

```

*****
43425 Sat Aug 24 02:37:58 2013
new/usr/src/uts/common/Makefile.files
fsh webrev (August 24th)
*****
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) 1991, 2010, Oracle and/or its affiliates. All rights reserved.
24 # Copyright (c) 2012 Nexenta Systems, Inc. All rights reserved.
25 # Copyright (c) 2012 by Delphix. All rights reserved.
26 # Copyright (c) 2013 by Saso Kiselkov. All rights reserved.
27 #
28 #
29 #
30 # This Makefile defines all file modules for the directory uts/common
31 # and its children. These are the source files which may be considered
32 # common to all SunOS systems.
33 #
34 i386_CORE_OBJS += \
35     atomic.o          \
36     avintr.o          \
37     pic.o
38 #
39 sparc_CORE_OBJS +=
40 #
41 COMMON_CORE_OBJS += \
42     beep.o            \
43     bitset.o          \
44     bp_map.o          \
45     brand.o           \
46     cpucaps.o         \
47     cmt.o             \
48     cmt_policy.o     \
49     cpu.o             \
50     cpu_event.o       \
51     cpu_intr.o        \
52     cpu_pm.o          \
53     cpupart.o         \
54     cap_util.o        \
55     disp.o            \
56     group.o           \
57     kstat_fr.o        \
58     iscsiboot_prop.o \
59     lgrp.o             \
60     lgrp_topo.o       \
61     mmapobj.o

```

```

62     mutex.o           \
63     page_lock.o       \
64     page_retire.o     \
65     panic.o           \
66     param.o           \
67     pg.o              \
68     pghw.o            \
69     putnext.o         \
70     rctl_proc.o       \
71     rwlock.o          \
72     seg_kmem.o        \
73     softint.o         \
74     string.o          \
75     strtol.o          \
76     strtoul.o         \
77     strtoll.o         \
78     strtoull.o        \
79     thread_intr.o     \
80     vm_page.o         \
81     vm_pagelist.o     \
82     zlib_obj.o        \
83     clock_tick.o
84 #
85 CORE_OBJS += $(COMMON_CORE_OBJS) $(MACH)_CORE_OBJS
86 #
87 ZLIB_OBJS = zutil.o zmod.o zmod_subr.o \
88     adler32.o crc32.o deflate.o inffast.o \
89     inflate.o inftrees.o trees.o
90 #
91 GENUNIX_OBJS += \
92     access.o          \
93     acl.o             \
94     acl_common.o      \
95     adjtime.o         \
96     alarm.o           \
97     aio_subr.o        \
98     auditsys.o        \
99     audit_core.o      \
100    audit_zone.o       \
101    audit_memory.o     \
102    autoconf.o         \
103    avl.o              \
104    bdev_dsort.o       \
105    bio.o              \
106    bitmap.o           \
107    blabel.o           \
108    brandsys.o         \
109    bz2blocksort.o    \
110    bz2compress.o     \
111    bz2decompress.o   \
112    bz2randtable.o    \
113    bz2bzip.o         \
114    bz2crctable.o     \
115    bz2huffman.o      \
116    callb.o           \
117    callout.o         \
118    chdir.o           \
119    chmod.o           \
120    chown.o           \
121    cladm.o           \
122    class.o           \
123    clock.o           \
124    clock_highres.o   \
125    clock_realtime.o \
126    close.o           \
127    compress.o

```

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

```

128 condvar.o \
129 conf.o \
130 console.o \
131 contract.o \
132 copyops.o \
133 core.o \
134 corectl.o \
135 cred.o \
136 cs_stubs.o \
137 dacf.o \
138 dacf_clnt.o \
139 damap.o \
140 cyclic.o \
141 ddi.o \
142 ddifm.o \
143 ddi_hp_impl.o \
144 ddi_hp_ndi.o \
145 ddi_intr.o \
146 ddi_intr_impl.o \
147 ddi_intr_irm.o \
148 ddi_nodeid.o \
149 ddi_timer.o \
150 devcfg.o \
151 devcache.o \
152 device.o \
153 devid.o \
154 devid_cache.o \
155 devid_scsi.o \
156 devid_smp.o \
157 devpolicy.o \
158 disp_lock.o \
159 dnlc.o \
160 driver.o \
161 dumpsubr.o \
162 driver_lyr.o \
163 dtrace_subr.o \
164 errorq.o \
165 etheraddr.o \
166 evchannels.o \
167 exacct.o \
168 exacct_core.o \
169 exec.o \
170 exit.o \
171 fbio.o \
172 fcntl.o \
173 fdbuffer.o \
174 fdsync.o \
175 fem.o \
176 ffs.o \
177 fio.o \
178 flock.o \
179 fm.o \
180 fork.o \
181 fsh.o \
182 vpm.o \
183 fs_reparse.o \
184 fs_subr.o \
185 fsflush.o \
186 ftrace.o \
187 getcwd.o \
188 getdents.o \
189 getloadavg.o \
190 getpagesizes.o \
191 getpid.o \
192 gfs.o \
193 rusagesys.o \

```

3

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

```

194 gid.o \
195 groups.o \
196 grow.o \
197 hat_refmod.o \
198 id32.o \
199 id_space.o \
200 inet_ntop.o \
201 instance.o \
202 ioctl.o \
203 ip_cksum.o \
204 issetugid.o \
205 ippconf.o \
206 kpcp.o \
207 kdi.o \
208 kiconv.o \
209 klpd.o \
210 kmem.o \
211 ksyms_snapshot.o \
212 l_strplumb.o \
213 labelsys.o \
214 link.o \
215 list.o \
216 lockstat_subr.o \
217 log_sysevent.o \
218 logsubr.o \
219 lookup.o \
220 lseek.o \
221 ltos.o \
222 lwp.o \
223 lwp_create.o \
224 lwp_info.o \
225 lwp_self.o \
226 lwp_obj.o \
227 lwp_timer.o \
228 lwpsys.o \
229 main.o \
230 mmapobjsys.o \
231 memcntl.o \
232 memstr.o \
233 lgrpsys.o \
234 mkdir.o \
235 mknod.o \
236 mount.o \
237 move.o \
238 msacct.o \
239 multidata.o \
240 nbmlock.o \
241 ndifm.o \
242 nice.o \
243 netstack.o \
244 ntptime.o \
245 nvpair.o \
246 nvpair_alloc_system.o \
247 nvpair_alloc_fixed.o \
248 fnvpair.o \
249 octet.o \
250 open.o \
251 p_online.o \
252 pathconf.o \
253 pathname.o \
254 pause.o \
255 serializer.o \
256 pci_intr_lib.o \
257 pci_cap.o \
258 pcifm.o \
259 pgrp.o \

```

4

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

```

260      pgrpsys.o  \
261      pid.o      \
262      pkp_hash.o \
263      policy.o   \
264      poll.o     \
265      pool.o     \
266      pool_pset.o \
267      port_subr.o \
268      ppriv.o    \
269      printf.o   \
270      prionctl.o \
271      priv.o     \
272      priv_const.o \
273      proc.o     \
274      procset.o  \
275      processor_bind.o \
276      processor_info.o \
277      profil.o   \
278      project.o  \
279      qsort.o    \
280      rctl.o     \
281      rctlsys.o  \
282      readlink.o \
283      refstr.o   \
284      rename.o   \
285      resolvepath.o \
286      retire_store.o \
287      process.o  \
288      rlimit.o   \
289      rmap.o     \
290      rw.o       \
291      rwstlock.o \
292      sad_conf.o \
293      sid.o      \
294      sidsys.o   \
295      sched.o    \
296      schedctl.o \
297      sctp_crc32.o \
298      seg_dev.o  \
299      seg_kp.o   \
300      seg_kpm.o  \
301      seg_map.o  \
302      seg_vn.o   \
303      seg_spt.o  \
304      semaphore.o \
305      sendfile.o \
306      session.o  \
307      share.o    \
308      shuttle.o  \
309      sig.o      \
310      sigaction.o \
311      sigaltstack.o \
312      signotify.o \
313      sigpending.o \
314      sigprocmask.o \
315      sigqueue.o \
316      sigsendset.o \
317      sigsuspend.o \
318      sigtimedwait.o \
319      sleepq.o   \
320      sock_conf.o \
321      space.o    \
322      sscanf.o   \
323      stat.o     \
324      statfs.o   \
325      statvfs.o  \

```

5

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

```

326      stol.o     \
327      str_conf.o \
328      strcalls.o \
329      stream.o   \
330      streamio.o \
331      strext.o   \
332      strsubr.o  \
333      strsun.o   \
334      subr.o     \
335      sunddi.o   \
336      sunmdi.o   \
337      sunndi.o   \
338      sunpci.o   \
339      sunpm.o    \
340      sundlpi.o  \
341      suntpi.o   \
342      swap_subr.o \
343      swap_vnops.o \
344      symlink.o  \
345      sync.o     \
346      sysclass.o \
347      sysconfig.o \
348      sysent.o   \
349      sysfs.o    \
350      systeminfo.o \
351      task.o     \
352      taskq.o    \
353      tasksys.o  \
354      time.o     \
355      timer.o    \
356      times.o    \
357      timers.o   \
358      thread.o   \
359      tlabel.o   \
360      tnf_res.o  \
361      turnstile.o \
362      tty_common.o \
363      u8_textprep.o \
364      uadmin.o   \
365      uconv.o    \
366      ucredsys.o \
367      uid.o      \
368      umask.o    \
369      umount.o   \
370      uname.o    \
371      unix_bb.o  \
372      unlink.o   \
373      urw.o      \
374      utime.o    \
375      utssys.o   \
376      uucopy.o   \
377      vfs.o      \
378      vfs_conf.o \
379      vmem.o     \
380      vm_anon.o  \
381      vm_as.o    \
382      vm_meter.o \
383      vm_pageout.o \
384      vm_pvn.o   \
385      vm_rm.o    \
386      vm_seg.o   \
387      vm_subr.o  \
388      vm_swap.o  \
389      vm_usage.o \
390      vnode.o    \
391      vuid_queue.o \

```

6

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

7

```

392          vuid_store.o  \
393          waitq.o       \
394          watchpoint.o  \
395          yield.o       \
396          scsi_confdata.o \
397          xattr.o       \
398          xattr_common.o \
399          xdr_mblk.o     \
400          xdr_mem.o     \
401          xdr.o         \
402          xdr_array.o   \
403          xdr_refer.o   \
404          xhat.o        \
405          zone.o

407 #
408 #     Stubs for the stand-alone linker/loader
409 #
410 sparc_GENSTUBS_OBJS = \
411     kobj_stubs.o

413 i386_GENSTUBS_OBJS =

415 COMMON_GENSTUBS_OBJS =

417 GENSTUBS_OBJS += $(COMMON_GENSTUBS_OBJS) $( $(MACH)_GENSTUBS_OBJS)

419 #
420 #     DTrace and DTrace Providers
421 #
422 DTRACE_OBJS += dtrace.o dtrace_isa.o dtrace_asm.o

424 SDT_OBJS += sdt_subr.o

426 PROFILE_OBJS += profile.o

428 SYSTRACE_OBJS += systrace.o

430 LOCKSTAT_OBJS += lockstat.o

432 FASTTRAP_OBJS += fasttrap.o fasttrap_isa.o

434 DCPC_OBJS += dcpc.o

436 #
437 #     Driver (pseudo-driver) Modules
438 #
439 IPP_OBJS += ippctl.o

441 AUDIO_OBJS += audio_client.o audio_ddi.o audio_engine.o \
442     audio_fltdata.o audio_format.o audio_ctrl.o \
443     audio_grc3.o audio_output.o audio_input.o \
444     audio_oss.o audio_sun.o

446 AUDIOEMU10K_OBJS += audioemu10k.o

448 AUDIOENS_OBJS += audioens.o

450 AUDIOVIA823X_OBJS += audiovia823x.o

452 AUDIOVIA97_OBJS += audiovia97.o

454 AUDIO1575_OBJS += audio1575.o

456 AUDIO810_OBJS += audio810.o

```

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

8

```

458 AUDIOCMI_OBJS += audiocmi.o

460 AUDIOCMIHD_OBJS += audiocmihd.o

462 AUDIOHD_OBJS += audiohd.o

464 AUDIOIXP_OBJS += audioixp.o

466 AUDIOLS_OBJS += audiols.o

468 AUDIOP16X_OBJS += audiop16x.o

470 AUDIOPCI_OBJS += audiopci.o

472 AUDIOSOLO_OBJS += audiosolo.o

474 AUDIOTS_OBJS += audiots.o

476 AC97_OBJS += ac97.o ac97_ad.o ac97_alc.o ac97_cmi.o

478 BLKDEV_OBJS += blkdev.o

480 CARDBUS_OBJS += cardbus.o cardbus_hp.o cardbus_cfg.o

482 CONSKBD_OBJS += conskbd.o

484 CONSMS_OBJS += consms.o

486 OLDPTY_OBJS += tty_ptyconf.o

488 PTC_OBJS += tty_pty.o

490 PTSL_OBJS += tty_pts.o

492 PTM_OBJS += ptm.o

494 MII_OBJS += mii.o mii_cicada.o mii_natsemi.o mii_intel.o mii_qualsemi.o \
495     mii_marvell.o mii_realtek.o mii_other.o

497 PTS_OBJS += pts.o

499 PTY_OBJS += ptms_conf.o

501 SAD_OBJS += sad.o

503 MD4_OBJS += md4.o md4_mod.o

505 MD5_OBJS += md5.o md5_mod.o

507 SHA1_OBJS += sha1.o sha1_mod.o

509 SHA2_OBJS += sha2.o sha2_mod.o

511 IPGPC_OBJS += classifierddi.o classifier.o filters.o trie.o table.o \
512     ba_table.o

514 DSCPMK_OBJS += dscpmk.o dscpmkddi.o

516 DLCOSMK_OBJS += dlcosmk.o dlcosmkddi.o

518 FLOWACCT_OBJS += flowacctddi.o flowacct.o

520 TOKENMT_OBJS += tokenmt.o tokenmtddi.o

522 TSWTCL_OBJS += tswtcl.o tswtclddi.o

```

```

524 ARP_OBJS += arpddi.o
526 ICMP_OBJS += icmpddi.o
528 ICMP6_OBJS += icmp6ddi.o
530 RTS_OBJS += rtsddi.o

532 IP_ICMP_OBJS = icmp.o icmp_opt_data.o
533 IP_RTS_OBJS = rts.o rts_opt_data.o
534 IP_TCP_OBJS = tcp.o tcp_fusion.o tcp_opt_data.o tcp_sack.o tcp_stats.o \
535 tcp_misc.o tcp_timers.o tcp_time_wait.o tcp_tpi.o tcp_output.o \
536 tcp_input.o tcp_socket.o tcp_bind.o tcp_cluster.o tcp_tunables.o
537 IP_UDP_OBJS = udp.o udp_opt_data.o udp_tunables.o udp_stats.o
538 IP_SCTP_OBJS = sctp.o sctp_opt_data.o sctp_output.o \
539 sctp_init.o sctp_input.o sctp_cookie.o \
540 sctp_conn.o sctp_error.o sctp_snmp.o \
541 sctp_tunables.o sctp_shutdown.o sctp_common.o \
542 sctp_timer.o sctp_heartbeat.o sctp_hash.o \
543 sctp_bind.o sctp_notify.o sctp_asconf.o \
544 sctp_addr.o tn_ipopt.o tnet.o ip_netinfo.o \
545 sctp_misc.o
546 IP_ILB_OBJS = ilb.o ilb_nat.o ilb_conn.o ilb_alg_hash.o ilb_alg_rr.o

548 IP_OBJS += igmp.o ipmp.o ip.o ip6.o ip6_asp.o ip6_if.o ip6_ire.o \
549 ip6_rts.o ip_if.o ip_ire.o ip_listutils.o ip_mrout.o \
550 ip_multi.o ip2mac.o ip_ndp.o ip_rts.o ip_srcid.o \
551 ipddi.o ipdrop.o mi.o nd.o tunables.o optcom.o snmpcom.o \
552 ipsec_loader.o spd.o ipclassifier.o inet_common.o ip_queue.o \
553 queue.o ip_sadb.o ip_ftable.o proto_set.o radix.o ip_dummy.o \
554 ip_helper_stream.o ip_tunables.o \
555 ip_output.o ip_input.o ip6_input.o ip6_output.o ip_arp.o \
556 conn_opt.o ip_attr.o ip_dce.o \
557 $(IP_ICMP_OBJS) \
558 $(IP_RTS_OBJS) \
559 $(IP_TCP_OBJS) \
560 $(IP_UDP_OBJS) \
561 $(IP_SCTP_OBJS) \
562 $(IP_ILB_OBJS)

564 IP6_OBJS += ip6ddi.o

566 HOOK_OBJS += hook.o

568 NETI_OBJS += neti_impl.o neti_mod.o neti_stack.o

570 KEYSOCK_OBJS += keysockddi.o keysock.o keysock_opt_data.o

572 IPNET_OBJS += ipnet.o ipnet_bpf.o

574 SPDSOCK_OBJS += spdsockddi.o spdsock.o spdsock_opt_data.o

576 IPSECESP_OBJS += ipsecespddi.o ipsecesp.o

578 IPSECAH_OBJS += ipsecahddi.o ipsecah.o sadb.o

580 SPPP_OBJS += sPPP.o sPPP_dlpi.o sPPP_mod.o s_common.o

582 SPPTUN_OBJS += spptun.o spptun_mod.o

584 SPASPASYN_OBJS += spaspasyn.o spaspasyn_mod.o

586 SPSPCOMP_OBJS += spspcomp.o spspcomp_mod.o deflate.o BSD-comp.o vjcompress.o \
587 zlib.o

589 TCP_OBJS += tcpddi.o

```

```

591 TCP6_OBJS += tcp6ddi.o

593 NCA_OBJS += ncaddi.o

595 SDP SOCK_MOD_OBJS += sockmod_sdp.o socksdp.o socksdpsubr.o

597 SCTP SOCK_MOD_OBJS += sockmod_sctp.o socksctp.o socksctpsubr.o

599 PFP SOCK_MOD_OBJS += sockmod_pfp.o

601 RDS SOCK_MOD_OBJS += sockmod_rds.o

603 RDS_OBJS += rdsddi.o rdssubr.o rds_opt.o rds_ioctl.o

605 RDSIB_OBJS += rdsib.o rdsib_ib.o rdsib_cm.o rdsib_ep.o rdsib_buf.o \
606 rdsib_debug.o rdsib_sc.o

608 RDSV3_OBJS += af_rds.o rdsV3_ddi.o bind.o loop.o threads.o connection.o \
609 transport.o cong.o sysctl.o message.o rds_recv.o send.o \
610 stats.o info.o page.o rdma_transport.o ib_ring.o ib_rdma.o \
611 ib_recv.o ib.o ib_send.o ib_sysctl.o ib_stats.o ib_cm.o \
612 rdsV3_sc.o rdsV3_debug.o rdsV3_impl.o rdma.o rdsV3_af_thr.o

614 ISER_OBJS += iser.o iser_cm.o iser_cq.o iser_ib.o iser_idm.o \
615 iser_resource.o iser_xfer.o

617 UDP_OBJS += udpddi.o

619 UDP6_OBJS += udp6ddi.o

621 SY_OBJS += gentyty.o

623 TCO_OBJS += ticots.o

625 TCOO_OBJS += ticotsord.o

627 TCL_OBJS += ticlts.o

629 TL_OBJS += tl.o

631 DUMP_OBJS += dump.o

633 BPF_OBJS += bpf.o bpf_filter.o bpf_mod.o bpf_dlt.o bpf_mac.o

635 CLONE_OBJS += clone.o

637 CN_OBJS += cons.o

639 DLD_OBJS += dld_drv.o dld_proto.o dld_str.o dld_flow.o

641 DLS_OBJS += dls.o dls_link.o dls_mod.o dls_stat.o dls_mgmt.o

643 GLD_OBJS += gld.o gldutil.o

645 MAC_OBJS += mac.o mac_bcast.o mac_client.o mac_datapath_setup.o mac_flow.o
646 mac_hio.o mac_mod.o mac_ndd.o mac_provider.o mac_sched.o \
647 mac_protect.o mac_soft_ring.o mac_stat.o mac_util.o

649 MAC_6TO4_OBJS += mac_6to4.o

651 MAC_ETHER_OBJS += mac_ether.o

653 MAC_IPV4_OBJS += mac_ipv4.o

655 MAC_IPV6_OBJS += mac_ipv6.o

```

```

657 MAC_WIFI_OBJS +=      mac_wifi.o
659 MAC_IB_OBJS +=       mac_ib.o
661 IPTUN_OBJS +=       iptun_dev.o iptun_ctl.o iptun.o
663 AGGR_OBJS +=       aggr_dev.o aggr_ctl.o aggr_grp.o aggr_port.o \
664                   aggr_send.o aggr_recv.o aggr_lacp.o
666 SOFTMAC_OBJS +=     softmac_main.o softmac_ctl.o softmac_capab.o \
667                   softmac_dev.o softmac_stat.o softmac_pkt.o softmac_fp.o
669 NET80211_OBJS +=    net80211.o net80211_proto.o net80211_input.o \
670                   net80211_output.o net80211_node.o net80211_crypto.o \
671                   net80211_crypto_none.o net80211_crypto_wep.o net80211_ioctl.o \
672                   net80211_crypto_tkip.o net80211_crypto_ccmp.o \
673                   net80211_ht.o
675 VNIC_OBJS +=       vnic_ctl.o vnic_dev.o
677 SIMNET_OBJS +=     simnet.o
679 IB_OBJS +=         ibnex.o ibnex_ioctl.o ibnex_hca.o
681 IBCM_OBJS +=       ibcm_impl.o ibcm_sm.o ibcm_ti.o ibcm_utils.o ibcm_path.o \
682                   ibcm_arp.o ibcm_arp_link.o
684 IBDM_OBJS +=       ibdm.o
686 IBDMA_OBJS +=      ibdma.o
688 IBMF_OBJS +=       ibmf.o ibmf_impl.o ibmf_dr.o ibmf_wqe.o ibmf_ud_dest.o ibmf_mod.o
689                   ibmf_send.o ibmf_recv.o ibmf_handlers.o ibmf_trans.o \
690                   ibmf_timers.o ibmf_msg.o ibmf_utils.o ibmf_rmpp.o \
691                   ibmf_saa.o ibmf_saa_impl.o ibmf_saa_utils.o ibmf_saa_events.o
693 IBTL_OBJS +=       ibtl_impl.o ibtl_util.o ibtl_mem.o ibtl_handlers.o ibtl_qp.o \
694                   ibtl_cg.o ibtl_wr.o ibtl_hca.o ibtl_chan.o ibtl_cm.o \
695                   ibtl_mcg.o ibtl_ibnex.o ibtl_srqp.o ibtl_part.o
697 TAVOR_OBJS +=     tavor.o tavor_agents.o tavor_cfg.o tavor_ci.o tavor_cmd.o \
698                   tavor_cq.o tavor_event.o tavor_ioctl.o tavor_misc.o \
699                   tavor_mr.o tavor_qp.o tavor_qpmod.o tavor_rsrc.o \
700                   tavor_srqp.o tavor_stats.o tavor_umap.o tavor_wr.o
702 HERMON_OBJS +=     hermon.o hermon_agents.o hermon_cfg.o hermon_ci.o hermon_cmd.o \
703                   hermon_cq.o hermon_event.o hermon_ioctl.o hermon_misc.o \
704                   hermon_mr.o hermon_qp.o hermon_qpmod.o hermon_rsrc.o \
705                   hermon_srqp.o hermon_stats.o hermon_umap.o hermon_wr.o \
706                   hermon_fcoib.o hermon_fm.o
708 DAPLT_OBJS +=      daplt.o
710 SOL_OFS_OBJS +=    sol_cma.o sol_ib_cma.o sol_uobj.o \
711                   sol_ofs_debug_util.o sol_ofs_gen_util.o \
712                   sol_kverbs.o
714 SOL_UCMA_OBJS +=   sol_ucma.o
716 SOL_UVERBS_OBJS += sol_uverbs.o sol_uverbs_comp.o sol_uverbs_event.o \
717                   sol_uverbs_hca.o sol_uverbs_qp.o
719 SOL_UMAD_OBJS +=   sol_umad.o
721 KSTAT_OBJS +=     kstat.o

```

```

723 KSYMS_OBJS +=      ksyms.o
725 INSTANCE_OBJS +=   inst_sync.o
727 IWSCN_OBJS +=      iwscons.o
729 LOFI_OBJS +=       lofi.o LzmaDec.o
731 FSSNAP_OBJS +=     fssnap.o
733 FSSNAPIF_OBJS +=   fssnap_if.o
735 MM_OBJS +=         mem.o
737 PHYSMEM_OBJS +=   physmem.o
739 OPTIONS_OBJS +=    options.o
741 WINLOCK_OBJS +=    winlockio.o
743 PM_OBJS +=         pm.o
744 SRN_OBJS +=        srn.o
746 PSEUDO_OBJS +=     pseudonex.o
748 RAMDISK_OBJS +=    ramdisk.o
750 LLC1_OBJS +=       llcl.o
752 USBKBM_OBJS +=     usbkbm.o
754 USBWCM_OBJS +=     usbwcm.o
756 BOFI_OBJS +=       bofi.o
758 HID_OBJS +=        hid.o
760 HWA_RC_OBJS +=     hwarc.o
762 USBSKEL_OBJS +=    usbskel.o
764 USBVC_OBJS +=      usbvc.o usbvc_v412.o
766 HIDPARSER_OBJS += hidparser.o
768 USB_AC_OBJS +=     usb_ac.o
770 USB_AS_OBJS +=     usb_as.o
772 USB_AH_OBJS +=     usb_ah.o
774 USBMS_OBJS +=      usbms.o
776 USBPRN_OBJS +=     usbprn.o
778 UGEN_OBJS +=       ugen.o
780 USBSER_OBJS +=     usbser.o usbser_rseq.o
782 USBSACM_OBJS +=    usb sacram.o
784 USBSER_KEYSPAN_OBJS += usbser_keyspan.o keyspan_dsd.o keyspan_pipe.o
786 USBS49_FW_OBJS +=  keyspan_49fw.o

