

new/usr/src/cmd/cmd-inet/usr.sbin/snoop/snoop_nlm.c

1

```
*****
23855 Fri Aug 16 09:04:27 2013
new/usr/src/cmd/cmd-inet/usr.sbin/snoop/snoop_nlm.c
195 Need replacement for nfs/lockd+klm
*****
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, Version 1.0 only
6  * (the "License"). You may not use this file except in compliance
7  * with the License.
8  *
9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 * or http://www.opensolaris.org/os/licensing.
11 * See the License for the specific language governing permissions
12 * and limitations under the License.
13 *
14 * When distributing Covered Code, include this CDDL HEADER in each
15 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 * If applicable, add the following below this CDDL HEADER, with the
17 * fields enclosed by brackets "[]" replaced with your own identifying
18 * information: Portions Copyright [yyyy] [name of copyright owner]
19 *
20 * CDDL HEADER END
21 */
22 /*
23  * Copyright (c) 1991, 1998, 2001 by Sun Microsystems, Inc.
24  * All rights reserved.
25  */
27 /*
28  * Copyright 2011 Nexenta Systems, Inc. All rights reserved.
29  * Copyright (c) 2012 by Delphix. All rights reserved.
30  */
32 #include <sys/types.h>
33 #include <setjmp.h>
34 #include <string.h>
36 #ifdef notdef
37 #include <rpc/xdr.h>
38 #include <rpc/auth.h>
39 #include <rpc/rpc_msg.h>
40 #endif /* notdef */
41 #include <rpcsvc/nlm_prot.h>
42 #include "snoop.h"
44 extern char *dlc_header;
45 extern jmp_buf xdr_err;
47 extern void check_retransmit();
48 static void interpret_nlm_1();
49 static void interpret_nlm_3();
50 static void interpret_nlm_4();
51 static char *nameof_access();
52 static char *nameof_mode();
53 static char *nameof_stat();
54 static char *nameof_stat4();
55 static void show_cancargs();
56 static void show_cancargs4();
57 static void show_lock();
58 static void show_lock4();
59 static void show_lockargs();
60 static void show_lockargs4();
61 static void show_netobj();
```

new/usr/src/cmd/cmd-inet/usr.sbin/snoop/snoop_nlm.c

2

```
62 static void show_nlm_access();
63 static void show_nlm_mode();
64 static void show_notify();
65 static void show_res();
66 static void show_res4();
67 static void show_share();
68 static void show_shareargs();
69 static void show_sharereres();
70 static void show_sharereres4();
71 static enum nlm_stats show_stat();
72 static enum nlm4_stats show_stat4();
73 static void show_testargs();
74 static void show_testargs4();
75 static void show_testres();
76 static void show_testres4();
77 static void show_unlockargs();
78 static void show_unlockargs4();
79 static void skip_netobj();
80 static char *sum_lock();
81 static char *sum_lock4();
82 static char *sum_netobj();
83 static char *sum_notify();
84 static char *sum_share();
86 void
87 interpret_nlm(flags, type, xid, vers, proc, data, len)
88     int flags, type, xid, vers, proc;
89     char *data;
90     int len;
91 {
92     switch (vers) {
93     case 1: interpret_nlm_1(flags, type, xid, vers, proc, data, len);
94             break;
95     case 3: interpret_nlm_3(flags, type, xid, vers, proc, data, len);
96             break;
97     case 4: interpret_nlm_4(flags, type, xid, vers, proc, data, len);
98             break;
99     }
100 }
unchanged_portion_omitted
796 /* Maximum procedure number for version 4. */
797 #define MAXPROC_4 23
799 /* ARGSUSED */
800 static void
801 interpret_nlm_4(flags, type, xid, vers, proc, data, len)
802     int flags, type, xid, vers, proc;
803     char *data;
804     int len;
805 {
806     char *line;
807     char *pl;
808     ulong_t i;
810     if (proc < 0 || proc > MAXPROC_4)
811         return;
813     if (flags & F_SUM) {
814         if (setjmp(xdr_err)) {
815             return;
816         }
818         line = get_sum_line();
820         if (type == CALL) {
```

```

821         (void) sprintf(line,
822             "NLM C %s",
823             procnames_short_4[proc]);
824         line += strlen(line);
825         switch (proc) {
826         case NLM4_TEST:
827         case NLM4_GRANTED:
828         case NLM4_TEST_MSG:
829         case NLM4_GRANTED_MSG:
830         case NLMPROC4_TEST:
831         case NLMPROC4_GRANTED:
832         case NLMPROC4_TEST_MSG:
833         case NLMPROC4_GRANTED_MSG:
834             /* testargs */
835             (void) strcat(line, sum_netobj("OH"));
836             (void) getxdr_bool(); /* Excl */
837             (void) strcat(line, sum_lock4());
838             break;
839         case NLM4_LOCK:
840         case NLM4_LOCK_MSG:
841         case NLMPROC4_LOCK:
842         case NLMPROC4_LOCK_MSG:
843             /* lockargs */
844             (void) strcat(line, sum_netobj("OH"));
845             (void) getxdr_bool(); /* Block */
846             (void) getxdr_bool(); /* Excl */
847             (void) strcat(line, sum_lock4());
848             /* ignore reclaim, state fields */
849             break;
850         case NLM4_CANCEL:
851         case NLM4_CANCEL_MSG:
852         case NLMPROC4_CANCEL:
853         case NLMPROC4_CANCEL_MSG:
854             /* cancelargs */
855             (void) strcat(line, sum_netobj("OH"));
856             (void) getxdr_bool(); /* Block */
857             (void) getxdr_bool(); /* Excl */
858             (void) strcat(line, sum_lock4());
859             break;
860         case NLM4_UNLOCK:
861         case NLM4_UNLOCK_MSG:
862         case NLMPROC4_UNLOCK:
863         case NLMPROC4_UNLOCK_MSG:
864             /* unlockargs */
865             (void) strcat(line, sum_netobj("OH"));
866             (void) strcat(line, " ");
867             (void) strcat(line,
868                 nameof_stat4(getxdr_u_long()));
869             break;
870         case NLM4_LOCK_RES:
871         case NLM4_CANCEL_RES:
872         case NLM4_UNLOCK_RES:
873         case NLM4_GRANTED_RES:
874         case NLMPROC4_LOCK_RES:
875         case NLMPROC4_CANCEL_RES:
876         case NLMPROC4_UNLOCK_RES:
877         case NLMPROC4_GRANTED_RES:
878             /* res */
879             (void) strcat(line, sum_netobj("OH"));
880             (void) strcat(line, " ");

```

```

881         (void) strcat(line,
882             nameof_stat4(getxdr_u_long()));
883         break;
884         case NLM4_SHARE:
885         case NLM4_UNSHARE:
886         case NLMPROC4_SHARE:
887         case NLMPROC4_UNSHARE:
888             (void) strcat(line, sum_netobj("OH"));
889             (void) strcat(line, sum_share());
890             break;
891         case NLM4_NM_LOCK:
892         case NLMPROC4_NM_LOCK:
893             /* lockargs */
894             skip_netobj(); /* Cookie */
895             (void) getxdr_bool(); /* Block */
896             (void) getxdr_bool(); /* Excl */
897             (void) strcat(line, sum_lock4());
898             /* skip reclaim & state fields */
899             break;
900         case NLM4_FREE_ALL:
901         case NLMPROC4_FREE_ALL:
902             (void) sprintf(line,
903                 " %s", sum_notify());
904             break;
905     }
906     check_retransmit(line, (ulong_t)xid);
907 } else {
908     (void) sprintf(line, "NLM R %s",
909         procnames_short_4[proc]);
910     line += strlen(line);
911     switch (proc) {
912     case NLM4_TEST:
913     case NLMPROC4_TEST:
914         /* testres */
915         (void) strcat(line, sum_netobj("OH"));
916         (void) strcat(line, " ");
917         (void) strcat(line,
918             nameof_stat4(getxdr_u_long()));
919         break;
920     case NLM4_LOCK:
921     case NLM4_CANCEL:
922     case NLM4_UNLOCK:
923     case NLM4_GRANTED:
924     case NLM4_NM_LOCK:
925     case NLMPROC4_LOCK:
926     case NLMPROC4_CANCEL:
927     case NLMPROC4_UNLOCK:
928     case NLMPROC4_GRANTED:
929     case NLMPROC4_NM_LOCK:
930         /* res */
931         (void) strcat(line, sum_netobj("OH"));
932         (void) strcat(line, " ");
933         (void) strcat(line,
934             nameof_stat4(getxdr_u_long()));
935         break;
936     case NLM4_SHARE:
937     case NLM4_UNSHARE:
938     case NLMPROC4_SHARE:
939     case NLMPROC4_UNSHARE:
940         /* shareres */
941         pl = sum_netobj("OH");
942         i = getxdr_u_long();
943         sprintf(line, "%s %s %ld",
944             pl, nameof_stat4(i), getxdr_long());
945         break;
946     case NLM4_FREE_ALL:

```



```
1094     case nlm4_denied_grace_period: return ("denied (grace period)");
1095     case nlm4_deadlck:           return ("deadlock");
1096     case nlm4_rofs:             return ("read-only fs");
1097     case nlm4_stale_fh:        return ("stale fh");
1098     case nlm4_fbig:           return ("file too big");
1099     case nlm4_failed:         return ("failed");
1085     case NLM4_GRANTED:        return ("granted");
1086     case NLM4_DENIED:         return ("denied");
1087     case NLM4_DENIED_NOLOCKS: return ("denied (no locks)");
1088     case NLM4_BLOCKED:        return ("blocked");
1089     case NLM4_DENIED_GRACE_PERIOD: return ("denied (grace period)");
1090     case NLM4_DEADLCK:        return ("deadlock");
1091     case NLM4_ROFS:           return ("read-only fs");
1092     case NLM4_STALE_FH:       return ("stale fh");
1093     case NLM4_FBIG:           return ("file too big");
1094     case NLM4_FAILED:         return ("failed");
1100     default:                  return ("?");
1101     }
1102 }
_____unchanged_portion_omitted_____
```

new/usr/src/cmd/fs.d/nfs/Makefile

1

1801 Fri Aug 16 09:04:29 2013

new/usr/src/cmd/fs.d/nfs/Makefile

195 Need replacement for nfs/lockd+klm

```
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 2009 Sun Microsystems, Inc. All rights reserved.
24 # Use is subject to license terms.
25 #
26 # cmd/fs.d/nfs/Makefile
27 #
28 # cmd/fs.d/nfs is the directory of all nfs specific commands
29 # whose executable reside in $(INSDIR1) and $(INSDIR2).
30 #
```

```
32 include $(SRC)/Makefile.master
```

```
34 SUBDIR1=      exportfs nfsd rquotad \
35               statd nfsstat mountd dfshares \
36               nfsfind nfs4cbd share
```

```
38 # These do "make catalog"
```

```
39 SUBDIR2=      clear_locks lockd umount showmount \
37 SUBDIR2=      clear_locks umount showmount \
40               mount dfmounts nfslog nfsmapid \
41               nfsref rp_basic
```

```
43 SUBDIR3=      etc svc
44 SUBDIRS=      $(SUBDIR1) $(SUBDIR2) $(SUBDIR3)
```

```
46 # for messaging catalog files
```

```
47 #
48 POFILES=      $(SUBDIR2:%=%/.po)
49 POFILE=       nfs.po
```

```
49 LOCKD=        $(CLOSED)/cmd/fs.d/nfs/lockd
50 $(CLOSED_BUILD)CLOSED_SUBDIR2= $(LOCKD)
51 $(CLOSED_BUILD)POFILES +=      $(LOCKD)/lockd.po
52 $(CLOSED_BUILD)SUBDIRS +=      $(CLOSED_SUBDIR2)
```

```
51 all:=         TARGET= all
52 install:=     TARGET= install
53 clean:=       TARGET= clean
54 clobber:=     TARGET= clobber
55 lint:=        TARGET= lint
```

new/usr/src/cmd/fs.d/nfs/Makefile

2

```
56 catalog:=     TARGET= catalog
```

```
58 .KEEP_STATE:
```

```
60 .PARALLEL:    $(SUBDIRS)
```

```
62 all install clean clobber lint: $(SUBDIRS)
```

```
64 catalog: $(SUBDIR2)
```

```
67 catalog: $(SUBDIR2) $(CLOSED_SUBDIR2)
```

```
65 $(RM) $(POFILE)
```

```
66 cat $(POFILES) > $(POFILE)
```

```
68 $(SUBDIRS): FRC
```

```
69 @cd $@; pwd; $(MAKE) $(TARGET)
```

```
71 FRC:
```

new/usr/src/cmd/fs.d/nfs/lib/nfs_tbind.c

1

```
*****
44643 Fri Aug 16 09:04:30 2013
new/usr/src/cmd/fs.d/nfs/lib/nfs_tbind.c
195 Need replacement for nfs/lockd+klm
*****
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) 1996, 2010, Oracle and/or its affiliates. All rights reserved.
23 */
24 /*
24  * Copyright 2011 Nexenta Systems, Inc. All rights reserved.
25  * Copyright (c) 2012 by Delphix. All rights reserved.
26 */

29 /*
30  * nfs_tbind.c, common part for nfsd and lockd.
31 */

33 #include <tiuser.h>
34 #include <fcntl.h>
35 #include <netconfig.h>
36 #include <stropts.h>
37 #include <errno.h>
38 #include <syslog.h>
39 #include <rpc/rpc.h>
40 #include <sys/time.h>
41 #include <sys/resource.h>
42 #include <signal.h>
43 #include <netdir.h>
44 #include <unistd.h>
45 #include <string.h>
46 #include <netinet/tcp.h>
47 #include <malloc.h>
48 #include <stdlib.h>
49 #include "nfs_tbind.h"
50 #include <nfs/nfs.h>
51 #include <nfs/nfs_acl.h>
52 #include <nfs/nfssys.h>
53 #include <nfs/nfs4.h>
54 #include <zone.h>
55 #include <sys/socket.h>
56 #include <tsol/label.h>

58 /*
59  * Determine valid semantics for most applications.
```

