \
717  _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x10, 0x0008)
718 #define X86_CHIPREV_AMD_10_REV_D0 \
719  _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x10, 0x0010)

```

```

720 #define X86_CHIPREV_AMD_10_REV_D1 \
721  _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x10, 0x0020)
722 #define X86_CHIPREV_AMD_10_REV_E \
723  _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x10, 0x0040)

725 /*
726 * Definitions for AMD Family 0x11.
727 */
728 #define X86_CHIPREV_AMD_11_REV_B \
729  _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x11, 0x0002)

731 /*
732 * Definitions for AMD Family 0x12.
733 */
734 #define X86_CHIPREV_AMD_12_REV_B \
735  _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x12, 0x0002)

737 /*
738 * Definitions for AMD Family 0x14.
739 */
740 #define X86_CHIPREV_AMD_14_REV_B \
741  _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x14, 0x0002)
742 #define X86_CHIPREV_AMD_14_REV_C \
743  _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x14, 0x0004)

745 /*
746 * Definitions for AMD Family 0x15
747 */
748 #define X86_CHIPREV_AMD_15OR_REV_B2 \
749  _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x15, 0x0001)

751 #define X86_CHIPREV_AMD_15TN_REV_A1 \
752  _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x15, 0x0002)

754 #define X86_CHIPREV_AMD_15OR_REV_C0 \
755  _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x15, 0x0003)

757 #define X86_CHIPREV_AMD_15KV_REV_A1 \
758  _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x15, 0x0004)

760 #define X86_CHIPREV_AMD_15F60 \
761  _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x15, 0x0005)

763 #define X86_CHIPREV_AMD_15ST_REV_A0 \
764  _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x15, 0x0006)

766 /*
767 * Definitions for AMD Family 0x16
768 */
769 #define X86_CHIPREV_AMD_16_KB_A1 \
770  _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x16, 0x0001)

772 #define X86_CHIPREV_AMD_16_ML_A1 \
773  _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x16, 0x0002)

775 /*
776 * Definitions for AMD Family 0x17
777 */
779 #define X86_CHIPREV_AMD_17_ZP_B1 \
780  _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x17, 0x0001)

782 #define X86_CHIPREV_AMD_17_ZP_B2 \
783  _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x17, 0x0002)

785 #define X86_CHIPREV_AMD_17_PiR_B2 \

```

```

786     _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x17, 0x0003)

788 /*
789  * Various socket/package types, extended as the need to distinguish
790  * a new type arises. The top 8 byte identifies the vendor and the
791  * remaining 24 bits describe 24 socket types.
792  */

794 #define _X86_SOCKET_VENDOR_SHIFT      24
795 #define _X86_SOCKET_VENDOR(x)        ((x) >> _X86_SOCKET_VENDOR_SHIFT)
796 #define _X86_SOCKET_TYPE_MASK        0x00ffffff
797 #define _X86_SOCKET_TYPE(x)          ((x) & _X86_SOCKET_TYPE_MASK)

799 #define _X86_SOCKET_MKVAL(vendor, bitval) \
800     ((uint32_t)(vendor) << _X86_SOCKET_VENDOR_SHIFT | (bitval))

802 #define X86_SOCKET_MATCH(s, mask) \
803     (_X86_SOCKET_VENDOR(s) == _X86_SOCKET_VENDOR(mask) && \
804     (_X86_SOCKET_TYPE(s) & _X86_SOCKET_TYPE(mask)) != 0)