```

```

788 USBSPRL_OBJS += usbser_pl2303.o pl2303_dsd.o
790 WUSB_CA_OBJS += wusb_ca.o
792 USBFTDI_OBJS += usbser_uftdi.o uftdi_dsd.o
794 USBECM_OBJS += usbecm.o
796 WC_OBJS += wscons.o vcons.o
798 VCONS_CONF_OBJS += vcons_conf.o
800 SCSI_OBJS +=      scsi_capabilities.o scsi_confsubr.o scsi_control.o \
801                scsi_data.o scsi_fm.o scsi_hba.o scsi_reset_notify.o \
802                scsi_resource.o scsi_subr.o scsi_transport.o scsi_watch.o \
803                smp_transport.o
805 SCSI_VHCI_OBJS +=      scsi_vhci.o mpapi_impl.o scsi_vhci_tpgs.o
807 SCSI_VHCI_F_SYM_OBJS +=      sym.o
809 SCSI_VHCI_F_TPGS_OBJS +=      tpgs.o
811 SCSI_VHCI_F_ASYM_SUN_OBJS +=  asym_sun.o
813 SCSI_VHCI_F_SYM_HDS_OBJS +=  sym_hds.o
815 SCSI_VHCI_F_TAPE_OBJS +=      tape.o
817 SCSI_VHCI_F_TPGS_TAPE_OBJS += tpgs_tape.o
819 SGEN_OBJS +=      sgen.o
821 SMP_OBJS +=      smp.o
823 SATA_OBJS +=      sata.o
825 USBA_OBJS +=      hcidi.o  usba.o  usbai.o  hubdi.o  parser.o  genconsole.o \
826                usbai_pipe_mgmt.o usbai_req.o usbai_util.o usbai_register.o \
827                usba_devdb.o usba10_calls.o usba_uugen.o whcdi.o wa.o
828 USBA_WITHOUT_WUSB_OBJS +=      hcidi.o  usba.o  usbai.o  hubdi.o  parser.o  gencons
829                usbai_pipe_mgmt.o usbai_req.o usbai_util.o usbai_register.o \
830                usba_devdb.o usba10_calls.o usba_uugen.o
832 USBA10_OBJS +=      usba10.o
834 RSM_OBJS +=      rsm.o  rsmka_pathmanager.o  rsmka_util.o
836 RSMOPS_OBJS +=      rsmops.o
838 S1394_OBJS +=      t1394.o t1394_errmsg.o s1394.o s1394_addr.o s1394_async.o \
839                s1394_bus_reset.o s1394_cmp.o s1394_csr.o s1394_dev_disc.o \
840                s1394_fa.o s1394_fcp.o \
841                s1394_hotplug.o s1394_isoch.o s1394_misc.o h1394.o nx1394.o
843 HCI1394_OBJS +=      hcil1394.o hcil1394_async.o hcil1394_attach.o hcil1394_buf.o \
844                hcil1394_csr.o hcil1394_detach.o hcil1394_extern.o \
845                hcil1394_ioctl.o hcil1394_isoch.o hcil1394_isr.o \
846                hcil1394_ixl_comp.o hcil1394_ixl_isr.o hcil1394_ixl_misc.o \
847                hcil1394_ixl_update.o hcil1394_misc.o hcil1394_ohci.o \
848                hcil1394_q.o hcil1394_s1394if.o hcil1394_tlabel.o \
849                hcil1394_tlist.o hcil1394_vendor.o
851 AV1394_OBJS +=      avl1394.o avl1394_as.o avl1394_async.o avl1394_cfgrom.o \
852                avl1394_cmp.o avl1394_fcp.o avl1394_isoch.o avl1394_isoch_chan.o \
853                avl1394_isoch_recv.o avl1394_isoch_xmit.o avl1394_list.o \

```

```

854                avl1394_queue.o
856 DCAM1394_OBJS +=      dcam.o dcam_frame.o dcam_param.o dcam_reg.o \
857                dcam_ring_buff.o
859 SCSA1394_OBJS +=      hba.o  sbp2_driver.o  sbp2_bus.o
861 SBP2_OBJS +=      cfgrom.o  sbp2.o
863 PMODEM_OBJS +=      pmodem.o pmodem_cis.o cis.o cis_callout.o cis_handlers.o cis_para
865 DSW_OBJS +=      dsw.o dsw_dev.o ii_tree.o
867 NCALL_OBJS +=      ncall.o \
868                ncall_stub.o
870 RDC_OBJS +=      rdc.o \
871                rdc_dev.o \
872                rdc_io.o \
873                rdc_clnt.o \
874                rdc_prot_xdr.o \
875                rdc_svc.o \
876                rdc_bitmap.o \
877                rdc_health.o \
878                rdc_subr.o \
879                rdc_diskq.o
881 RDCSRV_OBJS +=      rdcsrv.o
883 RDCSTUB_OBJS +=      rdc_stub.o
885 SDBC_OBJS +=      sd_bcache.o \
886                sd_bio.o \
887                sd_conf.o \
888                sd_ft.o \
889                sd_hash.o \
890                sd_io.o \
891                sd_misc.o \
892                sd_pcu.o \
893                sd_tdaemon.o \
894                sd_trace.o \
895                sd_iob_impl0.o \
896                sd_iob_impl1.o \
897                sd_iob_impl2.o \
898                sd_iob_impl3.o \
899                sd_iob_impl4.o \
900                sd_iob_impl5.o \
901                sd_iob_impl6.o \
902                sd_iob_impl7.o \
903                safestore.o \
904                safestore_ram.o
906 NSCTL_OBJS +=      nsctl.o \
907                nsc_cache.o \
908                nsc_disk.o \
909                nsc_dev.o \
910                nsc_freeze.o \
911                nsc_gen.o \
912                nsc_mem.o \
913                nsc_ncallio.o \
914                nsc_power.o \
915                nsc_resv.o \
916                nsc_rmspin.o \
917                nsc_solaris.o \
918                nsc_trap.o \
919                nsc_list.o

```

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

15

```

920 UNISTAT_OBJS += spuni.o \
921                spcs_s_k.o

923 NSKERN_OBJS += nsc_ddi.o \
924                nsc_proc.o \
925                nsc_raw.o \
926                nsc_thread.o \
927                nskernd.o

929 SV_OBJS +=      sv.o

931 PMCS_OBJS += pmcs_attach.o pmcs_ds.o pmcs_intr.o pmcs_nvram.o pmcs_sata.o \
932                pmcs_scsa.o pmcs_smhba.o pmcs_subr.o pmcs_fwlog.o

934 PMCS8001FW_C_OBJS += pmcs_fw_hdr.o
935 PMCS8001FW_OBJS +=      $(PMCS8001FW_C_OBJS) SPCBoot.o ila.o firmware.o

937 #
938 #      Build up defines and paths.

940 ST_OBJS +=      st.o      st_conf.o

942 EMLXS_OBJS +=   emlxs_clock.o emlxs_dfc.o emlxs_dhchap.o emlxs_diag.o \
943                 emlxs_download.o emlxs_dump.o emlxs_eis.o emlxs_event.o \
944                 emlxs_fcf.o emlxs_fcp.o emlxs_fct.o emlxs_hba.o emlxs_ip.o \
945                 emlxs_mbox.o emlxs_mem.o emlxs_msg.o emlxs_node.o \
946                 emlxs_pkt.o emlxs_sli3.o emlxs_sli4.o emlxs_solaris.o \
947                 emlxs_thread.o

949 EMLXS_FW_OBJS +=      emlxs_fw.o

951 OCE_OBJS +=     oce_buf.o oce_fm.o oce_gld.o oce_hw.o oce_intr.o oce_main.o \
952                 oce_mbx.o oce_mq.o oce_queue.o oce_rx.o oce_stat.o oce_tx.o \
953                 oce_utils.o

955 FCT_OBJS +=     discovery.o fct.o

957 QLT_OBJS +=     2400.o 2500.o 8100.o qlt.o qlt_dma.o

959 SRPT_OBJS +=     srpt_mod.o srpt_ch.o srpt_cm.o srpt_ioc.o srpt_stp.o

961 FCOE_OBJS +=     fcoe.o fcoe_eth.o fcoe_fc.o

963 FCOET_OBJS +=    fcoet.o fcoet_eth.o fcoet_fc.o

965 FCOEI_OBJS +=    fcoei.o fcoei_eth.o fcoei_lv.o

967 ISCSIT_SHARED_OBJS += \
968                 iscsit_common.o

970 ISCSIT_OBJS +=   $(ISCSIT_SHARED_OBJS) \
971                 iscsit.o iscsit_tgt.o iscsit_sess.o iscsit_login.o \
972                 iscsit_text.o iscsit_isns.o iscsit_radiusauth.o \
973                 iscsit_radiuspacket.o iscsit_auth.o iscsit_authclient.o

975 PPPT_OBJS +=     alua_ic_if.o pppt.o pppt_msg.o pppt_tgt.o

977 STMF_OBJS +=     lun_map.o stmf.o

979 STMF_SBD_OBJS += sbd.o sbd_scsi.o sbd_pgr.o sbd_zvol.o

981 SYMSG_OBJS +=     sysmsg.o

983 SES_OBJS +=       ses.o ses_sen.o ses_safte.o ses_ses.o

985 TNF_OBJS +=       tnf_buf.o      tnf_trace.o      tnf_writer.o      trace_init.o \

```

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

16

```

986                trace_funcs.o tnf_probe.o      tnf.o

988 LOGINDMUX_OBJS += logindmux.o

990 DEVINFO_OBJS += devinfo.o

992 DEVPOLL_OBJS += devpoll.o

994 DEVPOOL_OBJS += devpool.o

996 I8042_OBJS +=    i8042.o

998 KB8042_OBJS +=   \
999                 at_keyprocess.o \
1000                kb8042.o \
1001                kb8042_keytables.o

1003 MOUSE8042_OBJS += mouse8042.o

1005 FDC_OBJS +=      fdc.o

1007 ASY_OBJS +=      asy.o

1009 ECPP_OBJS +=      ecpp.o

1011 VUIDM3P_OBJS +=    vuidmice.o vuidm3p.o

1013 VUIDM4P_OBJS +=    vuidmice.o vuidm4p.o

1015 VUIDM5P_OBJS +=    vuidmice.o vuidm5p.o

1017 VUIDPS2_OBJS +=   vuidmice.o vuidps2.o

1019 HPCSVCS_OBJS +=   hpcsvc.o

1021 PCIE_MISC_OBJS += pcie.o pcie_fault.o pcie_hp.o pciehp.o pcishpc.o pcie_pwr.o p

1023 PCIHPNEXUS_OBJS += pcihp.o

1025 OPENEEPR_OBJS += openprom.o

1027 RANDOM_OBJS +=   random.o

1029 PSHOT_OBJS +=     pshot.o

1031 GEN_DRV_OBJS +=    gen_drv.o

1033 TCLIENT_OBJS +=    tclient.o

1035 TPHCI_OBJS +=      tphci.o

1037 TVHCI_OBJS +=      tvhci.o

1039 EMUL64_OBJS +=     emul64.o emul64_bsd.o

1041 FCP_OBJS +=         fcp.o

1043 FCIP_OBJS +=        fcip.o

1045 FCSM_OBJS +=        fcsm.o

1047 FCTL_OBJS +=        fctl.o

1049 FP_OBJS +=          fp.o

1051 QLC_OBJS +=         ql_api.o ql_debug.o ql_hba_fru.o ql_init.o ql_iocb.o ql_ioctl.o \

```


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

17

```

1052      ql_isr.o ql_mbx.o ql_nx.o ql_xioctl.o ql_fw_table.o
1054 QLC_FW_2200_OBJS += ql_fw_2200.o
1056 QLC_FW_2300_OBJS += ql_fw_2300.o
1058 QLC_FW_2400_OBJS += ql_fw_2400.o
1060 QLC_FW_2500_OBJS += ql_fw_2500.o
1062 QLC_FW_6322_OBJS += ql_fw_6322.o
1064 QLC_FW_8100_OBJS += ql_fw_8100.o
1066 QLGE_OBJS += qlge.o qlge_dbg.o qlge_flash.o qlge_fm.o qlge_gld.o qlge_mpi.o
1068 ZCONS_OBJS += zcons.o
1070 NV_SATA_OBJS += nv_sata.o
1072 SI3124_OBJS += si3124.o
1074 AHCI_OBJS += ahci.o
1076 PCIIDE_OBJS += pci-ide.o
1078 PCEPP_OBJS += pcepp.o
1080 CPC_OBJS += cpc.o
1082 CPUID_OBJS += cpuid_drv.o
1084 SYSEVENT_OBJS += sysevent.o
1086 BL_OBJS += bl.o
1088 DRM_OBJS += drm_sunmod.o drm_kstat.o drm_agpsupport.o \
1089      drm_auth.o drm_bufs.o drm_context.o drm_dma.o \
1090      drm_drawable.o drm_drv.o drm_fops.o drm_ioctl.o drm_irq.o \
1091      drm_lock.o drm_memory.o drm_msg.o drm_pci.o drm_scatter.o \
1092      drm_cache.o drm_gem.o drm_mm.o ati_pcigart.o
1094 FM_OBJS += devfm.o devfm_machdep.o
1096 RTLS_OBJS += rtls.o
1098 #
1099 #           exec modules
1100 #
1101 AOUTEXEC_OBJS += aout.o
1103 ELFEXEC_OBJS += elf.o elf_notes.o old_notes.o
1105 INTPEXEC_OBJS += intp.o
1107 SHBINEXEC_OBJS += shbin.o
1109 JAVAEXEC_OBJS += java.o
1111 #
1112 #           file system modules
1113 #
1114 AUTOFNS_OBJS += auto_vfsops.o auto_vnops.o auto_subr.o auto_xdr.o auto_sys.o
1116 CACHEFS_OBJS += cachefs_cnode.o          cachefs_cod.o \
1117      cachefs_dir.o          cachefs_dlog.o  cachefs_filegrp.o \

```

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

18

```

1118      cachefs_fscache.o          cachefs_ioctl.o cachefs_log.o \
1119      cachefs_module.o \
1120      cachefs_noopc.o          cachefs_resource.o \
1121      cachefs_strict.o \
1122      cachefs_subr.o          cachefs_vfsops.o \
1123      cachefs_vnops.o
1125 DCFS_OBJS += dc_vnops.o
1127 DEVFS_OBJS += devfs_subr.o  devfs_vfsops.o  devfs_vnops.o
1129 DEV_OBJS += sdev_subr.o      sdev_vfsops.o  sdev_vnops.o \
1130      sdev_ptsops.o          sdev_zvolops.o sdev_comm.o \
1131      sdev_profile.o         sdev_ncache.o  sdev_netops.o \
1132      sdev_ipnetops.o \
1133      sdev_vtops.o
1135 CTFS_OBJS += ctfs_all.o ctfs_cdir.o ctfs_ctl.o ctfs_event.o \
1136      ctfs_latest.o ctfs_root.o ctfs_sym.o ctfs_tdir.o ctfs_tmpl.o
1138 OBJFS_OBJS += objfs_vfs.o    objfs_root.o  objfs_common.o \
1139      objfs_odir.o          objfs_data.o
1141 FDFS_OBJS += fdops.o
1143 FIFO_OBJS += fifosubr.o     fifovnops.o
1145 PIPE_OBJS += pipe.o
1147 HSFS_OBJS += hsfs_node.o    hsfs_subr.o   hsfs_vfsops.o  hsfs_vnops.o \
1148      hsfs_susp.o          hsfs_rrip.o   hsfs_susp_subr.o
1150 LOFS_OBJS += lofs_subr.o     lofs_vfsops.o  lofs_vnops.o
1152 NAMEFS_OBJS += namevfs.o     namevno.o
1154 NFS_OBJS += nfs_client.o     nfs_common.o   nfs_dump.o \
1155      nfs_subr.o          nfs_vfsops.o   nfs_vnops.o \
1156      nfs_xdr.o          nfs_sys.o     nfs_strerror.o \
1157      nfs3_vfsops.o     nfs3_vnops.o  nfs3_xdr.o \
1158      nfs_acl_vnops.o   nfs_acl_xdr.o  nfs4_vfsops.o \
1159      nfs4_vnops.o     nfs4_xdr.o    nfs4_idmap.o \
1160      nfs4_shadow.o    nfs4_subr.o \
1161      nfs4_attr.o      nfs4_rnode.o  nfs4_client.o \
1162      nfs4_acache.o    nfs4_common.o  nfs4_client_state.o \
1163      nfs4_callback.o  nfs4_recovery.o  nfs4_client_secinfo.o \
1164      nfs4_client_debug.o  nfs_stats.o \
1165      nfs4_acl.o       nfs4_stub_vnops.o  nfs_cmd.o
1167 NFSSRV_OBJS += nfs_server.o   nfs_srv.o     nfs3_srv.o \
1168      nfs_acl_srv.o     nfs_auth.o    nfs_auth_xdr.o \
1169      nfs_export.o      nfs_log.o     nfs_log_xdr.o \
1170      nfs4_srv.o        nfs4_state.o  nfs4_srv_attr.o \
1171      nfs4_srv_ns.o     nfs4_db.o     nfs4_srv_deleg.o \
1172      nfs4_deleg_ops.o  nfs4_srv_readdir.o  nfs4_dispatch.o
1174 SMBSRV_SHARED_OBJS += \
1175      smb_inet.o \
1176      smb_match.o \
1177      smb_msgbuf.o \
1178      smb_oem.o \
1179      smb_string.o \
1180      smb_utf8.o \
1181      smb_door_legacy.o \
1182      smb_xdr.o \
1183      smb_token.o \

```

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

19

```

1184         smb_token_xdr.o \
1185         smb_sid.o \
1186         smb_native.o \
1187         smb_netbios_util.o

1189 SMBSRV_OBJS += $(SMBSRV_SHARED_OBJS)
1190         smb_acl.o
1191         smb_alloc.o
1192         smb_close.o
1193         smb_common_open.o
1194         smb_common_transact.o
1195         smb_create.o
1196         smb_delete.o
1197         smb_directory.o
1198         smb_dispatch.o
1199         smb_echo.o
1200         smb_fem.o
1201         smb_find.o
1202         smb_flush.o
1203         smb_fsinfo.o
1204         smb_fsops.o
1205         smb_init.o
1206         smb_kdoor.o
1207         smb_kshare.o
1208         smb_kutil.o
1209         smb_lock.o
1210         smb_lock_byte_range.o
1211         smb_locking_andx.o
1212         smb_logoff_andx.o
1213         smb_mangle_name.o
1214         smb_mbuf_marshallng.o
1215         smb_mbuf_util.o
1216         smb_negotiate.o
1217         smb_net.o
1218         smb_node.o
1219         smb_nt_cancel.o
1220         smb_nt_create_andx.o
1221         smb_nt_transact_create.o
1222         smb_nt_transact_ioctl.o
1223         smb_nt_transact_notify_change.o
1224         smb_nt_transact_quota.o
1225         smb_nt_transact_security.o
1226         smb_odir.o
1227         smb_ofile.o
1228         smb_open_andx.o
1229         smb_opipe.o
1230         smb_oplock.o
1231         smb_pathname.o
1232         smb_print.o
1233         smb_process_exit.o
1234         smb_query_fileinfo.o
1235         smb_read.o
1236         smb_rename.o
1237         smb_sd.o
1238         smb_seek.o
1239         smb_server.o
1240         smb_session.o
1241         smb_session_setup_andx.o
1242         smb_set_fileinfo.o
1243         smb_signing.o
1244         smb_tree.o
1245         smb_trans2_create_directory.o
1246         smb_trans2_dfs.o
1247         smb_trans2_find.o
1248         smb_tree_connect.o
1249         smb_unlock_byte_range.o

```

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

20

```

1250         smb_user.o
1251         smb_vfs.o
1252         smb_vops.o
1253         smb_vss.o
1254         smb_write.o
1255         smb_write_raw.o

1257 PCFS_OBJS += pc_alloc.o pc_dir.o pc_node.o pc_subr.o \
1258         pc_vfsops.o pc_vnops.o

1260 PROC_OBJS += prcontrol.o prioctl.o prsubr.o prusr.o \
1261         prvfsops.o prvnops.o

1263 MNTFS_OBJS += mntvfsops.o mntvnops.o

1265 SHAREFS_OBJS += sharetab.o sharefs_vfsops.o sharefs_vnops.o

1267 SPEC_OBJS += specsubr.o specvfsops.o specvnops.o

1269 SOCK_OBJS += socksubr.o sockvfsops.o sockparams.o \
1270         socksyscalls.o socktpi.o sockstr.o \
1271         sockcommon_vnops.o sockcommon_subr.o \
1272         sockcommon_sops.o sockcommon.o \
1273         sock_notsupp.o socknotify.o \
1274         nl7c.o nl7curi.o nl7chttp.o nl7clogd.o \
1275         nl7cnca.o sodirect.o sockfilter.o

1277 TMPFS_OBJS += tmp_dir.o tmp_subr.o tmp_tnode.o tmp_vfsops.o \
1278         tmp_vnops.o

1280 UDFS_OBJS += udf_alloc.o udf_bmap.o udf_dir.o \
1281         udf_inode.o udf_subr.o udf_vfsops.o \
1282         udf_vnops.o

1284 UFS_OBJS += ufs_alloc.o ufs_bmap.o ufs_dir.o ufs_xattr.o \
1285         ufs_inode.o ufs_subr.o ufs_tables.o ufs_vfsops.o \
1286         ufs_vnops.o quota.o quotacalls.o quota_ufs.o \
1287         ufs_filio.o ufs_lockfs.o ufs_thread.o ufs_trans.o \
1288         ufs_acl.o ufs_panic.o ufs_directio.o ufs_log.o \
1289         ufs_extvnops.o ufs_snap.o lufs.o lufs_thread.o \
1290         lufs_log.o lufs_map.o lufs_top.o lufs_debug.o \
1291         vscan_drv.o vscan_svc.o vscan_door.o

1293 NSMB_OBJS += smb_conn.o smb_dev.o smb_iod.o smb_pass.o \
1294         smb_rq.o smb_sign.o smb_smb.o smb_subrs.o \
1295         smb_time.o smb_tran.o smb_trantcp.o smb_usr.o \
1296         subr_mchain.o

1298 SMBFS_COMMON_OBJS += smbfs_ntacl.o
1299 SMBFS_OBJS += smbfs_vfsops.o smbfs_vnops.o smbfs_node.o \
1300         smbfs_acl.o smbfs_client.o smbfs_smb.o \
1301         smbfs_subr.o smbfs_subr2.o \
1302         smbfs_rwlock.o smbfs_xattr.o \
1303         $(SMBFS_COMMON_OBJS)

1306 #
1307 #
1308 #
1309 MD_OBJS += md.o md_error.o md_ioctl.o md_mddb.o md_names.o \
1310         md_med.o md_rename.o md_subr.o

1312 MD_COMMON_OBJS = md_convert.o md_crc.o md_revchk.o

1314 MD_DERIVED_OBJS = metamed_xdr.o meta_basic_xdr.o

```

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

21

```

1316 SOFTPART_OBJS += sp.o sp_ioctl.o
1318 STRIPE_OBJS += stripe.o stripe_ioctl.o
1320 HOTSPARES_OBJS += hotspares.o
1322 RAID_OBJS += raid.o raid_ioctl.o raid_replay.o raid_resync.o raid_hotspare.o
1324 MIRROR_OBJS += mirror.o mirror_ioctl.o mirror_resync.o
1326 NOTIFY_OBJS += md_notify.o
1328 TRANS_OBJS += mdtrans.o trans_ioctl.o trans_log.o

1330 ZFS_COMMON_OBJS += \
1331     arc.o \
1332     bplist.o \
1333     bpobj.o \
1334     bptree.o \
1335     dbuf.o \
1336     ddt.o \
1337     ddt_zap.o \
1338     dmuf.o \
1339     dmuf_diff.o \
1340     dmuf_send.o \
1341     dmuf_object.o \
1342     dmuf_objset.o \
1343     dmuf_traverse.o \
1344     dmuf_tx.o \
1345     dnode.o \
1346     dnode_sync.o \
1347     dsl_dir.o \
1348     dsl_dataset.o \
1349     dsl_deadlist.o \
1350     dsl_destroy.o \
1351     dsl_pool.o \
1352     dsl_synctask.o \
1353     dsl_userhold.o \
1354     dmuf_zfetch.o \
1355     dsl_deleg.o \
1356     dsl_prop.o \
1357     dsl_scan.o \
1358     zfeature.o \
1359     gzip.o \
1360     lz4.o \
1361     lzjb.o \
1362     metaslab.o \
1363     refcount.o \
1364     rrwlock.o \
1365     sa.o \
1366     sha256.o \
1367     spa.o \
1368     spa_config.o \
1369     spa_errlog.o \
1370     spa_history.o \
1371     spa_misc.o \
1372     space_map.o \
1373     txg.o \
1374     uberblock.o \
1375     unique.o \
1376     vdev.o \
1377     vdev_cache.o \
1378     vdev_file.o \
1379     vdev_label.o \
1380     vdev_mirror.o \
1381     vdev_missing.o \

```

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

22

```

1382     vdev_queue.o \
1383     vdev_raidz.o \
1384     vdev_root.o \
1385     zap.o \
1386     zap_leaf.o \
1387     zap_micro.o \
1388     zfs_byteswap.o \
1389     zfs_debug.o \
1390     zfs_fm.o \
1391     zfs_fuid.o \
1392     zfs_sa.o \
1393     zfs_znode.o \
1394     zil.o \
1395     zio.o \
1396     zio_checksum.o \
1397     zio_compress.o \
1398     zio_inject.o \
1399     zle.o \
1400     zlock.o

1402 ZFS_SHARED_OBJS += \
1403     zfeature_common.o \
1404     zfs_comutil.o \
1405     zfs_deleg.o \
1406     zfs_fletcher.o \
1407     zfs_namecheck.o \
1408     zfs_prop.o \
1409     zpool_prop.o \
1410     zprop_common.o

1412 ZFS_OBJS += \
1413     $(ZFS_COMMON_OBJS) \
1414     $(ZFS_SHARED_OBJS) \
1415     vdev_disk.o \
1416     zfs_acl.o \
1417     zfs_ctldir.o \
1418     zfs_dir.o \
1419     zfs_ioctl.o \
1420     zfs_log.o \
1421     zfs_onexit.o \
1422     zfs_replay.o \
1423     zfs_rlock.o \
1424     zfs_vfsops.o \
1425     zfs_vnops.o \
1426     zvol.o

1428 ZUT_OBJS += \
1429     zut.o

1431 #
1432 #           streams modules
1433 #
1434 BUFMOD_OBJS += bufmod.o

1436 CONNLD_OBJS += connld.o

1438 DEDUMP_OBJS += dedump.o

1440 DRCOMPAT_OBJS += drcompat.o

1442 LDLINUX_OBJS += ldlinux.o

1444 LDTERM_OBJS += ldterm.o uwidth.o

1446 PCKT_OBJS += pckt.o