new/usr/src/cmd/fs.d/nfs/lib/nfs_tbind.c

2

```
60 */
61 #define OK_TPI_TYPE(_nconf) \
62     (_nconf->nc_semantics == NC_TPI_CLTS || \
63      _nconf->nc_semantics == NC_TPI_COTS || \
64      _nconf->nc_semantics == NC_TPI_COTS_ORD)

66 #define BE32_TO_U32(a) \
67     (((ulong_t)((uchar_t *)a)[0] & 0xFF) << (ulong_t)24) | \
68     (((ulong_t)((uchar_t *)a)[1] & 0xFF) << (ulong_t)16) | \
69     (((ulong_t)((uchar_t *)a)[2] & 0xFF) << (ulong_t)8) | \
70     ((ulong_t)((uchar_t *)a)[3] & 0xFF)

72 /*
73  * Number of elements to add to the poll array on each allocation.
74 */
75 #define POLL_ARRAY_INC_SIZE    64

77 /*
78  * Number of file descriptors by which the process soft limit may be
79  * increased on each call to nofile_increas(0).
80 */
81 #define NOFILE_INC_SIZE 64

83 /*
84  * Default TCP send and receive buffer size of NFS server.
85 */
86 #define NFSD_TCP_BUFSZ    (1024*1024)

88 struct conn_ind {
89     struct conn_ind *conn_next;
90     struct conn_ind *conn_prev;
91     struct t_call    *conn_call;
92 };
_____unchanged_portion_omitted_____

1690 #include <netinet/in.h>

1692 /*
1693  * Create an address mask appropriate for the transport.
1694  * The mask is used to obtain the host-specific part of
1695  * a network address when comparing addresses.
1696  * For an internet address the host-specific part is just
1697  * the 32 bit IP address and this part of the mask is set
1698  * to all-ones. The port number part of the mask is zeroes.
1699 */
1700 static int
1701 set_addrmask(int fd,
1702              struct netconfig *nconf,
1703              struct netbuf *mask)
1701 set_addrmask(fd, nconf, mask)
1702     struct netconfig *nconf;
1703     struct netbuf *mask;
1704 {
1705     struct t_info info;

1707     /*
1708      * Find the size of the address we need to mask.
1709      */
1710     if (t_getinfo(fd, &info) < 0) {
1711         t_error("t_getinfo");
1712         return (-1);
1713     }
1714     mask->len = mask->maxlen = info.addr;
1715     if (info.addr <= 0) {
1716         /*
1717          * loopback devices have infinite addr size
```

```
1718     * (it is identified by -1 in addr field of t_info structure),
1719     * so don't build the netmask for them. It's a special case
1720     * that should be handled properly.
1721     */
1722     if ((info.addr == -1) &&
1723         (0 == strcmp(nconf->nc_protofmly, NC_LOOPBACK))) {
1724         memset(mask, 0, sizeof (*mask));
1725         return (0);
1726     }
1727
1728     syslog(LOG_ERR, "set_addrmask: address size: %ld", info.addr);
1729     syslog(LOG_ERR, "set_addrmask: address size: %ld",
1730            info.addr);
1731     return (-1);
1732 }
1733
1734 mask->buf = (char *)malloc(mask->len);
1735 if (mask->buf == NULL) {
1736     syslog(LOG_ERR, "set_addrmask: no memory");
1737     return (-1);
1738 }
1739 (void) memset(mask->buf, 0, mask->len); /* reset all mask bits */
1740
1741 if (strcmp(nconf->nc_protofmly, NC_INET) == 0) {
1742     /*
1743      * Set the mask so that the port is ignored.
1744      */
1745     /* LINTED pointer alignment */
1746     ((struct sockaddr_in *)mask->buf)->sin_addr.s_addr =
1747         (ulong_t)~0;
1748     /* LINTED pointer alignment */
1749     ((struct sockaddr_in *)mask->buf)->sin_family =
1750         (ushort_t)~0;
1751 } else if (strcmp(nconf->nc_protofmly, NC_INET6) == 0) {
1752     /* LINTED pointer alignment */
1753     (void) memset(&((struct sockaddr_in6 *)mask->buf)->sin6_addr,
1754                 (uchar_t)~0, sizeof (struct in6_addr));
1755     /* LINTED pointer alignment */
1756     ((struct sockaddr_in6 *)mask->buf)->sin6_family =
1757         (ushort_t)~0;
1758 } else {
1759     /*
1760      * Set all mask bits.
1761      */
1762     (void) memset(mask->buf, 0xFF, mask->len);
1763 }
1764 return (0);
1765 }
1766 }
1767 }
1768 }
1769 }
1770 }
1771 }
1772 }
1773 }
1774 }
1775 }
1776 }
1777 }
1778 }
1779 }
1780 }
1781 }
1782 }
1783 }
1784 }
1785 }
1786 }
1787 }
1788 }
1789 }
1790 }
1791 }
1792 }
1793 }
1794 }
1795 }
1796 }
1797 }
1798 }
1799 }
1800 }
1801 }
1802 }
1803 }
1804 }
1805 }
1806 }
1807 }
1808 }
1809 }
1810 }
1811 }
1812 }
1813 }
1814 }
1815 }
1816 }
1817 }
1818 }
1819 }
1820 }
1821 }
1822 }
1823 }
1824 }
1825 }
1826 }
1827 }
1828 }
1829 }
1830 }
1831 }
1832 }
1833 }
1834 }
1835 }
1836 }
1837 }
1838 }
1839 }
1840 }
1841 }
1842 }
1843 }
1844 }
1845 }
1846 }
1847 }
1848 }
1849 }
1850 }
1851 }
1852 }
1853 }
1854 }
1855 }
1856 }
1857 }
1858 }
1859 }
1860 }
1861 }
1862 }
1863 }
1864 }
1865 }
1866 }
1867 }
1868 }
1869 }
1870 }
1871 }
1872 }
1873 }
1874 }
1875 }
1876 }
1877 }
1878 }
1879 }
1880 }
1881 }
1882 }
1883 }
1884 }
1885 }
1886 }
1887 }
1888 }
1889 }
1890 }
1891 }
1892 }
1893 }
1894 }
1895 }
1896 }
1897 }
1898 }
1899 }
1900 }
1901 }
1902 }
1903 }
1904 }
1905 }
1906 }
1907 }
1908 }
1909 }
1910 }
1911 }
1912 }
1913 }
1914 }
1915 }
1916 }
1917 }
1918 }
1919 }
1920 }
1921 }
1922 }
1923 }
1924 }
1925 }
1926 }
1927 }
1928 }
1929 }
1930 }
1931 }
1932 }
1933 }
1934 }
1935 }
1936 }
1937 }
1938 }
1939 }
1940 }
1941 }
1942 }
1943 }
1944 }
1945 }
1946 }
1947 }
1948 }
1949 }
1950 }
1951 }
1952 }
1953 }
1954 }
1955 }
1956 }
1957 }
1958 }
1959 }
1960 }
1961 }
1962 }
1963 }
1964 }
1965 }
1966 }
1967 }
1968 }
1969 }
1970 }
1971 }
1972 }
1973 }
1974 }
1975 }
1976 }
1977 }
1978 }
1979 }
1980 }
1981 }
1982 }
1983 }
1984 }
1985 }
1986 }
1987 }
1988 }
1989 }
1990 }
1991 }
1992 }
1993 }
1994 }
1995 }
1996 }
1997 }
1998 }
1999 }
2000 }
```

unchanged_portion_omitted_

```

*****
10383 Fri Aug 16 09:04:31 2013
new/usr/src/cmd/fs.d/nfs/lib/smfcfg.c
195 Need replacement for nfs/lockd+klm
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
24  * Copyright 2011 Nexenta Systems, Inc. All rights reserved.
25  */
26 #include <stdio.h>
27 #include <stdlib.h>
28 #include <syslog.h>
29 #include <stdarg.h>
30 #include "smfcfg.h"

32 fs_smfhandle_t *
33 fs_smf_init(char *fmri, char *instance)
34 {
35     fs_smfhandle_t *handle = NULL;
36     char *svcname, srv[MAXPATHLEN];

38     /*
39      * svc name is of the form svc://network/fs/server:instancel
40      * FMRI portion is /network/fs/server
41      */
42     snprintf(srv, MAXPATHLEN, "%s", fmri + strlen("svc:/"));
43     svcname = strrchr(srv, ':');
44     if (svcname != NULL)
45         *svcname = '\\0';
46     svcname = srv;

48     handle = calloc(1, sizeof (fs_smfhandle_t));
49     if (handle != NULL) {
50         handle->fs_handle = scf_handle_create(SCF_VERSION);
51         if (handle->fs_handle == NULL)
52             goto out;
53         if (scf_handle_bind(handle->fs_handle) != 0)
54             goto out;
55         handle->fs_service =
56             scf_service_create(handle->fs_handle);
57         handle->fs_scope =
58             scf_scope_create(handle->fs_handle);
59         if (scf_handle_get_local_scope(handle->fs_handle,
60             handle->fs_scope) != 0)
61             goto out;

```

```

62         if (scf_scope_get_service(handle->fs_scope,
63             svcname, handle->fs_service) != SCF_SUCCESS) {
64             goto out;
65         }
66         handle->fs_pg =
67             scf_pg_create(handle->fs_handle);
68         handle->fs_instance =
69             scf_instance_create(handle->fs_handle);
70         handle->fs_property =
71             scf_property_create(handle->fs_handle);
72         handle->fs_value =
73             scf_value_create(handle->fs_handle);
74     } else {
75         fprintf(stderr,
76             gettext("Cannot access SMF repository: %s\n"), fmri);
77     }
78     return (handle);

80 out:
81     fs_smf_fini(handle);
82     fprintf(stderr, gettext("SMF Initialization problems..%s\n"), fmri);
83     return (NULL);
84 }

unchanged_portion_omitted

363 /* Get an integer (base 10) property */
364 int
365 nfs_smf_get_iprop(char *prop_name, int *rvp, char *instance,
366     scf_type_t sctype, char *svc_name)
367 {
368     char propbuf[32];
369     int bufisz, rc, val;

371     bufisz = sizeof (propbuf);
372     rc = fs_smf_get_prop(NFS_SMF, prop_name, propbuf,
373         instance, sctype, svc_name, &bufisz);
374     if (rc != SA_OK)
375         return (rc);
376     errno = 0;
377     val = strtol(propbuf, NULL, 10);
378     if (errno != 0)
379         return (SA_BAD_VALUE);
380     *rvp = val;
381     return (SA_OK);
382 }

384 int
385 nfs_smf_set_prop(char *prop_name, char *value, char *instance,
386     scf_type_t type, char *svc_name)
387 {
388     return (fs_smf_set_prop(NFS_SMF, prop_name, value, instance,
389         type, svc_name));
390 }

unchanged_portion_omitted

```



```
*****
3044 Fri Aug 16 09:04:32 2013
new/usr/src/cmd/fs.d/nfs/lib/smfcfg.h
195 Need replacement for nfs/lockd+klm
*****
_____unchanged_portion_omitted_

65 #define DEFAULT_INSTANCE      "default"

67 /*
68  * NFS Property Group names.
69  */
70 #define SMF_PG_NFSPROPS        ((const char *)"com.oracle.nfs,props")
71 #define NFS_PROPS_PGNAME      ((const char *)"nfs-props")
72 #define SVC_NFS_CLIENT        "svc:/network/nfs/client"

74 /*
75  * AUTOFS Property Group Names.
76  */
77 #define SMF_PG_AUTOFS        ((const char *)"com.oracle.autofs,props")
78 #define AUTOFS_PROPS_PGNAME  ((const char *)"autofs-props")

80 #define AUTOFS_FMRI            "svc:/system/filesystem/autofs"
81 #define AUTOFS_DEFAULT_FMRI   "svc:/system/filesystem/autofs:default"
82 #define MAXDIGITS              32

84 /*
85  * ERRORS
86  */
87 #define SMF_OK                  0
88 #define SMF_SYSTEM_ERR        -1
89 #define STATE_INITIALIZING     1
90 #define SMF_NO_PERMISSION     2
91 #define SMF_NO_PGTYPE         3

93 extern int nfs_smf_get_iprop(char *, int *, char *, scf_type_t, char *);
94 extern int nfs_smf_get_prop(char *, char *, char *, scf_type_t, char *, int *);
95 extern int fs_smf_get_prop(smf_fstype_t, char *, char *, char *, scf_type_t,
96     char *, int *);
97 extern int nfs_smf_set_prop(char *, char *, char *, scf_type_t, char *);
98 extern int fs_smf_set_prop(smf_fstype_t, char *, char *,
99     char *, scf_type_t, char *);
100 extern int autofs_smf_set_prop(char *, char *, char *, scf_type_t, char *);
101 extern int autofs_smf_get_prop(char *, char *, char *, scf_type_t,
102     char *, int *);
103 extern void fs_smf_fini(fs_smfhandle_t *);
104 extern boolean_t string_to_boolean(const char *);

106 #ifdef __cplusplus
107 }
_____unchanged_portion_omitted_
```

```
*****
```

```
2092 Fri Aug 16 09:04:33 2013
```

```
new/usr/src/cmd/fs.d/nfs/lockd/Makefile
```

```
195 Need replacement for nfs/lockd+klm
```

```
*****
```

```
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) 1990, 2010, Oracle and/or its affiliates. All rights reserved.
23 # Copyright (c) 2012 by Delphix. All rights reserved.
24 #
```

```
26 FSTYPE = nfs
27 TYPEPROG = lockd
28 ATTKM = $(TYPEPROG)
```

```
30 include ../../Makefile.fstype
```

```
32 LOCAL = lockd.o
33 OBJS = $(LOCAL) daemon.o nfs_tbind.o smfcfg.o thrpool.o
```

```
35 POFILE = lockd.po
```

```
37 SRCS = $(LOCAL:%o=%c) ../lib/daemon.c ../lib/nfs_tbind.c \
38 ../lib/smfcfg.c ../lib/thrpool.c
39 LDLIBS += -lnsl -lscf
40 CPPFLAGS += -I../lib
41 C99MODE = $(C99_ENABLE)
```

```
43 CERRWARN += _gcc=-Wno-parentheses
44 CERRWARN += _gcc=-Wno-switch
45 CERRWARN += _gcc=-Wno-unused-variable
46 CERRWARN += _gcc=-Wno-uninitialized
```

```
48 $(TYPEPROG): $(OBJS)
49 $(LINK.c) -o $@ $(OBJS) $(LDLIBS)
50 $(POST_PROCESS)
```

```
52 lockd.o: lockd.c
53 $(COMPILE.c) lockd.c
```

```
55 nfs_tbind.o: ../lib/nfs_tbind.c
56 $(COMPILE.c) ../lib/nfs_tbind.c
```

```
58 thrpool.o: ../lib/thrpool.c
59 $(COMPILE.c) ../lib/thrpool.c
```

```
61 daemon.o: ../lib/daemon.c
```

```
62 $(COMPILE.c) ../lib/daemon.c
```

```
64 smfcfg.o: ../lib/smfcfg.c
65 $(COMPILE.c) ../lib/smfcfg.c
```

```
67 #
68 # message catalog
69 #
70 catalog: $(POFILE)
```

```
72 $(POFILE): $(SRCS)
73 $(RM) $@
74 $(COMPILE.cpp) $(SRCS) > $(POFILE).i
75 $(XGETTEXT) $(XGETFLAGS) $(POFILE).i
76 sed "/^domain/d" messages.po > $@
77 $(RM) $(POFILE).i messages.po
```

```
79 lint:
80 $(LINT.c) $(SRCS) $(LDLIBS)
```

```
82 clean:
83 $(RM) $(OBJS) $(DOBJ)
```

```

*****
12718 Fri Aug 16 09:04:34 2013
new/usr/src/cmd/fs.d/nfs/lockd/lockd.c
195 Need replacement for nfs/lockd+klm
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright (c) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
24  * Copyright 2011 Nexenta Systems, Inc. All rights reserved.
25  * Copyright (c) 2012 by Delphix. All rights reserved.
26  */

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

31 /*
32  * University Copyright- Copyright (c) 1982, 1986, 1988
33  * The Regents of the University of California
34  * All Rights Reserved
35  *
36  * University Acknowledgment- Portions of this document are derived from
37  * software developed by the University of California, Berkeley, and its
38  * contributors.
39  */

41 /* LINTLIBRARY */
42 /* PROTOLIB1 */

44 /*
45  * NLM server
46  *
47  * Most of this copied from ../nfsd/nfsd.c
48  * and then s:NFS:NLM: applied, etc.
49  */

51 #include <sys/param.h>
52 #include <sys/types.h>
53 #include <sys/stat.h>
54 #include <syslog.h>
55 #include <tiuser.h>
56 #include <rpc/rpc.h>
57 #include <errno.h>
58 #include <thread.h>
59 #include <sys/time.h>
60 #include <sys/file.h>
61 #include <nfs/nfs.h>