806 #define X86_SOCKET_UNKNOWN 0x0
807 /*
808  * AMD socket types
809  */
810 #define X86_SOCKET_754          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x01)
811 #define X86_SOCKET_939          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x02)
812 #define X86_SOCKET_940          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x03)
813 #define X86_SOCKET_S1g1        _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x04)
814 #define X86_SOCKET_AM2          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x05)
815 #define X86_SOCKET_F1207        _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x06)
816 #define X86_SOCKET_S1g2        _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x07)
817 #define X86_SOCKET_S1g3        _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x08)
818 #define X86_SOCKET_AM           _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x09)
819 #define X86_SOCKET_AM2R2        _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x0a)
820 #define X86_SOCKET_AM3          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x0b)
821 #define X86_SOCKET_G34          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x0c)
822 #define X86_SOCKET_ASB2        _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x0d)
823 #define X86_SOCKET_C32          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x0e)
824 #define X86_SOCKET_S1g4        _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x0f)
825 #define X86_SOCKET_FT1          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x10)
826 #define X86_SOCKET_FM1          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x11)
827 #define X86_SOCKET_FS1          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x12)
828 #define X86_SOCKET_AM3R2        _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x13)
829 #define X86_SOCKET_FP2          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x14)
830 #define X86_SOCKET_FS1R2        _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x15)
831 #define X86_SOCKET_FM2          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x16)
832 #define X86_SOCKET_FP3          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x17)
833 #define X86_SOCKET_FM2R2        _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x18)
834 #define X86_SOCKET_FP4          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x19)
835 #define X86_SOCKET_AM4          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x1a)
836 #define X86_SOCKET_FT3          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x1b)
837 #define X86_SOCKET_FT4          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x1c)
838 #define X86_SOCKET_FS1B         _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x1d)
839 #define X86_SOCKET_FT3B         _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x1e)
840 #define X86_SOCKET_SP3          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x1f)
841 #define X86_SOCKET_SP3R2        _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x20)
842 #define X86_NUM_SOCKETS_AMD     0x21

845 /*
846  * Definitions for Intel processor models. These are all for Family 6
847  * processors. This list and the Atom set below it are not exhaustive.
848  */
849 #define INTC_MODEL_MEROM          0x0f
850 #define INTC_MODEL_PENRYN        0x17
851 #define INTC_MODEL_DUNNINGTON    0x1d

```

```

853 #define INTC_MODEL_NEHALEM        0x1e
854 #define INTC_MODEL_NEHALEM2      0x1f
855 #define INTC_MODEL_NEHALEM_EP    0x1a
856 #define INTC_MODEL_NEHALEM_EX    0x2e

858 #define INTC_MODEL_WESTMERE       0x25
859 #define INTC_MODEL_WESTMERE_EP   0x2c
860 #define INTC_MODEL_WESTMERE_EX   0x2f

862 #define INTC_MODEL_SANDYBRIDGE    0x2a
863 #define INTC_MODEL_SANDYBRIDGE_XEON 0x2d
864 #define INTC_MODEL_IVYBRIDGE     0x3a
865 #define INTC_MODEL_IVYBRIDGE_XEON 0x3e

867 #define INTC_MODEL_HASWELL        0x3c
868 #define INTC_MODEL_HASWELL_ULT    0x45
869 #define INTC_MODEL_HASWELL_GT3E   0x46
870 #define INTC_MODEL_HASWELL_XEON   0x3f

872 #define INTC_MODEL_BROADWELL      0x3d
873 #define INTC_MODEL_BROADWELL_2    0x47
874 #define INTC_MODEL_BROADWELL_XEON 0x4f
875 #define INTC_MODEL_BROADWELL_XEON_D 0x56

877 #define INCC_MODEL_SKYLAKE_MOBILE 0x4e
878 #define INTC_MODEL_SKYLAKE_XEON   0x55
879 #define INTC_MODEL_SKYLAKE_DESKTOP 0x5e

881 #define INTC_MODEL_KABYLAKE_MOBILE 0x8e
882 #define INTC_MODEL_KABYLAKE_DESKTOP 0x9e

884 /*
885  * Atom Processors
886  */
887 #define INTC_MODEL_SILVERTHORNE    0x1c
888 #define INTC_MODEL_LINCROFT        0x26
889 #define INTC_MODEL_PENWELL         0x27
890 #define INTC_MODEL_CLOVERVIEW      0x35
891 #define INTC_MODEL_CEDARVIEW       0x36
892 #define INTC_MODEL_BAY_TRAIL       0x37
893 #define INTC_MODEL_AVATON          0x4d
894 #define INTC_MODEL_AIRMONT         0x4c
895 #define INTC_MODEL_GOLDMONT        0x5c
896 #define INTC_MODEL_DENVERTON       0x5f
897 #define INTC_MODEL_GEMINI_LAKE     0x7a