```

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

23

```

1448 PFMOD_OBJS += pfmmod.o
1450 PTEM_OBJS += ptem.o
1452 REDIRMOD_OBJS += strredirm.o
1454 TIMOD_OBJS += timod.o
1456 TIRDWR_OBJS += tirdwr.o
1458 TTCOMPAT_OBJS +=ttcompat.o
1460 LOG_OBJS += log.o
1462 PIPEMOD_OBJS += pipemod.o

1464 RPCMOD_OBJS += rpcmod.o      clnt_cots.o      clnt_clts.o \
1465                  clnt_gen.o      clnt_perr.o      mt_rpcinit.o   rpc_calmsg.o \
1466                  rpc_prot.o      rpc_sztypes.o   rpc_subr.o     rpcb_prot.o \
1467                  svc.o           svc_clts.o      svc_gen.o      svc_cots.o \
1468                  rpcsys.o      xdr_sizeof.o   xdr_rdma.o     svc_rdma.o \
1469                  xdr_rdma.o      rdma_subr.o    xdrdma_sizeof.o

1471 TLIMOD_OBJS += tlimod.o      t_kalloc.o      t_kbind.o      t_kclose.o \
1472                  t_kconnect.o    t_kfree.o       t_kgtstate.o   t_kopen.o \
1473                  t_krcvudat.o    t_ksndudat.o   t_kspoll.o     t_kunbind.o \
1474                  t_kutil.o

1476 RLMOD_OBJS += rlmod.o
1478 TELMOD_OBJS += telmod.o
1480 CRYPTMOD_OBJS += cryptmod.o
1482 KB_OBJS += kbd.o          keytables.o

1484 #
1485 #           ID mapping module
1486 #
1487 IDMAP_OBJS += idmap_mod.o   idmap_kapi.o   idmap_xdr.o   idmap_cache.o

1489 #
1490 #           scheduling class modules
1491 #
1492 SDC_OBJS += sysdc.o

1494 RT_OBJS += rt.o
1495 RT_DPTBL_OBJS += rt_dptbl.o

1497 TS_OBJS += ts.o
1498 TS_DPTBL_OBJS += ts_dptbl.o

1500 IA_OBJS += ia.o

1502 FSS_OBJS += fss.o

1504 FX_OBJS += fx.o
1505 FX_DPTBL_OBJS += fx_dptbl.o

1507 #
1508 #           Inter-Process Communication (IPC) modules
1509 #
1510 IPC_OBJS += ipc.o

1512 IPCMSG_OBJS += msg.o

```

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

24

```

1514 IPCSEM_OBJS += sem.o
1516 IPCSHM_OBJS += shm.o

1518 #
1519 #           bignum module
1520 #
1521 COMMON_BIGNUM_OBJS += bignum_mod.o bignumimpl.o

1523 BIGNUM_OBJS += $(COMMON_BIGNUM_OBJS) $(BIGNUM_PSR_OBJS)

1525 #
1526 #           kernel cryptographic framework
1527 #
1528 KCF_OBJS += kcf.o kcf_callprov.o kcf_cbufoall.o kcf_cipher.o kcf_crypto.o \
1529             kcf_cryptoadm.o kcf_ctxops.o kcf_digest.o kcf_dual.o \
1530             kcf_keys.o kcf_mac.o kcf_mech_tabs.o kcf_miscapi.o \
1531             kcf_object.o kcf_policy.o kcf_prov_lib.o kcf_prov_tabs.o \
1532             kcf_sched.o kcf_session.o kcf_sign.o kcf_spi.o kcf_verify.o \
1533             kcf_random.o modes.o ecb.o cbc.o ctr.o ccm.o gcm.o \
1534             fips_random.o

1536 CRYPTOADM_OBJS += cryptoadm.o
1538 CRYPTO_OBJS += crypto.o

1540 DPROV_OBJS += dprov.o

1542 DCA_OBJS += dca.o dca_3des.o dca_debug.o dca_dsa.o dca_kstat.o dca_rng.o \
1543             dca_rsa.o

1545 AESPROV_OBJS += aes.o aes_impl.o aes_modes.o
1547 ARCFOURPROV_OBJS += arcfour.o arcfour_crypt.o
1549 BLOWFISHPROV_OBJS += blowfish.o blowfish_impl.o

1551 ECCPROV_OBJS += ecc.o ec.o ec2_163.o ec2_mont.o ecdecode.o ecl_mult.o \
1552             ecp_384.o ecp_jac.o ec2_193.o ecl.o ecp_192.o ecp_521.o \
1553             ecp_jm.o ec2_233.o ecl_curve.o ecp_224.o ecp_aff.o \
1554             ecp_mont.o ec2_aff.o ec_naf.o ecl_gf.o ecp_256.o mp_gf2m.o \
1555             mpi.o mplogic.o mpmontg.o mprime.o oid.o \
1556             secitem.o ec2_test.o ecp_test.o

1558 RSAPROV_OBJS += rsa.o rsa_impl.o pkcs1.o

1560 SWRANDPROV_OBJS += swrand.o

1562 #
1563 #           kernel SSL
1564 #
1565 KSSL_OBJS += kssl.o ksslioc1.o

1567 KSSL_SOCKETFIL_MOD_OBJS += ksslfilter.o ksslapi.o ksslrec.o

1569 #
1570 #           misc. modules
1571 #

1573 C2AUDIT_OBJS += adr.o audit.o audit_event.o audit_io.o \
1574             audit_path.o audit_start.o audit_syscalls.o audit_token.o \
1575             audit_mem.o

1577 PCIC_OBJS += pcic.o

1579 RPCSEC_OBJS += secmod.o      sec_clnt.o      sec_svc.o      sec_gen.o \

```

```

1580          auth_des.o      auth_kern.o      auth_none.o      auth_loopb.o\
1581          authdesprt.o    authdesubr.o    authu_prot.o \
1582          key_call.o      key_prot.o      svc_authu.o      svcauthdes.o

1584 RPCSEC_GSS_OBJS +=      rpcsec_gssmod.o rpcsec_gss.o rpcsec_gss_misc.o \
1585          rpcsec_gss_utils.o svc_rpcsec_gss.o

1587 CONSCONFIG_OBJS += consconfig.o

1589 CONSCONFIG_DACF_OBJS += consconfig_dacf.o consplat.o

1591 TEM_OBJS += tem.o tem_safe.o 6x10.o 7x14.o 12x22.o

1593 KBTRANS_OBJS +=          \
1594          kbtrans.o        \
1595          kbtrans_keytables.o \
1596          kbtrans_polled.o  \
1597          kbtrans_streams.o \
1598          usb_keytables.o

1600 KGSSD_OBJS +=      gssd_clnt_stubs.o gssd_handle.o gssd_prot.o \
1601          gss_display_name.o gss_release_name.o gss_import_name.o \
1602          gss_release_buffer.o gss_release_oid_set.o gen_oids.o gssdmod.o

1604 KGSSD_DERIVED_OBJS = gssd_xdr.o

1606 KGSS_DUMMY_OBJS += dmech.o

1608 KSOCKET_OBJS += ksocket.o ksocket_mod.o

1610 CRYPTO= cksmtypes.o decrypt.o encrypt.o encrypt_length.o etypes.o \
1611          nfold.o verify_checksum.o prng.o block_size.o make_checksum.o\
1612          checksum_length.o hmac.o default_state.o mandatory_sumtype.o

1614 # crypto/des
1615 CRYPTO_DES= f_cbc.o f_cksum.o f_parity.o weak_key.o d3_cbc.o ef_crypto.o

1617 CRYPTO_DK= checksum.o derive.o dk_decrypt.o dk_encrypt.o

1619 CRYPTO_ARCFOUR= k5_arcfour.o

1621 # crypto/enc_provider
1622 CRYPTO_ENC= des.o des3.o arcfour_provider.o aes_provider.o

1624 # crypto/hash_provider
1625 CRYPTO_HASH= hash_kef_generic.o hash_kmd5.o hash_crc32.o hash_kshal.o

1627 # crypto/keyhash_provider
1628 CRYPTO_KEYHASH= descbc.o k5_kmd5des.o k_hmac_md5.o

1630 # crypto/crc32
1631 CRYPTO_CRC32= crc32.o

1633 # crypto/old
1634 CRYPTO_OLD= old_decrypt.o old_encrypt.o

1636 # crypto/raw
1637 CRYPTO_RAW= raw_decrypt.o raw_encrypt.o

1639 K5_KRB= kfree.o copy_key.o \
1640          parse.o init_ctx.o \
1641          ser_adata.o ser_addr.o \
1642          ser_auth.o ser_cksum.o \
1643          ser_key.o ser_princ.o \
1644          serialize.o unparse.o \
1645          ser_actx.o

```

```

1647 K5_OS=   timeofday.o toffset.o \
1648          init_os_ctx.o c_ustime.o

1650 SEAL=
1651 # EXPORT DELETE START
1652 SEAL=    seal.o unseal.o
1653 # EXPORT DELETE END

1655 MECH=    delete_sec_context.o \
1656          import_sec_context.o \
1657          gssapi_krb5.o \
1658          k5seal.o k5unseal.o k5sealv3.o \
1659          ser_sctx.o \
1660          sign.o \
1661          util_crypt.o \
1662          util_validate.o util_ordering.o \
1663          util_seqnum.o util_set.o util_seed.o \
1664          wrap_size_limit.o verify.o

1668 MECH_GEN= util_token.o

1671 KGSS_KRB5_OBJS += krb5mech.o \
1672          $(MECH) $(SEAL) $(MECH_GEN) \
1673          $(CRYPTO) $(CRYPTO_DES) $(CRYPTO_DK) $(CRYPTO_ARCFOUR) \
1674          $(CRYPTO_ENC) $(CRYPTO_HASH) \
1675          $(CRYPTO_KEYHASH) $(CRYPTO_CRC32) \
1676          $(CRYPTO_OLD) \
1677          $(CRYPTO_RAW) $(K5_KRB) $(K5_OS)

1679 DES_OBJS +=      des_crypt.o des_impl.o des_ks.o des_soft.o

1681 DLBOOT_OBJS +=  bootparam_xdr.o nfs_dlinet.o scan.o

1683 KRTLD_OBJS +=   kobj_bootflags.o getoptstr.o \
1684          kobj.o kobj_kdi.o kobj_lm.o kobj_subr.o

1686 MOD_OBJS +=     modctl.o modsubr.o modsysfile.o modconf.o modhash.o

1688 STRPLUMB_OBJS += strplumb.o

1690 CPR_OBJS +=      cpr_driver.o cpr_dump.o \
1691          cpr_main.o cpr_misc.o cpr_mod.o cpr_stat.o \
1692          cpr_uthread.o

1694 PROF_OBJS +=    prf.o

1696 SE_OBJS +=      se_driver.o

1698 SYSACCT_OBJS += acct.o

1700 ACCTCTL_OBJS += acctctl.o

1702 EXACCTSYS_OBJS += exacctsys.o

1704 KAIO_OBJS +=    aio.o

1706 PCMCIA_OBJS +=  pcmcia.o cs.o cis.o cis_callout.o cis_handlers.o cis_params.o

1708 BUSRA_OBJS +=   busra.o

1710 PCS_OBJS +=     pcs.o

```

1712 PCAN_OBJS += pcan.o
1714 PCATA_OBJS += pcide.o pcdisk.o pclabel.o pcata.o
1716 PCSER_OBJS += pcser.o pcser_cis.o
1718 PCWL_OBJS += pcwl.o
1720 PSET_OBJS += pset.o
1722 OHCI_OBJS += ohci.o ohci_hub.o ohci_polled.o
1724 UHCI_OBJS += uhci.o uhciutil.o uhcitgt.o uhcihub.o uhcipolled.o
1726 EHCI_OBJS += ehci.o ehci_hub.o ehci_xfer.o ehci_intr.o ehci_util.o ehci_polled.o
1728 HUBD_OBJS += hubd.o
1730 USB_MID_OBJS += usb_mid.o
1732 USB_IA_OBJS += usb_ia.o
1734 UWBA_OBJS += uwba.o uwbai.o
1736 SCSA2USB_OBJS += scsa2usb.o usb_ms_bulkonly.o usb_ms_cbi.o
1738 HWAHC_OBJS += hwahc.o hwahc_util.o
1740 WUSB_DF_OBJS += wusb_df.o
1741 WUSB_FWMOD_OBJS += wusb_fwmod.o
1743 IPF_OBJS += ip_fil_solaris.o fil.o solaris.o ip_state.o ip_frag.o ip_nat.o \
1744 ip_proxy.o ip_auth.o ip_pool.o ip_htable.o ip_lookup.o \
1745 ip_log.o misc.o ip_compat.o ip_nat6.o drand48.o
1747 IBD_OBJS += ibd.o ibd_cm.o
1749 EIBNX_OBJS += enx_main.o enx_hdlrs.o enx_ibt.o enx_log.o enx_fip.o \
1750 enx_misc.o enx_q.o enx_ctl.o
1752 EOIB_OBJS += eib_adm.o eib_chan.o eib_cmn.o eib_ctl.o eib_data.o \
1753 eib_fip.o eib_ibt.o eib_log.o eib_mac.o eib_main.o \
1754 eib_rsrc.o eib_svc.o eib_vnic.o
1756 DLPSTUB_OBJS += dlpistub.o
1758 SDP_OBJS += sdppi.o
1760 TRILL_OBJS += trill.o
1762 CTF_OBJS += ctf_create.o ctf_decl.o ctf_error.o ctf_hash.o ctf_labels.o \
1763 ctf_lookup.o ctf_open.o ctf_types.o ctf_util.o ctf_subr.o ctf_mod.o
1765 SMBIOS_OBJS += smb_error.o smb_info.o smb_open.o smb_subr.o smb_dev.o
1767 RPCIB_OBJS += rpcib.o
1769 KMDB_OBJS += kdrv.o
1771 AFE_OBJS += afe.o
1773 BGE_OBJS += bge_main2.o bge_chip2.o bge_kstats.o bge_log.o bge_ndd.o \
1774 bge_atomic.o bge_mii.o bge_send.o bge_recv2.o bge_mii_5906.o
1776 DMFE_OBJS += dmfe_log.o dmfe_main.o dmfe_mii.o

1778 EFE_OBJS += efe.o
1780 ELXL_OBJS += elxl.o
1782 HME_OBJS += hme.o
1784 IXGB_OBJS += ixgb.o ixgb_atomic.o ixgb_chip.o ixgb_gld.o ixgb_kstats.o \
1785 ixgb_log.o ixgb_ndd.o ixgb_rx.o ixgb_tx.o ixgb_xmii.o
1787 NGE_OBJS += nge_main.o nge_atomic.o nge_chip.o nge_ndd.o nge_kstats.o \
1788 nge_log.o nge_rx.o nge_tx.o nge_xmii.o
1790 PCN_OBJS += pcn.o
1792 RGE_OBJS += rge_main.o rge_chip.o rge_ndd.o rge_kstats.o rge_log.o rge_rxtx.o
1794 URTW_OBJS += urtw.o
1796 ARN_OBJS += arn_hw.o arn_eeeprom.o arn_mac.o arn_calib.o arn_ani.o arn_phy.o arn_\
1797 arn_main.o arn_recv.o arn_xmit.o arn_rc.o
1799 ATH_OBJS += ath_aux.o ath_main.o ath_osdep.o ath_rate.o
1801 ATU_OBJS += atu.o
1803 IPW_OBJS += ipw2100_hw.o ipw2100.o
1805 IWI_OBJS += ipw2200_hw.o ipw2200.o
1807 IWH_OBJS += iwh.o
1809 IWK_OBJS += iw2.o
1811 IWP_OBJS += iwp.o
1813 MWL_OBJS += mwl.o
1815 MWLFW_OBJS += mwlfw_mode.o
1817 WPI_OBJS += wpi.o
1819 RAL_OBJS += rt2560.o ral_rate.o
1821 RUM_OBJS += rum.o
1823 RWD_OBJS += rt2661.o
1825 RWN_OBJS += rt2860.o
1827 UATH_OBJS += uath.o
1829 UATHFW_OBJS += uathfw_mod.o
1831 URAL_OBJS += ural.o
1833 RTW_OBJS += rtw.o smc93cx6.o rtwphy.o rtwphyio.o
1835 ZYD_OBJS += zyd.o zyd_usb.o zyd_hw.o zyd_fw.o
1837 MXFE_OBJS += mxfe.o
1839 MPTSAS_OBJS += mptsas.o mptsas_impl.o mptsas_init.o mptsas_raid.o mptsas_smhba.o
1841 SFE_OBJS += sfe.o sfe_util.o
1843 BFE_OBJS += bfe.o

```

1845 BRIDGE_OBJS += bridge.o
1847 IDM_SHARED_OBJS += base64.o
1849 IDM_OBJS += $(IDM_SHARED_OBJS) \
1850 idm.o idm_impl.o idm_text.o idm_conn_sm.o idm_so.o
1852 VR_OBJS += vr.o
1854 ATGE_OBJS += atge_main.o atge_lle.o atge_mii.o atge_ll.o atge_llc.o
1856 YGE_OBJS = yge.o
1858 #
1859 # Build up defines and paths.
1860 #
1861 LINT_DEFS += -Dunix
1863 #
1864 # This duality can be removed when the native and target compilers
1865 # are the same (or at least recognize the same command line syntax!)
1866 # It is a bug in the current compilation system that the assembler
1867 # can't process the -Y I, flag.
1868 #
1869 NATIVE_INC_PATH += $(INC_PATH) $(CCYFLAG)$(UTSBASE)/common
1870 AS_INC_PATH += $(INC_PATH) -I$(UTSBASE)/common
1871 INCLUDE_PATH += $(INC_PATH) $(CCYFLAG)$(UTSBASE)/common
1873 PCIEB_OBJS += pcieb.o
1875 # Chelsio N110 10G NIC driver module
1876 #
1877 CH_OBJS = ch.o glue.o pe.o sge.o
1879 CH_COM_OBJS = ch_mac.o ch_subr.o cspi.o espi.o ixfl1010.o mc3.o mc4.o mc5.o \
1880 mv88elxxx.o mv88x20lx.o my3126.o pm3393.o tp.o ulp.o \
1881 vsc7321.o vsc7326.o xpak.o
1883 #
1884 # Chelsio Terminator 4 10G NIC nexus driver module
1885 #
1886 CXGBE_FW_OBJS = t4_fw.o t4_cfg.o
1887 CXGBE_COM_OBJS = t4_hw.o common.o
1888 CXGBE_NEX_OBJS = t4_nexus.o t4_sge.o t4_mac.o t4_ioctl.o shared.o \
1889 t4_l2t.o adapter.o osdep.o
1891 #
1892 # Chelsio Terminator 4 10G NIC driver module
1893 #
1894 CXGBE_OBJS = cxgbe.o
1896 #
1897 # PCI strings file
1898 #
1899 PCI_STRING_OBJS = pci_strings.o
1901 NET_DACF_OBJS += net_dacf.o
1903 #
1904 # Xframe 10G NIC driver module
1905 #
1906 XGE_OBJS = xge.o xgell.o
1908 XGE_HAL_OBJS = xgehal-channel.o xgehal-fifo.o xgehal-ring.o xgehal-config.o \
1909 xgehal-driver.o xgehal-mm.o xgehal-stats.o xgehal-device.o \

```

```

1910 xge-queue.o xgehal-mgmt.o xgehal-mgmtaux.o
1912 #
1913 # e1000g module
1914 #
1915 E1000G_OBJS += e1000_80003es2lan.o e1000_82540.o e1000_82541.o e1000_82542.o \
1916 e1000_82543.o e1000_82571.o e1000_api.o e1000_ich8lan.o \
1917 e1000_mac.o e1000_manage.o e1000_nvm.o e1000_osdep.o \
1918 e1000_phy.o e1000g_debug.o e1000g_main.o e1000g_alloc.o \
1919 e1000g_tx.o e1000g_rx.o e1000g_stat.o
1921 #
1922 # Intel 82575 1G NIC driver module
1923 #
1924 IGB_OBJS = igb_82575.o igb_api.o igb_mac.o igb_manage.o \
1925 igb_nvm.o igb_osdep.o igb_phy.o igb_buf.o \
1926 igb_debug.o igb_gld.o igb_log.o igb_main.o \
1927 igb_rx.o igb_stat.o igb_tx.o
1929 #
1930 # Intel Pro/100 NIC driver module
1931 #
1932 IPRB_OBJS = iprb.o
1934 #
1935 # Intel 10GbE PCIE NIC driver module
1936 #
1937 IXGBE_OBJS = ixgbe_82598.o ixgbe_82599.o ixgbe_api.o \
1938 ixgbe_common.o ixgbe_phy.o \
1939 ixgbe_buf.o ixgbe_debug.o ixgbe_gld.o \
1940 ixgbe_log.o ixgbe_main.o \
1941 ixgbe_osdep.o ixgbe_rx.o ixgbe_stat.o \
1942 ixgbe_tx.o ixgbe_x540.o ixgbe_mbx.o
1944 #
1945 # NIU 10G/1G driver module
1946 #
1947 NXGE_OBJS = nxge_mac.o nxge_ipp.o nxge_rxdma.o \
1948 nxge_txdma.o nxge_txc.o nxge_main.o \
1949 nxge_hw.o nxge_fzc.o nxge_virtual.o \
1950 nxge_send.o nxge_classify.o nxge_fflp.o \
1951 nxge_fflp_hash.o nxge_ndd.o nxge_kstats.o \
1952 nxge_zcp.o nxge_fm.o nxge_espc.o nxge_hv.o \
1953 nxge_hio.o nxge_hio_guest.o nxge_intr.o
1955 NXGE_NPI_OBJS = \
1956 npi.o npi_mac.o npi_ipp.o \
1957 npi_txdma.o npi_rxdma.o npi_txc.o \
1958 npi_zcp.o npi_espc.o npi_fflp.o \
1959 npi_vir.o
1961 NXGE_HCALL_OBJS = \
1962 nxge_hcall.o
1964 #
1965 # Virtio modules
1966 #
1968 # Virtio core
1969 VIRTIO_OBJS = virtio.o
1971 # Virtio block driver
1972 VIOBLK_OBJS = vioblk.o
1974 #
1975 # kiconv modules

```

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

31

```
1976 #
1977 KICONV_EMEA_OBJS += kiconv_emea.o

1979 KICONV_JA_OBJS += kiconv_ja.o

1981 KICONV_KO_OBJS += kiconv_cck_common.o kiconv_ko.o

1983 KICONV_SC_OBJS += kiconv_cck_common.o kiconv_sc.o

1985 KICONV_TC_OBJS += kiconv_cck_common.o kiconv_tc.o

1987 #
1988 #     AAC module
1989 #
1990 AAC_OBJS = aac.o aac_ioctl.o

1992 #
1993 #     sdc card modules
1994 #
1995 SDA_OBJS =     sda_cmd.o sda_host.o sda_init.o sda_mem.o sda_mod.o sda_slot.o
1996 SDHOST_OBJS = sdhost.o

1998 #
1999 #     hxge 10G driver module
2000 #
2001 HXGE_OBJS =     hxge_main.o hxge_vmac.o hxge_send.o           \
2002                hxge_txdma.o hxge_rxdma.o hxge_virtual.o     \
2003                hxge_fm.o hxge_fzc.o hxge_hw.o hxge_kstats.o \
2004                hxge_ndd.o hxge_pfc.o                         \
2005                hpi.o hpi_vmac.o hpi_rxdma.o hpi_txdma.o     \
2006                hpi_vir.o hpi_pfc.o

2008 #
2009 #     MEGARAID_SAS module
2010 #
2011 MEGA_SAS_OBJS = megaraid_sas.o

2013 #
2014 #     MR_SAS module
2015 #
2016 MR_SAS_OBJS = ld_pd_map.o mr_sas.o mr_sas_tbolt.o mr_sas_list.o

2018 #
2019 #     ISCSI_INITIATOR module
2020 #
2021 ISCSI_INITIATOR_OBJS = chap.o iscsi_io.o iscsi_thread.o      \
2022                       iscsi_ioctl.o iscsid.o iscsi.o         \
2023                       iscsi_login.o isns_client.o iscsiAuthClient.o \
2024                       iscsi_lun.o iscsiAuthClientGlue.o     \
2025                       iscsi_net.o nvfile.o iscsi_cmd.o       \
2026                       iscsi_queue.o persistent.o iscsi_conn.o \
2027                       iscsi_sess.o radius_auth.o iscsi_crc.o \
2028                       iscsi_stats.o radius_packet.o iscsi_doorclt.o \
2029                       iscsi_targetparam.o utils.o kifconf.o

2031 #
2032 #     ntxn 10Gb/1Gb NIC driver module
2033 #
2034 NTXN_OBJS =     unm_nic_init.o unm_gem.o unm_nic_hw.o unm_ndd.o \
2035                unm_nic_main.o unm_nic_isr.o unm_nic_ctx.o niu.o

2037 #
2038 #     Myricom 10Gb NIC driver module
2039 #
2040 MYRI10GE_OBJS = myri10ge.o myri10ge_lro.o
```

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

32

```
2042 #     nulldriver module
2043 #
2044 NULLDRIVER_OBJS =     nulldriver.o

2046 TPM_OBJS =     tpm.o tpm_hcall.o
```



```

*****
28843 Sat Aug 24 02:37:58 2013
new/usr/src/uts/common/fs/fsh.c
fsh webrev (August 24th)
*****
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 2013 Damian Bogel. All rights reserved.
14 */

16 #include <sys/debug.h>
17 #include <sys/errno.h>
18 #include <sys/fsh.h>
19 #include <sys/fsh_impl.h>
20 #include <sys/id_space.h>
21 #include <sys/kmem.h>
22 #include <sys/ksynch.h>
23 #include <sys/list.h>
24 #include <sys/sunddi.h>
25 #include <sys/sysmacros.h>
26 #include <sys/types.h>
27 #include <sys/vfs.h>
28 #include <sys/vnode.h>

30 /*
31  * Filesystem hook framework (fsh)
32  *
33  * 1. Abstract.
34  * The main goal of the filesystem hook framework is to provide an easy way to
35  * inject client-defined behaviour into vfs/vnode calls. fsh works on
36  * vfs_t granularity.
37  *
38  *
39  * 2. Overview.
40  * fsh_t is the main object in the fsh. An fsh_t is a structure containing:
41  * - pointers to hooking functions (named after corresponding
42  *   vnodeops/vfsops)
43  * - a pointer to an argument to pass (this is shared for all the
44  *   hooks in a given fsh_t)
45  *
46  * The information from fsh_t is copied by the fsh and an fsh_handle_t
47  * is returned. It should be used for further removing.
48  *
49  *
50  * 3. Usage.
51  * It is expected that vfs_t/vnode_t that are passed to fsh_foo() functions
52  * are held by the caller when needed. fsh does no vfs_t/vnode_t locking.
53  *
54  * fsh_t is a structure filled out by the client. If a client does not want
55  * to add/remove a hook for function foo(), he should fill the foo field of
56  * fsh_t with NULL. Every hook has a type of corresponding vfsop/vnodeop with
57  * two additional arguments:
58  * - fsh_int_t *fsh_int - this argument MUST be passed to
59  *   hook_next_foo(). fsh wouldn't know which hook to execute next
60  *   without it
61  * - void *arg - this is the argument passed with fsh_t during