```

```

62 #include <nfs/nfssys.h>
63 #include <stdio.h>
64 #include <stdio_ext.h>
65 #include <stdlib.h>
66 #include <signal.h>
67 #include <netconfig.h>
68 #include <netdir.h>
69 #include <string.h>
70 #include <unistd.h>
71 #include <stropts.h>
72 #include <sys/tihdr.h>
73 #include <poll.h>
74 #include <priv_utils.h>
75 #include <sys/tiuser.h>
76 #include <netinet/tcp.h>
77 #include <deflt.h>
78 #include <rpcsvc/daemon_utils.h>
79 #include <rpcsvc/nlm_prot.h>
80 #include <libintl.h>
81 #include <libscf.h>
82 #include <libshare.h>
83 #include "nfs_tbind.h"
84 #include "thrpool.h"
85 #include "smfcfg.h"

87 /* Option defaults. See nfssys.h */
88 struct lm_svc_args lmargs = {
89     .version = LM_SVC_CUR_VERS,
90     /* fd, n_fmly, n_proto, n_rdev (below) */
91     .debug = 0,
92     .timeout = 5 * 60,
93     .grace = 60,
94     .retransmittimeout = 15
95 };
96 int max_servers = 20;

99 #define RET_OK          0      /* return code for no error */
100 #define RET_ERR        33     /* return code for error(s) */

102 static int      nlmsvc(int fd, struct netbuf addrmask,
103                      struct netconfig *nconf);
104 static int nlmsvcpool(int max_servers);
105 static void      usage(void);

107 extern int      _nfssys(int, void *);
108 static void sigterm_handler(void);
109 static void shutdown_lockd(void);

111 extern int      daemonize_init(void);
112 extern void     daemonize_fini(int fd);

114 static char     *MyName;

116 /*
117  * We want to bind to these TLI providers, and in this order,
118  * because the kernel NLM needs the loopback first for its
119  * initialization. (It uses it to talk to statd.)
120  */
121 static NETSELDECL(defaultproviders)[] = {
122     "/dev/ticotsord",
123     "/dev/tcp",
124     "/dev/udp",
125     "/dev/tcp6",
126     "/dev/udp6",
127     NULL

```

```

128 };
129
130 /*
131  * The following are all globals used by routines in nfs_tbind.c.
132  */
133 size_t end_listen_fds; /* used by conn_close_oldest() */
134 size_t num_fds = 0; /* used by multiple routines */
135 int listen_backlog = 32; /* used by bind_to_{provider,proto}() */
136 int (*Mysvc)(int, struct netbuf, struct netconfig *) = nlmvc;
137 /* used by cots_listen_event() */
138 int max_conns_allowed = -1; /* used by cots_listen_event() */
139
140 int
141 main(int ac, char *av[])
142 {
143     char *propname = NULL;
144     char *dir = "/";
145     char *provider = (char *)NULL;
146     struct protob *protobp;
147     NETSELDECL(providerp);
148     sigset_t sgset;
149     int i, c, pid, ret, val;
150     int pipe_fd = -1;
151     struct sigaction act;
152
153     MyName = *av;
154
155     /*
156      * Initializations that require more privileges than we need to run.
157      */
158     (void) _create_daemon_lock(LOCKD, DAEMON_UID, DAEMON_GID);
159     svcsetprio();
160
161     if (__init_daemon_priv(PU_RESETGROUPS|PU_CLEARLIMITSET,
162     DAEMON_UID, DAEMON_GID, PRIV_SYS_NFS, NULL) == -1) {
163         (void) fprintf(stderr, "%s should be run with"
164         " sufficient privileges\n", av[0]);
165         exit(1);
166     }
167
168     (void) enable_extended_FILE_stdio(-1, -1);
169
170     /*
171      * Read in the values from SMF first before we check
172      * command line options so the options override SMF values.
173      */
174
175     /* How long to keep idle connections. */
176     propname = "conn_idle_timeout"; /* also -t */
177     ret = nfs_smf_get_iprop(propname, &val,
178     DEFAULT_INSTANCE, SCF_TYPE_INTEGER, LOCKD);
179     if (ret == SA_OK) {
180         if (val <= 0)
181             fprintf(stderr, gettext(
182             "Invalid %s from SMF"), propname);
183         else
184             lmargs.timeout = val;
185     }
186
187     /* Note: debug_level can only be set by args. */
188
189     /* How long to wait for clients to re-establish locks. */
190     propname = "grace_period"; /* also -g */
191     ret = nfs_smf_get_iprop(propname, &val,
192     DEFAULT_INSTANCE, SCF_TYPE_INTEGER, LOCKD);
193     if (ret == SA_OK) {

```

```

194         if (val <= 0)
195             fprintf(stderr, gettext(
196             "Invalid %s from SMF"), propname);
197         else
198             lmargs.grace = val;
199     }
200
201     propname = "listen_backlog"; /* also -l */
202     ret = nfs_smf_get_iprop(propname, &val,
203     DEFAULT_INSTANCE, SCF_TYPE_INTEGER, LOCKD);
204     if (ret == SA_OK) {
205         if (val <= 0)
206             fprintf(stderr, gettext(
207             "Invalid %s from SMF"), propname);
208         else
209             listen_backlog = val;
210     }
211
212     propname = "max_connections"; /* also -c */
213     ret = nfs_smf_get_iprop(propname, &val,
214     DEFAULT_INSTANCE, SCF_TYPE_INTEGER, LOCKD);
215     if (ret == SA_OK) {
216         if (val <= 0)
217             fprintf(stderr, gettext(
218             "Invalid %s from SMF"), propname);
219         else
220             max_conns_allowed = val;
221     }
222
223     propname = "max_servers"; /* also argv[1] */
224     ret = nfs_smf_get_iprop(propname, &val,
225     DEFAULT_INSTANCE, SCF_TYPE_INTEGER, LOCKD);
226     if (ret == SA_OK) {
227         if (val <= 0)
228             fprintf(stderr, gettext(
229             "Invalid %s from SMF"), propname);
230         else
231             max_servers = val;
232     }
233
234     propname = "retrans_timeout"; /* also -r */
235     ret = nfs_smf_get_iprop(propname, &val,
236     DEFAULT_INSTANCE, SCF_TYPE_INTEGER, LOCKD);
237     if (ret == SA_OK) {
238         if (val <= 0)
239             fprintf(stderr, gettext(
240             "Invalid %s from SMF"), propname);
241         else
242             lmargs.retransmittimeout = val;
243     }
244
245     while ((c = getopt(ac, av, "c:d:g:l:r:t:")) != EOF)
246         switch (c) {
247             case 'c': /* max_connections */
248                 if ((val = atoi(optarg)) <= 0)
249                     goto badval;
250                 max_conns_allowed = val;
251                 break;
252
253             case 'd': /* debug */
254                 lmargs.debug = atoi(optarg);
255                 break;
256
257             case 'g': /* grace_period */
258                 if ((val = atoi(optarg)) <= 0)

```

```

260         goto badval;
261         lmargs.grace = val;
262         break;

264     case 'l': /* listen_backlog */
265         if ((val = atoi(optarg)) <= 0)
266             goto badval;
267         listen_backlog = val;
268         break;

270     case 'r': /* retrans_timeout */
271         if ((val = atoi(optarg)) <= 0)
272             goto badval;
273         lmargs.retransmittimeout = val;
274         break;

276     case 't': /* conn_idle_timeout */
277         if ((val = atoi(optarg)) <= 0)
278             goto badval;
279         lmargs.timeout = val;
280         break;

282     badval:
283         fprintf(stderr, gettext(
284             "Invalid -%c option value"), c);
285         /* FALLTHROUGH */
286     default:
287         usage();
288         /* NOTREACHED */
289     }

291 /*
292  * If there is exactly one more argument, it is the number of
293  * servers.
294  */
295 if (optind < ac) {
296     val = atoi(av[optind]);
297     if (val <= 0) {
298         fprintf(stderr, gettext(
299             "Invalid max_servers argument"));
300         usage();
301     }
302     max_servers = val;
303     optind++;
304 }
305 /*
306  * If there are two or more arguments, then this is a usage error.
307  */
308 if (optind != ac)
309     usage();

311 if (lmargs.debug) {
312     printf("%s: debug= %d, conn_idle_timeout= %d,"
313         " grace_period= %d, listen_backlog= %d,"
314         " max_connections= %d, max_servers= %d,"
315         " retrans_timeout= %d\n",
316         MyName, lmargs.debug, lmargs.timeout,
317         lmargs.grace, listen_backlog,
318         max_conns_allowed, max_servers,
319         lmargs.retransmittimeout);
320 }

322 /*
323  * Set current dir to server root
324  */
325 if (chdir(dir) < 0) {

```

```

326         (void) fprintf(stderr, "%s: ", MyName);
327         perror(dir);
328         exit(1);
329     }

331 /* Daemonize, if not debug. */
332 if (lmargs.debug == 0)
333     pipe_fd = daemonize_init();

335     openlog(MyName, LOG_PID | LOG_NDELAY, LOG_DAEMON);

337 /*
338  * establish our lock on the lock file and write our pid to it.
339  * exit if some other process holds the lock, or if there's any
340  * error in writing/locking the file.
341  */
342 pid = _enter_daemon_lock(LOCKD);
343 switch (pid) {
344     case 0:
345         break;
346     case -1:
347         fprintf(stderr, "error locking for %s: %s", LOCKD,
348             strerror(errno));
349         exit(2);
350     default:
351         /* daemon was already running */
352         exit(0);
353 }

355 /*
356  * Block all signals till we spawn other
357  * threads.
358  */
359 (void) sigfillset(&sgset);
360 (void) thr_sigsetmask(SIG_BLOCK, &sgset, NULL);

362 /* Unregister any previous versions. */
363 for (i = NLM_VERS; i < NLM4_VERS; i++) {
364     svc_unreg(NLM_PROG, i);
365 }

367 /*
368  * Set up kernel RPC thread pool for the NLM server.
369  */
370 if (nlmsvcpool(max_servers)) {
371     fprintf(stderr, "Can't set up kernel NLM service: %s. Exiting",
372         strerror(errno));
373     exit(1);
374 }

376 /*
377  * Set up blocked thread to do LWP creation on behalf of the kernel.
378  */
379 if (svcwait(NLM_SVCPPOOL_ID)) {
380     fprintf(stderr, "Can't set up NLM pool creator: %s. Exiting",
381         strerror(errno));
382     exit(1);
383 }

385 /*
386  * Install atexit and sigterm handlers
387  */
388 act.sa_handler = sigterm_handler;
389 act.sa_flags = 0;

391 (void) sigaction(SIGTERM, &act, NULL);

```

```

392     (void) atexit(shutdown_lockd);
393
394     /*
395     * Now open up for signal delivery
396     */
397     (void) thr_sigsetmask(SIG_UNBLOCK, &sgset, NULL);
398
399     /*
400     * Build a protocol block list for registration.
401     */
402     protobp = (struct protob *)malloc(sizeof (struct protob));
403     protobp->serv = "NLM";
404     protobp->versmin = NLM_VERS;
405     protobp->versmax = NLM4_VERS;
406     protobp->program = NLM_PROG;
407     protobp->next = (struct protob *)NULL;
408
409     for (providerp = defaultproviders;
410         *providerp != NULL; providerp++) {
411         provider = *providerp;
412         do_one(provider, NULL, protobp, nlmsvc);
413     }
414
415     free(protobp);
416
417     if (num_fds == 0) {
418         fprintf(stderr, "Could not start NLM service for any protocol."
419                 " Exiting");
420         exit(1);
421     }
422
423     end_listen_fds = num_fds;
424
425     /*
426     * lockd is up and running as far as we are concerned.
427     */
428     if (lmargs.debug == 0)
429         daemonize_fini(pipe_fd);
430
431     /*
432     * Get rid of unneeded privileges.
433     */
434     __fini_daemon_priv(PRIV_PROC_FORK, PRIV_PROC_EXEC, PRIV_PROC_SESSION,
435                      PRIV_FILE_LINK_ANY, PRIV_PROC_INFO, (char *)NULL);
436
437     /*
438     * Poll for non-data control events on the transport descriptors.
439     */
440     poll_for_action();
441
442     /*
443     * If we get here, something failed in poll_for_action().
444     */
445     return (1);
446 }
447
448 static int
449 nlmsvcpool(int maxservers)
450 {
451     struct svcpool_args npa;
452
453     npa.id = NLM_SVCPPOOL_ID;
454     npa.maxthreads = maxservers;
455     npa.redline = 0;
456     npa.qsize = 0;
457     npa.timeout = 0;

```

```

458     npa.stksize = 0;
459     npa.max_same_xprt = 0;
460     return (_nfssys(SVCPOOL_CREATE, &npa));
461 }
462
463 static int
464 ncfmly_to_lmfmly(const char *ncfmly)
465 {
466     if (0 == strcmp(ncfmly, NC_INET))
467         return (LM_INET);
468     if (0 == strcmp(ncfmly, NC_INET6))
469         return (LM_INET6);
470     if (0 == strcmp(ncfmly, NC_LOOPBACK))
471         return (LM_LOOPBACK);
472     return (-1);
473 }
474
475 static int
476 nctype_to_lmprot(uint_t semantics)
477 {
478     switch (semantics) {
479     case NC_TPI_CLTS:
480         return (LM_UDP);
481     case NC_TPI_COTS_ORD:
482         return (LM_TCP);
483     }
484     return (-1);
485 }
486
487 static dev_t
488 ncdev_to_rdev(const char *ncdev)
489 {
490     struct stat st;
491
492     if (stat(ncdev, &st) < 0)
493         return (NODEV);
494     return (st.st_rdev);
495 }
496
497 static void
498 sigterm_handler(void)
499 {
500     /* to call atexit handler */
501     exit(0);
502 }
503
504 static void
505 shutdown_lockd(void)
506 {
507     (void) _nfssys(KILL_LOCKMGR, NULL);
508 }
509
510
511 /*
512 * Establish NLM service thread.
513 */
514 static int
515 nlmsvc(int fd, struct netbuf addrmask, struct netconfig *nconf)
516 {
517     struct lm_svc_args lma;
518
519     lma = lmargs; /* init by struct copy */
520
521     /*
522     * The kernel code needs to reconstruct a complete
523     * knetconfig from n_fmly, n_proto. We use these

```

```
524     * two fields to convey the family and semantics.
525     */
526     lma.fd = fd;
527     lma.n_fmly = ncfmly_to_lmfmly(nconf->nc_protomly);
528     lma.n_proto = nctype_to_lmprot(nconf->nc_semantics);
529     lma.n_rdev = ncdev_to_rdev(nconf->nc_device);
531     return (_nfssys(LM_SVC, &lma));
532 }
534 static void
535 usage(void)
536 {
537     (void) fprintf(stderr, gettext(
538         "usage: %s [options] [max_servers]\n"), MyName);
539     (void) fprintf(stderr, gettext(
540         "options: (see SMF property descriptions)\n"));
541     /* Note: don't translate these */
542     (void) fprintf(stderr, "\t-c max_connections\n");
543     (void) fprintf(stderr, "\t-d debug_level\n");
544     (void) fprintf(stderr, "\t-g grace_period\n");
545     (void) fprintf(stderr, "\t-l listen_backlog\n");
546     (void) fprintf(stderr, "\t-r retrans_timeout\n");
547     (void) fprintf(stderr, "\t-t conn_idle_timeout\n");
549     exit(1);
550 }
```

new/usr/src/cmd/fs.d/nfs/mount/Makefile

1

```
*****  
    3154 Fri Aug 16 09:04:34 2013  
new/usr/src/cmd/fs.d/nfs/mount/Makefile  
195 Need replacement for nfs/lockd+klm  
*****  
_____unchanged_portion_omitted_
```



```

*****
33822 Fri Aug 16 09:04:36 2013
new/usr/src/cmd/fs.d/nfs/statd/sm_proc.c
195 Need replacement for nfs/lockd+klm
*****
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, Version 1.0 only
6  * (the "License"). You may not use this file except in compliance
7  * with the License.
8  *
9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 * or http://www.opensolaris.org/os/licensing.
11 * See the License for the specific language governing permissions
12 * and limitations under the License.
13 *
14 * When distributing Covered Code, include this CDDL HEADER in each
15 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 * If applicable, add the following below this CDDL HEADER, with the
17 * fields enclosed by brackets "[]" replaced with your own identifying
18 * information: Portions Copyright [yyyy] [name of copyright owner]
19 *
20 * CDDL HEADER END
21 */
22 /*
23 * Copyright 2005 Sun Microsystems, Inc. All rights reserved.
24 * Use is subject to license terms.
25 */
26 /*
27 * Copyright 2013 Nexenta Systems, Inc. All rights reserved.
28 * Copyright (c) 2012 by Delphix. All rights reserved.
29 */

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

34 /*
35 * University Copyright- Copyright (c) 1982, 1986, 1988
36 * The Regents of the University of California
37 * All Rights Reserved
38 *
39 * University Acknowledgment- Portions of this document are derived from
40 * software developed by the University of California, Berkeley, and its
41 * contributors.
42 */

44 #include <stdio.h>
45 #include <sys/types.h>
46 #include <stdlib.h>
47 #include <unistd.h>
48 #include <string.h>
49 #include <syslog.h>
50 #include <rpc/rpc.h>
51 #include <rpcsvc/sm_inter.h>
52 #include <rpcsvc/nsm_addr.h>
53 #include <memory.h>
54 #include <net/if.h>
55 #include <sys/socket.h>
56 #include <sys/socket.h>
57 #include <netinet/in.h>
58 #include <arpa/inet.h>
59 #include <netdb.h>
60 #include <netdir.h>
61 #include <synch.h>