899 /*
900  * xgetbv/xsetbv support
901  * See section 13.3 in vol. 1 of the Intel developers manual.
902  */

904 #define XFEATURE_ENABLED_MASK     0x0
905 /*
906  * XFEATURE_ENABLED_MASK values (eax)
907  * See setup_xfem().
908  */
909 #define XFEATURE_LEGACY_FP        0x1
910 #define XFEATURE_SSE              0x2
911 #define XFEATURE_AVX              0x4
912 #define XFEATURE_MPX              0x18 /* 2 bits, both 0 or 1 */
913 #define XFEATURE_AVX512           0xe0 /* 3 bits, all 0 or 1 */
914 /* bit 8 unused */
915 #define XFEATURE_PKRU              0x200
916 #define XFEATURE_FP_ALL \
917     (XFEATURE_LEGACY_FP | XFEATURE_SSE | XFEATURE_AVX | XFEATURE_MPX | \

```

```
918     XFEATURE_AVX512 | XFEATURE_PKRU)
920 /*
921  * Define the set of xfeature flags that should be considered valid in the xsave
922  * state vector when we initialize an lwp. This is distinct from the full set so
923  * that all of the processor's normal logic and tracking of the xsave state is
924  * usable. This should correspond to the state that's been initialized by the
925  * ABI to hold meaningful values. Adding additional bits here can have serious
926  * performance implications and cause performance degradations when using the
927  * FPU vector (xmm) registers.
928  */
929 #define XFEATURE_FP_INITIAL      (XFEATURE_LEGACY_FP | XFEATURE_SSE)
931 #if !defined(_ASM)
933 #if defined(_KERNEL) || defined(_KMEMUSER)
935 #define NUM_X86_FEATURES      92
936 extern uchar_t x86_featureset[];
938 extern void free_x86_featureset(void *featureset);
939 extern boolean_t is_x86_feature(void *featureset, uint_t feature);
940 extern void add_x86_feature(void *featureset, uint_t feature);
941 extern void remove_x86_feature(void *featureset, uint_t feature);
942 extern boolean_t compare_x86_featureset(void *setA, void *setB);
943 extern void print_x86_featureset(void *featureset);
946 extern uint_t x86_type;
947 extern uint_t x86_vendor;
948 extern uint_t x86_clflush_size;
950 extern uint_t pentiumpro_bug4046376;
952 extern const char CyrixInstead[];
954 extern void (*spec_lld_flush)(void);
956 #endif
958 #if defined(_KERNEL)
960 /*
961  * This structure is used to pass arguments and get return values back
962  * from the CPUID instruction in __cpuid_insn() routine.
963  */
964 struct cpuid_regs {
965     uint32_t      cp_eax;
966     uint32_t      cp_ebx;
967     uint32_t      cp_ecx;
968     uint32_t      cp_edx;
969 };
unchanged_portion_omitted
```

new/usr/src/uts/intel/zfs/Makefile

1

```

*****
3551 Tue May 7 04:50:02 2019
new/usr/src/uts/intel/zfs/Makefile
OS-7125 Need mitigation of LITF (CVE-2018-3646)
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
*****
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 2009 Sun Microsystems, Inc. All rights reserved.
23 # Use is subject to license terms.
24 #
25 # This makefile drives the production of the zfs file system
26 # kernel module.
27 #
28 # Copyright 2013 Saso Kiselkov. All rights reserved.
29 #
30 # Copyright (c) 2016 by Delphix. All rights reserved.
31 #
32 # Copyright 2018 Joyent, Inc.
32 # Copyright (c) 2018, Joyent, Inc.