```

```

62  * installation
63  * After installation, an fsh_handle_t is returned to the caller.
64  *
65  * A client might want to fire callbacks when vfs_t's are being mounted
66  * or freed. There's an fsh_callback_t structure provided to install such
67  * callbacks along with the API.
68  * It is legal to call fsh_hook_{install,remove}() inside a mount callback
69  * WITHOUT holding the vfs_t.
70  *
71  * After vfs_t's free callback returns, all the handles associated with the
72  * hooks installed on this vfs_t are invalid and must not be used.
73  *
74  *
75  * Every hook function is responsible for passing the control to the next
76  * hook associated with a particular call. In order to provide an easy way to
77  * modify the behaviour of a function call both before and after the
78  * underlying vfsop/vnodeop (or next hook) execution, a hook has to call
79  * fsh_next_foo() at some point. This function does necessary internal
80  * operations and calls the next hook, until there's no hook left, then it
81  * calls the underlying vfsop/vnodeop.
82  * Example:
83  * my_freefs(fsh_int_t *fsh_int, void *arg, vfs_t *vfsp) {
84  *     cmn_err(CE_NOTE, "freefs called!\n");
85  *     return (fsh_next_freefs(fsh_int, vfsp));
86  * }
87  *
88  *
89  * 4. API
90  * None of the APIs should be called during interrupt context above lock
91  * level. The only exceptions are fsh_next_foo() functions, which do not use
92  * locks.
93  *
94  * a) fsh.h
95  * None of the functions listed below should be called inside of a hook
96  * Doing so will cause a deadlock. The only exceptions are fsh_next_foo() and
97  * fsh_callback_{install,remove}().
98  *
99  * fsh_callback_{install,remove}() should not be called inside of a {mount,free}
100 * callback. Doing so will cause a deadlock.
101 *
102 * fsh_fs_enable(vfs_t *vfsp)
103 * fsh_fs_disable(vfs_t *vfsp)
104 *     Enables/disables fsh for a given vfs_t.
105 *
106 * fsh_hook_install(vfs_t *vfsp, fsh_t *hooks)
107 *     Installs hooks on vfsp filesystem. It's important that hooks are
108 *     executed in LIFO installation order, which means that if there are
109 *     hooks A and B installed in this order, B is going to be executed
110 *     before A. It returns a correct handle, or (-1) if hook/callback
111 *     limit exceeded. The handle is valid until a free callback returns
112 *     or an explicit call to fsh_hook_remove().
113 *
114 * fsh_hook_remove(fsh_handle_t handle)
115 *     Removes a hook and invalidates the handle. It is guaranteed that after
116 *     this function returns, hook associated with this handle won't be
117 *     executing.
118 *
119 * fsh_next_foo(fsh_int_t *fsh_int, void *arg, ARGUMENTS)
120 *     This is the function which should be called once in every hook. It
121 *     does the necessary internal operations and passes control to the
122 *     next hook or, if there's no hook left, to the underlying
123 *     vfsop/vnodeop.
124 *
125 * fsh_callback_install(fsh_callback_t *callback)
126 * fsh_callback_remove(fsh_callback_handle_t handle)
127 *     Installs/removes callbacks for vfs_t mount/free. The mount callback

```

```

128 * is executed right before domount() returns. The free callback is
129 * called right before VFS_FREEVFS() is called. The
130 * fsh_callback_install() returns a correct handle, or (-1) if
131 * hook/callback limit exceeded.
132 *
133 * b) fsh_impl.h (for vfs.c and vnode.c only)
134 * fsh_init()
135 * This call has to be done in vfsinit(). It initialises the fsh. It
136 * is absolutely necessary that this call is made before any other fsh
137 * operation.
138 *
139 * fsh_exec_mount_callbacks(vfs_t *vfsp)
140 * fsh_exec_free_callbacks(vfs_t *vfsp)
141 * Used to execute all fsh callbacks for {mount,free} of a vfs_t.
142 *
143 * fsh_fsrec_destroy(struct fsh_fsrecord *fsrecp)
144 * Destroys an fsh_fsrecord structure. All the hooks installed on this
145 * vfs_t are then destroyed. free callback is called before this function.
146 *
147 * fsh_foo(ARGUMENTS)
148 * Function used to start executing the hook chain for a given call.
149 *
150 * 5. Internals.
151 * fsh_fsrecord_t is a structure which lives inside a vfs_t.
152 * fsh_fsrecord_t contains:
153 * - an rw-lock that protects the structure
154 * - a list of hooks installed on this vfs_t
155 * - a flag which tells whether fsh is enabled on this vfs_t
156 *
157 * Unfortunately, because of unexpected behaviour of some filesystems (no use of
158 * vfs_alloc()/vfs_init()) there's no good place to initialise the
159 * fsh_fsrecord_t structure. The approach being used here is to check if it's
160 * initialised in every call. Because of the fact that no lock could be used
161 * here (the same problem with initialisation), a spinlock is used. This is
162 * explained in more detail in a comment before fsh_prepare_fsrec(), a function
163 * that MUST be used whenever a vfsp->vfs_fsrecord needs to be accessed. After
164 * doing that, it's completely safe to keep this pointer locally, because it
165 * won't be changed until vfs_free() is called.
166 *
167 * The only exception from the fsh_prepare_fsrec() rule is vfs_free(),
168 * where there is expected that no other fsh calls would be made for the
169 * vfs_t that's being freed. That's why vfs_fsrecord could be only NULL or a
170 * valid pointer and could not be concurrently accessed.
171 *
172 * When there are no fsh functions (that use a particular fsh_fsrecord_t)
173 * executing, the vfs_fsrecord pointer won't be equal to fsh_res_ptr. It
174 * would be NULL or a pointer to an initialised fsh_fsrecord_t.
175 *
176 *
177 * Mount callbacks are executed by a call to fsh_exec_mount_callbacks() right
178 * before returning from domount()@vfs.c.
179 *
180 * Free callbacks are executed by a call to fsh_exec_free_callbacks() right
181 * before calling VFS_FREEVFS(), after vfs_t's reference count drops to 0.
182 *
183 *
184 * fsh_next_foo(fsh_int_t *fshi, ARGUMENTS)
185 * This function is quite simple. It takes the fsh_int_t and passes control
186 * to the next hook or to the underlying vnodeop/vfsop. fshi is always the
187 * next hook to be executed.
188 *
189 *
190 * 6. Concurrency
191 * fsh does no vfs_t nor vnode_t locking. It is expected that whenever it is
192 * needed, the client does that.
193 *

```

```

194 * An fsh_fsrecord_t of a vfs_t is read-locked (fshfsr_lock) by every
195 * fsh_foo() function (with the mentioned vfs_t as a parameter, of course).
196 * This means that fsh_hook_{install,remove}() must NOT be called inside of
197 * a hook, because it will cause a deadlock.
198 *
199 * The same thing applies to callbacks. fsh_cblst is read-locked by
200 * fsh_exec_{mount,free}(). This means that fsh_callback_{install,remove}
201 * must not be called inside a callback, because it will cause a deadlock.
202 *
203 * Solution to concurrency issues involving vfs_fsrecord are explained
204 * both in chapter 5th "Internals" and before fsh_prepare_fsrec() function.
205 *
206 * Concurrency issues between fsh_hook_remove() and fsh_fsrec_destroy() are
207 * solved by fsh_remove_lock. For more info see: fsh_remove_lock,
208 * fsh_hook_remove(), fsh_fsrec_destroy().
209 */
210
211 /* Internals */
212 /* Used for mapping an fsh_handle_t to fsh_int_t. */
213 typedef struct fsh_mapping {
214     fsh_handle_t    fshm_handle;
215     fsh_int_t       *fshm_fshi;
216     vfs_t           *fshm_vfsp;
217     list_node_t     fshm_next;
218 } fsh_mapping_t;
219
220 struct fsh_int {
221     fsh_handle_t     fshi_handle;
222     fsh_t            fshi_hooks;
223     fsh_mapping_t    *fshi_mapping;
224     list_node_t      fshi_next;
225 };
226
227 typedef struct fsh_callback_int {
228     fsh_callback_t   fshci_cb;
229     fsh_callback_handle_t fshci_handle;
230     list_node_t      fshci_next;
231 } fsh_callback_int_t;
232
233
234 /*
235 * fsh_fsrecord_t is the main internal structure. It's content is protected
236 * by fshfsr_lock. The fshfsr_list is a list of fsh_int_t hook entries for
237 * the vfs_t that contains the fsh_fsrecord_t.
238 *
239 * It is guaranteed by the fsh_prepare_fsrec() that outside the fsh,
240 * a pointer to fsh_fsrecord inside a vfs_t is never equal to fsh_res_ptr.
241 */
242 struct fsh_fsrecord {
243     krwlock_t        fshfsr_lock;
244     int              fshfsr_enabled;
245     list_t           fshfsr_list; /* list of fsh_int_t */
246 };
247
248 /*
249 * It's a list of fsh_mapping_t's used to map fsh_handle_t's to
250 * fsh_int_t's. This is needed because of the fact that we'd like an opaque
251 * handle returned to the fsh API client after a hook is successfully
252 * installed. We'd like to make the handle the only thing that is needed
253 * after the hooks are installed, for further actions on them. This means,
254 * that there is no easy way to search for the hooks matching a handle,
255 * without having the vfs_t on which they are installed.
256 * The same problem doesn't apply to callbacks, that's why fsh_map handles
257 * only fsh_handle_t to fsh_int_t translation.
258 */
259 static kmutex_t fsh_map_lock;

```

```

260 static list_t fsh_map;

262 /*
263  * The lock is used when there is a need to lock both the fsh_fsrecord_t and
264  * fsh_map.
265  * fsh_hook_remove() and fsh_fsrec_destroy() use this lock to protect both the
266  * fsh_map and the fsh_fsrecord_t associated with a vfs_t on which the removing
267  * is performed. Because fsh_hook_remove() starts with handle, through fsh_map
268  * to vfs_fshrecord and fsh_fsrec_destroy() from vfs_fshrecord to fsh_map, such
269  * lock is necessary.
270  * For more info see: fsh_fsrec_destroy() and fsh_hook_remove()
271  */
272 static kmutex_t fsh_remove_lock;

274 /*
275  * It's a list of fsh_callback_int_t's. Unlike hooks, there is no need to
276  * keep a separate list for translating handles to fsh_callback_int_t's,
277  * because a callback list is global for all the vfs_t's.
278  */
279 static krwlock_t fsh_cblst_lock;
280 static list_t fsh_cblst;

282 /*
283  * A reserved pointer for fsh purposes. It is used because of the method
284  * chosen for solving concurrency issues with vfs_fshrecord. The full
285  * explanation is in the big theory statement at the beginning of this
286  * file. It is initialised in fsh_init().
287  */
288 static void *fsh_res_ptr;

290 static fsh_fsrecord_t *fsh_fsrec_create();

292 int fsh_limit = INT_MAX;
293 static id_space_t *fsh_idspace;

295 /*
296  * Important note:
297  * Before using this function, fsh_init() MUST be called. We do that in
298  * vfsinit()@vfs.c.
299  *
300  * One would ask, why isn't the vfsp->vfs_fshrecord initialised when the
301  * vfs_t is created. Unfortunately, some filesystems (e.g. fifofs) do not
302  * call vfs_init() or even vfs_alloc(), It's possible that some unbundled
303  * filesystems could do the same thing. That's why this solution is
304  * introduced. It should be called before any code that needs access to
305  * vfs_fshrecord.
306  *
307  * Locking:
308  * There are no locks here, because there's no good place to initialise
309  * the lock. Concurrency issues are solved by using atomic instructions
310  * and a spinlock, which is spinning only once for a given vfs_t. Because
311  * of that, the usage of the spinlock isn't bad at all.
312  *
313  * How it works:
314  * a) if vfsp->vfs_fshrecord equals NULL, atomic_cas_ptr() changes it to
315  *    fsh_res_ptr. That's a signal for other threads, that the structure
316  *    is being initialised.
317  * b) if vfsp->vfs_fshrecord equals fsh_res_ptr, that means we have to wait,
318  *    because vfs_fshrecord is being initialised by another call.
319  * c) other cases:
320  *    vfs_fshrecord is already initialised, so we can use it. It won't change
321  *    until vfs_free() is called. It can't happen when someone is holding
322  *    the vfs_t, which is expected from the caller of fsh API.
323  */
324 static void
325 fsh_prepare_fsrec(vfs_t *vfsp)

```

```

326 {
327     fsh_fsrecord_t *fsrec;

329     while ((fsrec = atomic_cas_ptr(&vfsp->vfs_fshrecord, NULL,
330         fsh_res_ptr)) == fsh_res_ptr)
331         ;

333     if (fsrec == NULL)
334         atomic_swap_ptr(&vfsp->vfs_fshrecord, fsh_fsrec_create());
335 }

337 /*
338  * API for enabling/disabling fsh per vfs_t.
339  *
340  * A newly created vfs_t has fsh enabled by default. If one would want to change
341  * this behaviour, mount callbacks could be used.
342  *
343  * The caller is expected to hold the vfs_t.
344  *
345  * These functions must NOT be called in a hook.
346  */
347 void
348 fsh_fs_enable(vfs_t *vfsp)
349 {
350     fsh_prepare_fsrec(vfsp);

352     rw_enter(&vfsp->vfs_fshrecord->fshfsr_lock, RW_WRITER);
353     vfsp->vfs_fshrecord->fshfsr_enabled = 1;
354     rw_exit(&vfsp->vfs_fshrecord->fshfsr_lock);
355 }

357 void
358 fsh_fs_disable(vfs_t *vfsp)
359 {
360     fsh_prepare_fsrec(vfsp);

362     rw_enter(&vfsp->vfs_fshrecord->fshfsr_lock, RW_WRITER);
363     vfsp->vfs_fshrecord->fshfsr_enabled = 0;
364     rw_exit(&vfsp->vfs_fshrecord->fshfsr_lock);
365 }

367 /*
368  * API used for installing hooks. fsh_handle_t is returned for further
369  * actions (currently just removing) on this set of hooks.
370  *
371  * fsh_t fields:
372  * - arg - argument passed to every hook
373  * - read, write, ... - pointers to hooks for corresponding vnodeops/vfsops;
374  *   if there is no hook desired for an operation, it should be set to
375  *   NULL
376  *
377  * It's important that the hooks are executed in LIFO installation order (they
378  * are added to the head of the hook list).
379  *
380  * The caller is expected to hold the vfs_t.
381  *
382  * This function must NOT be called in a hook.
383  *
384  * Returns (-1) if hook/callback limit exceeded, handle otherwise.
385  */
386 fsh_handle_t
387 fsh_hook_install(vfs_t *vfsp, fsh_t *hooks)
388 {
389     fsh_handle_t    handle;
390     fsh_int_t       *fshi;
391     fsh_mapping_t   *mapping;

```

```

393     fsh_prepare_fsrec(vfsp);
395     if ((handle = id_alloc(fsh_idspace)) == -1)
396         return (-1);
398     fshi = kmem_alloc(sizeof (*fshi), KM_SLEEP);
399     (void) memcpy(&fshi->fshi_hooks, hooks, sizeof (fshi->fshi_hooks));
400     fshi->fshi_handle = handle;
402     /*
403     * Although we lock vfs_fshrecord and fsh_map in this function,
404     * there is no need to use fsh_remove_lock. Since it is expected that a
405     * vfs_t is held across this call, fsh_fsrec_destroy() cannot be
406     * executing concurrently. fsh_hook_remove() cannot be called for this
407     * hook set, because a handle passed to that function doesn't yet exist.
408     */
410     /* If it is called inside of a hook, causes deadlock. */
411     rw_enter(&vfsp->vfs_fshrecord->fshfsr_lock, RW_WRITER);
412     list_insert_head(&vfsp->vfs_fshrecord->fshfsr_list, fshi);
413     rw_exit(&vfsp->vfs_fshrecord->fshfsr_lock);
415     mapping = kmem_alloc(sizeof (*mapping), KM_SLEEP);
416     mapping->fshm_handle = handle;
417     mapping->fshm_vfsp = vfsp;
418     mapping->fshm_fshi = fshi;
419     fshi->fshi_mapping = mapping;
421     mutex_enter(&fsh_map_lock);
422     list_insert_head(&fsh_map, mapping);
423     mutex_exit(&fsh_map_lock);
425     return (handle);
426 }
428 /*
429 * Used for removing a hook set.
430 *
431 * This function must NOT be called in a hook.
432 *
433 * Returns (-1) if hook wasn't found, 0 otherwise.
434 */
435 int
436 fsh_hook_remove(fsh_handle_t handle)
437 {
438     fsh_fsrecord_t *fsrecp;
439     fsh_mapping_t *mapping;
441     /* For more info about the fsh_remove_lock, see fsh_fsrec_destroy() */
442     mutex_enter(&fsh_remove_lock);
443     mutex_enter(&fsh_map_lock);
444     for (mapping = list_head(&fsh_map); mapping != NULL;
445          mapping = list_next(&fsh_map, mapping)) {
446         if (mapping->fshm_handle == handle) {
447             list_remove(&fsh_map, mapping);
448             break;
449         }
450     }
451     mutex_exit(&fsh_map_lock);
452     if (mapping == NULL) {
453         mutex_exit(&fsh_remove_lock);
454         return (-1);
455     }
457     ASSERT(mapping->fshm_fshi->fshi_handle == handle);

```

```

459     /*
460     * We don't have to call fsh_prepare_fsrec() here. fsh_fsrecord_t
461     * is already initialised, because we've found a mapping for the given
462     * handle. We instead make two ASSERTs.
463     */
464     fsrecp = mapping->fshm_vfsp->vfs_fshrecord;
465     ASSERT(fsrecp != NULL);
466     ASSERT(fsrecp != fsh_res_ptr);
468     /* If it is called inside of a hook, causes deadlock. */
469     rw_enter(&fsrecp->fshfsr_lock, RW_WRITER);
470     list_remove(&fsrecp->fshfsr_list, mapping->fshm_fshi);
471     rw_exit(&fsrecp->fshfsr_lock);
473     mutex_exit(&fsh_remove_lock);
475     id_free(fsh_idspace, handle);
477     kmem_free(mapping->fshm_fshi, sizeof (*mapping->fshm_fshi));
478     kmem_free(mapping, sizeof (*mapping));
480     return (0);
481 }
483 /*
484 * API for installing global mount/free callbacks.
485 *
486 * fsh_callback_t fields:
487 * fshc_arg - argument passed to the callbacks
488 * fshc_free - callback fired before VFS_FREEVFS() is called, after vfs_count
489 *             drops to 0
490 * fshc_mount - callback fired right before returning from domount()
491 * The first argument of these callbacks is the vfs_t that is mounted/freed.
492 * The second one is the fshc_arg.
493 *
494 * fsh_callback_handle_t is filled out by this function.
495 *
496 * This function must NOT be called in a callback, because it will cause
497 * a deadlock.
498 *
499 * Returns (-1) if hook/callback limit exceeded.
500 */
501 fsh_callback_handle_t
502 fsh_callback_install(fsh_callback_t *callback)
503 {
504     fsh_callback_int_t *fshci;
505     fsh_callback_handle_t handle;
507     if ((handle = id_alloc(fsh_idspace)) == -1)
508         return (-1);
510     fshci = (fsh_callback_int_t *)kmem_alloc(sizeof (*fshci), KM_SLEEP);
511     (void) memcpy(&fshci->fshci_cb, callback, sizeof (fshci->fshci_cb));
512     fshci->fshci_handle = handle;
514     /* If it is called in a {mount,free} callback, causes deadlock. */
515     rw_enter(&fsh_cblst_lock, RW_WRITER);
516     list_insert_head(&fsh_cblst, fshci);
517     rw_exit(&fsh_cblst_lock);
519     return (handle);
520 }
522 /*
523 * API for removing global mount/free callbacks.

```

```

524 *
525 * This function must NOT be called in a callback, because it will cause
526 * a deadlock.
527 *
528 * Returns (-1) if callback wasn't found, 0 otherwise.
529 */
530 int
531 fsh_callback_remove(fsh_callback_handle_t handle)
532 {
533     fsh_callback_int_t *fshci;
534
535     /* If it is called in a {mount,free} callback, causes deadlock. */
536     rw_enter(&fsh_cblst_lock, RW_WRITER);
537     for (fshci = list_head(&fsh_cblst); fshci != NULL;
538         fshci = list_next(&fsh_cblst, fshci)) {
539         if (fshci->fshci_handle == handle) {
540             list_remove(&fsh_cblst, fshci);
541             break;
542         }
543     }
544     rw_exit(&fsh_cblst_lock);
545
546     if (fshci == NULL)
547         return (-1);
548
549     kmem_free(fshci, sizeof (*fshci));
550     id_free(fsh_idspace, handle);
551
552     return (0);
553 }
554
555 /*
556 * This function is executed right before returning from domount()@vfs.c.
557 * We are sure that it's called only after fsh_init().
558 * It executes all the mount callbacks installed in the fsh.
559 *
560 * Since fsh_exec_mount_callbacks() is called only inside domount(), it is legal
561 * to call fsh_hook_{install,remove}() inside a mount callback WITHOUT holding
562 * this vfs_t. This guarantee should be preserved, because it's in the "Usage"
563 * section in the big theory statement at the top of this file.
564 */
565 void
566 fsh_exec_mount_callbacks(vfs_t *vfsp)
567 {
568     fsh_callback_int_t *fshci;
569     fsh_callback_t *cb;
570
571     rw_enter(&fsh_cblst_lock, RW_READER);
572     for (fshci = list_head(&fsh_cblst); fshci != NULL;
573         fshci = list_next(&fsh_cblst, fshci)) {
574         cb = &fshci->fshci_cb;
575         if (cb->fshc_mount != NULL)
576             (*(cb->fshc_mount))(vfsp, cb->fshc_arg);
577     }
578     rw_exit(&fsh_cblst_lock);
579 }
580
581 /*
582 * This function is executed right before VFS_FREEVFS() is called in
583 * vfs_rele()@vfs.c. We are sure that it's called only after fsh_init().
584 * It executes all the free callbacks installed in the fsh.
585 *
586 * free() callback is the point where the handles associated with the hooks
587 * installed on this vfs_t become invalid
588 */
589 void

```

```

590 fsh_exec_free_callbacks(vfs_t *vfsp)
591 {
592     fsh_callback_int_t *fshci;
593     fsh_callback_t *cb;
594
595     rw_enter(&fsh_cblst_lock, RW_READER);
596     for (fshci = list_head(&fsh_cblst); fshci != NULL;
597         fshci = list_next(&fsh_cblst, fshci)) {
598         cb = &fshci->fshci_cb;
599         if (cb->fshc_free != NULL)
600             (*(cb->fshc_free))(vfsp, cb->fshc_arg);
601     }
602     rw_exit(&fsh_cblst_lock);
603 }
604
605 /*
606 * API for vnode.c/vfs.c to start executing the fsh for a given operation.
607 *
608 * These interfaces are using fsh_res_ptr (in fsh_prepare_fsrec()), so it's
609 * absolutely necessary to call fsh_init() before using them. That's done in
610 * vfsinit().
611 *
612 * While these functions are executing, it's expected that necessary vfs_t's
613 * are held so that vfs_free() isn't called. vfs_free() expects that noone
614 * else accesses vfs_fshrecord of a given vfs_t.
615 * It's also the caller responsibility to keep vnode_t passed to fsh_foo()
616 * alive and valid.
617 * All these expectations are met because these functions are used only in
618 * corresponding {fop,fsop}_foo() functions.
619 */
620 int
621 fsh_read(vnode_t *vp, uio_t *uiop, int ioflag, cred_t *cr,
622         caller_context_t *ct)
623 {
624     int ret;
625     fsh_fsrecord_t *fsrecp;
626
627     fsh_prepare_fsrec(vp->v_vfsp);
628     fsrecp = vp->v_vfsp->vfs_fshrecord;
629
630     rw_enter(&fsrecp->fshfsr_lock, RW_READER);
631     if (!(fsrecp->fshfsr_enabled)) {
632         rw_exit(&fsrecp->fshfsr_lock);
633         return ((*vp->v_op->vop_read)(vp, uiop, ioflag, cr, ct));
634     }
635
636     ret = fsh_next_read(list_head(&fsrecp->fshfsr_list), vp, uiop, ioflag,
637                         cr, ct);
638     rw_exit(&fsrecp->fshfsr_lock);
639
640     return (ret);
641 }
642
643 int
644 fsh_write(vnode_t *vp, uio_t *uiop, int ioflag, cred_t *cr,
645         caller_context_t *ct)
646 {
647     int ret;
648     fsh_fsrecord_t *fsrecp;
649
650     fsh_prepare_fsrec(vp->v_vfsp);
651     fsrecp = vp->v_vfsp->vfs_fshrecord;
652
653     rw_enter(&fsrecp->fshfsr_lock, RW_READER);
654     if (!(vp->v_vfsp->vfs_fshrecord->fshfsr_enabled)) {
655         rw_exit(&fsrecp->fshfsr_lock);

```

```

656         return ((*vp->v_op->vop_write))(vp, uiop, ioflag, cr, ct);
657     }

659     ret = fsh_next_write(list_head(&fsrecp->fshfsr_list), vp, uiop, ioflag,
660                          cr, ct);
661     rw_exit(&fsrecp->fshfsr_lock);

663     return (ret);
664 }

666 int
667 fsh_mount(vfs_t *vfsp, vnode_t *mvp, struct mounta *uap, cred_t *cr)
668 {
669     fsh_fsrecord_t *fsrecp;
670     int ret;

672     fsh_prepare_fsrec(vfsp);
673     fsrecp = vfsp->vfs_fshrecord;

675     rw_enter(&fsrecp->fshfsr_lock, RW_READER);
676     if (!(fsrecp->fshfsr_enabled)) {
677         rw_exit(&fsrecp->fshfsr_lock);
678         return ((*vfsp->vfs_op->vfs_mount))(vfsp, mvp, uap, cr);
679     }

681     ret = fsh_next_mount(list_head(&fsrecp->fshfsr_list), vfsp, mvp, uap,
682                          cr);
683     rw_exit(&fsrecp->fshfsr_lock);

685     return (ret);
686 }

688 int
689 fsh_unmount(vfs_t *vfsp, int flag, cred_t *cr)
690 {
691     fsh_fsrecord_t *fsrecp;
692     int ret;

694     fsh_prepare_fsrec(vfsp);
695     fsrecp = vfsp->vfs_fshrecord;

697     rw_enter(&fsrecp->fshfsr_lock, RW_READER);
698     if (!(fsrecp->fshfsr_enabled)) {
699         rw_exit(&fsrecp->fshfsr_lock);
700         return ((*vfsp->vfs_op->vfs_unmount))(vfsp, flag, cr);
701     }

703     ret = fsh_next_unmount(list_head(&fsrecp->fshfsr_list), vfsp, flag, cr);
704     rw_exit(&fsrecp->fshfsr_lock);

706     return (ret);
707 }

709 /*
710  * This is the funtion used by fsh_prepare_fsrec() to allocate a new
711  * fsh_fsrecord. This function is called by the first function which
712  * access the vfs_fshrecord and finds out it's NULL.
713  */
714 static fsh_fsrecord_t *
715 fsh_fsrec_create()
716 {
717     fsh_fsrecord_t *fsrecp;

719     fsrecp = (fsh_fsrecord_t *)kmem_zalloc(sizeof (*fsrecp), KM_SLEEP);
720     list_create(&fsrecp->fshfsr_list, sizeof (fsh_int_t),
721               offsetof(fsh_int_t, fshi_next));