```

```

62 #include <thread.h>
63 #include <ifaddrs.h>
64 #include <errno.h>
65 #include <assert.h>
66 #include "sm_statd.h"

68 static int local_state;          /* fake local sm state */
69                                /* client name-to-address translation table */
70 static name_addr_entry_t *name_addr = NULL;

73 #define LOGHOST "loghost"

75 static void delete_mon(char *mon_name, my_id *my_idp);
76 static void insert_mon(mon *monp);
77 static void pr_mon(char *);
78 static int statd_call_lockd(mon *monp, int state);
79 static int hostname_eq(char *host1, char *host2);
80 static char *get_system_id(char *hostname);
81 static void add_aliases(struct hostent *phost);
82 static void *thr_send_notice(void *);
83 static void delete_onemon(char *mon_name, my_id *my_idp,
84                            mon_entry **monitor_q);
85 static void send_notice(char *mon_name, int state);
86 static void add_to_host_array(char *host);
87 static int in_host_array(char *host);
88 static void pr_name_addr(name_addr_entry_t *name_addr);

90 extern int self_check(char *hostname);
91 extern struct lifconf *getmyaddrs(void);

93 /* ARGSUSED */
94 void
95 sm_stat_svc(sm_name *namep, sm_stat_res *resp)
96 {
97     sm_status(namep, resp)
98     sm_name *namep;
99     sm_stat_res *resp;
100
101     if (debug)
102         (void) printf("proc sm_stat: mon_name = %s\n",
103                      namep->mon_name);
104
105     resp->res_stat = stat_succ;
106     resp->state = LOCAL_STATE;
107 }

108 /* ARGSUSED */
109 void
110 sm_mon_svc(mon *monp, sm_stat_res *resp)
111 {
112     sm_mon(monp, resp)
113     mon *monp;
114     sm_stat_res *resp;
115
116     {
117         mon_id *monidp;
118         monidp = &monp->mon_id;
119
120         rw_rdlock(&thr_rwlock);
121         if (debug) {
122             (void) printf("proc sm_mon: mon_name = %s, id = %d\n",
123                          monidp->mon_name, * ((int *)monp->priv));
124             pr_mon(monp->mon_id.mon_name);
125         }
126
127         /* only monitor other hosts */
128         if (self_check(monp->mon_id.mon_name) == 0) {

```

```

122         /* store monitor request into monitor_q */
123         insert_mon(monp);
124     }

126     pr_mon(monp->mon_id.mon_name);
127     resp->res_stat = stat_succ;
128     resp->state = local_state;
129     rw_unlock(&thr_rwlock);
130 }

132 /* ARGSUSED */
133 void
134 sm_unmon_svc(mon_id *monidp, sm_stat *resp)
135 sm_unmon(monidp, resp)
136     mon_id *monidp;
137     sm_stat *resp;
138 {
139     rw_rdlock(&thr_rwlock);
140     if (debug) {
141         (void) printf(
142             "proc sm_unmon: mon_name = %s, [%s, %d, %d, %d]\n",
143             monidp->mon_name, monidp->my_id.my_name,
144             monidp->my_id.my_prog, monidp->my_id.my_vers,
145             monidp->my_id.my_proc);
146     }
147     pr_mon(monidp->mon_name);
148     delete_mon(monidp->mon_name, &monidp->my_id);
149     resp->state = local_state;
150     rw_unlock(&thr_rwlock);
151 }

152 /* ARGSUSED */
153 void
154 sm_unmon_all_svc(my_id *myidp, sm_stat *resp)
155 sm_unmon_all(myidp, resp)
156     my_id *myidp;
157     sm_stat *resp;
158 {
159     rw_rdlock(&thr_rwlock);
160     if (debug) {
161         (void) printf("proc sm_unmon_all: [%s, %d, %d, %d]\n",
162             myidp->my_name,
163             myidp->my_prog, myidp->my_vers,
164             myidp->my_proc);
165     }
166     delete_mon((char *)NULL, myidp);
167     pr_mon(NULL);
168     resp->state = local_state;
169     rw_unlock(&thr_rwlock);
170 }

171 /*
172  * Notifies lockd specified by name that state has changed for this server.
173  */
174 void
175 sm_notify_svc(stat_chge *ntfp)
176 sm_notify(ntfp)
177     stat_chge *ntfp;
178 {
179     rw_rdlock(&thr_rwlock);
180     if (debug) {
181         (void) printf("sm_notify: %s state = %d\n",
182             ntfp->mon_name, ntfp->state);
183     }
184     send_notice(ntfp->mon_name, ntfp->state);
185     rw_unlock(&thr_rwlock);

```

```

180 }

182 /* ARGSUSED */
183 void
184 sm_simu_crash_svc(void *myidp)
185 sm_simu_crash(myidp)
186     void *myidp;
187 {
188     int i;
189     struct mon_entry *monitor_q;
190     int found = 0;

191     /* Only one crash should be running at a time. */
192     mutex_lock(&crash_lock);
193     if (debug) {
194         (void) printf("proc sm_simu_crash\n");
195     }
196     if (in_crash) {
197         cond_wait(&crash_finish, &crash_lock);
198         mutex_unlock(&crash_lock);
199     } else {
200         in_crash = 1;
201     }
202     mutex_unlock(&crash_lock);

203     for (i = 0; i < MAX_HASHSIZE; i++) {
204         mutex_lock(&mon_table[i].lock);
205         monitor_q = mon_table[i].sm_monhdp;
206         if (monitor_q != (struct mon_entry *)NULL) {
207             mutex_unlock(&mon_table[i].lock);
208             found = 1;
209             break;
210         }
211         mutex_unlock(&mon_table[i].lock);
212     }
213     /*
214      * If there are entries found in the monitor table,
215      * initiate a crash, else zero out the in_crash variable.
216      */
217     if (found) {
218         mutex_lock(&crash_lock);
219         die = 1;
220         /* Signal sm_retry() thread if sleeping. */
221         cond_signal(&retrywait);
222         mutex_unlock(&crash_lock);
223         rw_wrlock(&thr_rwlock);
224         sm_crash();
225         rw_unlock(&thr_rwlock);
226     } else {
227         mutex_lock(&crash_lock);
228         in_crash = 0;
229         mutex_unlock(&crash_lock);
230     }
231 }

232 unchanged_portion_omitted

233 /*
234  * Work thread created to do the actual statd_call_lockd
235  */
236 static void *
237 thr_send_notice(void *arg)
238 {
239     moninfo_t *minfop;

240     minfop = (moninfo_t *)arg;

```

```

723     if (statd_call_lockd(&minfop->id, minfop->state) == -1) {
724         if (debug && minfop->id.mon_id.mon_name)
725             (void) printf("problem with notifying %s failure, "
726                 "give up\n", minfop->id.mon_id.mon_name);
727     } else {
728         if (debug)
729             (void) printf("send_notice: %s, %d notified.\n",
730                 minfop->id.mon_id.mon_name, minfop->state);
731     }

733     free(minfop->id.mon_id.mon_name);
734     free(minfop->id.mon_id.my_id.my_name);
735     free(minfop);

737     thr_exit((void *) 0);
738 #ifdef lint
739     /*NOTREACHED*/
740     return ((void *)0);
741 #endif
742 }

744 /*
745  * Contact lockd specified by monp.
746  */
747 static int
748 statd_call_lockd(monp, state)
749     mon *monp;
750     int state;
751 {
752     enum clnt_stat clnt_stat;
753     struct timeval tottimeout;
754     struct sm_status stat;
755     struct status stat;
756     my_idp *my_idp;
757     char *mon_name;
758     int i;
759     int rc = 0;
760     CLIENT *clnt;

761     mon_name = monp->mon_id.mon_name;
762     my_idp = &monp->mon_id.my_id;
763     (void) memset(&stat, 0, sizeof (stat));
764     (void) memset(&stat, 0, sizeof (struct status));
765     stat.mon_name = mon_name;
766     stat.state = state;
767     for (i = 0; i < 16; i++) {
768         stat.priv[i] = monp->priv[i];
769     }
770     if (debug)
771         (void) printf("statd_call_lockd: %s state = %d\n",
772             stat.mon_name, stat.state);

773     tottimeout.tv_sec = SM_RPC_TIMEOUT;
774     tottimeout.tv_usec = 0;

776     clnt = create_client(my_idp->my_name, my_idp->my_prog, my_idp->my_vers,
777         "ticotsord", &tottimeout);
778     if (clnt == NULL) {
779         if ((clnt = create_client(my_idp->my_name, my_idp->my_prog,
780             my_idp->my_vers, &tottimeout)) == (CLIENT *) NULL) {
781             return (-1);
782         }
783     }

782     clnt_stat = clnt_call(clnt, my_idp->my_proc,
783         xdr_sm_status, (char *)&stat,
784         clnt_stat = clnt_call(clnt, my_idp->my_proc, xdr_status, (char *)&stat,

```

```

784         xdr_void, NULL, tottimeout);
785     if (debug) {
786         (void) printf("clnt_stat=%s(%d)\n",
787             clnt_sperrno(clnt_stat), clnt_stat);
788     }
789     if (clnt_stat != (int)RPC_SUCCESS) {
790         syslog(LOG_WARNING,
791             "statd: cannot talk to lockd at %s, %s(%d)\n",
792             my_idp->my_name, clnt_sperrno(clnt_stat), clnt_stat);
793         rc = -1;
794     }

796     clnt_destroy(clnt);
797     return (rc);

799 }

801 /*
802  * Client handle created.
803  */
804 CLIENT *
805 create_client(char *host, int prognum, int versnum, char *netid,
806     struct timeval *utimeout)
807 create_client(host, prognum, versnum, utimeout)
808     char *host;
809     int prognum;
810     int versnum;
811     struct timeval *utimeout;
812 {
813     int fd;
814     struct timeval timeout;
815     CLIENT *client;
816     struct t_info tinfo;

817     if (netid == NULL) {
818         client = clnt_create_timed(host, prognum, versnum,
819             "netpath", utimeout);
820     } else {
821         struct netconfig *nconf;

822         nconf = getnetconfig(netid);
823         if (nconf == NULL) {
824             if ((client = clnt_create_timed(host, prognum, versnum,
825                 "netpath", utimeout)) == NULL) {
826                 return (NULL);
827             }
828         }

829         client = clnt_tp_create_timed(host, prognum, versnum, nconf,
830             utimeout);

831         freenetconfig(nconf);
832     }

833     if (client == NULL) {
834         return (NULL);
835     }

836     (void) CLNT_CONTROL(client, CLGET_FD, (caddr_t)&fd);
837     if (t_getinfo(fd, &tinfo) != -1) {
838         if (tinfo.servtype == T_CLTS) {
839             /*
840              * Set time outs for connectionless case
841              */
842             timeout.tv_usec = 0;
843             timeout.tv_sec = SM_CLTS_TIMEOUT;
844             (void) CLNT_CONTROL(client,

```

```

843             CLSET_RETRY_TIMEOUT, (caddr_t)&timeout);
844         }
845     } else
846         return (NULL);

848     return (client);
849 }
unchanged_portion_omitted

1248 /*
1249  * Compares <family>.<address-specifier> ASCII names for hosts. Returns
1250  * 0 if the addresses match, and 1 if the addresses fail to match.
1251  * If the args are indeed specifiers, they should look like this:
1252  *
1253  *     ipv4.192.9.200.1 or ipv6.::C009:C801
1254  */
1255 int
1256 str_cmp_address_specifier(char *specifier1, char *specifier2)
1257 {
1258     size_t unq_len1, unq_len2;
1259     char *rawaddr1, *rawaddr2;
1260     int af1, af2, len;

1262     if (debug) {
1263         (void) printf("str_cmp_addr: specifier1= %s, specifier2= %s\n",
1264             specifier1, specifier2);
1265     }

1267     /*
1268     * Verify that:
1269     * 1. The family tokens match;
1270     * 2. The IP addresses following the '.' are legal; and
1271     * 3. These addresses match.
1272     */
1273     unq_len1 = strcspn(specifier1, ".");
1274     unq_len2 = strcspn(specifier2, ".");
1275     rawaddr1 = strchr(specifier1, '.');
1276     rawaddr2 = strchr(specifier2, '.');

1278     if (strncmp(specifier1, SM_ADDR_IPV4, unq_len1) == 0) {
1279         af1 = AF_INET;
1280         len = 4;
1281     } else if (strncmp(specifier1, SM_ADDR_IPV6, unq_len1) == 0) {
1282         af1 = AF_INET6;
1283         len = 16;
1284     }
1285     else
1286         return (1);

1288     if (strncmp(specifier2, SM_ADDR_IPV4, unq_len2) == 0)
1289         af2 = AF_INET;
1290     else if (strncmp(specifier2, SM_ADDR_IPV6, unq_len2) == 0)
1291         af2 = AF_INET6;
1292     else
1293         return (1);

1295     if (af1 != af2)
1296         return (1);

1298     if (rawaddr1 != NULL && rawaddr2 != NULL) {
1299         char dst1[16];
1300         char dst2[16];
1301         ++rawaddr1;
1302         ++rawaddr2;

1304         if (inet_pton(af1, rawaddr1, dst1) == 1 &&