34 #
35 # Path to the base of the uts directory tree (usually /usr/src/uts).
36 #
37 UTSBASE = ../../

39 ARCHDIR:sh = cd ../; basename `pwd`

41 #
42 # Define the module and object file sets.
43 #
44 MODULE = zfs
45 OBJECTS = $(ZFS_OBJS:%=$(OBJSDIR)/%) $(LUA_OBJS:%=$(OBJSDIR)/%)
46 # LUA_OBJS are intentionally omitted from LINTS
47 LINTS = $(ZFS_OBJS:%.o=$(LINTSDIR)/%.ln)
48 ROOTMODULE = $(ROOT_DRV_DIR)/$(MODULE)
49 ROOTLINK = $(ROOT_FS_DIR)/$(MODULE)
50 CONF_SRCDIR = $(UTSBASE)/common/fs/zfs

52 #
53 # Include common rules.
54 #
55 include ../Makefile.$(ARCHDIR)

57 #
58 # Define targets

```

new/usr/src/uts/intel/zfs/Makefile

2

```

59 #
60 ALL_TARGET = $(BINARY) $(SRC_CONFILE)
61 LINT_TARGET = $(MODULE).lint
62 INSTALL_TARGET = $(BINARY) $(ROOTMODULE) $(ROOTLINK) $(ROOT_CONFFILE)

64 #
65 # Overrides and depends_on
66 #
67 MODSTUBS_DIR = $(OBJSDIR)
68 LDFLAGS += -dy -Nfs/specfs -Ncrypto/swrand -Nmisc/idmap -Nmisc/sha2 \
69 -Nmisc/skein -Nmisc/edonr

71 INC_PATH += -I$(UTSBASE)/common/fs/zfs
72 INC_PATH += -I$(UTSBASE)/common/fs/zfs/lua
73 INC_PATH += -I$(SRC)/common
74 INC_PATH += -I$(COMMONBASE)/zfs

76 CPPFLAGS += -I$(UTSBASE)/i86pc
77 C99LMODE= -Xc99=%all

79 #
80 # For now, disable these lint checks; maintainers should endeavor
81 # to investigate and remove these for maximum lint coverage.
82 # Please do not carry these forward to new Makefiles.
83 #
84 LINTTAGS += -erroff=E_SUSPICIOUS_COMPARISON
85 LINTTAGS += -erroff=E_BAD_PTR_CAST_ALIGN
86 LINTTAGS += -erroff=E_SUPPRESSION_DIRECTIVE_UNUSED
87 LINTTAGS += -erroff=E_STATIC_UNUSED
88 LINTTAGS += -erroff=E_PTRDIFF_OVERFLOW
89 LINTTAGS += -erroff=E_ASSIGN_NARROW_CONV

91 CERRWARN += _gcc=-Wno-type-limits
92 CERRWARN += _gcc=-Wno-switch
93 CERRWARN += _gcc=-Wno-parentheses
94 CERRWARN += _gcc=-Wno-unused-variable
95 CERRWARN += _gcc=-Wno-unused-function
96 CERRWARN += _gcc=-Wno-unused-label

98 # needs work
99 SMOFF += all_func_returns,indenting
100 $(OBJSDIR)/llex.o := SMOFF += index_overflow
101 $(OBJSDIR)/metaslab.o := SMOFF += no_if_block
102 $(OBJSDIR)/zfs_vnops.o := SMOFF += signed
103 # needs work
104 $(OBJSDIR)/zvol.o := SMOFF += deref_check,signed

106 # false positives
107 $(OBJSDIR)/zfs_ctldir.o := SMOFF += strcpy_overflow
108 $(OBJSDIR)/zfs_ioctl.o := SMOFF += strcpy_overflow

110 #
111 # Default build targets.
112 #
113 .KEEP_STATE:

115 def: $(DEF_DEPS)

117 all: $(ALL_DEPS)

119 clean: $(CLEAN_DEPS)

121 clobber: $(CLOBBER_DEPS)

123 lint: $(LINT_DEPS)

```

new/usr/src/uts/intel/zfs/Makefile

3

```
125 modlintlib:    $(MODLINTLIB_DEPS)
127 clean.lint:   $(CLEAN_LINT_DEPS)
129 install:      $(INSTALL_DEPS)

131 $(ROOTLINK):  $(ROOT_FS_DIR) $(ROOTMODULE)
132             -$(RM) $@; ln $(ROOTMODULE) $@

134 #
135 #             Include common targets.
136 #
137 include ../Makefile.targ
```