```

```

722     rw_init(&fsrecp->fshfsr_lock, NULL, RW_DRIVER, NULL);
723     fsrecp->fshfsr_enabled = 1;
724     return (fsrecp);
725 }

728 /*
729  * This call can be used ONLY in vfs_free(). It's assumed that no other
730  * fsh calls using the vfs_t that owns the fsh_fsrecord to be destroyed
731  * are executing while a call to fsh_fsrec_destroy() is made. With this
732  * assumptions, no concurrency issues occur.
733  *
734  * Before calling this function outside the fsh, it's sufficient and
735  * required to check if the passed fsh_fsrecord * is not NULL. We don't
736  * have to check if it is not equal to fsh_res_ptr, because all the fsh API
737  * calls involving this vfs_t should end before vfs_free() is called
738  * (outside the fsh, fsh_fsrecord is never equal to fsh_res_ptr). That is
739  * guaranteed by the explicit requirement that the caller of fsh API holds
740  * the vfs_t when needed.
741  *
742  * All the remaining hooks are being removed.
743  */
744 void
745 fsh_fsrec_destroy(struct fsh_fsrecord *volatile fsrecp)
746 {
747     fsh_int_t *fshi;

749     VERIFY(fsrecp != NULL);

751     /*
752      * Although it is expected that no fsh calls using this vfs_t
753      * would be executing, we would not like to panic if that happens.
754      * That's a client's bug obviously, but we'd like to minimise the
755      * possibility of a system crash. That's why fsh_remove_lock is used
756      * here. Even if fsh_hook_remove() would be called with a handle that is
757      * invalid, it will just return (-1), because it won't find the handle
758      * in fsh_map. here is of course a possibility, that an invalid handle
759      * (id_t internally) would become valid (it would be assigned again),
760      * but this is very rare, because of the fact that id_alloc() uses next
761      * fit strategy to alloc the id_t's.
762      * For more info see fsh_hook_remove()
763      */
764     mutex_enter(&fsh_remove_lock);
765     while ((fshi = list_remove_head(&fsrecp->fshfsr_list)) != NULL) {
766         fsh_mapping_t *mapping = fshi->fshi_mapping;

768         ASSERT(mapping->fshm_handle == fshi->fshi_handle);
769         ASSERT(mapping->fshm_fshi == fshi);

771         mutex_enter(&fsh_map_lock);
772         list_remove(&fsh_map, mapping);
773         mutex_exit(&fsh_map_lock);

775         id_free(fsh_idspace, fshi->fshi_handle);

777         kmem_free(fshi, sizeof (*fshi));
778         kmem_free(mapping, sizeof (*mapping));
779     }
780     mutex_exit(&fsh_remove_lock);

782     list_destroy(&fsrecp->fshfsr_list);
783     rw_destroy(&fsrecp->fshfsr_lock);
784     kmem_free(fsrecp, sizeof (*fsrecp));
785 }

787 /*

```

```

788 * fsh_init() is called in vfsinit()@vfs.c. This function MUST be called
789 * before every other fsh call.
790 */
791 void
792 fsh_init(void)
793 {
794     rw_init(&fsh_cblst_lock, NULL, RW_DRIVER, NULL);
795     list_create(&fsh_cblst, sizeof (fsh_callback_int_t),
796               offsetof(fsh_callback_int_t, fshci_next));
797
798     mutex_init(&fsh_map_lock, NULL, MUTEX_DRIVER, NULL);
799     mutex_init(&fsh_remove_lock, NULL, MUTEX_DRIVER, NULL);
800
801     list_create(&fsh_map, sizeof (fsh_mapping_t),
802               offsetof(fsh_mapping_t, fshm_next));
803
804     /* See comment above fsh_prepare_fsrec() */
805     fsh_res_ptr = (void *)-1;
806
807     fsh_idspace = id_space_create("fsh", 0, fsh_limit);
808 }
809
810 /*
811 * These functions are used to pass control to the next hook or underlying
812 * vop or vfsop. It's client doesn't have to worry about any locking, because
813 * all the necessities are guaranteed by the fsh_foo().
814 *
815 * In fsh_next_foo() we execute the hook passed in the first argument and
816 * try to find the next one. It is guaranteed that the passed hook is still
817 * valid, because of fshfsr_lock held by fsh_foo().
818 */
819 int
820 fsh_next_read(fsh_int_t *fshi, vnode_t *vp, uio_t *uiop, int ioflag,
821              cred_t *cr, caller_context_t *ct)
822 {
823     while (fshi != NULL && fshi->fshi_hooks.read == NULL)
824         fshi = list_next(&vp->v_vfsp->vfs_fshrecord->fshfsr_list,
825                          fshi);
826
827     if (fshi != NULL)
828         return ((*fshi->fshi_hooks.read)(
829             list_next(&vp->v_vfsp->vfs_fshrecord->fshfsr_list, fshi),
830             fshi->fshi_hooks.arg, vp, uiop, ioflag, cr, ct));
831     else
832         return ((*vp->v_op->vop_read)(vp, uiop, ioflag, cr, ct));
833 }
834
835 int
836 fsh_next_write(fsh_int_t *fshi, vnode_t *vp, uio_t *uiop, int ioflag,
837               cred_t *cr, caller_context_t *ct)
838 {
839     while (fshi != NULL && fshi->fshi_hooks.write == NULL)
840         fshi = list_next(&vp->v_vfsp->vfs_fshrecord->fshfsr_list,
841                          fshi);
842
843     if (fshi != NULL)
844         return ((*fshi->fshi_hooks.write)(
845             list_next(&vp->v_vfsp->vfs_fshrecord->fshfsr_list, fshi),
846             fshi->fshi_hooks.arg, vp, uiop, ioflag, cr, ct));
847     else
848         return ((*vp->v_op->vop_write)(vp, uiop, ioflag, cr, ct));
849 }
850
851 int
852 fsh_next_mount(fsh_int_t *fshi, vfs_t *vfsp, vnode_t *mvp, struct mounta *uap,
853               cred_t *cr)

```

```

854 {
855     while (fshi != NULL && fshi->fshi_hooks.mount == NULL)
856         fshi = list_next(&vfsp->vfs_fshrecord->fshfsr_list, fshi);
857
858     if (fshi != NULL)
859         return ((*fshi->fshi_hooks.mount)(
860             list_next(&vfsp->vfs_fshrecord->fshfsr_list, fshi),
861             fshi->fshi_hooks.arg, vfsp, mvp, uap, cr));
862     else
863         return ((*vfsp->vfs_op->vfs_mount)(vfsp, mvp, uap, cr));
864 }
865
866 int
867 fsh_next_unmount(fsh_int_t *fshi, vfs_t *vfsp, int flag, cred_t *cr)
868 {
869     while (fshi != NULL && fshi->fshi_hooks.unmount == NULL)
870         fshi = list_next(&vfsp->vfs_fshrecord->fshfsr_list, fshi);
871
872     if (fshi != NULL)
873         return ((*fshi->fshi_hooks.unmount)(
874             list_next(&vfsp->vfs_fshrecord->fshfsr_list, fshi),
875             fshi->fshi_hooks.arg, vfsp, flag, cr));
876     else
877         return ((*vfsp->vfs_op->vfs_unmount)(vfsp, flag, cr));
878 }
879 }

```

new/usr/src/uts/common/fs/vfs.c

1

```
*****
118568 Sat Aug 24 02:37:58 2013
new/usr/src/uts/common/fs/vfs.c
fsh webrev (August 24th)
*****
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) 1988, 2010, Oracle and/or its affiliates. All rights reserved.
23 */

25 /*      Copyright (c) 1983, 1984, 1985, 1986, 1987, 1988, 1989 AT&T      */
26 /*      All Rights Reserved      */

28 /*
29 * University Copyright- Copyright (c) 1982, 1986, 1988
30 * The Regents of the University of California
31 * All Rights Reserved
32 *
33 * University Acknowledgment- Portions of this document are derived from
34 * software developed by the University of California, Berkeley, and its
35 * contributors.
36 */

38 #include <sys/types.h>
39 #include <sys/t_lock.h>
40 #include <sys/param.h>
41 #include <sys/errno.h>
42 #include <sys/user.h>
43 #include <sys/fstyp.h>
44 #include <sys/kmem.h>
45 #include <sys/system.h>
46 #include <sys/proc.h>
47 #include <sys/mount.h>
48 #include <sys/vfs.h>
49 #include <sys/vfs_opreg.h>
50 #include <sys/fem.h>
51 #include <sys/mntent.h>
52 #include <sys/stat.h>
53 #include <sys/statvfs.h>
54 #include <sys/statfs.h>
55 #include <sys/cred.h>
56 #include <sys/vnode.h>
57 #include <sys/rwstlock.h>
58 #include <sys/dnld.h>
59 #include <sys/file.h>
60 #include <sys/time.h>
61 #include <sys/atomic.h>
```

new/usr/src/uts/common/fs/vfs.c

2

```
62 #include <sys/cmn_err.h>
63 #include <sys/buf.h>
64 #include <sys/swap.h>
65 #include <sys/debug.h>
66 #include <sys/vnode.h>
67 #include <sys/modctl.h>
68 #include <sys/ddi.h>
69 #include <sys/pathname.h>
70 #include <sys/bootconf.h>
71 #include <sys/dumphdr.h>
72 #include <sys/dc_ki.h>
73 #include <sys/poll.h>
74 #include <sys/sunddi.h>
75 #include <sys/sysmacros.h>
76 #include <sys/zone.h>
77 #include <sys/policy.h>
78 #include <sys/ctfs.h>
79 #include <sys/objfs.h>
80 #include <sys/console.h>
81 #include <sys/reboot.h>
82 #include <sys/attr.h>
83 #include <sys/zio.h>
84 #include <sys/spa.h>
85 #include <sys/lofi.h>
86 #include <sys/bootprops.h>
87 #include <sys/fsh.h>
88 #include <sys/fsh_impl.h>

90 #include <vm/page.h>

92 #include <fs/fs_subr.h>
93 /* Private interfaces to create vopstats-related data structures */
94 extern void initialize_vopstats(vopstats_t *);
95 extern vopstats_t *get_fstype_vopstats(struct vfs *, struct vfssw *);
96 extern vsk_anchor_t *get_vskstat_anchor(struct vfs *);

98 static void vfs_clearmntopt_nolock(mntopts_t *, const char *, int);
99 static void vfs_setmntopt_nolock(mntopts_t *, const char *,
100 const char *, int, int);
101 static int vfs_optionisset_nolock(const mntopts_t *, const char *, char **);
102 static void vfs_freemnttab(struct vfs *);
103 static void vfs_freeopt(mntopt_t *);
104 static void vfs_swapopttbl_nolock(mntopts_t *, mntopts_t *);
105 static void vfs_swapopttbl(mntopts_t *, mntopts_t *);
106 static void vfs_copyopttbl_extend(const mntopts_t *, mntopts_t *, int);
107 static void vfs_createopttbl_extend(mntopts_t *, const char *,
108 const mntopts_t *);
109 static char **vfs_copycancelopt_extend(char **const, int);
110 static void vfs_freecancelopt(char **);
111 static void getrootfs(char **, char **);
112 static int getmacpath(dev_info_t *, void *);
113 static void vfs_mnttabvop_setup(void);

115 struct ipmnt {
116     struct ipmnt *mip_next;
117     dev_t mip_dev;
118     struct vfs *mip_vfsp;
119 };

----- unchanged portion omitted -----

214 /*
215  * File system operation dispatch functions.
216  */

218 int
219 fsop_mount(vfs_t *vfsp, vnode_t *mvp, struct mounta *uap, cred_t *cr)
```



```

220 {
221     return (fsh_mount(vfsp, mvp, uap, cr));
219     return (*(vfs_op->vfs_mount)(vfs, mvp, uap, cr);
222 }

224 int
225 fsop_unmount(vfs_t *vfs, int flag, cred_t *cr)
226 {
227     return (fsh_unmount(vfsp, flag, cr));
225     return (*(vfs_op->vfs_unmount)(vfs, flag, cr);
228 }

    unchanged_portion_omitted

1085 /*
1086 * Common mount code. Called from the system call entry point, from autofs,
1087 * nfsv4 trigger mounts, and from pxfs.
1088 *
1089 * Takes the effective file system type, mount arguments, the mount point
1090 * vnode, flags specifying whether the mount is a remount and whether it
1091 * should be entered into the vfs list, and credentials. Fills in its vfssp
1092 * parameter with the mounted file system instance's vfs.
1093 *
1094 * Note that the effective file system type is specified as a string. It may
1095 * be null, in which case it's determined from the mount arguments, and may
1096 * differ from the type specified in the mount arguments; this is a hook to
1097 * allow interposition when instantiating file system instances.
1098 *
1099 * The caller is responsible for releasing its own hold on the mount point
1100 * vp (this routine does its own hold when necessary).
1101 * Also note that for remounts, the mount point vp should be the vnode for
1102 * the root of the file system rather than the vnode that the file system
1103 * is mounted on top of.
1104 */
1105 int
1106 domount(char *fsname, struct mounta *uap, vnode_t *vp, struct cred *credp,
1107         struct vfs **vfssp)
1108 {
1109     struct vfsw       *vswp;
1110     vfsops_t         *vfops;
1111     struct vfs       *vfs;
1112     struct vnode     *bvp;
1113     dev_t            bdev = 0;
1114     mntopts_t        mntopts;
1115     int              error = 0;
1116     int              copyout_error = 0;
1117     int              ovflags;
1118     char             *opts = uap->optptr;
1119     char             *inargs = opts;
1120     int              optlen = uap->optlen;
1121     int              remount;
1122     int              rdonly;
1123     int              nbmand = 0;
1124     int              delmip = 0;
1125     int              addmip = 0;
1126     int              splice = ((uap->flags & MS_NOSPLICE) == 0);
1127     int              fromspace = (uap->flags & MS_SYSSPACE) ?
1128         UIO_SYSSPACE : UIO_USERSPACE;
1129     char             *resource = NULL, *mountpt = NULL;
1130     refstr_t         *oldresource, *oldmntpt;
1131     struct pathname  pn, rpn;
1132     vsk_anchor_t     *vskap;
1133     char             fstname[FSTYPSZ];

1135 /*
1136 * The v_flag value for the mount point vp is permanently set
1137 * to VVFSLOCK so that no one bypasses the vn_vfs*locks routine

```

```

1138     * for mount point locking.
1139     */
1140     mutex_enter(&vp->v_lock);
1141     vp->v_flag |= VVFSLOCK;
1142     mutex_exit(&vp->v_lock);

1144     mnt_mntopts.mo_count = 0;
1145     /*
1146     * Find the ops vector to use to invoke the file system-specific mount
1147     * method. If the fsname argument is non-NULL, use it directly.
1148     * Otherwise, dig the file system type information out of the mount
1149     * arguments.
1150     *
1151     * A side effect is to hold the vfsw entry.
1152     *
1153     * Mount arguments can be specified in several ways, which are
1154     * distinguished by flag bit settings. The preferred way is to set
1155     * MS_OPTIONSTR, indicating an 8 argument mount with the file system
1156     * type supplied as a character string and the last two arguments
1157     * being a pointer to a character buffer and the size of the buffer.
1158     * On entry, the buffer holds a null terminated list of options; on
1159     * return, the string is the list of options the file system
1160     * recognized. If MS_DATA is set arguments five and six point to a
1161     * block of binary data which the file system interprets.
1162     * A further wrinkle is that some callers don't set MS_FSS and MS_DATA
1163     * consistently with these conventions. To handle them, we check to
1164     * see whether the pointer to the file system name has a numeric value
1165     * less than 256. If so, we treat it as an index.
1166     */
1167     if (fsname != NULL) {
1168         if ((vswp = vfs_getvfsw(fsname)) == NULL) {
1169             return (EINVAL);
1170         }
1171     } else if (uap->flags & (MS_OPTIONSTR | MS_DATA | MS_FSS)) {
1172         size_t n;
1173         uint_t fstype;

1175         fsname = fstname;

1177         if ((fstype = (uintptr_t)uap->fstype) < 256) {
1178             RLOCK_VFSSW();
1179             if (fstype == 0 || fstype >= nfstype ||
1180                 !ALLOCATED_VFSSW(&vfsw[fstype])) {
1181                 RUNLOCK_VFSSW();
1182                 return (EINVAL);
1183             }
1184             (void) strcpy(fsname, vfsw[fstype].vsw_name);
1185             RUNLOCK_VFSSW();
1186             if ((vswp = vfs_getvfsw(fsname)) == NULL)
1187                 return (EINVAL);
1188         } else {
1189             /*
1190             * Handle either kernel or user address space.
1191             */
1192             if (uap->flags & MS_SYSSPACE) {
1193                 error = copystr(uap->fstype, fsname,
1194                               FSTYPSZ, &n);
1195             } else {
1196                 error = copyinstr(uap->fstype, fsname,
1197                                 FSTYPSZ, &n);
1198             }
1199             if (error) {
1200                 if (error == ENAMETOOLONG)
1201                     return (EINVAL);
1202                 return (error);
1203             }

```

```

1204         if ((vswp = vfs_getvfsw(fsname)) == NULL)
1205             return (EINVAL);
1206     }
1207 } else {
1208     if ((vswp = vfs_getvfswbyvfsops(vfs_getops(rootvfs))) == NULL)
1209         return (EINVAL);
1210     fsname = vswp->vsw_name;
1211 }
1212 if (!VFS_INSTALLED(vswp))
1213     return (EINVAL);
1214
1215 if ((error = secpolicy_fs_allowed_mount(fsname)) != 0) {
1216     vfs_unrefvfsw(vswp);
1217     return (error);
1218 }
1219
1220 vfsops = &vswp->vsw_vfsops;
1221
1222 vfs_copyopttbl(&vswp->vsw_optproto, &mnt_mntopts);
1223 /*
1224  * Fetch mount options and parse them for generic vfs options
1225  */
1226 if (uap->flags & MS_OPTIONSTR) {
1227     /*
1228      * Limit the buffer size
1229      */
1230     if (optlen < 0 || optlen > MAX_MNTOPT_STR) {
1231         error = EINVAL;
1232         goto errout;
1233     }
1234     if ((uap->flags & MS_SYSSPACE) == 0) {
1235         inargs = kmem_alloc(MAX_MNTOPT_STR, KM_SLEEP);
1236         inargs[0] = '\0';
1237         if (optlen) {
1238             error = copyinstr(opts, inargs, (size_t)optlen,
1239                 NULL);
1240             if (error) {
1241                 goto errout;
1242             }
1243         }
1244     }
1245     vfs_parsemntopts(&mnt_mntopts, inargs, 0);
1246 }
1247 /*
1248  * Flag bits override the options string.
1249  */
1250 if (uap->flags & MS_REMOUNT)
1251     vfs_setmntopt_nolock(&mnt_mntopts, MNTOPT_REMOUNT, NULL, 0, 0);
1252 if (uap->flags & MS_RDONLY)
1253     vfs_setmntopt_nolock(&mnt_mntopts, MNTOPT_RO, NULL, 0, 0);
1254 if (uap->flags & MS_NOSUID)
1255     vfs_setmntopt_nolock(&mnt_mntopts, MNTOPT_NOSUID, NULL, 0, 0);
1256
1257 /*
1258  * Check if this is a remount; must be set in the option string and
1259  * the file system must support a remount option.
1260  */
1261 if (remount = vfs_optionisset_nolock(&mnt_mntopts,
1262     MNTOPT_REMOUNT, NULL)) {
1263     if (!(vswp->vsw_flag & VSW_CANREMOUNT)) {
1264         error = ENOTSUP;
1265         goto errout;
1266     }
1267     uap->flags |= MS_REMOUNT;
1268 }

```

```

1270     /*
1271      * uap->flags and vfs_optionisset() should agree.
1272      */
1273     if (rdonly = vfs_optionisset_nolock(&mnt_mntopts, MNTOPT_RO, NULL)) {
1274         uap->flags |= MS_RDONLY;
1275     }
1276     if (vfs_optionisset_nolock(&mnt_mntopts, MNTOPT_NOSUID, NULL)) {
1277         uap->flags |= MS_NOSUID;
1278     }
1279     nbmand = vfs_optionisset_nolock(&mnt_mntopts, MNTOPT_NBMAND, NULL);
1280     ASSERT(splice || !remount);
1281     /*
1282      * If we are splicing the fs into the namespace,
1283      * perform mount point checks.
1284      *
1285      * We want to resolve the path for the mount point to eliminate
1286      * '.' and '..' and symlinks in mount points; we can't do the
1287      * same for the resource string, since it would turn
1288      * "/dev/dsk/c0t0d0s0" into "/devices/pci@...". We need to do
1289      * this before grabbing vn_vfswlock(), because otherwise we
1290      * would deadlock with lookuppn().
1291      */
1292     if (splice) {
1293         ASSERT(vp->v_count > 0);
1294
1295         /*
1296          * Pick up mount point and device from appropriate space.
1297          */
1298         if (pn_get(uap->spec, fromspace, &pn) == 0) {
1299             resource = kmem_alloc(pn.pn_pathlen + 1,
1300                 KM_SLEEP);
1301             (void) strcpy(resource, pn.pn_path);
1302             pn_free(&pn);
1303         }
1304         /*
1305          * Do a lookupname prior to taking the
1306          * writelock. Mark this as completed if
1307          * successful for later cleanup and addition to
1308          * the mount in progress table.
1309          */
1310         if ((uap->flags & MS_GLOBAL) == 0 &&
1311             lookupname(uap->spec, fromspace,
1312                 FOLLOW, NULL, &bpv) == 0) {
1313             addmip = 1;
1314         }
1315
1316         if ((error = pn_get(uap->dir, fromspace, &pn)) == 0) {
1317             pathname_t *pnp;
1318
1319             if (*pn.pn_path != '/') {
1320                 error = EINVAL;
1321                 pn_free(&pn);
1322                 goto errout;
1323             }
1324             pn_alloc(&rpn);
1325             /*
1326              * Kludge to prevent autofs from deadlocking with
1327              * itself when it calls domount().
1328              *
1329              * If autofs is calling, it is because it is doing
1330              * (autofs) mounts in the process of an NFS mount. A
1331              * lookuppn() here would cause us to block waiting for
1332              * said NFS mount to complete, which can't since this
1333              * is the thread that was supposed to do it.
1334              */
1335             if (fromspace == UIO_USERSPACE) {

```

```

1336         if ((error = lookuppn(&pn, &rp, FOLLOW, NULL,
1337             NULL)) == 0) {
1338             pnp = &rp;
1339         } else {
1340             /*
1341              * The file disappeared or otherwise
1342              * became inaccessible since we opened
1343              * it; might as well fail the mount
1344              * since the mount point is no longer
1345              * accessible.
1346              */
1347             pn_free(&rp);
1348             pn_free(&pn);
1349             goto errout;
1350         }
1351     } else {
1352         pnp = &pn;
1353     }
1354     mountpt = kmem_alloc(pnp->pn_pathlen + 1, KM_SLEEP);
1355     (void) strcpy(mountpt, pnp->pn_path);
1356
1357     /*
1358      * If the addition of the zone's rootpath
1359      * would push us over a total path length
1360      * of MAXPATHLEN, we fail the mount with
1361      * ENAMETOOLONG, which is what we would have
1362      * gotten if we were trying to perform the same
1363      * mount in the global zone.
1364      *
1365      * strlen() doesn't count the trailing
1366      * '\0', but zone_rootpathlen counts both a
1367      * trailing '/' and the terminating '\0'.
1368      */
1369     if (((curproc->p_zone->zone_rootpathlen - 1 +
1370         strlen(mountpt)) > MAXPATHLEN ||
1371         (resource != NULL &&
1372          (curproc->p_zone->zone_rootpathlen - 1 +
1373           strlen(resource)) > MAXPATHLEN)) {
1374         error = ENAMETOOLONG;
1375     }
1376
1377     pn_free(&rp);
1378     pn_free(&pn);
1379 }
1380
1381 if (error)
1382     goto errout;
1383
1384 /*
1385  * Prevent path name resolution from proceeding past
1386  * the mount point.
1387  */
1388 if (vn_vfswlock(vp) != 0) {
1389     error = EBUSY;
1390     goto errout;
1391 }
1392
1393 /*
1394  * Verify that it's legitimate to establish a mount on
1395  * the prospective mount point.
1396  */
1397 if (vn_mountedvfs(vp) != NULL) {
1398     /*
1399      * The mount point lock was obtained after some
1400      * other thread raced through and established a mount.
1401      */

```

```

1402         vn_vfsunlock(vp);
1403         error = EBUSY;
1404         goto errout;
1405     }
1406     if (vp->v_flag & VNOMOUNT) {
1407         vn_vfsunlock(vp);
1408         error = EINVAL;
1409         goto errout;
1410     }
1411 }
1412 if ((uap->flags & (MS_DATA | MS_OPTIONSTR)) == 0) {
1413     uap->dataptr = NULL;
1414     uap->datalen = 0;
1415 }
1416
1417 /*
1418  * If this is a remount, we don't want to create a new VFS.
1419  * Instead, we pass the existing one with a remount flag.
1420  */
1421 if (remount) {
1422     /*
1423      * Confirm that the mount point is the root vnode of the
1424      * file system that is being remounted.
1425      * This can happen if the user specifies a different
1426      * mount point directory pathname in the (re)mount command.
1427      *
1428      * Code below can only be reached if splice is true, so it's
1429      * safe to do vn_vfsunlock() here.
1430      */
1431     if ((vp->v_flag & VROOT) == 0) {
1432         vn_vfsunlock(vp);
1433         error = ENOENT;
1434         goto errout;
1435     }
1436     /*
1437      * Disallow making file systems read-only unless file system
1438      * explicitly allows it in its vfssw. Ignore other flags.
1439      */
1440     if (rdonly && vn_is_readonly(vp) == 0 &&
1441         (vswp->vsw_flag & VSW_CANRWRO) == 0) {
1442         vn_vfsunlock(vp);
1443         error = EINVAL;
1444         goto errout;
1445     }
1446     /*
1447      * Disallow changing the NBMAND disposition of the file
1448      * system on remounts.
1449      */
1450     if ((nbmand && ((vp->v_vfsp->vfs_flag & VFS_NBMAND) == 0)) ||
1451         (!nbmand && (vp->v_vfsp->vfs_flag & VFS_NBMAND))) {
1452         vn_vfsunlock(vp);
1453         error = EINVAL;
1454         goto errout;
1455     }
1456     vfsp = vp->v_vfsp;
1457     ovflags = vfsp->vfs_flag;
1458     vfsp->vfs_flag |= VFS_REMOUNT;
1459     vfsp->vfs_flag &= ~VFS_RDONLY;
1460 } else {
1461     vfsp = vfs_alloc(KM_SLEEP);
1462     VFS_INIT(vfsp, vfsops, NULL);
1463 }
1464
1465 VFS_HOLD(vfsp);
1466
1467 if ((error = lofi_add(fsname, vfsp, &mnt_mntopts, uap)) != 0) {