```

```

1305         inet_pton(af2, rawaddr1, dst2) == 1 &&
1306         memcmp(dst1, dst2, len) == 0) {
1307             return (0);
1308         }
1309     }
1310     return (1);
1311 }

1313 /*
1314  * Add IP address strings to the host_name list.
1315  */
1316 void
1317 merge_ips(void)
1318 {
1319     struct ifaddrs *ifap, *cifap;
1320     int error;

1322     error = getifaddrs(&ifap);
1323     if (error) {
1324         syslog(LOG_WARNING, "getifaddrs error: '%s'",
1325             strerror(errno));
1326         return;
1327     }

1329     for (cifap = ifap; cifap != NULL; cifap = cifap->ifa_next) {
1330         struct sockaddr *sa = cifap->ifa_addr;
1331         char addr_str[INET6_ADDRSTRLEN];
1332         void *addr = NULL;

1334         switch (sa->sa_family) {
1335             case AF_INET: {
1336                 struct sockaddr_in *sin = (struct sockaddr_in *)sa;

1338                 /* Skip loopback addresses. */
1339                 if (sin->sin_addr.s_addr == htonl(INADDR_LOOPBACK)) {
1340                     continue;
1341                 }

1343                 addr = &sin->sin_addr;
1344                 break;
1345             }

1347             case AF_INET6: {
1348                 struct sockaddr_in6 *sin6 = (struct sockaddr_in6 *)sa;

1350                 /* Skip loopback addresses. */
1351                 if (IN6_IS_ADDR_LOOPBACK(&sin6->sin6_addr)) {
1352                     continue;
1353                 }

1355                 addr = &sin6->sin6_addr;
1356                 break;
1357             }

1359             default:
1360                 syslog(LOG_WARNING, "Unknown address family %d for "
1361                     "interface %s", sa->sa_family, cifap->ifa_name);
1362                 continue;
1363         }

1365         if (inet_ntop(sa->sa_family, addr, addr_str, sizeof (addr_str))
1366             == NULL) {
1367             syslog(LOG_WARNING, "Failed to convert address into "
1368                 "string representation for interface '%s' "
1369                 "address family %d", cifap->ifa_name,
1370                 sa->sa_family);

```

```
1371         continue;
1372     }
1374     if (!in_host_array(addr_str)) {
1375         add_to_host_array(addr_str);
1376     }
1377 }
1379     freeifaddrs(ifap);
1380 }
unchanged_portion_omitted_
```

```

*****
36756 Fri Aug 16 09:04:37 2013
new/usr/src/cmd/fs.d/nfs/statd/sm_statd.c
195 Need replacement for nfs/lockd+klm
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2007 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

26 /*      Copyright (c) 1984, 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 /*
40 * Copyright (c) 2012 by Delphix. All rights reserved.
41 */

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

45 /*
46 * sm_statd.c consists of routines used for the intermediate
47 * statd implementation(3.2 rpc.statd);
48 * it creates an entry in "current" directory for each site that it monitors;
49 * after crash and recovery, it moves all entries in "current"
50 * to "backup" directory, and notifies the corresponding statd of its recovery.
51 */

53 #include <stdio.h>
54 #include <stdlib.h>
55 #include <unistd.h>
56 #include <string.h>
57 #include <syslog.h>
58 #include <netdb.h>
59 #include <sys/types.h>
60 #include <sys/stat.h>
61 #include <sys/file.h>

```

```

62 #include <sys/param.h>
63 #include <arpa/inet.h>
64 #include <dirent.h>
65 #include <rpc/rpc.h>
66 #include <rpcsvc/sm_inter.h>
67 #include <rpcsvc/nsm_addr.h>
68 #include <errno.h>
69 #include <memory.h>
70 #include <signal.h>
71 #include <synch.h>
72 #include <thread.h>
73 #include <limits.h>
74 #include <strings.h>
75 #include "sm_statd.h"

78 int LOCAL_STATE;

80 sm_hash_t      mon_table[MAX_HASHSIZE];
81 static sm_hash_t      record_table[MAX_HASHSIZE];
82 static sm_hash_t      recov_q;

84 static name_entry *find_name(name_entry **namepp, char *name);
85 static name_entry *insert_name(name_entry **namepp, char *name,
86                               int need_alloc);
87 static void delete_name(name_entry **namepp, char *name);
88 static void remove_name(char *name, int op, int startup);
89 static int statd_call_statd(char *name);
90 static void pr_name(char *name, int flag);
91 static void *thr_statd_init();
92 static void *sm_try();
93 static void *thr_call_statd(void *);
94 static void remove_single_name(char *name, char *dir1, char *dir2);
95 static int move_file(char *fromdir, char *file, char *todir);
96 static int count_symlinks(char *dir, char *name, int *count);
97 static char *family2string(sa_family_t family);

99 /*
100 * called when statd first comes up; it searches /etc/sm to gather
101 * all entries to notify its own failure
102 */
103 void
104 statd_init()
105 {
106     struct dirent *dirp;
107     DIR      *dp;
108     FILE *fp, *fp_tmp;
109     int i, tmp_state;
110     char state_file[MAXPATHLEN+SM_MAXPATHLEN];

112     if (debug)
113         (void) printf("enter statd_init\n");

115     /*
116      * First try to open the file.  If that fails, try to create it.
117      * If that fails, give up.
118      */
119     if ((fp = fopen(STATE, "r+")) == (FILE *)NULL)
120         if ((fp = fopen(STATE, "w+")) == (FILE *)NULL) {
121             syslog(LOG_ERR, "can't open %s: %m", STATE);
122             exit(1);
123         } else
124             (void) chmod(STATE, 0644);
125     if ((fscanf(fp, "%d", &LOCAL_STATE)) == EOF) {
126         if (debug >= 2)
127             (void) printf("empty file\n");

```

```

128         LOCAL_STATE = 0;
129     }

131     /*
132     * Scan alternate paths for largest "state" number
133     */
134     for (i = 0; i < pathix; i++) {
135         (void) sprintf(state_file, "%s/statmon/state", path_name[i]);
136         if ((fp_tmp = fopen(state_file, "r+")) == (FILE *)NULL) {
137             if ((fp_tmp = fopen(state_file, "w+"))
138                 == (FILE *)NULL) {
139                 if (debug)
140                     syslog(LOG_ERR,
141                         "can't open %s: %m",
142                         state_file);
143             } else
144                 continue;
145             (void) chmod(state_file, 0644);
146         }
147         if ((fscanf(fp_tmp, "%d", &tmp_state)) == EOF) {
148             if (debug)
149                 syslog(LOG_ERR,
150                     "statd: %s: file empty\n", state_file);
151             (void) fclose(fp_tmp);
152             continue;
153         }
154         if (tmp_state > LOCAL_STATE) {
155             LOCAL_STATE = tmp_state;
156             if (debug)
157                 (void) printf("Update LOCAL STATE: %d\n",
158                             tmp_state);
159         }
160         (void) fclose(fp_tmp);
161     }

163     LOCAL_STATE = ((LOCAL_STATE%2) == 0) ? LOCAL_STATE+1 : LOCAL_STATE+2;

165     /* IF local state overflows, reset to value 1 */
166     if (LOCAL_STATE < 0) {
167         LOCAL_STATE = 1;
168     }

170     /* Copy the LOCAL_STATE value back to all stat files */
171     if (fseek(fp, 0, 0) == -1) {
172         syslog(LOG_ERR, "statd: fseek failed\n");
173         exit(1);
174     }

176     (void) fprintf(fp, "%-10d", LOCAL_STATE);
177     (void) fflush(fp);
178     if (fsync(fileno(fp)) == -1) {
179         syslog(LOG_ERR, "statd: fsync failed\n");
180         exit(1);
181     }
182     (void) fclose(fp);

184     for (i = 0; i < pathix; i++) {
185         (void) sprintf(state_file, "%s/statmon/state", path_name[i]);
186         if ((fp_tmp = fopen(state_file, "r+")) == (FILE *)NULL) {
187             if ((fp_tmp = fopen(state_file, "w+"))
188                 == (FILE *)NULL) {
189                 syslog(LOG_ERR,
190                     "can't open %s: %m", state_file);
191                 continue;
192             } else
193                 (void) chmod(state_file, 0644);

```

```

194     }
195     (void) fprintf(fp_tmp, "%-10d", LOCAL_STATE);
196     (void) fflush(fp_tmp);
197     if (fsync(fileno(fp_tmp)) == -1) {
198         syslog(LOG_ERR,
199             "statd: %s: fsync failed\n", state_file);
200         (void) fclose(fp_tmp);
201         exit(1);
202     }
203     (void) fclose(fp_tmp);
204 }

206     if (debug)
207         (void) printf("local state = %d\n", LOCAL_STATE);

209     if ((mkdir(CURRENT, SM_DIRECTORY_MODE)) == -1) {
210         if (errno != EEXIST) {
211             syslog(LOG_ERR, "statd: mkdir current, error %m\n");
212             exit(1);
213         }
214     }
215     if ((mkdir(BACKUP, SM_DIRECTORY_MODE)) == -1) {
216         if (errno != EEXIST) {
217             syslog(LOG_ERR, "statd: mkdir backup, error %m\n");
218             exit(1);
219         }
220     }

222     /* get all entries in CURRENT into BACKUP */
223     if ((dp = opendir(CURRENT)) == (DIR *)NULL) {
224         syslog(LOG_ERR, "statd: open current directory, error %m\n");
225         exit(1);
226     }

228     while ((dirp = readdir(dp)) != NULL) {
229         if (strcmp(dirp->d_name, ".") != 0 &&
230             strcmp(dirp->d_name, "..") != 0) {
231             /* rename all entries from CURRENT to BACKUP */
232             (void) move_file(CURRENT, dirp->d_name, BACKUP);
233         }
234     }

236     (void) closedir(dp);

238     /* Contact hosts' statd */
239     if (thr_create(NULL, NULL, thr_statd_init, NULL, THR_DETACHED, 0)) {
240         syslog(LOG_ERR,
241             "statd: unable to create thread for thr_statd_init\n");
242         exit(1);
243     }
244 }

    unchanged portion omitted

536 /*
537  * Notifies the statd of host specified by name to indicate that
538  * state has changed for this server.
539  */
540 static int
541 statd_call_statd(name)
542     char *name;
543 {
544     enum clnt_stat clnt_stat;
545     struct timeval tottimeout;
546     CLIENT *clnt;
547     char *name_or_addr;
548     stat_chge ntf;

```

```

549     int i;
550     int rc;
551     int dummy1, dummy2, dummy3, dummy4;
552     char ascii_addr[MAXNAMELEN];
553     size_t unq_len;

555     ntf.mon_name = hostname;
556     ntf.state = LOCAL_STATE;
557     if (debug)
558         (void) printf("statd_call_statd at %s\n", name);

560     /*
561      * If it looks like an ASCII <address family>.<address> specifier,
562      * strip off the family - we just want the address when obtaining
563      * a client handle.
564      * If it's anything else, just pass it on to create_client().
565      */
566     unq_len = strcspn(name, ".");

568     if ((strncmp(name, SM_ADDR_IPV4, unq_len) == 0) ||
569         (strncmp(name, SM_ADDR_IPV6, unq_len) == 0)) {
570         name_or_addr = strchr(name, '.') + 1;
571     } else {
572         name_or_addr = name;
573     }

575     /*
576      * NOTE: We depend here upon the fact that the RPC client code
577      * allows us to use ASCII dotted quad 'names', i.e. "192.9.200.1".
578      * This may change in a future release.
579      */
580     if (debug) {
581         (void) printf("statd_call_statd: calling create_client(%s)\n",
582                     name_or_addr);
583     }

585     tottimeout.tv_sec = SM_RPC_TIMEOUT;
586     tottimeout.tv_usec = 0;

588     if ((clnt = create_client(name_or_addr, SM_PROG, SM_VERS, NULL,
589                             &tottimeout)) == NULL) {
590     if ((clnt = create_client(name_or_addr, SM_PROG, SM_VERS,
591                             &tottimeout)) == (CLIENT *) NULL) {
592         return (-1);
593     }

593     /* Perform notification to client */
594     rc = 0;
595     clnt_stat = clnt_call(clnt, SM_NOTIFY, xdr_stat_chge, (char *)&ntf,
596                          xdr_void, NULL, tottimeout);
597     if (debug) {
598         (void) printf("clnt_stat=%s(%d)\n",
599                     clnt_sperrno(clnt_stat), clnt_stat);
600     }
601     if (clnt_stat != (int)RPC_SUCCESS) {
602         syslog(LOG_WARNING,
603              "statd: cannot talk to statd at %s, %s(%d)\n",
604              name_or_addr, clnt_sperrno(clnt_stat), clnt_stat);
605         rc = -1;
606     }

608     /* For HA systems and multi-homed hosts */
609     ntf.state = LOCAL_STATE;
610     for (i = 0; i < addrix; i++) {
611         ntf.mon_name = host_name[i];
612         if (debug)

```

```

613         (void) printf("statd_call_statd at %s\n", name_or_addr);
614         clnt_stat = clnt_call(clnt, SM_NOTIFY, xdr_stat_chge,
615                              (char *)&ntf, xdr_void, NULL,
616                              tottimeout);
617         if (clnt_stat != (int)RPC_SUCCESS) {
618             syslog(LOG_WARNING,
619                  "statd: cannot talk to statd at %s, %s(%d)\n",
620                  name_or_addr, clnt_sperrno(clnt_stat), clnt_stat);
621             rc = -1;
622         }
623     }
624     clnt_destroy(clnt);
625     return (rc);
626 }
_____unchanged_portion_omitted_

```



```

*****
6976 Fri Aug 16 09:04:38 2013
new/usr/src/cmd/fs.d/nfs/statd/sm_statd.h
195 Need replacement for nfs/lockd+klm
*****
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, Version 1.0 only
6  * (the "License"). You may not use this file except in compliance
7  * with the License.
8  *
9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 * or http://www.opensolaris.org/os/licensing.
11 * See the License for the specific language governing permissions
12 * and limitations under the License.
13 *
14 * When distributing Covered Code, include this CDDL HEADER in each
15 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 * If applicable, add the following below this CDDL HEADER, with the
17 * fields enclosed by brackets "[]" replaced with your own identifying
18 * information: Portions Copyright [yyyy] [name of copyright owner]
19 *
20 * CDDL HEADER END
21 */
22 /*
23 * Copyright 2005 Sun Microsystems, Inc. All rights reserved.
24 * Use is subject to license terms.
25 */
27 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
28 /*      All Rights Reserved */

30 /*
31 * University Copyright- Copyright (c) 1982, 1986, 1988
32 * The Regents of the University of California
33 * All Rights Reserved
34 *
35 * University Acknowledgment- Portions of this document are derived from
36 * software developed by the University of California, Berkeley, and its
37 * contributors.
38 */

40 /*
41 * Copyright 2011 Nexenta Systems, Inc. All rights reserved.
42 * Copyright (c) 2012 by Delphix. All rights reserved.
43 */

45 #ifndef _SM_STATD_H
46 #define _SM_STATD_H

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

48 #ifdef __cplusplus
49 extern "C" {
50 #endif

52 /* Limit defines */
53 #define SM_DIRECTORY_MODE 00755
54 #define MAX_HASHSIZE 50
55 #define SM_RPC_TIMEOUT 15
56 #define PERCENT_MINJOIN 10
57 #define MAX_FDS 256
58 #define MAX_THR 25
59 #define INC_DELAYTIME 30