```

*****
3303 Tue May 7 04:50:02 2019
new/usr/src/uts/sparc/zfs/Makefile
OS-7125 Need mitigation of LITF (CVE-2018-3646)
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
*****
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 2009 Sun Microsystems, Inc. All rights reserved.
23 # Use is subject to license terms.
24 #
25 # This makefile drives the production of the zfs file system
26 # kernel module.
27 #
28 # Copyright 2013 Saso Kiselkov. All rights reserved.
29 #
30 # Copyright (c) 2016 by Delphix. All rights reserved.
31 #
32 # Copyright 2018 Joyent, Inc.
33 #

35 #
36 # Path to the base of the uts directory tree (usually /usr/src/uts).
37 #
38 UTSBASE = ../../

40 ARCHDIR:sh = cd ..; basename `pwd`

42 #
43 # Define the module and object file sets.
44 #
45 MODULE = zfs
46 OBJECTS = ${ZFS_OBJS:=${OBJSDIR}/%} ${LUA_OBJS:=${OBJSDIR}/%}
47 # LUA_OBJS are intentionally omitted from LINTS
48 LINTS = ${ZFS_OBJS:%.o=${LINTSDIR}/%.ln}
49 ROOTMODULE = ${ROOTDRVDIR}/${MODULE}
50 ROOTLINK = ${ROOTFSDIR}/${MODULE}
51 CONF_SRCDIR = ${UTSBASE}/common/fs/zfs

53 #
54 # Include common rules.
55 #
56 include ../Makefile.${ARCHDIR}

58 #
59 # Define targets

```

```

60 #
61 ALL_TARGET = ${BINARY} ${SRC_CONFILE}
62 LINT_TARGET = ${MODULE}.lint
63 INSTALL_TARGET = ${BINARY} ${ROOTMODULE} ${ROOTLINK} ${ROOT_CONFFILE}

65 #
66 # Overrides and depends_on
67 #
68 # We require sched/SDC because by the time vfs_mountroot() runs,
69 # we can no longer load modules through OBP.
70 #
71 MODSTUBS_DIR = ${OBJSDIR}
72 LDPLAGS += -dy -Nfs/specfs -Ncrypto/swrand -Nmisc/idmap \
73 -Nsched/SDC -Nmisc/sha2 -Nmisc/skein -Nmisc/edonr

75 INC_PATH += -I${UTSBASE}/common/fs/zfs
76 INC_PATH += -I${UTSBASE}/common/fs/zfs/lua
77 INC_PATH += -I${SRC}/common
78 INC_PATH += -I${COMMONBASE}/zfs
79 INC_PATH += -I${UTSBASE}/sun4

81 C99LMODE= -Xc99=%all

83 #
84 # For now, disable these lint checks; maintainers should endeavor
85 # to investigate and remove these for maximum lint coverage.
86 # Please do not carry these forward to new Makefiles.
87 #
88 LINTTAGS += -erroff=E_SUSPICIOUS_COMPARISON
89 LINTTAGS += -erroff=E_BAD_PTR_CAST_ALIGN
90 LINTTAGS += -erroff=E_SUPPRESSION_DIRECTIVE_UNUSED
91 LINTTAGS += -erroff=E_STATIC_UNUSED
92 LINTTAGS += -erroff=E_PTRDIFF_OVERFLOW
93 LINTTAGS += -erroff=E_ASSIGN_NARROW_CONV

95 CERRWARN += -_gcc=-Wno-type-limits
96 CERRWARN += -_gcc=-Wno-switch
97 CERRWARN += -_gcc=-Wno-parentheses
98 CERRWARN += -_gcc=-Wno-unused-variable
99 CERRWARN += -_gcc=-Wno-unused-function
100 CERRWARN += -_gcc=-Wno-unused-label

102 #
103 # Default build targets.
104 #
105 .KEEP_STATE:

107 def: ${DEF_DEPS}

109 all: ${ALL_DEPS}

111 clean: ${CLEAN_DEPS}

113 clobber: ${CLOBBER_DEPS}

115 lint: ${LINT_DEPS}

117 modlintlib: ${MODLINTLIB_DEPS}

119 clean.lint: ${CLEAN_LINT_DEPS}

121 install: ${INSTALL_DEPS}

123 ${ROOTLINK}: ${ROOTFSDIR} ${ROOTMODULE}
124 -$(RM) $@; ln ${ROOTMODULE} $@

```

new/usr/src/uts/sparc/zfs/Makefile

3

```
126 #  
127 #     Include common targets.  
128 #  
129 include ../Makefile.targ
```


new/usr/src/uts/sun4/sys/ht.h

1

824 Tue May 7 04:50:02 2019

new/usr/src/uts/sun4/sys/ht.h

OS-7125 Need mitigation of LITF (CVE-2018-3646)

Reviewed by: Robert Mustacchi <rm@joyent.com>

Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>

```
1 /*
2  * This file and its contents are supplied under the terms of the
3  * Common Development and Distribution License ("CDDL"), version 1.0.
4  * You may only use this file in accordance with the terms of version
5  * 1.0 of the CDDL.
6  *
7  * A full copy of the text of the CDDL should have accompanied this
8  * source. A copy of the CDDL is also available via the Internet at
9  * http://www.illumos.org/license/CDDL.
10 */
```

```
12 /*
13  * Copyright 2018 Joyent, Inc.
14 */
```

```
16 #ifndef _SYS_HT_H
17 #define _SYS_HT_H
```

```
19 #include <sys/types.h>
20 #include <sys/thread.h>
```

```
22 #ifdef __cplusplus
23 extern "C" {
24 #endif
```

```
26 #define ht_init() {}
```

```
28 #define ht_should_run(t, c) (B_TRUE)
29 #define ht_adjust_cpu_score(t, c, p) (p)
30 #define ht_begin_unsafe(void) {}
31 #define ht_end_unsafe(void) {}
32 #define ht_end_intr(void) {}
```

```
34 #ifdef __cplusplus
35 }
36 #endif
```

```
38 #endif /* _SYS_HT_H */
```

```

*****
3735 Tue May 7 04:50:02 2019
new/usr/src/uts/sun4u/sys/Makefile
OS-7125 Need mitigation of LITF (CVE-2018-3646)
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
*****
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 # Copyright 2009 Sun Microsystems, Inc. All rights reserved.
22 # Use is subject to license terms.
23 # Copyright 2019 Peter Tribble.
24 #
25 # Copyright 2018 Joyent, Inc.
25 # uts/sun4u/sys/Makefile
26 #
27 UTSBATCH = ../..

29 #
30 # include global definitions
31 #
32 include ../Makefile.sun4u

34 #
35 # Override defaults.
36 #
37 FILEMODE = 644

39 SUN4_HDRS= \
40     async.h \
41     clock.h \
42     cmp.h \
43     cpc_ultra.h \
44     cpu_sgnblk_defs.h \
45     ddi_subrdefs.h \
46     dvma.h \
47     eeprom.h \
48     errclassify.h \
49     fcode.h \
50     fc_plat.h \
51     ht.h \
52     idprom.h \
53     intr.h \
54     intreg.h \
55     ivintr.h \
56     memlist_plat.h \
57     memnode.h \
58     nexusdebug.h \

```

```

59     prom_debug.h \
60     scb.h \
61     sun4asi.h \
62     tod.h \
63     trapstat.h \
64     vis.h \
65     vm_machparam.h \
66     x_call.h \
67     xc_impl.h \
68     zsmach.h

70 HDRS= \
71     cheetahregs.h \
72     cpr_impl.h \
73     cpu_impl.h \
74     ecc_kstat.h \
75     envctrl.h \
76     envctrl_gen.h \
77     envctrl_ue250.h \
78     envctrl_ue450.h \
79     gpio_87317.h \
80     iocache.h \
81     iommu.h \
82     machasi.h \
83     machclock.h \
84     machcpuvar.h \
85     machparam.h \
86     machsystem.h \
87     machthread.h \
88     mem_cache.h \
89     mmu.h \
90     opl_module.h \
91     prom_plat.h \
92     pte.h \
93     sbd_ioctl.h \
94     spitregs.h \
95     sysioerr.h \
96     sysiosbus.h \
97     todmostek.h \
98     traptrace.h