```

```

1468         if (!remount) {
1469             if (splice)
1470                 vn_vfsunlock(vp);
1471             vfs_free(vfsp);
1472         } else {
1473             vn_vfsunlock(vp);
1474             VFS_RELE(vfsp);
1475         }
1476         goto errout;
1477     }
1479     /*
1480     * PRIV_SYS_MOUNT doesn't mean you can become root.
1481     */
1482     if (vfsp->vfs_lofi_minor != 0) {
1483         uap->flags |= MS_NOSUID;
1484         vfs_setmntopt_nolock(&mnt_mntopts, MNTOPT_NOSUID, NULL, 0, 0);
1485     }
1487     /*
1488     * The vfs_reflock is not used anymore the code below explicitly
1489     * holds it preventing others accessing it directly.
1490     */
1491     if ((sema_tryv(&vfsp->vfs_reflock) == 0) &&
1492         !(vfsp->vfs_flag & VFS_REMOUNT))
1493         cmn_err(CE_WARN,
1494             "mount type %s couldn't get vfs_reflock", vswp->vsw_name);
1496     /*
1497     * Lock the vfs. If this is a remount we want to avoid spurious amount
1498     * failures that happen as a side-effect of fsflush() and other mount
1499     * and unmount operations that might be going on simultaneously and
1500     * may have locked the vfs currently. To not return EBUSY immediately
1501     * here we use vfs_lock_wait() instead vfs_lock() for the remount case.
1502     */
1503     if (!remount) {
1504         if (error = vfs_lock(vfsp)) {
1505             vfsp->vfs_flag = ovflags;
1507             lofi_remove(vfsp);
1509             if (splice)
1510                 vn_vfsunlock(vp);
1511             vfs_free(vfsp);
1512             goto errout;
1513         }
1514     } else {
1515         vfs_lock_wait(vfsp);
1516     }
1518     /*
1519     * Add device to mount in progress table, global mounts require special
1520     * handling. It is possible that we have already done the lookupname
1521     * on a spliced, non-global fs. If so, we don't want to do it again
1522     * since we cannot do a lookupname after taking the
1523     * wlock above. This case is for a non-spliced, non-global filesystem.
1524     */
1525     if (!addmip) {
1526         if ((uap->flags & MS_GLOBAL) == 0 &&
1527             lookupname(uap->spec, fromspace, FOLLOW, NULL, &bvp) == 0) {
1528             addmip = 1;
1529         }
1530     }
1532     if (addmip) {
1533         vnode_t *lvp = NULL;

```

```

1535         error = vfs_get_lofi(vfsp, &lvp);
1536         if (error > 0) {
1537             lofi_remove(vfsp);
1539             if (splice)
1540                 vn_vfsunlock(vp);
1541             vfs_unlock(vfsp);
1543             if (remount) {
1544                 VFS_RELE(vfsp);
1545             } else {
1546                 vfs_free(vfsp);
1547             }
1549             goto errout;
1550         } else if (error == -1) {
1551             bdev = bvp->v_rdev;
1552             VN_RELE(bvp);
1553         } else {
1554             bdev = lvp->v_rdev;
1555             VN_RELE(lvp);
1556             VN_RELE(bvp);
1557         }
1559         vfs_addmip(bdev, vfsp);
1560         addmip = 0;
1561         delmip = 1;
1562     }
1563     /*
1564     * Invalidate cached entry for the mount point.
1565     */
1566     if (splice)
1567         dnlc_purge_vp(vp);
1569     /*
1570     * If have an option string but the filesystem doesn't supply a
1571     * prototype options table, create a table with the global
1572     * options and sufficient room to accept all the options in the
1573     * string. Then parse the passed in option string
1574     * accepting all the options in the string. This gives us an
1575     * option table with all the proper cancel properties for the
1576     * global options.
1577     *
1578     * Filesystems that supply a prototype options table are handled
1579     * earlier in this function.
1580     */
1581     if (uap->flags & MS_OPTIONSTR) {
1582         if (!(vswp->vsw_flag & VSW_HASPROTO)) {
1583             mntopts_t tmp_mntopts;
1585             tmp_mntopts.mo_count = 0;
1586             vfs_createopttbl_extend(&tmp_mntopts, inargs,
1587                 &mnt_mntopts);
1588             vfs_parsemntopts(&tmp_mntopts, inargs, 1);
1589             vfs_swapopttbl_nolock(&mnt_mntopts, &tmp_mntopts);
1590             vfs_freeopttbl(&tmp_mntopts);
1591         }
1592     }
1594     /*
1595     * Serialize with zone creations.
1596     */
1597     mount_in_progress();
1598     /*
1599     * Instantiate (or reinstantiate) the file system. If appropriate,

```

```

1600     * splice it into the file system name space.
1601     *
1602     * We want VFS_MOUNT() to be able to override the vfs_resource
1603     * string if necessary (ie, mntfs), and also for a remount to
1604     * change the same (necessary when remounting '/' during boot).
1605     * So we set up vfs_mntpt and vfs_resource to what we think they
1606     * should be, then hand off control to VFS_MOUNT() which can
1607     * override this.
1608     *
1609     * For safety's sake, when changing vfs_resource or vfs_mntpt of
1610     * a vfs which is on the vfs list (i.e. during a remount), we must
1611     * never set those fields to NULL. Several bits of code make
1612     * assumptions that the fields are always valid.
1613     */
1614     vfs_swapopttbl(&mnt_mntopts, &vfsp->vfs_mntopts);
1615     if (remount) {
1616         if ((oldresource = vfsp->vfs_resource) != NULL)
1617             refstr_hold(oldresource);
1618         if ((oldmntpt = vfsp->vfs_mntpt) != NULL)
1619             refstr_hold(oldmntpt);
1620     }
1621     vfs_setresource(vfsp, resource, 0);
1622     vfs_setmntpoint(vfsp, mountpt, 0);
1623
1624     /*
1625     * going to mount on this vnode, so notify.
1626     */
1627     vnevent_mountedover(vp, NULL);
1628     error = VFS_MOUNT(vfsp, vp, uap, credp);
1629
1630     if (uap->flags & MS_RDONLY)
1631         vfs_setmntopt(vfsp, MNTOPT_RO, NULL, 0);
1632     if (uap->flags & MS_NOSUID)
1633         vfs_setmntopt(vfsp, MNTOPT_NOSUID, NULL, 0);
1634     if (uap->flags & MS_GLOBAL)
1635         vfs_setmntopt(vfsp, MNTOPT_GLOBAL, NULL, 0);
1636
1637     if (error) {
1638         lofi_remove(vfsp);
1639
1640         if (remount) {
1641             /* put back pre-remount options */
1642             vfs_swapopttbl(&mnt_mntopts, &vfsp->vfs_mntopts);
1643             vfs_setmntpoint(vfsp, refstr_value(oldmntpt),
1644                             VFSPP_VERBATIM);
1645             if (oldmntpt)
1646                 refstr_rele(oldmntpt);
1647             vfs_setresource(vfsp, refstr_value(oldresource),
1648                             VFSPP_VERBATIM);
1649             if (oldresource)
1650                 refstr_rele(oldresource);
1651             vfsp->vfs_flag = ovflags;
1652             vfs_unlock(vfsp);
1653             VFS_RELE(vfsp);
1654         } else {
1655             vfs_unlock(vfsp);
1656             vfs_freemnttab(vfsp);
1657             vfs_free(vfsp);
1658         }
1659     } else {
1660         /*
1661         * Set the mount time to now
1662         */
1663         vfsp->vfs_mtime = ddi_get_time();
1664         if (remount) {
1665             vfsp->vfs_flag &= ~VFS_REMOUNT;

```

```

1666             if (oldresource)
1667                 refstr_rele(oldresource);
1668             if (oldmntpt)
1669                 refstr_rele(oldmntpt);
1670         } else if (splice) {
1671             /*
1672             * Link vfsp into the name space at the mount
1673             * point. Vfs_add() is responsible for
1674             * holding the mount point which will be
1675             * released when vfs_remove() is called.
1676             */
1677             vfs_add(vp, vfsp, uap->flags);
1678         } else {
1679             /*
1680             * Hold the reference to file system which is
1681             * not linked into the name space.
1682             */
1683             vfsp->vfs_zone = NULL;
1684             VFS_HOLD(vfsp);
1685             vfsp->vfs_vnodecovered = NULL;
1686         }
1687     /*
1688     * Set flags for global options encountered
1689     */
1690     if (vfs_optionisset(vfsp, MNTOPT_RO, NULL))
1691         vfsp->vfs_flag |= VFS_RDONLY;
1692     else
1693         vfsp->vfs_flag &= ~VFS_RDONLY;
1694     if (vfs_optionisset(vfsp, MNTOPT_NOSUID, NULL)) {
1695         vfsp->vfs_flag |= (VFS_NOSETUID|VFS_NODEVICES);
1696     } else {
1697         if (vfs_optionisset(vfsp, MNTOPT_NODEVICES, NULL))
1698             vfsp->vfs_flag |= VFS_NODEVICES;
1699         else
1700             vfsp->vfs_flag &= ~VFS_NODEVICES;
1701         if (vfs_optionisset(vfsp, MNTOPT_NOSETUID, NULL))
1702             vfsp->vfs_flag |= VFS_NOSETUID;
1703         else
1704             vfsp->vfs_flag &= ~VFS_NOSETUID;
1705     }
1706     if (vfs_optionisset(vfsp, MNTOPT_NBMAND, NULL))
1707         vfsp->vfs_flag |= VFS_NBMAND;
1708     else
1709         vfsp->vfs_flag &= ~VFS_NBMAND;
1710
1711     if (vfs_optionisset(vfsp, MNTOPT_XATTR, NULL))
1712         vfsp->vfs_flag |= VFS_XATTR;
1713     else
1714         vfsp->vfs_flag &= ~VFS_XATTR;
1715
1716     if (vfs_optionisset(vfsp, MNTOPT_NOEXEC, NULL))
1717         vfsp->vfs_flag |= VFS_NOEXEC;
1718     else
1719         vfsp->vfs_flag &= ~VFS_NOEXEC;
1720
1721     /*
1722     * Now construct the output option string of options
1723     * we recognized.
1724     */
1725     if (uap->flags & MS_OPTIONSTR) {
1726         vfs_list_read_lock();
1727         copyout_error = vfs_buildoptionstr(
1728             &vfsp->vfs_mntopts, inargs, optlen);
1729         vfs_list_unlock();
1730         if (copyout_error == 0 &&
1731             (uap->flags & MS_SYSSPACE) == 0) {

```

```

1732         copyout_error = copyoutstr(inargs, opts,
1733         optlen, NULL);
1734     }
1735 }
1736
1737 /*
1738 * If this isn't a remount, set up the vopstats before
1739 * anyone can touch this. We only allow spliced file
1740 * systems (file systems which are in the namespace) to
1741 * have the VFS_STATS flag set.
1742 * NOTE: PxFs mounts the underlying file system with
1743 * MS_NOSPLICE set and copies those vfs_flags to its private
1744 * vfs structure. As a result, PxFs should never have
1745 * the VFS_STATS flag or else we might access the vfs
1746 * statistics-related fields prior to them being
1747 * properly initialized.
1748 */
1749 if (!remount && (vswp->vsw_flag & VSW_STATS) && splice) {
1750     initialize_vopstats(&vfsp->vfs_vopstats);
1751     /*
1752     * We need to set vfs_vskap to NULL because there's
1753     * a chance it won't be set below. This is checked
1754     * in teardown_vopstats() so we can't have garbage.
1755     */
1756     vfs->vfs_vskap = NULL;
1757     vfs->vfs_flag |= VFS_STATS;
1758     vfs->vfs_fstypevswp = get_fstype_vopstats(vfsp, vswp);
1759 }
1760
1761 if (vswp->vsw_flag & VSW_XID)
1762     vfs->vfs_flag |= VFS_XID;
1763
1764     vfs_unlock(vfsp);
1765 }
1766 mount_completed();
1767 if (splice)
1768     vn_vfsunlock(vp);
1769
1770 if ((error == 0) && (copyout_error == 0)) {
1771     if (!remount) {
1772         /*
1773         * Don't call get_vskstat_anchor() while holding
1774         * locks since it allocates memory and calls
1775         * VFS_STATVFS(). For NFS, the latter can generate
1776         * an over-the-wire call.
1777         */
1778         vskap = get_vskstat_anchor(vfsp);
1779         /* Only take the lock if we have something to do */
1780         if (vskap != NULL) {
1781             vfs_lock_wait(vfsp);
1782             if (vfs->vfs_flag & VFS_STATS) {
1783                 vfs->vfs_vskap = vskap;
1784             }
1785             vfs_unlock(vfsp);
1786         }
1787     }
1788     /* Return vfs to caller. */
1789     *vfsp = vfs;
1790     fsh_exec_mount_callbacks(vfsp);
1791 }
1792 errout:
1793     vfs_freeopttbl(&mnt_mntopts);
1794     if (resource != NULL)
1795         kmem_free(resource, strlen(resource) + 1);
1796     if (mountpt != NULL)
1797         kmem_free(mountpt, strlen(mountpt) + 1);

```

```

1798     /*
1799     * It is possible we errored prior to adding to mount in progress
1800     * table. Must free vnode we acquired with successful lookupname.
1801     */
1802     if (addmip)
1803         VN_RELE(bvp);
1804     if (delmip)
1805         vfs_delmip(vfsp);
1806     ASSERT(vswp != NULL);
1807     vfs_unrefvfsw(vswp);
1808     if (inargs != opts)
1809         kmem_free(inargs, MAX_MNTOPT_STR);
1810     if (copyout_error) {
1811         lofi_remove(vfsp);
1812         VFS_RELE(vfsp);
1813         error = copyout_error;
1814     }
1815     return (error);
1816 }
1817
1818 unchanged_portion_omitted
1819
1820 vfs_t EIO_vfs;
1821 vfsops_t *EIO_vfsops;
1822
1823 /*
1824 * Called from startup() to initialize all loaded vfs's
1825 */
1826 void
1827 vfsinit(void)
1828 {
1829     struct vfsw *vswp;
1830     int error;
1831     extern int vopstats_enabled;
1832     extern void vopstats_startup();
1833
1834     static const fs_operation_def_t EIO_vfsops_template[] = {
1835         VFSNAME_MOUNT,           { .error = vfs_EIO },
1836         VFSNAME_UNMOUNT,        { .error = vfs_EIO },
1837         VFSNAME_ROOT,           { .error = vfs_EIO },
1838         VFSNAME_STATVFS,        { .error = vfs_EIO },
1839         VFSNAME_SYNC,           { .vfs_sync = vfs_EIO_sync },
1840         VFSNAME_VGET,           { .error = vfs_EIO },
1841         VFSNAME_MOUNTROOT,      { .error = vfs_EIO },
1842         VFSNAME_FREEVFS,        { .error = vfs_EIO },
1843         VFSNAME_VNSTATE,        { .error = vfs_EIO },
1844         NULL, NULL
1845     };
1846
1847     static const fs_operation_def_t stray_vfsops_template[] = {
1848         VFSNAME_MOUNT,           { .error = vfststray },
1849         VFSNAME_UNMOUNT,        { .error = vfststray },
1850         VFSNAME_ROOT,           { .error = vfststray },
1851         VFSNAME_STATVFS,        { .error = vfststray },
1852         VFSNAME_SYNC,           { .vfs_sync = vfststray_sync },
1853         VFSNAME_VGET,           { .error = vfststray },
1854         VFSNAME_MOUNTROOT,      { .error = vfststray },
1855         VFSNAME_FREEVFS,        { .error = vfststray },
1856         VFSNAME_VNSTATE,        { .error = vfststray },
1857         NULL, NULL
1858     };
1859
1860     /* Create vfs cache */
1861     vfs_cache = kmem_cache_create("vfs_cache", sizeof (struct vfs),
1862         sizeof (uintptr_t), NULL, NULL, NULL, NULL, 0);
1863
1864     /* Initialize the vnode cache (file systems may use it during init). */

```

```

4236     vn_create_cache();
4238     /* Setup event monitor framework */
4239     fem_init();
4241     /* Setup filesystem hook framework */
4242     fsh_init();
4244     /* Initialize the dummy stray file system type. */
4245     error = vfs_setfsops(0, stray_vfsops_template, NULL);
4247     /* Initialize the dummy EIO file system. */
4248     error = vfs_makefsops(EIO_vfsops_template, &EIO_vfsops);
4249     if (error != 0) {
4250         cmn_err(CE_WARN, "vfsinit: bad EIO vfs ops template");
4251         /* Shouldn't happen, but not bad enough to panic */
4252     }
4254     VFS_INIT(&EIO_vfs, EIO_vfsops, (caddr_t)NULL);
4256     /*
4257     * Default EIO_vfs.vfs_flag to VFS_UNMOUNTED so a lookup
4258     * on this vfs can immediately notice it's invalid.
4259     */
4260     EIO_vfs.vfs_flag |= VFS_UNMOUNTED;
4262     /*
4263     * Call the init routines of non-loadable filesystems only.
4264     * Filesystems which are loaded as separate modules will be
4265     * initialized by the module loading code instead.
4266     */
4268     for (vswp = &vfssw[1]; vswp < &vfssw[infstypel]; vswp++) {
4269         RLOCK_VFSSW();
4270         if (vswp->vsw_init != NULL)
4271             (*vswp->vsw_init)(vswp - vfssw, vswp->vsw_name);
4272         RUNLOCK_VFSSW();
4273     }
4275     vopstats_startup();
4277     if (vopstats_enabled) {
4278         /* EIO_vfs can collect stats, but we don't retrieve them */
4279         initialize_vopstats(&EIO_vfs.vfs_vopstats);
4280         EIO_vfs.vfs_fstypevsp = NULL;
4281         EIO_vfs.vfs_vskap = NULL;
4282         EIO_vfs.vfs_flag |= VFS_STATS;
4283     }
4285     xattr_init();
4287     reparse_point_init();
4288 }
    unchanged portion omitted
4305 void
4306 vfs_free(vfs_t *vfsp)
4307 {
4308     /*
4309     * One would be tempted to assert that "vfsp->vfs_count == 0".
4310     * The problem is that this gets called out of domount() with
4311     * a partially initialized vfs and a vfs_count of 1. This is
4312     * also called from vfs_rele() with a vfs_count of 0. We can't
4313     * call VFS_RELE() from domount() if VFS_MOUNT() hasn't successfully
4314     * returned. This is because VFS_MOUNT() fully initializes the
4315     * vfs structure and its associated data. VFS_RELE() will call

```

```

4316     * VFS_FREEVFS() which may panic the system if the data structures
4317     * aren't fully initialized from a successful VFS_MOUNT().
4318     */
4320     /* If FEM was in use, make sure everything gets cleaned up */
4321     if (vfsp->vfs_femhead) {
4322         ASSERT(vfsp->vfs_femhead->femh_list == NULL);
4323         mutex_destroy(&vfsp->vfs_femhead->femh_lock);
4324         kmem_free(vfsp->vfs_femhead, sizeof (*vfsp->vfs_femhead));
4325         vfsp->vfs_femhead = NULL;
4326     }
4328     /*
4329     * fsh cleanup
4330     * There's no need here to use atomic operations on vfs_fshrecord.
4331     */
4332     if (vfsp->vfs_fshrecord != NULL) {
4333         fsh_fsrec_destroy(vfsp->vfs_fshrecord);
4334         vfsp->vfs_fshrecord = NULL;
4335     }
4337     if (vfsp->vfs_implp)
4338         vfsimpl_tear_down(vfsp);
4339     sema_destroy(&vfsp->vfs_reflock);
4340     kmem_cache_free(vfs_cache, vfsp);
4341 }
    unchanged portion omitted
4353 /*
4354 * Decrements the vfs reference count by one atomically. When
4355 * vfs reference count becomes zero, it calls the file system
4356 * specific vfs_freevfs() to free up the resources.
4357 */
4358 void
4359 vfs_rele(vfs_t *vfsp)
4360 {
4361     ASSERT(vfsp->vfs_count != 0);
4362     if (atomic_add_32_nv(&vfsp->vfs_count, -1) == 0) {
4363         fsh_exec_free_callbacks(vfsp);
4364         VFS_FREEVFS(vfsp);
4365         lofi_remove(vfsp);
4366         if (vfsp->vfs_zone)
4367             zone_rele_ref(&vfsp->vfs_implp->vi_zone_ref,
4368                 ZONE_REF_VFS);
4369         vfs_freemttab(vfsp);
4370         vfs_free(vfsp);
4371     }
4372 }
    unchanged portion omitted

```

```

*****
105122 Sat Aug 24 02:37:59 2013
new/usr/src/uts/common/fs/vnode.c
fsh webrev (August 24th)
*****
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) 1988, 2010, Oracle and/or its affiliates. All rights reserved.
24 */

26 /*      Copyright (c) 1983, 1984, 1985, 1986, 1987, 1988, 1989 AT&T      */
27 /*      All Rights Reserved      */

29 /*
30 * University Copyright- Copyright (c) 1982, 1986, 1988
31 * The Regents of the University of California
32 * All Rights Reserved
33 *
34 * University Acknowledgment- Portions of this document are derived from
35 * software developed by the University of California, Berkeley, and its
36 * contributors.
37 */

39 #include <sys/types.h>
40 #include <sys/param.h>
41 #include <sys/t_lock.h>
42 #include <sys/errno.h>
43 #include <sys/cred.h>
44 #include <sys/user.h>
45 #include <sys/uio.h>
46 #include <sys/file.h>
47 #include <sys/pathname.h>
48 #include <sys/vfs.h>
49 #include <sys/vfs_opreg.h>
50 #include <sys/vnode.h>
51 #include <sys/rwstlock.h>
52 #include <sys/fem.h>
53 #include <sys/stat.h>
54 #include <sys/mode.h>
55 #include <sys/conf.h>
56 #include <sys/sysmacros.h>
57 #include <sys/cmn_err.h>
58 #include <sys/system.h>
59 #include <sys/kmem.h>
60 #include <sys/debug.h>
61 #include <c2/audit.h>

```

```

62 #include <sys/acl.h>
63 #include <sys/nbmlck.h>
64 #include <sys/fcntl.h>
65 #include <fs/fs_subr.h>
66 #include <sys/taskq.h>
67 #include <fs/fs_reparse.h>
68 #include <sys/fsh_impl.h>

70 /* Determine if this vnode is a file that is read-only */
71 #define ISROFILE(vp) \
72     ((vp)->v_type != VCHR && (vp)->v_type != VBLK && \
73      (vp)->v_type != VFIFO && vn_is_readonly(vp))

75 /* Tunable via /etc/system; used only by admin/install */
76 int nfs_global_client_only;

78 /*
79  * Array of vopstats_t for per-FS-type vopstats. This array has the same
80  * number of entries as and parallel to the vfssw table. (Arguably, it could
81  * be part of the vfssw table.) Once it's initialized, it's accessed using
82  * the same fstype index that is used to index into the vfssw table.
83  */
84 vopstats_t **vopstats_fstype;

86 /* vopstats initialization template used for fast initialization via bcopy() */
87 static vopstats_t *vs_templatep;

89 /* Kmem cache handle for vsk_anchor_t allocations */
90 kmem_cache_t *vsk_anchor_cache;

92 /* file events cleanup routine */
93 extern void free_fopdata(vnode_t *);

95 /*
96  * Root of AVL tree for the kstats associated with vopstats. Lock protects
97  * updates to vsktat_tree.
98  */
99 avl_tree_t      vskstat_tree;
100 kmutex_t        vskstat_tree_lock;

102 /* Global variable which enables/disables the vopstats collection */
103 int vopstats_enabled = 1;

105 /*
106  * forward declarations for internal vnode specific data (vsd)
107  */
108 static void *vsd_realloc(void *, size_t, size_t);

110 /*
111  * forward declarations for reparse point functions
112  */
113 static int fs_reparse_mark(char *target, vattn_t *vap, xvattr_t *xvattr);

115 /*
116  * VSD -- VNODE SPECIFIC DATA
117  * The v_data pointer is typically used by a file system to store a
118  * pointer to the file system's private node (e.g. ufs inode, nfs rnode).
119  * However, there are times when additional project private data needs
120  * to be stored separately from the data (node) pointed to by v_data.
121  * This additional data could be stored by the file system itself or
122  * by a completely different kernel entity. VSD provides a way for
123  * callers to obtain a key and store a pointer to private data associated
124  * with a vnode.
125  *
126  * Callers are responsible for protecting the vsd by holding v_vsd_lock
127  * for calls to vsd_set() and vsd_get().