```

```

60 #define MAX_DELAYTIME 300
61 #define SM_CLTS_TIMEOUT 15
62 /* max strlen of /statmon/state, /statmon/sm.bak, /statmon/sm */
63 #define SM_MAXPATHLEN 17
64 /* Increment size for realloc of array host_name */
65 #define HOST_NAME_INCR 5

67 /* supported address family names in /var/statmon symlinks */
68 #define SM_ADDR_IPV4 "ipv4"
69 #define SM_ADDR_IPV6 "ipv6"

71 /* Supported for readdir_r() */
72 #define MAXDIRENT (sizeof (struct dirent) + _POSIX_PATH_MAX + 1)

74 /* Structure entry for monitor table (mon_table) */
75 struct mon_entry {
76     mon_id; /* mon information: mon_name, my_id */
77     struct mon_entry *prev; /* Prev ptr to prev entry in hash */
78     struct mon_entry *nxt; /* Next ptr to next entry in hash */
79 };
    unchanged_portion_omitted

163 extern int debug; /* Prints out debug information if set. */

165 extern char hostname[MAXHOSTNAMELEN];

167 /*
168 * These variables will be used to store all the
169 * alias names for the host, as well as the -a
170 * command line hostnames.
171 */
172 extern char **host_name; /* store -a opts */
173 extern int host_name_count;
174 extern int addrinx; /* # of -a entries */

176 /*
177 * The following 2 variables are meaningful
178 * only under a HA configuration.
179 */
180 extern char **path_name; /* store -p opts */
181 extern int pathinx; /* # of -p entries */

183 /* Function prototypes used in program */
184 extern int create_file(char *name);
185 extern void delete_file(char *name);
186 extern void record_name(char *name, int op);
187 extern void sm_crash(void);
185 extern void sm_notify(stat_chge *ntfp);
188 extern void statd_init();
189 extern void merge_hosts(void);
190 extern void merge_ips(void);
191 extern CLIENT *create_client(char *, int, int, char *, struct timeval *);
188 extern CLIENT *create_client(char *, int, int, struct timeval *);
192 extern char *xmalloc(unsigned);

194 /*
195 * RPC service functions, slightly different here than the
196 * generated ones in sm_inter.h
197 */
198 extern void nsmaddrprocl_reg(reglargs *, reglres *);
199 extern void sm_stat_svc(sm_name *namep, sm_stat_res *resp);
200 extern void sm_mon_svc(mon *monp, sm_stat_res *resp);
201 extern void sm_unmon_svc(mon_id *monidp, sm_stat *resp);
202 extern void sm_unmon_all_svc(my_id *myidp, sm_stat *resp);
203 extern void sm_simu_crash_svc(void *myidp);
204 extern void sm_notify_svc(stat_chge *ntfp);

```

```
190 extern void    sm_status(sm_name *namep, sm_stat_res *resp);
191 extern void    sm_mon(mon *monp, sm_stat_res *resp);
192 extern void    sm_unmon(mon_id *monidp, sm_stat *resp);
193 extern void    sm_unmon_all(my_id *myidp, sm_stat *resp);
194 extern void    sm_simu_crash(void *myidp);
206 extern void    sm_inithash();
207 extern void    copydir_from_to(char *from_dir, char *to_dir);
208 extern int     str_cmp_unqual_hostname(char *, char *);
198 extern void    nsmaddrproc1_reg(reglargs *, reglres *);
209 extern void    record_addr(char *name, sa_family_t family, struct netobj *ah);
210 extern int     is_symlink(char *file);
211 extern int     create_symlink(char *todir, char *rname, char *lname);
212 extern int     str_cmp_address_specifier(char *specifier1, char *specifier2);

214 #ifdef __cplusplus
215 }
_____unchanged_portion_omitted_____
```

new/usr/src/cmd/fs.d/nfs/statd/sm_svc.c

1

```
*****
21174 Fri Aug 16 09:04:39 2013
new/usr/src/cmd/fs.d/nfs/statd/sm_svc.c
195 Need replacement for nfs/lockd+klm
*****
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 * Copyright 2011 Nexenta Systems, Inc. All rights reserved.
24 */

26 /*      Copyright (c) 1984, 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 /*
40 * Copyright (c) 2012 by Delphix. All rights reserved.
41 */

43 #include <stdio.h>
44 #include <stdio_ext.h>
45 #include <stdlib.h>
46 #include <ftw.h>
47 #include <signal.h>
48 #include <string.h>
49 #include <syslog.h>
50 #include <netconfig.h>
51 #include <unistd.h>
52 #include <netdb.h>
53 #include <rpc/rpc.h>
54 #include <netinet/in.h>
55 #include <sys/param.h>
56 #include <sys/resource.h>
57 #include <sys/file.h>
58 #include <sys/types.h>
59 #include <sys/stat.h>
60 #include <sys/sockio.h>
61 #include <dirent.h>
```

new/usr/src/cmd/fs.d/nfs/statd/sm_svc.c

2

```
62 #include <errno.h>
63 #include <rpcsvc/sm_inter.h>
64 #include <rpcsvc/nsm_addr.h>
65 #include <thread.h>
66 #include <synch.h>
67 #include <net/if.h>
68 #include <limits.h>
69 #include <rpcsvc/daemon_utils.h>
70 #include <priv_utils.h>
71 #include "sm_statd.h"

74 #define home0          "/var/statmon"
75 #define current0       "/var/statmon/sm"
76 #define backup0        "/var/statmon/sm.bak"
77 #define state0         "/var/statmon/state"

79 #define home1          "statmon"
80 #define current1       "statmon/sm/"
81 #define backup1        "statmon/sm.bak/"
82 #define state1         "statmon/state"

84 /*
85 * User and group IDs to run as. These are hardwired, rather than looked
86 * up at runtime, because they are very unlikely to change and because they
87 * provide some protection against bogus changes to the passwd and group
88 * files.
89 */
90 uid_t  daemon_uid = DAEMON_UID;
91 gid_t  daemon_gid = DAEMON_GID;

93 char STATE[MAXPATHLEN], CURRENT[MAXPATHLEN], BACKUP[MAXPATHLEN];
94 static char statd_home[MAXPATHLEN];

96 int debug;
97 int regfiles_only = 0;          /* 1 => use symlinks in statmon, 0 => don't */
98 char hostname[MAXHOSTNAMELEN];

100 /*
101 * These variables will be used to store all the
102 * alias names for the host, as well as the -a
103 * command line hostnames.
104 */
105 int host_name_count;
106 char **host_name; /* store -a opts */
107 int  addrix; /* # of -a entries */

110 /*
111 * The following 2 variables are meaningful
112 * only under a HA configuration.
113 * The path_name array is dynamically allocated in main() during
114 * command line argument processing for the -p options.
115 */
116 char **path_name = NULL; /* store -p opts */
117 int  pathix = 0; /* # of -p entries */

119 /* Global variables. Refer to sm_statd.h for description */
120 mutex_t crash_lock;
121 int die;
122 int in_crash;
123 cond_t crash_finish;
124 mutex_t sm_trylock;
125 rwlock_t thr_rwlock;
126 cond_t retrywait;
127 mutex_t name_addrlock;
```

```

129 /* forward references */
130 static void set_statmon_owner(void);
131 static void copy_client_names(void);
132 static void one_statmon_owner(const char *);
133 static int nftw_owner(const char *, const struct stat *, int, struct FTW *);

135 /*
136 * statd protocol
137 *   commands:
138 *     SM_STAT      returns stat_fail to caller
139 *
140 *     SM_MON       adds an entry to the monitor_q and the record_q
141 *                 This message is sent by the server lockd to the server
142 *                 statd, to indicate that a new client is to be monitored.
143 *                 It is also sent by the server lockd to the client statd
144 *                 to indicate that a new server is to be monitored.
145 *
146 *     SM_UNMON     removes an entry from the monitor_q and the record_q
147 *
148 *     SM_UNMON_ALL removes all entries from a particular host from the
149 *                 monitor_q and the record_q. Our statd has this
150 *                 disabled.
151 *
152 *     SM_SIMU_CRASH simulate a crash. removes everything from the
153 *                 record_q and the recovery_q, then calls statd_init()
154 *                 to restart things. This message is sent by the server
155 *                 lockd to the server statd to have all clients notified
156 *                 that they should reclaim locks.
157 *
158 *     SM_NOTIFY    Sent by statd on server to statd on client during
159 *                 crash recovery. The client statd passes the info
160 *                 to its lockd so it can attempt to reclaim the locks
161 *                 held on the server.
162 *
163 * There are three main hash tables used to keep track of things.
164 *   mon_table     table that keeps track hosts statd must watch. If one of
165 *                 these hosts crashes, then any locks held by that host must
166 *                 be released.
167 *   record_table  used to keep track of all the hostname files stored in
168 *                 the directory /var/statmon/sm. These are client hosts who
169 *                 are holding or have held a lock at some point. Needed
170 *                 to determine if a file needs to be created for host in
171 *                 /var/statmon/sm.
172 *   recov_q       used to keep track hostnames during a recovery
173 *
174 * The entries are hashed based upon the name.
175 *
176 * There is a directory /var/statmon/sm which holds a file named
177 * for each host that is holding (or has held) a lock. This is
178 * used during initialization on startup, or after a simulated
179 * crash.
180 */

186 static void
187 sm_prog_l(rqstp, transp)
188     struct svc_req *rqstp;
189     SVCXPRT *transp;
190 {
191     union {
192         struct sm_name sm_stat_l_arg;
193         struct mon sm_mon_l_arg;

```

```

194         struct mon_id sm_unmon_l_arg;
195         struct my_id sm_unmon_all_l_arg;
196         struct stat_chge ntf_arg;
197         struct reglargs regl_arg;
198     } argument;

200     union {
201         sm_stat_res stat_resp;
202         sm_stat mon_resp;
203         struct reglres regl_resp;
204     } result;

206     bool_t (*xdr_argument)(), (*xdr_result)();
207     char *(*local)();

209     /*
210     * Dispatch according to which protocol is being used:
211     *   NSM_ADDR_PROGRAM is the private lockd address
212     *   registration protocol.
213     *   SM_PROG is the normal statd (NSM) protocol.
214     */
215     if (rqstp->rq_prog == NSM_ADDR_PROGRAM) {
216         switch (rqstp->rq_proc) {
217             case NULLPROC:
218                 svc_sendreply(transp, xdr_void, (caddr_t)NULL);
219                 return;

221             case NSMADDRPROC1_REG:
222                 xdr_argument = xdr_reglargs;
223                 xdr_result = xdr_reglres;
224                 local = (char *(*)()) nsmaddrproc1_reg;
225                 break;

227             case NSMADDRPROC1_UNREG: /* Not impl. */
228                 default:
229                     svcerr_noproc(transp);
230                     return;
231         }
232     } else {
233         /* Must be SM_PROG */
234         switch (rqstp->rq_proc) {
235             case NULLPROC:
236                 svc_sendreply(transp, xdr_void, (caddr_t)NULL);
237                 return;

239             case SM_STAT:
240                 xdr_argument = xdr_sm_name;
241                 xdr_result = xdr_sm_stat_res;
242                 local = (char *(*)()) sm_stat_svc;
243                 local = (char *(*)()) sm_status;
244                 break;

245             case SM_MON:
246                 xdr_argument = xdr_mon;
247                 xdr_result = xdr_sm_stat_res;
248                 local = (char *(*)()) sm_mon_svc;
249                 local = (char *(*)()) sm_mon;
250                 break;

251             case SM_UNMON:
252                 xdr_argument = xdr_mon_id;
253                 xdr_result = xdr_sm_stat;
254                 local = (char *(*)()) sm_unmon_svc;
255                 local = (char *(*)()) sm_unmon;
256                 break;

```

```

257     case SM_UNMON_ALL:
258         xdr_argument = xdr_my_id;
259         xdr_result = xdr_sm_stat;
260         local = (char *(*()) sm_unmon_all_svc;
253         local = (char *(*()) sm_unmon_all;
261         break;

263     case SM_SIMU_CRASH:
264         xdr_argument = xdr_void;
265         xdr_result = xdr_void;
266         local = (char *(*()) sm_simu_crash_svc;
259         local = (char *(*()) sm_simu_crash;
267         break;

269     case SM_NOTIFY:
270         xdr_argument = xdr_stat_chge;
271         xdr_result = xdr_void;
272         local = (char *(*()) sm_notify_svc;
265         local = (char *(*()) sm_notify;
273         break;

275     default:
276         svcerr_noproc(transp);
277         return;
278     }
279 }

281 (void) memset(&argument, 0, sizeof (argument));
282 if (!svc_getargs(transp, xdr_argument, (caddr_t)&argument)) {
283     svcerr_decode(transp);
284     return;
285 }

287 (void) memset(&result, 0, sizeof (result));
288 (*local)(amp;argument, &result);
289 if (!svc_sendreply(transp, xdr_result, (caddr_t)&result)) {
290     svcerr_systemerr(transp);
291 }

293 if (!svc_freeargs(transp, xdr_argument, (caddr_t)&argument)) {
294     syslog(LOG_ERR, "statd: unable to free arguments\n");
295 }
296 }

```

unchanged portion omitted

```

435 int
436 main(int argc, char *argv[])
437 {
438     int c;
439     int ppid;
440     extern char *optarg;
441     int choice = 0;
442     struct rlimit rl;
443     int mode;
444     int sz;
445     int connmaxrec = RPC_MAXDATASIZE;

447     addrix = 0;
448     pathix = 0;

450     (void) gethostname(hostname, MAXHOSTNAMELEN);
451     if (init_hostname() < 0)
452         exit(1);

454     while ((c = getopt(argc, argv, "Dd:a:G:p:rU:")) != EOF)
455         switch (c) {

```

```

456     case 'd':
457         (void) sscanf(optarg, "%d", &debug);
458         break;
459     case 'D':
460         choice = 1;
461         break;
462     case 'a':
463         if (addrix < host_name_count) {
464             if (strcmp(hostname, optarg) != 0) {
465                 sz = strlen(optarg);
466                 if (sz < MAXHOSTNAMELEN) {
467                     host_name[addrix] =
468                         (char *)xmalloc(sz+1);
469                     if (host_name[addrix] !=
470                         NULL) {
471                         (void) sscanf(optarg, "%s",
472                             host_name[addrix]);
473                         addrix++;
474                     }
475                 } else
476                     (void) fprintf(stderr,
477                         "statd: -a name of host is too long.\n");
478             }
479         } else
480             (void) fprintf(stderr,
481                 "statd: -a exceeding maximum hostnames\n");
482         break;
483     case 'U':
484         (void) sscanf(optarg, "%d", &daemon_uid);
485         break;
486     case 'G':
487         (void) sscanf(optarg, "%d", &daemon_gid);
488         break;
489     case 'p':
490         if (strlen(optarg) < MAXPATHLEN) {
491             /* If the path_name array has not yet */
492             /* been malloc'ed, do that. The array */
493             /* should be big enough to hold all of the */
494             /* -p options we might have. An upper */
495             /* bound on the number of -p options is */
496             /* argc/2, because each -p option consumes */
497             /* two arguments. Here the upper bound */
498             /* is supposing that all the command line */
499             /* arguments are -p options, which would */
500             /* actually never be the case. */
501             if (path_name == NULL) {
502                 size_t sz = (argc/2) * sizeof (char *);
503                 path_name = (char **)malloc(sz);
504                 if (path_name == NULL) {
505                     (void) fprintf(stderr,
506                         "statd: malloc failed\n");
507                     exit(1);
508                 }
509                 (void) memset(path_name, 0, sz);
510             }
511             path_name[pathix] = optarg;
512             pathix++;
513         } else {
514             (void) fprintf(stderr,
515                 "statd: -p pathname is too long.\n");
516         }
517         break;
518     case 'r':
519         regfiles_only = 1;
520         break;
521 }