100 I2CHDRS = clients/max1617.h misc/i2c_svc.h clients/i2c_client.h \
101           clients/hpc3130.h clients/lm75.h \
102           clients/pcf8591.h clients/ssc050.h

104 I2C_DIRS= clients misc
105 USR_PSM_ISYS_I2C_ROOT= $(USR_PSM_ISYS_DIR)/i2c
106 USR_PSM_ISYS_I2C_DIRS= $(USR_PSM_ISYS_I2C_ROOT) \
107                        $(I2C_DIRS:%=$(USR_PSM_ISYS_I2C_ROOT)/%)

109 ROOTI2CHDRS= $(I2CHDRS:%=$(USR_PSM_ISYS_I2C_ROOT)/%)

111 MONHDRS=
112 #MONHDRS= eeprom.h idprom.h keyboard.h password.h

114 USR_PSM_MON_DIR= $(USR_PSM_ISYS_DIR)/mon

116 ROOTHDRS= $(HDRS:%=$(USR_PSM_ISYS_DIR)/%)

118 SUN4_ROOTHDRS= $(SUN4_HDRS:%=$(USR_PSM_ISYS_DIR)/%)

120 ROOTMONHDRS= $(MONHDRS:%=$(USR_PSM_MON_DIR)/%)

122 ROOTDIR= $(ROOT)/usr/share/src
123 ROOTDIRS= $(ROOTDIR)/uts $(ROOTDIR)/uts/$(PLATFORM)

```

new/usr/src/uts/sun4u/sys/Makefile

3

```
125 ROOTLINK=          $(ROOTDIR)/uts/$(PLATFORM)/sys
126 LINKDEST=          ../../../../platform/$(PLATFORM)/include/sys

128 CHECKHDRS=         $(HDRS:%.h=%.check) \
129                   $(MONHDRS:%.h=mon/%.check) \
130                   $(SUN4_HDRS:%.h=%.cmncheck)

132 .KEEP_STATE:

134 .PARALLEL: $(CHECKHDRS) $(ROOTHDRS) $(ROOTMONHDRS) $(SUN4_ROOTHDRS)

136 install_h: $(ROOTDIRS) $(USR_PSM_ISYS_I2C_DIRS) .WAIT \
137             $(ROOTHDRS) $(ROOTI2CHDRS) \
138             $(ROOTMONHDRS) \
139             $(SUN4_ROOTHDRS) $(ROOTLINK)

141 check: $(CHECKHDRS)

143 #
144 # install rules
145 #
146 $(USR_PSM_MON_DIR): $(USR_PSM_ISYS_DIR)
147     $(INS.dir)

149 $(USR_PSM_ISYS_I2C_DIRS):
150     $(INS.dir)

152 $(USR_PSM_ISYS_DIR)/%: ../../sfmmu/sys/% $(USR_PSM_ISYS_DIR)
153     $(INS.file)

155 $(USR_PSM_ISYS_DIR)/%: ../../sun4/sys/% $(USR_PSM_ISYS_DIR)
156     $(INS.file)

158 $(USR_PSM_MON_DIR)/%: mon/% $(USR_PSM_MON_DIR)
159     $(INS.file)

161 $(ROOTDIRS):
162     $(INS.dir)

164 # -r because this used to be a directory and is now a link.
165 $(ROOTLINK): $(ROOTDIRS)
166     -$(RM) -r $@; $(SYMLINK) $(LINKDEST) $@

168 mon/%.check: mon/%.h
169     $(DOT_H_CHECK)

171 %.check: ../../sfmmu/sys/%.h
172     $(DOT_H_CHECK)
173 %.cmncheck: ../../sun4/sys/%.h
174     $(DOT_H_CHECK)