```



```

128 */
130 /*
131  * vsd_lock protects:
132  *   vsd_nkeys - creation and deletion of vsd keys
133  *   vsd_list - insertion and deletion of vsd_node in the vsd_list
134  *   vsd_destructor - adding and removing destructors to the list
135  */
136 static kmutex_t      vsd_lock;
137 static uint_t        vsd_nkeys;      /* size of destructor array */
138 /* list of vsd_node's */
139 static list_t *vsd_list = NULL;
140 /* per-key destructor funcs */
141 static void          (**vsd_destructor)(void *);

143 /*
144  * The following is the common set of actions needed to update the
145  * vopstats structure from a vnode op.  Both VOPSTATS_UPDATE() and
146  * VOPSTATS_UPDATE_IO() do almost the same thing, except for the
147  * recording of the bytes transferred.  Since the code is similar
148  * but small, it is nearly a duplicate.  Consequently any changes
149  * to one may need to be reflected in the other.
150  * Rundown of the variables:
151  * vp - Pointer to the vnode
152  * counter - Partial name structure member to update in vopstats for counts
153  * bytecounter - Partial name structure member to update in vopstats for bytes
154  * bytesval - Value to update in vopstats for bytes
155  * fstype - Index into vsanchor_fstype[], same as index into vfssw[]
156  * vsp - Pointer to vopstats structure (either in vfs or vsanchor_fstype[i])
157  */

159 #define VOPSTATS_UPDATE(vp, counter) {
160     vfs_t *vfsp = (vp)->v_vfsp;
161     if (vfsp && vfsp->vfs_implp &&
162         (vfsp->vfs_flag & VFS_STATS) && (vp)->v_type != VBAD) {
163         vopstats_t *vsp = &vfsp->vfs_vopstats;
164         uint64_t *stataddr = &(vsp->n##counter.value.ui64);
165         extern void __dtrace_probe__fsinfo_##counter(vnode_t *,
166             size_t, uint64_t *);
167         __dtrace_probe__fsinfo_##counter(vp, 0, stataddr);
168         (*stataddr)++;
169         if ((vsp = vfsp->vfs_fstypevsp) != NULL) {
170             vsp->n##counter.value.ui64++;
171         }
172     }
173 }

```

unchanged portion omitted

```

3216 int
3217 fop_read(
3218     vnode_t *vp,
3219     uio_t *uiop,
3220     int ioflag,
3221     cred_t *cr,
3222     caller_context_t *ct)
3223 {
3224     int err;
3225     ssize_t resid_start = uiop->uio_resid;

3227     VOPXID_MAP_CR(vp, cr);

3229     err = fsh_read(vp, uiop, ioflag, cr, ct);
3228     err = (*(vp)->v_op->vop_read)(vp, uiop, ioflag, cr, ct);
3230     VOPSTATS_UPDATE_IO(vp, read,
3231         read_bytes, (resid_start - uiop->uio_resid));
3232     return (err);

```

```

3233 }

3235 int
3236 fop_write(
3237     vnode_t *vp,
3238     uio_t *uiop,
3239     int ioflag,
3240     cred_t *cr,
3241     caller_context_t *ct)
3242 {
3243     int err;
3244     ssize_t resid_start = uiop->uio_resid;

3246     VOPXID_MAP_CR(vp, cr);

3248     err = fsh_write(vp, uiop, ioflag, cr, ct);
3247     err = (*(vp)->v_op->vop_write)(vp, uiop, ioflag, cr, ct);
3249     VOPSTATS_UPDATE_IO(vp, write,
3250         write_bytes, (resid_start - uiop->uio_resid));
3251     return (err);
3252 }

```

unchanged portion omitted

```

*****
22739 Sat Aug 24 02:37:59 2013
new/usr/src/uts/common/sys/Makefile
fsh webrev (August 24th)
*****
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) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
23 #

25 include $(SRC)/uts/Makefile.uts

27 FILEMODE=644

29 #
30 # Note that the following headers are present in the kernel but
31 # neither installed or shipped as part of the product:
32 # cpuid_drv.h: Private interface for cpuid consumers
33 # unix_bb_info.h: Private interface to kcov
34 #

36 i386_HDRS= \
37 agp/agpamd64gart_io.h \
38 agp/agpdefs.h \
39 agp/agpgart_impl.h \
40 agp/agpmaster_io.h \
41 agp/agptarget_io.h \
42 agpgart.h \
43 asy.h \
44 fd_debug.h \
45 fdc.h \
46 fdmedia.h \
47 mouse.h \
48 ucode.h

50 sparc_HDRS= \
51 mouse.h \
52 scsi/targets/ssddef.h \
53 $(MDESCHDRS)

55 # Generated headers
56 GENHDRS= \
57 priv_const.h \
58 priv_names.h \
59 usb/usbdevs.h

61 CHKHDRS= \

```

```

62 acpi_drv.h \
63 acct.h \
64 acctctl.h \
65 acl.h \
66 acl_impl.h \
67 aggr.h \
68 aggr_impl.h \
69 aio.h \
70 aio_impl.h \
71 aio_req.h \
72 aiocb.h \
73 ascii.h \
74 asynch.h \
75 atomic.h \
76 attr.h \
77 audio.h \
78 audioio.h \
79 autoconf.h \
80 auxv.h \
81 auxv_386.h \
82 auxv_SPARC.h \
83 avl.h \
84 avl_impl.h \
85 bitmap.h \
86 bitset.h \
87 bl.h \
88 blkdev.h \
89 bofi.h \
90 bofi_impl.h \
91 bpp_io.h \
92 bootstat.h \
93 brand.h \
94 buf.h \
95 bufmod.h \
96 bustypes.h \
97 byteorder.h \
98 callb.h \
99 callo.h \
100 cap_util.h \
101 cpucaps.h \
102 cpucaps_impl.h \
103 ccompile.h \
104 cdio.h \
105 cladm.h \
106 class.h \
107 clconf.h \
108 clock_impl.h \
109 cmlb.h \
110 cmn_err.h \
111 compress.h \
112 condvar.h \
113 condvar_impl.h \
114 conf.h \
115 consdev.h \
116 console.h \
117 consplat.h \
118 vt.h \
119 vtdaemon.h \
120 kd.h \
121 contract.h \
122 contract_impl.h \
123 copyops.h \
124 core.h \
125 corectl.h \
126 cpc_impl.h \
127 cpc_pcbe.h \

```

```

128     cpr.h                \|
129     cpupart.h           \|
130     cpuvar.h            \|
131     crc32.h             \|
132     cred.h              \|
133     cred_impl.h        \|
134     crtctl.h            \|
135     cryptmod.h         \|
136     csiiioctl.h        \|
137     ctf.h               \|
138     ctfs.h              \|
139     ctfs_impl.h        \|
140     ctf_api.h           \|
141     ctype.h             \|
142     cyclic.h            \|
143     cyclic_impl.h      \|
144     dacf.h              \|
145     dacf_impl.h        \|
146     damap.h             \|
147     damap_impl.h       \|
148     dc_ki.h             \|
149     ddi.h               \|
150     ddifm.h             \|
151     ddifm_impl.h       \|
152     ddi_hp.h            \|
153     ddi_hp_impl.h      \|
154     ddi_intr.h          \|
155     ddi_intr_impl.h    \|
156     ddi_impldefs.h     \|
157     ddi_implfuncs.h    \|
158     ddi_obsolete.h     \|
159     ddi_timer.h         \|
160     ddidevmap.h         \|
161     ddidmareq.h         \|
162     ddimapreq.h         \|
163     ddipropdefs.h     \|
164     ddiypes.h           \|
165     debug.h             \|
166     des.h               \|
167     devctl.h            \|
168     devcache.h          \|
169     devcache_impl.h    \|
170     devfm.h             \|
171     devid_cache.h       \|
172     devinfo_impl.h     \|
173     devops.h            \|
174     devpolicy.h         \|
175     devpoll.h           \|
176     dirent.h            \|
177     disp.h              \|
178     dkbad.h             \|
179     dkio.h              \|
180     dklabel.h           \|
181     dl.h                \|
182     dlpi.h              \|
183     dld.h               \|
184     dld_impl.h         \|
185     dld_ioc.h           \|
186     dls.h               \|
187     dls_mgmt.h          \|
188     dls_impl.h          \|
189     dma_i8237A.h        \|
190     dnlc.h              \|
191     door.h              \|
192     door_data.h         \|
193     door_impl.h         \|

```

```

194     dtrace.h            \|
195     dtrace_impl.h      \|
196     dumpadm.h           \|
197     dumphdr.h           \|
198     ecppsys.h           \|
199     ecppio.h            \|
200     ecppreg.h           \|
201     ecppvar.h           \|
202     efi_partition.h     \|
203     elf.h                \|
204     elf_386.h           \|
205     elf_SPARC.h         \|
206     elf_notes.h         \|
207     elf_amd64.h         \|
208     elftypes.h          \|
209     emul64.h            \|
210     emul64cmd.h         \|
211     emul64var.h         \|
212     epm.h                \|
213     errno.h             \|
214     errorq.h            \|
215     errorq_impl.h       \|
216     esunddi.h           \|
217     ethernet.h          \|
218     euc.h                \|
219     eucioctl.h          \|
220     exacct.h            \|
221     exacct_catalog.h    \|
222     exacct_impl.h       \|
223     exec.h              \|
224     exechnr.h           \|
225     extdirent.h         \|
226     fault.h             \|
227     fasttrap.h          \|
228     fasttrap_impl.h    \|
229     fbio.h              \|
230     fbuf.h              \|
231     fcntl.h             \|
232     fct.h               \|
233     fct_defines.h       \|
234     fctio.h             \|
235     fdbuffer.h          \|
236     fdio.h              \|
237     feature_tests.h     \|
238     fem.h                \|
239     file.h               \|
240     filio.h             \|
241     flock.h             \|
242     flock_impl.h        \|
243     fork.h              \|
244     fsh.h                \|
245     fsh_impl.h          \|
246     fss.h               \|
247     fssprioctl.h        \|
248     fsid.h              \|
249     fssnap.h            \|
250     fssnap_if.h         \|
251     fstyp.h             \|
252     ftrace.h            \|
253     fx.h                 \|
254     fxprioctl.h         \|
255     gfs.h               \|
256     gld.h               \|
257     gldpriv.h           \|
258     group.h             \|
259     hdio.h              \|

```

```

260 hook.h \
261 hook_event.h \
262 hook_impl.h \
263 hwconf.h \
264 ia.h \
265 iapriocntl.h \
266 ibpart.h \
267 id32.h \
268 idmap.h \
269 ieeeep.h \
270 id_space.h \
271 instance.h \
272 int_const.h \
273 int_fmio.h \
274 int_limits.h \
275 int_types.h \
276 inttypes.h \
277 ioccom.h \
278 ioctl.h \
279 ipc.h \
280 ipc_impl.h \
281 ipc_rctl.h \
282 ipmi.h \
283 isa_defs.h \
284 iscsi_authclient.h \
285 iscsi_authclientglue.h \
286 iscsi_protocol.h \
287 jioctl.h \
288 kbd.h \
289 kbdrreg.h \
290 kbio.h \
291 kcpic.h \
292 kdi.h \
293 kdi_impl.h \
294 kiconv.h \
295 kiconv_big5_utf8.h \
296 kiconv_ckk_common.h \
297 kiconv_cp950hkscs_utf8.h \
298 kiconv_emeal.h \
299 kiconv_emea2.h \
300 kiconv_euckr_utf8.h \
301 kiconv_euctw_utf8.h \
302 kiconv_gb18030_utf8.h \
303 kiconv_gb2312_utf8.h \
304 kiconv_hkscs_utf8.h \
305 kiconv_ja.h \
306 kiconv_ja_jis_to_unicode.h \
307 kiconv_ja_unicode_to_jis.h \
308 kiconv_ko.h \
309 kiconv_latin1.h \
310 kiconv_sc.h \
311 kiconv_tc.h \
312 kiconv_uhc_utf8.h \
313 kiconv_utf8_big5.h \
314 kiconv_utf8_cp950hkscs.h \
315 kiconv_utf8_euckr.h \
316 kiconv_utf8_euctw.h \
317 kiconv_utf8_gb18030.h \
318 kiconv_utf8_gb2312.h \
319 kiconv_utf8_hkscs.h \
320 kiconv_utf8_uhc.h \
321 kidmap.h \
322 klpd.h \
323 klpw.h \
324 kmdb.h \
325 kmem.h \

```

```

326 kmem_impl.h \
327 kobj.h \
328 kobj_impl.h \
329 ksocket.h \
330 kstat.h \
331 kstr.h \
332 ksyms.h \
333 ksynch.h \
334 ldterm.h \
335 lgrp.h \
336 lgrp_user.h \
337 libc_kernel.h \
338 link.h \
339 list.h \
340 list_impl.h \
341 llc1.h \
342 loadavg.h \
343 lock.h \
344 lockfs.h \
345 lockstat.h \
346 lofi.h \
347 log.h \
348 logindmux.h \
349 logindmux_impl.h \
350 lwp.h \
351 lwp_timer_impl.h \
352 lwp_upimutex_impl.h \
353 lpif.h \
354 mac.h \
355 mac_client.h \
356 mac_client_impl.h \
357 mac_ether.h \
358 mac_flow.h \
359 mac_flow_impl.h \
360 mac_impl.h \
361 mac_provider.h \
362 mac_soft_ring.h \
363 mac_stat.h \
364 machelf.h \
365 map.h \
366 md4.h \
367 md5.h \
368 md5_consts.h \
369 mdi_impldefs.h \
370 mem.h \
371 mem_config.h \
372 memlist.h \
373 mkdev.h \
374 mhd.h \
375 mii.h \
376 miiregs.h \
377 mixer.h \
378 mman.h \
379 mmapobj.h \
380 mntent.h \
381 mntio.h \
382 mnttab.h \
383 modctl.h \
384 mode.h \
385 model.h \
386 modhash.h \
387 modhash_impl.h \
388 mount.h \
389 mouse.h \
390 msacct.h \
391 msg.h \

```

```

392     msg_impl.h      \|
393     msio.h          \|
394     msreg.h         \|
395     mtio.h          \|
396     multidata.h    \|
397     multidata_impl.h \|
398     mutex.h        \|
399     nbmlock.h      \|
400     ndifm.h        \|
401     ndi_impldefs.h \|
402     net80211.h     \|
403     net80211_crypto.h \|
404     net80211_ht.h \|
405     net80211_proto.h \|
406     netconfig.h    \|
407     neti.h         \|
408     netstack.h     \|
409     nexusdefs.h   \|
410     note.h         \|
411     nvpair.h       \|
412     nvpair_impl.h \|
413     objfs.h        \|
414     objfs_impl.h  \|
415     ontrap.h      \|
416     open.h        \|
417     openpromio.h \|
418     panic.h       \|
419     param.h       \|
420     pathconf.h    \|
421     pathname.h    \|
422     pattr.h       \|
423     queue.h       \|
424     serializer.h  \|
425     pbio.h        \|
426     pccard.h      \|
427     pci.h         \|
428     pcie.h        \|
429     pci_impl.h    \|
430     pci_tools.h   \|
431     pcmcia.h      \|
432     ptypes.h      \|
433     pfmod.h       \|
434     pg.h          \|
435     pghw.h        \|
436     physmem.h     \|
437     pkp_hash.h    \|
438     pm.h          \|
439     policy.h      \|
440     poll.h        \|
441     poll_impl.h  \|
442     pool.h        \|
443     pool_impl.h  \|
444     pool_pset.h  \|
445     port.h        \|
446     port_impl.h  \|
447     port_kernel.h \|
448     portif.h     \|
449     ppmio.h       \|
450     pppt_ic_if.h \|
451     pppt_ioctl.h \|
452     priocntl.h   \|
453     priv.h       \|
454     priv_impl.h  \|
455     prnio.h      \|
456     proc.h       \|
457     processor.h  \|

```

```

458     procfh.h     \|
459     procset.h    \|
460     project.h    \|
461     protosw.h    \|
462     prsystem.h   \|
463     pset.h       \|
464     pshot.h      \|
465     ptem.h       \|
466     ptms.h       \|
467     ptyvar.h     \|
468     raidioctl.h \|
469     ramdisk.h    \|
470     random.h     \|
471     rctl.h       \|
472     rctl_impl.h \|
473     rds.h        \|
474     reboot.h    \|
475     refstr.h     \|
476     refstr_impl.h \|
477     resource.h   \|
478     rliocntl.h  \|
479     rt.h         \|
480     rtpriocntl.h \|
481     rwlock.h     \|
482     rwlock_impl.h \|
483     rwstlock.h  \|
484     sad.h        \|
485     schedctl.h  \|
486     sdt.h        \|
487     select.h     \|
488     sem.h        \|
489     sem_impl.h   \|
490     sema_impl.h  \|
491     semaphore.h  \|
492     sendfile.h   \|
493     ser_sync.h   \|
494     session.h    \|
495     shal.h       \|
496     shal_consts.h \|
497     sha2.h       \|
498     sha2_consts.h \|
499     share.h      \|
500     shm.h        \|
501     shm_impl.h   \|
502     sid.h        \|
503     siginfo.h    \|
504     signal.h     \|
505     sleepq.h     \|
506     smbios.h     \|
507     smbios_impl.h \|
508     subject.h    \|
509     socket.h     \|
510     socket_impl.h \|
511     socket_proto.h \|
512     socketvar.h  \|
513     sockfilter.h \|
514     sockio.h     \|
515     soundcard.h  \|
516     queue.h      \|
517     queue_impl.h \|
518     srn.h        \|
519     sservice.h   \|
520     stat.h       \|
521     statfs.h     \|
522     statvfs.h    \|
523     stdbool.h    \|

```

```

524      stdint.h          \|
525      stermio.h        \|
526      stmf.h           \|
527      stmf_defines.h   \|
528      stmf_ioctl.h     \|
529      stmf_sbd_ioctl.h \|
530      stream.h         \|
531      strft.h          \|
532      strlog.h         \|
533      strmddep.h       \|
534      stropts.h        \|
535      strredir.h       \|
536      strstat.h        \|
537      strsubr.h        \|
538      strsun.h         \|
539      strtty.h         \|
540      sunddi.h         \|
541      sunldi.h         \|
542      sunldi_impl.h    \|
543      sunmdi.h         \|
544      sunndi.h         \|
545      sunos_dhcp_class.h \|
546      sunpm.h          \|
547      suntpl.h         \|
548      suntty.h         \|
549      swap.h           \|
550      synch.h          \|
551      sysdc.h          \|
552      sysdc_impl.h     \|
553      syscall.h        \|
554      sysconf.h        \|
555      sysconfig.h      \|
556      sysevent.h       \|
557      sysevent_impl.h \|
558      sysinfo.h        \|
559      syslog.h         \|
560      sysmacros.h      \|
561      sysmsg_impl.h    \|
562      systeminfo.h     \|
563      system.h         \|
564      task.h           \|
565      taskq.h          \|
566      taskq_impl.h     \|
567      t_kuser.h        \|
568      t_lock.h         \|
569      telioctl.h       \|
570      termio.h         \|
571      termios.h        \|
572      termiox.h        \|
573      thread.h         \|
574      ticlts.h         \|
575      ticots.h         \|
576      ticotsord.h     \|
577      tihdr.h          \|
578      time.h           \|
579      time_impl.h      \|
580      time_std_impl.h \|
581      timeb.h          \|
582      timer.h          \|
583      times.h          \|
584      timex.h          \|
585      timod.h          \|
586      tirdwr.h         \|
587      tiuser.h         \|
588      tl.h             \|
589      tnf.h            \|

```

```

590      tnf_com.h        \|
591      tnf_probe.h      \|
592      tnf_writer.h     \|
593      todio.h          \|
594      tpicommon.h     \|
595      ts.h             \|
596      tspriocntl.h    \|
597      ttcompat.h      \|
598      ttold.h          \|
599      tty.h            \|
600      ttychars.h      \|
601      ttydev.h        \|
602      tuneable.h      \|
603      turnstile.h     \|
604      types.h          \|
605      types32.h       \|
606      tzfile.h        \|
607      u8_textprep.h   \|
608      u8_textprep_data.h \|
609      uadmin.h         \|
610      ucred.h         \|
611      uio.h           \|
612      ulimit.h        \|
613      un.h            \|
614      unistd.h        \|
615      user.h          \|
616      ustat.h         \|
617      utime.h         \|
618      utsname.h       \|
619      utssys.h        \|
620      uuid.h          \|
621      va_impl.h       \|
622      va_list.h       \|
623      var.h           \|
624      varargs.h       \|
625      vfs.h           \|
626      vfs_opreg.h     \|
627      vfstab.h        \|
628      vgapreg.h       \|
629      videodev2.h     \|
630      visual_io.h     \|
631      vlan.h          \|
632      vm.h            \|
633      vm_usage.h      \|
634      vmem.h          \|
635      vmem_impl.h    \|
636      vmsystem.h     \|
637      vnic.h          \|
638      vnic_impl.h    \|
639      vnode.h         \|
640      vscan.h         \|
641      vtoc.h          \|
642      vtrace.h        \|
643      vuid_event.h    \|
644      vuid_wheel.h    \|
645      vuid_queue.h    \|
646      vuid_state.h    \|
647      vuid_store.h   \|
648      wait.h          \|
649      waitq.h         \|
650      wanboot_impl.h \|
651      watchpoint.h    \|
652      winlockio.h     \|
653      zcons.h         \|
654      zone.h          \|
655      xti_inet.h      \|

```

```

656      xti_osi.h          \
657      xti_xtiopt.h     \
658      zmod.h           \

660 HDRS=                \
661      $(GENHDRS)       \
662      $(CHKHDRS)

664 AUDIOHDRS=          \
665      ac97.h           \
666      audio_common.h  \
667      audio_driver.h  \
668      audio_oss.h     \
669      g711.h

671 AVHDRS=              \
672      iec61883.h

674 BSCHDRS=            \
675      bscbus.h         \
676      bscv_impl.h     \
677      lom_ebuscodes.h \
678      lom_io.h         \
679      lom_priv.h      \
680      lombus.h

682 MDESCHDRS=          \
683      mdesc.h          \
684      mdesc_impl.h

686 CPUDRVHDRS=         \
687      cpudrv.h

689 CRYPTOHDRS=         \
690      elfsign.h        \
691      ioctl.h          \
692      ioctladmin.h    \
693      common.h         \
694      impl.h           \
695      spi.h            \
696      api.h            \
697      ops_impl.h      \
698      sched_impl.h

700 DCAMHDRS=           \
701      dcaml394_io.h

703 IBHDRS=              \
704      ib_types.h      \
705      ib_pkt_hdrs.h

707 IBTLHDRS=            \
708      ibtl_types.h    \
709      ibtl_status.h   \
710      ibti.h          \
711      ibti_cm.h       \
712      ibci.h          \
713      ibti_common.h   \
714      ibvti.h         \
715      ibtl_ci_types.h

717 IBTLIMPLHDRS=       \
718      ibtl_util.h

720 IBNEXHDRS=          \
721      ibnex_devctl.h

```

```

723 IBMFHDRS=           \
724      ibmf.h          \
725      ibmf_msg.h     \
726      ibmf_saa.h     \
727      ibmf_utils.h

729 IBMGTHDRS=          \
730      ib_dm_attr.h   \
731      ib_mad.h       \
732      sm_attr.h      \
733      sa_recvs.h

735 IBDHDRS=            \
736      ibd.h

738 OFHDRS=              \
739      ofa_solaris.h  \
740      ofed_kernel.h

742 RDMAHDRS=           \
743      ib_addr.h      \
744      ib_user_mad.h  \
745      ib_user_sa.h   \
746      ib_user_verbs.h \
747      ib_verbs.h     \
748      rdma_cm.h      \
749      rdma_user_cm.h

751 SOL_UVERBSHDRS=     \
752      sol_uverbs.h   \
753      sol_uverbs2ucma.h \
754      sol_uverbs_comp.h \
755      sol_uverbs_hca.h \
756      sol_uverbs_qp.h \
757      sol_uverbs_event.h

759 SOL_UMADHDRS=       \
760      sol_umad.h

762 SOL_UCMAHDRS=       \
763      sol_ucma.h     \
764      sol_rdma_user_cm.h

766 SOL_OFSHDRS=        \
767      sol_cma.h      \
768      sol_ib_cma.h   \
769      sol_ofs_common.h \
770      sol_kverb_impl.h

772 TAVORHDRS=          \
773      tavor_ioctl.h

775 HERMONHDRS=         \
776      hermon_ioctl.h

778 MLNXHDRS=           \
779      mlnx_umap.h

781 IDMHDRS=            \
782      idm.h          \
783      idm_impl.h     \
784      idm_so.h       \
785      idm_text.h     \
786      idm_transport.h \
787      idm_conn_sm.h

```

```

789 ISCSITHDRS= \
790     radius_packet.h \
791     radius_protocol.h \
792     chap.h \
793     isns_protocol.h \
794     iscsi_if.h \
795     iscsit_common.h \

797 ISOHDRS= \
798     signal_iso.h \

800 DERIVED_LVMHDRS= \
801     md_mdiox.h \
802     md_basic.h \
803     mdmed.h \
804     md_mhdx.h \
805     mdmn_commd.h \

807 LVMHDRS= \
808     md_convert.h \
809     md_crc.h \
810     md_hotspares.h \
811     md_mddb.h \
812     md_mirror.h \
813     md_mirror_shared.h \
814     md_names.h \
815     md_notify.h \
816     md_raid.h \
817     md_rename.h \
818     md_sp.h \
819     md_stripe.h \
820     md_trans.h \
821     mdio.h \
822     mdvar.h \

824 ALL_LVMHDRS= \
825     $(LVMHDRS) \
826     $(DERIVED_LVMHDRS) \

828 FMHDRS= \
829     protocol.h \
830     util.h \

832 FMFSHDRS= \
833     zfs.h \

835 FMIOHDRS= \
836     ddi.h \
837     disk.h \
838     pci.h \
839     scsi.h \
840     sun4upci.h \
841     opl_mc_fm.h \

843 FSHDRS= \
844     autofs.h \
845     cacheofs_dir.h \
846     cacheofs_dlog.h \
847     cacheofs_filegrp.h \
848     cacheofs_fs.h \
849     cacheofs_fscache.h \
850     cacheofs_ioctl.h \
851     cacheofs_log.h \
852     decomp.h \
853     dv_node.h \

```

```

854     sdev_impl.h \
855     fifonode.h \
856     hsfs_isospec.h \
857     hsfs_node.h \
858     hsfs_rrip.h \
859     hsfs_spec.h \
860     hsfs_susp.h \
861     lofs_info.h \
862     lofs_node.h \
863     mntdata.h \
864     namenode.h \
865     pc_dir.h \
866     pc_fs.h \
867     pc_label.h \
868     pc_node.h \
869     pxfi_ki.h \
870     snode.h \
871     swapnode.h \
872     tmp.h \
873     tmpnode.h \
874     udf_inode.h \
875     udf_volume.h \
876     ufs_acl.h \
877     ufs_bio.h \
878     ufs_filio.h \
879     ufs_fs.h \
880     ufs_fsdire.h \
881     ufs_inode.h \
882     ufs_lockfs.h \
883     ufs_log.h \
884     ufs_mount.h \
885     ufs_panic.h \
886     ufs_prot.h \
887     ufs_quota.h \
888     ufs_snap.h \
889     ufs_trans.h \
890     zfs.h \
891     zut.h \

893 PCMCIAHDRS= \
894     pcata.h \
895     pcser_conf.h \
896     pcser_io.h \
897     pcser_reg.h \
898     pcser_manuspec.h \
899     pcser_var.h \

901 SCSIHDRS= \
902     scsi.h \
903     scsi_address.h \
904     scsi_ctl.h \
905     scsi_fm.h \
906     scsi_params.h \
907     scsi_pkt.h \
908     scsi_resource.h \
909     scsi_types.h \
910     scsi_watch.h \

912 SCSSICONFHDRS= \
913     autoconf.h \
914     device.h \

916 SCSSIGENHDRS= \
917     commands.h \
918     dad_mode.h \
919     inquiry.h \