```

```

522         default:
523             (void) fprintf(stderr,
524                 "statd [-d level] [-D]\n");
525             return (1);
526         }
527
528     if (choice == 0) {
529         (void) strcpy(statd_home, home0);
530         (void) strcpy(CURRENT, current0);
531         (void) strcpy(BACKUP, backup0);
532         (void) strcpy(STATE, state0);
533     } else {
534         (void) strcpy(statd_home, homel);
535         (void) strcpy(CURRENT, current1);
536         (void) strcpy(BACKUP, backup1);
537         (void) strcpy(STATE, statel);
538     }
539     if (debug)
540         (void) printf("debug is on, create entry: %s, %s, %s\n",
541             CURRENT, BACKUP, STATE);
542
543     if (getrlimit(RLIMIT_NOFILE, &rl))
544         (void) printf("statd: getrlimit failed. \n");
545
546     /* Set maxfdlimit current soft limit */
547     rl.rlim_cur = rl.rlim_max;
548     if (setrlimit(RLIMIT_NOFILE, &rl) != 0)
549         syslog(LOG_ERR, "statd: unable to set RLIMIT_NOFILE to %d\n",
550             rl.rlim_cur);
551
552     (void) enable_extended_FILE_stdio(-1, -1);
553
554     if (!debug) {
555         ppid = fork();
556         if (ppid == -1) {
557             (void) fprintf(stderr, "statd: fork failure\n");
558             (void) fflush(stderr);
559             abort();
560         }
561         if (ppid != 0) {
562             exit(0);
563         }
564         closefrom(0);
565         (void) open("/dev/null", O_RDONLY);
566         (void) open("/dev/null", O_WRONLY);
567         (void) dup(1);
568         (void) setsid();
569         openlog("statd", LOG_PID, LOG_DAEMON);
570     }
571
572     (void) _create_daemon_lock(STATD, daemon_uid, daemon_gid);
573     /*
574      * establish our lock on the lock file and write our pid to it.
575      * exit if some other process holds the lock, or if there's any
576      * error in writing/locking the file.
577      */
578     ppid = _enter_daemon_lock(STATD);
579     switch (ppid) {
580     case 0:
581         break;
582     case -1:
583         syslog(LOG_ERR, "error locking for %s: %s", STATD,
584             strerror(errno));
585         exit(2);
586     default:
587         /* daemon was already running */

```

```

588         exit(0);
589     }
590
591     /* Get other aliases from each interface. */
592     merge_hosts();
593
594     /* Get all of the configured IP addresses. */
595     merge_ips();
596
597     /*
598      * Set to automatic mode such that threads are automatically
599      * created
600      */
601     mode = RPC_SVC_MT_AUTO;
602     if (!rpc_control(RPC_SVC_MTMODE_SET, &mode)) {
603         syslog(LOG_ERR,
604             "statd:unable to set automatic MT mode.");
605         exit(1);
606     }
607
608     /*
609      * Set non-blocking mode and maximum record size for
610      * connection oriented RPC transports.
611      */
612     if (!rpc_control(RPC_SVC_CONNMAXREC_SET, &connmaxrec)) {
613         syslog(LOG_INFO, "unable to set maximum RPC record size");
614     }
615
616     if (!svc_create(sm_prog_1, SM_PROG, SM_VERS, "netpath")) {
617         syslog(LOG_ERR,
618             "statd: unable to create (SM_PROG, SM_VERS) for netpath.");
619         exit(1);
620     }
621
622     if (!svc_create(sm_prog_1, NSM_ADDR_PROGRAM, NSM_ADDR_V1, "netpath")) {
623         syslog(LOG_ERR,
624             "statd: unable to create (NSM_ADDR_PROGRAM, NSM_ADDR_V1) for netpath.");
625     }
626
627     /*
628      * Make sure /var/statmon and any alternate (-p) statmon
629      * directories exist and are owned by daemon. Then change our uid
630      * to daemon. The uid change is to prevent attacks against local
631      * daemons that trust any call from a local root process.
632      */
633
634     set_statmon_owner();
635
636     /*
637      *
638      * statd now runs as a daemon rather than root and can not
639      * dump core under / because of the permission. It is
640      * important that current working directory of statd be
641      * changed to writable directory /var/statmon so that it
642      * can dump the core upon the receipt of the signal.
643      * One still need to set allow_setid_core to non-zero in
644      * /etc/system to get the core dump.
645      */
646
647     if (chdir(statd_home) < 0) {
648         syslog(LOG_ERR, "can't chdir %s: %m", statd_home);
649         exit(1);
650     }
651
652     copy_client_names();

```

```
655     rwlock_init(&thr_rwlock, USYNC_THREAD, NULL);
656     mutex_init(&crash_lock, USYNC_THREAD, NULL);
657     mutex_init(&name_addrlock, USYNC_THREAD, NULL);
658     cond_init(&crash_finish, USYNC_THREAD, NULL);
659     cond_init(&retrywait, USYNC_THREAD, NULL);
660     sm_inithash();
661     die = 0;
662     /*
663     * This variable is set to ensure that an sm_crash
664     * request will not be done at the same time
665     * when a statd_init is being done, since sm_crash
666     * can reset some variables that statd_init will be using.
667     */
668     in_crash = 1;
669     statd_init();
671     if (debug)
672         (void) printf("Starting svc_run\n");
673     svc_run();
674     syslog(LOG_ERR, "statd: svc_run returned\n");
675     /* NOTREACHED */
676     thr_exit((void *) 1);
677     return (0);
679 }
_____unchanged_portion_omitted_
```

```

*****
10978 Fri Aug 16 09:04:40 2013
new/usr/src/head/Makefile
195 Need replacement for nfs/lockd+klm
*****
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 2010 Sun Microsystems, Inc. All rights reserved.
23 # Use is subject to license terms.
24 #
25 # head/Makefile
26 #
27 # include global definitions
28 include ../Makefile.master

30 sparc_HDRS=
31 i386_HDRS=      stack_unwind.h

33 # Headers are listed one per line so that TeamWare can auto-merge most changes

35 KRB5HDRS= mit_copyright.h mit-sipb-copyright.h

37 ATTRDB_HDRS=  secdb.h auth_attr.h exec_attr.h prof_attr.h user_attr.h \
38             auth_list.h

40 HDRS=  $(($(MACH)_HDRS) $(ATTRDB_HDRS) \
41         aio.h \
42         alloca.h \
43         apptrace.h \
44         apptrace_impl.h \
45         ar.h \
46         archives.h \
47         assert.h \
48         atomic.h \
49         attr.h \
50         config_admin.h \
51         cpio.h \
52         crypt.h \
53         ctype.h \
54         deflt.h \
55         devid.h \
56         devgmt.h \
57         devpoll.h \
58         dial.h \
59         dirent.h \
60         dlfcn.h \
61         door.h \

```

```

62         elf.h \
63         err.h \
64         errno.h \
65         euc.h \
66         exacct.h \
67         exacct_impl.h \
68         execinfo.h \
69         fatal.h \
70         fcntl.h \
71         float.h \
72         fmtmsg.h \
73         fnmatch.h \
74         ftw.h \
75         gelf.h \
76         getopt.h \
77         getwidth.h \
78         glob.h \
79         grp.h \
80         iconv.h \
81         ieeeep.h \
82         ifaddrs.h \
83         inttypes.h \
84         iso646.h \
85         klpd.h \
86         langinfo.h \
87         lastlog.h \
88         lber.h \
89         ldap.h \
90         libelf.h \
91         libgen.h \
92         libintl.h \
93         libw.h \
94         libzonecfg.h \
95         limits.h \
96         linenum.h \
97         link.h \
98         listen.h \
99         locale.h \
100        macros.h \
101        malloc.h \
102        mdmn_changelog.h \
103        memory.h \
104        meta.h \
105        meta_runtime.h \
106        metadyn.h \
107        mon.h \
108        monetary.h \
109        mp.h \
110        mqueue.h \
111        nan.h \
112        ndbm.h \
113        ndpd.h \
114        netconfig.h \
115        netdb.h \
116        netdir.h \
117        nl_types.h \
118        nlist.h \
119        note.h \
120        nsctl.h \
121        nsswitch.h \
122        nss_common.h \
123        nss_dbdefs.h \
124        nss_netdir.h \
125        paths.h \
126        pcsample.h \
127        pfmt.h \

```


new/usr/src/head/Makefile

```

128 pkgdev.h \
129 pkginfo.h \
130 pkglocs.h \
131 pkgstrct.h \
132 pkgtrans.h \
133 poll.h \
134 port.h \
135 priv.h \
136 priv_utils.h \
137 proc_service.h \
138 procfs.h \
139 prof.h \
140 project.h \
141 pthread.h \
142 pw.h \
143 pwd.h \
144 rctl.h \
145 re_comp.h \
146 regex.h \
147 regexp.h \
148 resolv.h \
149 rje.h \
150 rtdb_db.h \
151 sac.h \
152 sched.h \
153 schedctl.h \
154 sdssc.h \
155 search.h \
156 semaphore.h \
157 setjmp.h \
158 sgtty.h \
159 shadow.h \
160 siginfo.h \
161 signal.h \
162 spawn.h \
163 stdarg.h \
164 stdbool.h \
165 stddef.h \
166 stdint.h \
167 stdio.h \
168 stdio_ext.h \
169 stdio_tag.h \
170 stdio_impl.h \
171 stdlib.h \
172 storclass.h \
173 string.h \
174 strings.h \
175 stropts.h \
176 synch.h \
177 sysexits.h \
178 syslog.h \
179 syms.h \
180 tar.h \
181 termio.h \
182 termios.h \
183 thread.h \
184 thread_db.h \
185 time.h \
186 tiuser.h \
187 tzfile.h \
188 ucontext.h \
189 ucred.h \
190 ulimit.h \
191 unistd.h \
192 userdefs.h \
193 ustat.h \

```

3

new/usr/src/head/Makefile

```

194 utime.h \
195 utmp.h \
196 utmpx.h \
197 valtools.h \
198 values.h \
199 varargs.h \
200 wait.h \
201 wchar.h \
202 wchar_impl.h \
203 wctype.h \
204 widec.h \
205 wordexp.h \
206 xti.h \
207 xti_inet.h \
208 zone.h \
210 ISOHDRS = \
211 ctype_c99.h \
212 ctype_iso.h \
213 limits_iso.h \
214 locale_iso.h \
215 setjmp_iso.h \
216 signal_iso.h \
217 stdarg_c99.h \
218 stdarg_iso.h \
219 stddef_iso.h \
220 stdio_c99.h \
221 stdio_iso.h \
222 stdlib_c99.h \
223 stdlib_iso.h \
224 string_iso.h \
225 time_iso.h \
226 wchar_c99.h \
227 wchar_iso.h \
228 wctype_c99.h \
229 wctype_iso.h \
231 ARPAHDRS = \
232 ftp.h \
233 inet.h \
234 nameser.h \
235 telnet.h \
236 tftp.h \
237 nameser_compat.h \
239 AUDIOHDRS = \
240 au.h \
242 UUIDHDRS = \
243 uuid.h \
245 # rpcsvc headers which are just headers (not derived from a .x file)
246 RPCSVC_SRC_HDRS = \
247 bootparam.h \
248 daemon_utils.h \
249 dbm.h \
250 nis_db.h \
251 nislib.h \
252 svc_dg_priv.h \
253 yp_prot.h \
254 ypclnt.h \
255 yppasswd.h \
256 ypuuid.h \
257 rpc_sztypes.h \
259 # rpcsvc headers which are generated from .x files

```

4

```

260 RPCSVC_GEN_HDRS = \
261     bootparam_prot.h \
262     mount.h \
263     nfs_prot.h \
264     nfs4_prot.h \
265     nis.h \
266     nlm_prot.h \
266     rex.h \
267     rquota.h \
268     rstat.h \
269     rusers.h \
270     rwall.h \
271     spray.h \
272     ufs_prot.h \
273     nfs_acl.h

275 LVMRPCHDRS = \
276     mhdx.h mdiox.h meta_basic.h metad.h metamed.h metamhd.h metaacl.h

278 SYMHDRASSERT = $(ROOT)/usr/include/iso/assert_iso.h
279 SYMHDRERRNO = $(ROOT)/usr/include/iso/errno_iso.h
280 SYMHDRFLOAT = $(ROOT)/usr/include/iso/float_iso.h
281 SYMHDRISO646 = $(ROOT)/usr/include/iso/iso646_iso.h

283 RPCGENFLAGS = -C -h
284 rpcsvc/rwall.h := RPCGENFLAGS += -M
285 meta_basic.h := RPCGENFLAGS += -M
286 metad.h := RPCGENFLAGS += -M
287 metamed.h := RPCGENFLAGS += -M
288 mhdx.h := RPCGENFLAGS += -M
289 mdiox.h := RPCGENFLAGS += -M
290 metamhd.h := RPCGENFLAGS += -M
291 metaacl.h := RPCGENFLAGS += -M

293 # rpcsvc rpcgen source (.x files)
294 #
295 # yp.x is an attempt at codifying what was hand coded in RPCL.
296 # Unfortunately it doesn't quite work. (The handcoded stuff isn't
297 # expressible in RPCL) this is due to the fact that YP was written
298 # before rpcgen existed. Hence, yp_prot.h cannot be derived from yp.x
299 #
300 # There is no `h' for nis_object.x because it is included by nis.x and
301 # the resulting .h is nis.h.

303 RPCSVCPROTS = \
304     $(RPCSVC_GEN_HDRS:%.h=%.x)      nis_object.x      yp.x

306 LVMSVCPROTS = \
307     $(LVMRPCHDRS:%.h=%.x)

309 RPCSVCHDRS= $(RPCSVC_SRC_HDRS) $(RPCSVC_GEN_HDRS)

311 PROTOHDRS=  dumprestore.h routed.h ripngd.h rwhod.h timed.h

313 ROOTHDRS= $(HDRS:%=$(ROOT)/usr/include/%) \
314     $(KRB5HDRS:%=$(ROOT)/usr/include/kerberos5/%) \
315     $(ISOHDRS:%=$(ROOT)/usr/include/iso/%) \
316     $(ARPAHDRS:%=$(ROOT)/usr/include/arpa/%) \
317     $(AUDIOHDRS:%=$(ROOT)/usr/include/audio/%) \
318     $(UIDHDRS:%=$(ROOT)/usr/include/uid/%) \
319     $(RPCSVCHDRS:%=$(ROOT)/usr/include/rpcsvc/%) \
320     $(RPCSVCPROTS:%=$(ROOT)/usr/include/rpcsvc/%) \
321     $(LVMRPCHDRS:%=$(ROOT)/usr/include/%) \
322     $(PROTOHDRS:%=$(ROOT)/usr/include/protocols/%)

324 DIRS= iso arpa audio rpcsvc protocols security uuid kerberosv5

```

```

325 ROOTDIRS= $(DIRS:%=$(ROOT)/usr/include/%)

327 SED= sed

329 # check files really don't exist
330 #
331 # should do something with the rpcsvc headers

333 iso/%.check:      iso/%.h
334     $(DOT_H_CHECK)

336 arpa/%.check:     arpa/%.h
337     $(DOT_H_CHECK)

339 audio/%.check:    audio/%.h
340     $(DOT_H_CHECK)

342 rpcsvc/%.check:   rpcsvc/%.h
343     $(DOT_H_CHECK)

345 rpcsvc/%.check:   rpcsvc/%.x
346     $(DOT_X_CHECK)

348 protocols/%.check: protocols/%.h
349     $(DOT_H_CHECK)

351 kerberosv5/%.check: kerberosv5/%.h
352     $(DOT_H_CHECK)

354 uuid/%.check:     uuid/%.h
355     $(DOT_H_CHECK)

357 # Note that the derived headers (rpcgen) are not checked at this time. These
358 # need work at the source level and rpcgen itself has a bug which causes a
359 # cstyle violation. Furthermore, there seems to be good reasons for the
360 # generated headers to not pass all of the hdrchk rules.
361 #
362 # Add the following to the CHECKHDRS list to activate the .x checks:
363 #     $(RPCSVCPROTS:%.x=rpcsvc/%.check) \
364 #
365 CHECKHDRS= $(HDRS:%.h=%.check) \
366     $(KRB5HDRS:%.h=kerberosv5/%.check) \
367     $(ISOHDRS:%.h=iso/%.check) \
368     $(ARPAHDRS:%.h=arpa/%.check) \
369     $(AUDIOHDRS:%.h=audio/%.check) \
370     $(UIDHDRS:%.h=uid/%.check) \
371     $(RPCSVC_SRC_HDRS:%.h=rpcsvc/%.check) \
372     $(PROTOHDRS:%.h=protocols/%.check)

374 # headers which won't quite meet the standards...
375 #
376 # assert.h is required by ansi-c to *not* be idempotent (section 4.1.2).
377 # Hence the trailing guard is not the last thing in the file nor can it
378 # be without playing silly games.

380 assert.check      := HDRCHK_TAIL = | grep -v "end guard wrong" | true

382 # install rules

384 $(ROOT)/usr/include/security/%: security/%
385     $(INS.file)

387 $(ROOT)/usr/include/protocols/%: protocols/%
388     $(INS.file)

390 $(ROOT)/usr/include/rpcsvc/%: rpcsvc/%