176 FRC:

178 include ../../Makefile.targ
```

```

*****
2813 Tue May 7 04:50:03 2019
new/usr/src/uts/sun4v/sys/Makefile
OS-7125 Need mitigation of L1TF (CVE-2018-3646)
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
*****
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 2009 Sun Microsystems, Inc. All rights reserved.
23 # Use is subject to license terms.
24 #
25 # Copyright 2018 Joyent, Inc.
26 #
27 # uts/sun4v/sys/Makefile
28 # include global definitions
29 UTSBASE = ../..
30 #
31 # include global definitions
32 #
33 include ../Makefile.sun4v
34 #
35 #
36 # Override defaults.
37 #
38 FILEMODE = 644
39 #
40 SUN4_HDRS= \
41     clock.h \
42     cmp.h \
43     cpc_ultra.h \
44     cpu_sgnblk_defs.h \
45     ddi_subrdefs.h \
46     dvma.h \
47     eeprom.h \
48     fcode.h \
49     ht.h \
50     idprom.h \
51     intr.h \
52     intreg.h \
53     ivintr.h \
54     memlist_plat.h \
55     memnode.h \
56     nexusdebug.h \
57     prom_debug.h \

```

```

58     scb.h \
59     sun4asi.h \
60     tod.h \
61     trapstat.h \
62     vis.h \
63     vm_machparam.h \
64     x_call.h \
65     xc_impl.h \
66     zsmach.h \
67 #
68 HDRS= \
69     ds_pri.h \
70     ds_snmp.h \
71     hypervisor_api.h \
72     hsvc.h \
73     machasi.h \
74     machclock.h \
75     machcpuvar.h \
76     mach_descrip.h \
77     machintreg.h \
78     machparam.h \
79     machsystem.h \
80     machthread.h \
81     mmu.h \
82     niagaraasi.h \
83     niagararegs.h \
84     ntwdt.h \
85     pte.h \
86     prom_plat.h \
87     qcn.h \
88     soft_state.h \
89     traptrace.h \
90     vlds.h \
91 #
92 ROOTHDRS= $(HDRS:%=$(USR_PSM_ISYS_DIR)/%)
93 #
94 SUN4_ROOTHDRS= $(SUN4_HDRS:%=$(USR_PSM_ISYS_DIR)/%)
95 #
96 ROOTDIR= $(ROOT)/usr/share/src
97 ROOTDIRS= $(ROOTDIR)/uts $(ROOTDIR)/uts/$(PLATFORM)
98 #
99 ROOTLINK= $(ROOTDIR)/uts/$(PLATFORM)/sys
100 LINKDEST= ../../../../platform/$(PLATFORM)/include/sys
101 #
102 CHECKHDRS= $(HDRS:%.h=%.check) \
103             $(SUN4_HDRS:%.h=%.cmncheck)
104 #
105 .KEEP_STATE:
106 #
107 .PARALLEL: $(CHECKHDRS) $(ROOTHDRS) $(SUN4_ROOTHDRS)
108 #
109 install_h: $(ROOTDIRS) .WAIT \
110             $(ROOTHDRS) .WAIT \
111             $(SUN4_ROOTHDRS) .WAIT $(ROOTLINK)
112 #
113 check: $(CHECKHDRS)
114 #
115 #
116 # install rules
117 #
118 $(USR_PSM_ISYS_DIR)/%: ../../sfmmu/sys/% $(USR_PSM_ISYS_DIR)
119     $(INS.file)
120 #
121 $(USR_PSM_ISYS_DIR)/%: ../../sun4/sys/% $(USR_PSM_ISYS_DIR)
122     $(INS.file)

```

new/usr/src/uts/sun4v/sys/Makefile

3

```
124 $(ROOTDIRS):
125     $(INS.dir)

127 # -r because this used to be a directory and is now a link.
128 $(ROOTLINK):  $(ROOTDIRS)
129     -$(RM) -r $@; $(SYMLINK) $(LINKDEST) $@

131 mon/%.check:  mon/%.h
132     $(DOT_H_CHECK)

134 %.check:      ../../sfmmu/sys/%.h
135     $(DOT_H_CHECK)
136 %.cmncheck:   ../../sun4/sys/%.h
137     $(DOT_H_CHECK)

139 FRC:

141 include ../../Makefile.targ
```