```



```

920     message.h      \
921     mode.h         \
922     persist.h     \
923     sense.h       \
924     sff_frames.h  \
925     smp_frames.h  \
926     status.h      \

928 SCSIIMPLHDRS=    \
929     commands.h    \
930     inquiry.h     \
931     mode.h        \
932     scsi_reset_notify.h \
933     scsi_sas.h    \
934     sense.h       \
935     services.h    \
936     smp_transport.h \
937     spc3_types.h  \
938     status.h      \
939     transport.h   \
940     types.h       \
941     uscsi.h       \
942     usmp.h        \

944 SCSTITARGETSHDRS= \
945     ses.h         \
946     sesio.h       \
947     sgendef.h     \
948     stdef.h       \
949     sddef.h       \
950     smp.h         \

952 SCSIADHDRS=

954 SCASICADHDRS=

956 SCIIISCSIHDRS=  \
957     iscsi_door.h \
958     iscsi_if.h   \

960 SCIVHCIHDRS=    \
961     scsi_vhci.h  \
962     mpapi_impl.h \
963     mpapi_scsi_vhci.h \

965 SDCARDHDRS=     \
966     sda.h        \
967     sda_impl.h   \
968     sda_ioctl.h  \

970 FC4HDRS=        \
971     fc_transport.h \
972     linkapp.h     \
973     fc.h          \
974     fcp.h         \
975     fcal_transport.h \
976     fcal.h        \
977     fcal_linkapp.h \
978     fcio.h        \

980 FCHDRS=         \
981     fc.h         \
982     fcio.h       \
983     fc_types.h  \
984     fc_appif.h  \

```

```

986 FCIMPLHDRS=     \
987     fc_error.h   \
988     fcph.h       \

990 FCULPHDRS=      \
991     fcp_util.h   \
992     fcsm.h       \

994 SATAGENHDRS=    \
995     sata_hba.h   \
996     sata_defs.h  \
997     sata_cfgadm.h \

999 SYSEVENTHDRS=   \
1000    ap_driver.h   \
1001    dev.h         \
1002    domain.h      \
1003    dr.h          \
1004    env.h         \
1005    eventdefs.h   \
1006    ipmp.h        \
1007    pwrctl.h      \
1008    svm.h         \
1009    vrrp.h        \

1011 CONTRACTHDRS=   \
1012    process.h     \
1013    process_impl.h \
1014    device.h      \
1015    device_impl.h \

1017 USBHDRS=        \
1018    usba.h        \
1019    usbai.h       \

1021 UWBHDRS=         \
1022    uwb.h         \
1023    uwbai.h       \

1025 UWBAHDRS=       \
1026    uwba.h       \

1028 USBAUDHDRS=     \
1029    usb_audio.h  \

1031 USBHUBDHDRS=    \
1032    hub.h        \
1033    hubd_impl.h  \

1035 USBHIDHDRS=     \
1036    hid.h        \

1038 USBHWARDHDRS=   \
1039    hwarc.h      \

1041 USBMSHDRS=      \
1042    usb_bulkonly.h \
1043    usb_cbi.h     \

1045 USBPRNHDRS=     \
1046    usb_printer.h \

1048 USBDCDCHDRS=    \
1049    usb_cdc.h     \

1051 USBVIDHDRS=     \

```

```

1052     usbvc.h
1054 USBWCMHDRS= \
1055     usbwcm.h
1057 UGENHDRS= \
1058     usb_ugen.h
1060 HOTPLUGHDRS= \
1061     hpcsvc.h \
1062     hpctrl.h
1064 HOTPLUGPCIHDRS= \
1065     pcicfg.h \
1066     pcihp.h
1068 RSMHDRS= \
1069     rsm.h \
1070     rsm_common.h \
1071     rsmapi_common.h \
1072     rsmapi.h \
1073     rsmapi_driver.h \
1074     rsmka_path_int.h
1076 TSOLHDRS= \
1077     label.h \
1078     label_macro.h \
1079     priv.h \
1080     tndb.h \
1081     tsyscall.h
1083 I1394HDRS= \
1084     cmd1394.h \
1085     idl1394.h \
1086     ieee1212.h \
1087     ieee1394.h \
1088     ixl1394.h \
1089     sl394_impl.h \
1090     tl394.h
1092 # "cmdk" headers used on sparc
1093 SDKTPHDRS= \
1094     dadkio.h \
1095     fdisk.h
1097 # "cmdk" headers used on i386
1098 DKTPHDRS= \
1099     altctr.h \
1100     bbh.h \
1101     cm.h \
1102     cmddev.h \
1103     cmdk.h \
1104     cmpkt.h \
1105     controller.h \
1106     dadev.h \
1107     dadk.h \
1108     dadkio.h \
1109     fctypes.h \
1110     fdisk.h \
1111     flowctrl.h \
1112     gda.h \
1113     quetypes.h \
1114     queue.h \
1115     tgcom.h \
1116     tgdk.h

```

```

1118 # "pc" header files used on i386
1119 PCHDRS= \
1120     avintr.h \
1121     dma_engine.h \
1122     i8272A.h \
1123     pcic_reg.h \
1124     pcic_var.h \
1125     pic.h \
1126     pit.h \
1127     rtc.h
1129 NXGEHDRS= \
1130     nxge.h \
1131     nxge_common.h \
1132     nxge_common_impl.h \
1133     nxge_defs.h \
1134     nxge_hw.h \
1135     nxge_impl.h \
1136     nxge_ipp.h \
1137     nxge_ipp_hw.h \
1138     nxge_mac.h \
1139     nxge_mac_hw.h \
1140     nxge_fflp.h \
1141     nxge_fflp_hw.h \
1142     nxge_mii.h \
1143     nxge_rxdma.h \
1144     nxge_rxdma_hw.h \
1145     nxge_txc.h \
1146     nxge_txc_hw.h \
1147     nxge_txdma.h \
1148     nxge_txdma_hw.h \
1149     nxge_virtual.h \
1150     nxge_espc.h
1152 include Makefile.sysshdrs
1154 dcam/%.check: dcam/%.h
1155     $(DOT_H_CHECK)
1157 CHECKHDRS= \
1158     $( $(MACH)_HDRS:%.h=% .check) \
1159     $(AUDIOHDRS:%.h=audio/%.check) \
1160     $(AVHDRS:%.h=av/%.check) \
1161     $(BSCHDRS:%.h=% .check) \
1162     $(CHKHDRS:%.h=% .check) \
1163     $(CPUDRVHDRS:%.h=% .check) \
1164     $(CRYPTOHDRS:%.h=crypto/%.check) \
1165     $(DCAMHDRS:%.h=dcam/%.check) \
1166     $(FC4HDRS:%.h=fc4/%.check) \
1167     $(FCHDRS:%.h=fibre-channel/%.check) \
1168     $(FCIMPLHDRS:%.h=fibre-channel/impl/%.check) \
1169     $(FCULPHDRS:%.h=fibre-channel/ulp/%.check) \
1170     $(IBHDRS:%.h=ib/%.check) \
1171     $(IBDHDRS:%.h=ib/clients/ibd/%.check) \
1172     $(IBTLHDRS:%.h=ib/ibt1/%.check) \
1173     $(IBTLIMPLHDRS:%.h=ib/ibt1/impl/%.check) \
1174     $(IBNEXHDRS:%.h=ib/ibnex/%.check) \
1175     $(IBMGTHDRS:%.h=ib/mgt/%.check) \
1176     $(IBMFHDRS:%.h=ib/mgt/ibmf/%.check) \
1177     $(OFHDRS:%.h=ib/clients/of/%.check) \
1178     $(RDMAHDRS:%.h=ib/clients/of/rdma/%.check) \
1179     $(SOL_UVERBSHDRS:%.h=ib/clients/of/sol_uverbs/%.check) \
1180     $(SOL_UCMAHDRS:%.h=ib/clients/of/sol_ucma/%.check) \
1181     $(SOL_OFSHDRS:%.h=ib/clients/of/sol_ofs/%.check) \
1182     $(TAVORHDRS:%.h=ib/adapters/tavor/%.check) \
1183     $(HERMONHDRS:%.h=ib/adapters/hermon/%.check)

```

```

1184 $(MLNXHDRS:%.h=ib/adapters/%.check) \
1185 $(IDMHDRS:%.h=idm/%.check) \
1186 $(ISCSIHDRS:%.h=iscsi/%.check) \
1187 $(ISCSITHDRS:%.h=iscsit/%.check) \
1188 $(ISOHDRS:%.h=iso/%.check) \
1189 $(FMHDRS:%.h=fm/%.check) \
1190 $(FMFHDRS:%.h=fm/fs/%.check) \
1191 $(FMIOHDRS:%.h=fm/io/%.check) \
1192 $(FSHDRS:%.h=fs/%.check) \
1193 $(LVMHDRS:%.h=lvm/%.check) \
1194 $(PCMCIAHDRS:%.h=pcmcia/%.check) \
1195 $(SCSIHDRS:%.h=scsi/%.check) \
1196 $(SCSIADHDRS:%.h=scsi/adapters/%.check) \
1197 $(SCSICONFHDRS:%.h=scsi/conf/%.check) \
1198 $(SCSIIMPLHDRS:%.h=scsi/impl/%.check) \
1199 $(SCSIISCSIHDRS:%.h=scsi/adapters/%.check) \
1200 $(SCSIGHDRS:%.h=scsi/generic/%.check) \
1201 $(SCSITARGETSHDRS:%.h=scsi/targets/%.check) \
1202 $(SCSIVHCIHDRS:%.h=scsi/adapters/%.check) \
1203 $(SATAGENHDRS:%.h=sata/%.check) \
1204 $(SDCARDHDRS:%.h=sdcard/%.check) \
1205 $(SYSEVENTHDRS:%.h=sysevent/%.check) \
1206 $(CONTRACTHDRS:%.h=contract/%.check) \
1207 $(USBAUDHDRS:%.h=usb/clients/audio/%.check) \
1208 $(USBHUBDHDRS:%.h=usb/hubd/%.check) \
1209 $(USBHIDHDRS:%.h=usb/clients/hid/%.check) \
1210 $(USBHWARDHDRS:%.h=usb/clients/hwarc/%.check) \
1211 $(USBMSHDRS:%.h=usb/clients/mass_storage/%.check) \
1212 $(USBPRNHDRS:%.h=usb/clients/printer/%.check) \
1213 $(USBCDCHDRS:%.h=usb/clients/usbcdc/%.check) \
1214 $(USBVIDHDRS:%.h=usb/clients/video/usbvc/%.check) \
1215 $(USBWCMHDRS:%.h=usb/clients/usbinput/usbwcm/%.check) \
1216 $(UGENHDRS:%.h=usb/clients/ugen/%.check) \
1217 $(USBHDRS:%.h=usb/%.check) \
1218 $(UWBHDRS:%.h=uwb/%.check) \
1219 $(UWBAHDRS:%.h=uwb/uwba/%.check) \
1220 $(I1394HDRS:%.h=1394/%.check) \
1221 $(RSMHDRS:%.h=rsm/%.check) \
1222 $(TSOLHDRS:%.h=tsol/%.check) \
1223 $(NXGEHDRS:%.h=nxge/%.check)

```

```
1226 .KEEP_STATE:
```

```

1228 .PARALLEL: \
1229 $(CHECKHDRS) \
1230 $(ROOTHDRS) \
1231 $(ROOTAUDHDRS) \
1232 $(ROOTAVHDRS) \
1233 $(ROOTCRYPTOHDRS) \
1234 $(ROOTDCAMHDRS) \
1235 $(ROOTISOHDRS) \
1236 $(ROOTIDMHDRS) \
1237 $(ROOTISCSIHDRS) \
1238 $(ROOTISCSITHDRS) \
1239 $(ROOTFC4HDRS) \
1240 $(ROOTFCHDRS) \
1241 $(ROOTFCIMPLHDRS) \
1242 $(ROOTFCULPHDRS) \
1243 $(ROOTFMHDRS) \
1244 $(ROOTFMIOHDRS) \
1245 $(ROOTFMFHDRS) \
1246 $(ROOTFSHDRS) \
1247 $(ROOTIBDHDRS) \
1248 $(ROOTIBHDRS) \
1249 $(ROOTIBTLHDRS) \

```

```

1250 $(ROOTIBTLIMPLHDRS) \
1251 $(ROOTIBNEXHDRS) \
1252 $(ROOTIBMGTHDRS) \
1253 $(ROOTIBMFHDRS) \
1254 $(ROOTOFHDRS) \
1255 $(ROOTRDMHDRS) \
1256 $(ROOTSOL_OFSHDRS) \
1257 $(ROOTSOL_UMADHDRS) \
1258 $(ROOTSOL_UVERBSHDRS) \
1259 $(ROOTSOL_UCMAHDRS) \
1260 $(ROOTTAVORHDRS) \
1261 $(ROOTTHERMONHDRS) \
1262 $(ROOTMLNXHDRS) \
1263 $(ROOTLVMHDRS) \
1264 $(ROOTPCMCIAHDRS) \
1265 $(ROOTSCSIHDRS) \
1266 $(ROOTSCSIADHDRS) \
1267 $(ROOTSCSICONFHDRS) \
1268 $(ROOTSCSIIISCSIHDRS) \
1269 $(ROOTSCSIGHDRS) \
1270 $(ROOTSCSIIMPLHDRS) \
1271 $(ROOTSCSIVHCIHDRS) \
1272 $(ROOTSDCARDHDRS) \
1273 $(ROOTSYSEVENTHDRS) \
1274 $(ROOTCONTRACTHDRS) \
1275 $(ROOTUSBHDRS) \
1276 $(ROOTUWBHDRS) \
1277 $(ROOTUWBAHDRS) \
1278 $(ROOTUSBAUDHDRS) \
1279 $(ROOTUSBHUBDHDRS) \
1280 $(ROOTUSBHIDHDRS) \
1281 $(ROOTUSBHWARDHDRS) \
1282 $(ROOTUSBMSHDRS) \
1283 $(ROOTUSBPRNHDRS) \
1284 $(ROOTUSBCDCHDRS) \
1285 $(ROOTUSBVIDHDRS) \
1286 $(ROOTUSBWCMHDRS) \
1287 $(ROOTUGENHDRS) \
1288 $(ROOTI1394HDRS) \
1289 $(ROOTHOTPLUGHDRS) \
1290 $(ROOTHOTPLUGPCIHDRS) \
1291 $(ROOTRSMHDRS) \
1292 $(ROOTTSOLHDRS) \
1293 $( $(MACH)_ROOTHDRS)

```

```

1296 install_h: \
1297 $(ROOTDIRS) \
1298 LVMDERIVED_H \
1299 .WAIT \
1300 $(ROOTHDRS) \
1301 $(ROOTAUDHDRS) \
1302 $(ROOTAVHDRS) \
1303 $(ROOTCRYPTOHDRS) \
1304 $(ROOTDCAMHDRS) \
1305 $(ROOTISOHDRS) \
1306 $(ROOTIDMHDRS) \
1307 $(ROOTISCSIHDRS) \
1308 $(ROOTISCSITHDRS) \
1309 $(ROOTFC4HDRS) \
1310 $(ROOTFCHDRS) \
1311 $(ROOTFCIMPLHDRS) \
1312 $(ROOTFCULPHDRS) \
1313 $(ROOTFMHDRS) \
1314 $(ROOTFMFHDRS) \
1315 $(ROOTFMIOHDRS) \

```

```

1316 $(ROOTFSHDRS) \
1317 $(ROOTIBDHDRS) \
1318 $(ROOTIBHDRS) \
1319 $(ROOTIBTLHDRS) \
1320 $(ROOTIBTLIMPLHDRS) \
1321 $(ROOTIBNEXHDRS) \
1322 $(ROOTIBMGTHDRS) \
1323 $(ROOTIBMFHDRS) \
1324 $(ROOTOFHDRS) \
1325 $(ROOTRDMHDRS) \
1326 $(ROOTSOL_OFSHDRS) \
1327 $(ROOTSOL_UMADHDRS) \
1328 $(ROOTSOL_UVERBSHDRS) \
1329 $(ROOTSOL_UCMAHDRS) \
1330 $(ROOTTAVORHDRS) \
1331 $(ROOTTHERMONHDRS) \
1332 $(ROOTMLNXHDRS) \
1333 $(ROOTLVMHDRS) \
1334 $(ROOTPCMCIAHDRS) \
1335 $(ROOTSCSIHDRS) \
1336 $(ROOTSCSIADHDRS) \
1337 $(ROOTSCSIISCSIIHDRS) \
1338 $(ROOTSCSICONFHDRS) \
1339 $(ROOTSCSIGENHDRS) \
1340 $(ROOTSCSIIMPLHDRS) \
1341 $(ROOTSCSIVHCIHDRS) \
1342 $(ROOTSDCARDHDRS) \
1343 $(ROOTSYSEVENTHDRS) \
1344 $(ROOTCONTRACTHDRS) \
1345 $(ROOTUWBHDRS) \
1346 $(ROOTUWBAHDRS) \
1347 $(ROOTUSBHDRS) \
1348 $(ROOTUSBAUDHDRS) \
1349 $(ROOTUSBHUBDHDRS) \
1350 $(ROOTUSBHIDHDRS) \
1351 $(ROOTUSBHRCHDRS) \
1352 $(ROOTUSBMSHDRS) \
1353 $(ROOTUSBPRNHDRS) \
1354 $(ROOTUSBCDCHDRS) \
1355 $(ROOTUSBVIDHDRS) \
1356 $(ROOTUSBWCMHDRS) \
1357 $(ROOTUGENHDRS) \
1358 $(ROOT1394HDRS) \
1359 $(ROOTHOTPLUGHDRS) \
1360 $(ROOTHOTPLUGPCIHDRS) \
1361 $(ROOTRSMHDRS) \
1362 $(ROOTTSOLHDRS) \
1363 $(MACH)_ROOTHDRS)

1365 all_h: $(GENHDRS)

1367 priv_const.h: $(PRIVS_AWK) $(PRIVS_DEF)
1368 $(NAWK) -f $(PRIVS_AWK) < $(PRIVS_DEF) -v privhfile=$@

1370 priv_names.h: $(PRIVS_AWK) $(PRIVS_DEF)
1371 $(NAWK) -f $(PRIVS_AWK) < $(PRIVS_DEF) -v pubhfile=$@

1373 usb/usbdevs.h: $(USBDEVS_AWK) $(USBDEVS_DATA)
1374 $(NAWK) -f $(USBDEVS_AWK) $(USBDEVS_DATA) -H > $@

1376 LVMDERIVED_H:
1377 cd $(SRC)/uts/common/sys/lvm; pwd; $(MAKE)

1379 clean:
1380 $(RM) $(GENHDRS)

```

```

1382 clobber: clean

1384 check: $(CHECKHDRS)

1386 FRC:

1388 # EXPORT DELETE START
1389 EXPORT_SRC:
1390 $(RM) wanboot_impl.h+ Makefile+
1391 sed -e "/EXPORT DELETE START/,/EXPORT DELETE END/d" \
1392 < wanboot_impl.h > wanboot_impl.h+
1393 $(MV) wanboot_impl.h+ wanboot_impl.h
1394 sed -e "/^# EXPORT DELETE START/,/^# EXPORT DELETE END/d" \
1395 < Makefile > Makefile+
1396 $(RM) Makefile
1397 $(MV) Makefile+ Makefile
1398 $(CHMOD) 444 Makefile wanboot_impl.h
1399 # EXPORT DELETE END

```

new/usr/src/uts/common/sys/fsh.h

1

```
*****
2034 Sat Aug 24 02:37:59 2013
new/usr/src/uts/common/sys/fsh.h
fsh webrev (August 24th)
*****
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 2013 Damian Bogel. All rights reserved.
14 */

16 #ifndef _FSH_H
17 #define _FSH_H

19 #include <sys/id_space.h>
20 #include <sys/types.h>
21 #include <sys/vfs.h>
22 #include <sys/vnode.h>

24 #ifdef __cplusplus
25 extern "C" {
26 #endif

28 typedef id_t fsh_handle_t;
29 typedef id_t fsh_callback_handle_t;

31 struct fsh_int;
32 typedef struct fsh_int fsh_int_t;

34 typedef struct fsh {
35     void *arg;

37     /* vnode */
38     int (*read)(fsh_int_t *, void *, vnode_t *, uio_t *, int, cred_t *,
39               caller_context_t *);
40     int (*write)(fsh_int_t *, void *, vnode_t *, uio_t *, int, cred_t *,
41               caller_context_t *);

43     /* vfs */
44     int (*mount)(fsh_int_t *, void *, vfs_t *, vnode_t *, struct mounta *,
45                cred_t *);
46     int (*unmount)(fsh_int_t *, void *, vfs_t *, int, cred_t *);
47 } fsh_t;

49 typedef struct fsh_callback {
50     void *fshc_arg;
51     void (*fshc_free)(vfs_t *, void *);
52     void (*fshc_mount)(vfs_t *, void *);
53 } fsh_callback_t;

55 /* API */
56 extern fsh_handle_t fsh_hook_install(vfs_t *, fsh_t *);
57 extern int fsh_hook_remove(fsh_handle_t);

59 extern fsh_callback_handle_t fsh_callback_install(fsh_callback_t *);
60 extern int fsh_callback_remove(fsh_callback_handle_t);
```

new/usr/src/uts/common/sys/fsh.h

2

```
62 extern void fsh_fs_enable(vfs_t *);
63 extern void fsh_fs_disable(vfs_t *);

65 /* fsh control passing */
66 extern int fsh_next_read(fsh_int_t *, vnode_t *, uio_t *, int, cred_t *,
67                          caller_context_t *);
68 extern int fsh_next_write(fsh_int_t *, vnode_t *, uio_t *, int, cred_t *,
69                          caller_context_t *);

71 extern int fsh_next_mount(fsh_int_t *, vfs_t *, vnode_t *, struct mounta *uap,
72                          cred_t *);
73 extern int fsh_next_unmount(fsh_int_t *, vfs_t *, int, cred_t *);

75 #ifdef __cplusplus
76 }
77 #endif

79 #endif /* _FSH_H */
```

new/usr/src/uts/common/sys/fsh_impl.h

1

1413 Sat Aug 24 02:37:59 2013

new/usr/src/uts/common/sys/fsh_impl.h

fsh webrev (August 24th)

```
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 2013 Damian Bogel. All rights reserved.
14 */

16 /*
17  * This file includes all the necessary declarations for vfs.c and vnode.c.
18 */

20 #ifndef _FSH_IMPL_H
21 #define _FSH_IMPL_H

23 #include <sys/fsh.h>
24 #include <sys/list.h>
25 #include <sys/pathname.h>
26 #include <sys/types.h>
27 #include <sys/vfs.h>
28 #include <sys/vnode.h>

30 #ifdef __cplusplus
31 extern "C" {
32 #endif

34 struct fsh_fshrecord;
35 typedef struct fsh_fsrecord fsh_fsrecord_t;

37 /* API for vnode.c and vfs.c only */
38 /* vnode.c */
39 extern int fsh_read(vnode_t *, uio_t *, int, cred_t *, caller_context_t *);
40 extern int fsh_write(vnode_t *, uio_t *, int, cred_t *, caller_context_t *);

42 /* vfs.c */
43 extern void fsh_init(void);
44 extern void fsh_exec_mount_callbacks(vfs_t *);
45 extern void fsh_exec_free_callbacks(vfs_t *);
46 extern void fsh_fsrec_destroy(fsh_fsrecord_t *volatile);

48 extern int fsh_mount(vfs_t *, vnode_t *, struct mounta *, cred_t *);
49 extern int fsh_unmount(vfs_t *, int, cred_t *);

51 #ifdef __cplusplus
52 }
53 #endif

55 #endif /* _FSH_IMPL_H */
```

```

*****
21197 Sat Aug 24 02:37:59 2013
new/usr/src/uts/common/sys/vfs.h
fsh webrev (August 24th)
*****
_____unchanged_portion_omitted_____

173 extern avl_tree_t      vskstat_tree;
174 extern kmutex_t       vskstat_tree_lock;

176 /*
177  * Structure per mounted file system.  Each mounted file system has
178  * an array of operations and an instance record.
179  *
180  * The file systems are kept on a doubly linked circular list headed by
181  * "rootvfs".
182  * File system implementations should not access this list;
183  * it's intended for use only in the kernel's vfs layer.
184  *
185  * Each zone also has its own list of mounts, containing filesystems mounted
186  * somewhere within the filesystem tree rooted at the zone's rootpath.  The
187  * list is doubly linked to match the global list.
188  *
189  * mnttab locking: the in-kernel mnttab uses the vfs_mntpt, vfs_resource and
190  * vfs_mntopts fields in the vfs_t.  mntpt and resource are refstr_ts that
191  * are set at mount time and can only be modified during a remount.
192  * It is safe to read these fields if you can prevent a remount on the vfs,
193  * or through the convenience funcs vfs_getmntpoint() and vfs_getresource().
194  * The mntopts field may only be accessed through the provided convenience
195  * functions, as it is protected by the vfs list lock.  Modifying a mount
196  * option requires grabbing the vfs list write lock, which can be a very
197  * high latency lock.
198  */
199 struct zone;          /* from zone.h */
200 struct fem_head;     /* from fem.h */
201 struct fsh_fsrecord; /* from fsh_impl.h */

203 typedef struct vfs {
204     struct vfs      *vfs_next;          /* next VFS in VFS list */
205     struct vfs      *vfs_prev;          /* prev VFS in VFS list */
206
207 /* vfs_op should not be used directly.  Accessor functions are provided */
208     vfsops_t        *vfs_op;           /* operations on VFS */
209
210     struct vnode    *vfs_vnodecovered; /* vnode mounted on */
211     uint_t          vfs_flag;          /* flags */
212     uint_t          vfs_bsize;         /* native block size */
213     int             vfs_fstype;        /* file system type index */
214     fsid_t          vfs_fsid;          /* file system id */
215     void            *vfs_data;         /* private data */
216     dev_t           vfs_dev;           /* device of mounted VFS */
217     ulong_t         vfs_bcount;        /* I/O count (accounting) */
218     struct vfs      *vfs_list;         /* sync list pointer */
219     struct vfs      *vfs_hash;         /* hash list pointer */
220     ksema_t         vfs_reflock;       /* mount/unmount/sync lock */
221     uint_t          vfs_count;         /* vfs reference count */
222     mntopts_t       vfs_mntopts;       /* options mounted with */
223     refstr_t        *vfs_resource;     /* mounted resource name */
224     refstr_t        *vfs_mntpt;       /* mount point name */
225     time_t          vfs_mtime;        /* time we were mounted */
226     struct vfs_impl *vfs_implp;       /* impl specific data */
227     /*
228     * Zones support.  Note that the zone that "owns" the mount isn't
229     * necessarily the same as the zone in which the zone is visible.
230     * That is, vfs_zone and (vfs_zone_next|vfs_zone_prev) may refer to
231     * different zones.

```

```

232     /*
233     struct zone      *vfs_zone;        /* zone that owns the mount */
234     struct vfs      *vfs_zone_next;   /* next VFS visible in zone */
235     struct vfs      *vfs_zone_prev;   /* prev VFS visible in zone */
236
237     struct fem_head *vfs_femhead;     /* fs monitoring */
238     minor_t         vfs_lofi_minor;    /* minor if lofi mount */
239
240     struct fsh_fsrecord *volatile
241         vfs_fshrecord;                /* fs hooking */
242 } vfs_t;
_____unchanged_portion_omitted_____

```