```

new/usr/src/head/Makefile

7

```

391     $(INS.file)
393 $(ROOT)/usr/include/kerberos5/%: kerberos5/%
394     $(INS.file)
396 $(ROOT)/usr/include/arpa/%: arpa/%
397     $(INS.file)
399 $(ROOT)/usr/include/audio/%: audio/%
400     $(INS.file)
402 $(ROOT)/usr/include/iso/%: iso/%
403     $(INS.file)
405 $(ROOT)/usr/include/uuid/%: uuid/%
406     $(INS.file)
408 $(ROOT)/usr/include/%: %
409     $(INS.file)
411 .KEEP_STATE:
413 .PARALLEL:    $(ROOTHDRS) $(CHECKHDRS)
415 install_h:    $(ROOTDIRS) .WAIT $(ROOTHDRS) $(SYMHDRASSERT) $(SYMHDRERRNO) \
416               $(SYMHDRFLOAT) $(SYMHDRISO646)
418 check:    $(CHECKHDRS)
420 clean clobber:
421     $(RM) $(LVMRPCHDRS);
422     cd rpcsvc ; $(RM) $(RPCSVC_GEN_HDRS)
424 $(ROOTDIRS):
425     $(INS.dir)
427 $(SYMHDRASSERT):
428     -$(RM) $@; $(SYMLINK) ../assert.h $@
430 $(SYMHDRERRNO):
431     -$(RM) $@; $(SYMLINK) ../errno.h $@
433 $(SYMHDRFLOAT):
434     -$(RM) $@; $(SYMLINK) ../float.h $@
436 $(SYMHDRISO646):
437     -$(RM) $@; $(SYMLINK) ../iso646.h $@
439 rpcsvc/%.h:    rpcsvc/%.x
440     $(RPCGEN) $(RPCGENFLAGS) $< -o $@
442 rpcsvc/nis.h:  rpcsvc/nis.x
443     $(RPCGEN) $(RPCGENFLAGS) rpcsvc/nis.x | \
444     $(SED) -e '/EDIT_START/,,$$ d' > $@
446 meta_basic.h:  ../uts/common/sys/lvm/meta_basic.x
447     $(RPCGEN) $(RPCGENFLAGS) ../uts/common/sys/lvm/meta_basic.x | \
448     awk '/<synch.h>/ { print "#ifdef _REENTRANT"; print $$0; print "#endif\t"
449     /<thread.h>/ { print "#ifdef _REENTRANT"; print $$0; print "#endif\t/* _
450     { print $0 } \
451     ' > $@
453 metad.h:      metad.x
454     $(RPCGEN) $(RPCGENFLAGS) metad.x | \
455     awk '/<synch.h>/ { print "#ifdef _REENTRANT"; print $$0; print "#endif\t"
456     /<thread.h>/ { print "#ifdef _REENTRANT"; print $$0; print "#endif\t/* _

```

new/usr/src/head/Makefile

8

```

457     { print $0 } \
458     ' > $@
460 mhd_x.h:      ../uts/common/sys/lvm/mhd_x.x
461     $(RPCGEN) $(RPCGENFLAGS) ../uts/common/sys/lvm/mhd_x.x | \
462     awk '/<synch.h>/ { print "#ifdef _REENTRANT"; print $$0; print "#endif\t"
463     /<thread.h>/ { print "#ifdef _REENTRANT"; print $$0; print "#endif\t/* _
464     { print $0 } \
465     ' > $@
467 mdiox.h:     ../uts/common/sys/lvm/mdiox.x
468     $(RPCGEN) $(RPCGENFLAGS) ../uts/common/sys/lvm/mdiox.x | \
469     nawk '{sub(/sys\/lvm\/md_mhd_x/, "mhd_x"); print $$0}' | \
470     nawk '{sub(/sys\/lvm\/md_basic/, "meta_basic"); print $$0}' | \
471     awk '/<synch.h>/ { print "#ifdef _REENTRANT"; print $$0; print "#endif\t"
472     /<thread.h>/ { print "#ifdef _REENTRANT"; print $$0; print "#endif\t/* _
473     { print $0 } \
474     ' > $@
476 metamed.h:   ../uts/common/sys/lvm/metamed.x
477     $(RPCGEN) $(RPCGENFLAGS) ../uts/common/sys/lvm/metamed.x | \
478     nawk '{sub(/sys\/lvm\/md_basic/, "meta_basic"); print $$0}' | \
479     awk '/<synch.h>/ { print "#ifdef _REENTRANT"; print $$0; print "#endif\t"
480     /<thread.h>/ { print "#ifdef _REENTRANT"; print $$0; print "#endif\t/* _
481     { print $0 } \
482     ' > $@
484 metamhd.h:   metamhd.x
485     $(RPCGEN) $(RPCGENFLAGS) metamhd.x | \
486     awk '/<synch.h>/ { print "#ifdef _REENTRANT"; print $$0; print "#endif\t"
487     /<thread.h>/ { print "#ifdef _REENTRANT"; print $$0; print "#endif\t/* _
488     { print $0 } \
489     ' > $@
491 metacl.h:    metacl.x
492     $(RPCGEN) $(RPCGENFLAGS) metacl.x | \
493     awk '/<synch.h>/ { print "#ifdef _REENTRANT"; print $$0; print "#endif\t"
494     /<thread.h>/ { print "#ifdef _REENTRANT"; print $$0; print "#endif\t/* _
495     { print $0 } \
496     ' > $@

```

```

*****
3113 Fri Aug 16 09:04:41 2013
new/usr/src/lib/librpcsvc/common/mapfile-vers
195 Need replacement for nfs/lockd+klm
*****
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) 2006, 2010, Oracle and/or its affiliates. All rights reserved.
23 # Copyright 2011 Nexenta Systems, Inc. All rights reserved.
24 #

26 #
27 # MAPFILE HEADER START
28 #
29 # WARNING: STOP NOW. DO NOT MODIFY THIS FILE.
30 # Object versioning must comply with the rules detailed in
31 #
32 #     usr/src/lib/README.mapfiles
33 #
34 # You should not be making modifications here until you've read the most current
35 # copy of that file. If you need help, contact a gatekeeper for guidance.
36 #
37 # MAPFILE HEADER END
38 #

40 # Due to mistakes made early in the history of this library, there are
41 # no SUNW_1.1 symbols, but the version is now kept as a placeholder.
42 # Don't add any symbols to this version.

44 $mapfile_version 2

46 SYMBOL_VERSION SUNW_1.1 {
47     global:
48         SUNW_1.1;
49 } SUNW_0.7;
    unchanged portion omitted

63 SYMBOL_VERSION SUNWprivate_1.1 {
64     global:
65         __clnt_bindresvport;
66         xdr_bp_address;
67         xdr_bp_fileid_t;
68         xdr_bp_getfile_arg;
69         xdr_bp_getfile_res;
70         xdr_bp_machine_name_t;
71         xdr_bp_path_t;
72         xdr_bp_whoami_arg;

```

```

73     xdr_bp_whoami_res;
74     xdr_dirpath;
75     xdr_exportnode;
76     xdr_exports;
77     xdr_fhandle;
78     xdr_fhandle3;
79     xdr_fhstatus;
80     xdr_fsh4_access;
81     xdr_fsh4_mode;
82     xdr_fsh_access;
83     xdr_fsh_mode;
84     xdr_groupnode;
85     xdr_groups;
86     xdr_int32;
87     xdr_int64;
88     xdr_ip_addr_t;
89     xdr_mon;
90     xdr_mon_id;
91     xdr_mountbody;
92     xdr_mountlist;
93     xdr_mountres3;
94     xdr_mountres3_ok;
95     xdr_mountstat3;
96     xdr_my_id;
97     xdr_name;
98     xdr_nlm4_cancargs;
99     xdr_nlm4_holder;
100    xdr_nlm4_lock;
101    xdr_nlm4_lockargs;
102    xdr_nlm4_notify;
103    xdr_nlm4_res;
104    xdr_nlm4_share;
105    xdr_nlm4_shareargs;
106    xdr_nlm4_sharerres;
107    xdr_nlm4_stat;
108    xdr_nlm4_stats;
109    xdr_nlm4_testargs;
110    xdr_nlm4_testres;
111    xdr_nlm4_testrply;
112    xdr_nlm4_unlockargs;
113    xdr_nlm_cancargs;
114    xdr_nlm_holder;
115    xdr_nlm_lock;
116    xdr_nlm_lockargs;
117    xdr_nlm_notify;
118    xdr_nlm_res;
119    xdr_nlm_share;
120    xdr_nlm_shareargs;
121    xdr_nlm_sharerres;
122    xdr_nlm_stat;
123    xdr_nlm_stats;
124    xdr_nlm_testargs;
125    xdr_nlm_testres;
126    xdr_nlm_testrply;
127    xdr_nlm_unlockargs;
128    xdr_ppathcnf;
129    xdr_reglargs;
130    xdr_reglres;
131    xdr_res;
132    xdr_rstat_timeval;
133    xdr_rusers_utmp;
134    xdr_sm_name;
135    xdr_sm_res;
136    xdr_sm_stat;
137    xdr_sm_stat_res;
138    xdr_sm_status;

```

```
136     xdr_sprayarr;  
137     xdr_spraycumul;  
138     xdr_spraytimeval;  
139     xdr_stat_chge;  
140     xdr_status;  
140     xdr_timeval;  
141     xdr_uint32;  
142     xdr_uint64;  
143     xdr_unreglargs;  
144     xdr_unreglres;  
145     xdr_utmp_array;  
146     local:  
147         *;  
148 };  
_____unchanged_portion_omitted_____
```

```

*****
6994 Fri Aug 16 09:04:42 2013
new/usr/src/uts/Makefile
195 Need replacement for nfs/lockd+klm
*****
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 #
24 # include global definitions
25 include ../Makefile.master

27 #
28 # List of architectures to build as part of the standard build.
29 #
30 # Some of these architectures are built in parallel (see i386_PARALLEL and
31 # sparc_PARALLEL). This requires building some parts first before parallel build
32 # can start. Platform make files know what should be built as a prerequisite for
33 # the parallel build to work. The i386_PREREQ and sparc_PREREQ variables tell
34 # which platform directory to enter to start making prerequisite dependencies.
35 #
36 sparc_ARCHITECTURES = sun4v sun4u sparc

38 i386_ARCHITECTURES = i86pc i86xpv intel

40 #
41 # For i386 all architectures can be compiled in parallel.
42 #
43 # intel/Makefile knows how to build prerequisites needed for parallel build.
44 #
45 i386_PREREQ = intel
46 i386_PARALLEL = $(i386_ARCHITECTURES)

48 #
49 # For sparc all architectures can be compiled in parallel.
50 #
51 # sun4/Makefile knows how to build prerequisites needed for parallel build.
52 # can start.
53 #
54 sparc_PREREQ = sun4
55 sparc_PARALLEL = $(sparc_ARCHITECTURES)

57 #
58 # Platforms defined in $(MACH)_PARALLEL are built in parallel. DUMMY is placed
59 # at the end in case $(MACH)_PARALLEL is empty to prevent everything going in
60 # parallel.
61 #

```

```

62 .PARALLEL: $(MACH)_PARALLEL) DUMMY

64 #
65 # For build prerequisites we use a special target which is constructed by adding
66 # '.prereq' suffix to the $(MACH)_PREREQ.
67 #
68 PREREQ_TARGET = $(MACH)_PREREQ:%=%prereq)

71 def := TARGET= def
72 all := TARGET= all
73 install := TARGET= install
74 install_h := TARGET= install_h
75 clean := TARGET= clean
76 clobber := TARGET= clobber
77 lint := TARGET= lint
78 clean.lint := TARGET= clean.lint
79 check := TARGET= check
80 modlist := TARGET= modlist
81 modlist := NO_STATE= -K $$MODSTATE$$$

83 .KEEP_STATE:

85 def all lint: all_h $(PMTMO_FILE) $(MACH)_ARCHITECTURES)

87 install: all_h install_dirs $(PMTMO_FILE) $(MACH)_ARCHITECTURES)

89 install_dirs:
90 @cd ..; pwd; $(MAKE) rootdirs
91 @pwd

93 #
94 # Rule to build prerequisites. The left part of the pattern will match
95 # PREREQ_TARGET.
96 #
97 # The location of the Makefile is determined by stripping '.prereq' suffix from
98 # the target name. We add '.prereq' suffix to the target passed to the child
99 # Makefile so that it can distinguish prerequisite build from the regular one.
100 #
101 #
102 %.prereq:
103 @cd $(@:%prereq=); pwd; $(MAKE) $(NO_STATE) $(TARGET).prereq

105 #
106 # Rule to build architecture files. Build all required prerequisites and then
107 # build the rest (potentially in parallel).
108 #
109 $(MACH)_ARCHITECTURES: $(PREREQ_TARGET) FRC
110 @cd $@; pwd; $(MAKE) $(NO_STATE) $(TARGET)

112 $(PMTMO_FILE) pmtmo_file: $(PATCH_MAKEUP_TABLE)
113 @if [ -z "$(PATCH_MAKEUP_TABLE)" ] ; then \
114 echo 'ERROR: $(PATCH_MAKEUP_TABLE) not set' \
115 'in environment' >&2 ; \
116 exit 1 ; \
117 fi
118 RELEASE="$(RELEASE)" MACH="$(MACH)" \
119 $(CTFCVTPBTL) -o $(PMTMO_FILE) $(PATCH_MAKEUP_TABLE)

121 #
122 # The following is the list of directories which contain Makefiles with
123 # targets to install header file. The machine independent headers are
124 # installed by invoking the Makefile in the directory containing the
125 # header files. Machine and architecture dependent headers are installed
126 # by invoking the main makefile for that architecture/machine which,
127 # in turn, is responsible for invoking the Makefiles which install headers.

```

new/usr/src/uts/Makefile

3

```

128 # It is done this way so as not to assume that all of the header files in
129 # the architecture/machine dependent subdirectories are in completely
130 # isomorphic locations.
131 #
132 COMMON_HDRDIRS= common/avs \
133                 common/c2 \
134                 common/des \
135                 common/fs \
136                 common/gssapi \
137                 common/idmap \
138                 common/klm \
139                 common/inet \
140                 common/inet/ipf/netinet \
141                 common/inet/kssl \
142                 common/inet/nca \
143                 common/inet/sockmods/netpacket \
144                 common/io/bpf/net \
145                 common/ipp \
146                 common/net \
147                 common/netinet \
148                 common/nfs \
149                 common/pcmcia/sys \
150                 common/rpc \
151                 common/rpcsvc \
152                 common/sharefs \
153                 common/smb \
154                 common/smbsrv \
155                 common/sys \
156                 common/vm

159 # These aren't the only headers in closed. But the other directories
160 # are simple enough that they can be driven from the src tree.
161 $(CLOSED_BUILD)COMMON_HDRDIRS += $(CLOSED)/uts/common/sys

163 #
164 # Subset of COMMON_HDRDIRS in which at least one header is generated
165 # at runtime (e.g., rpcgen), and in which "make clean" should run.
166 # Other directories should be included here, but do not yet have the
167 # necessary Makefile support (make clean). See 6414855.
168 # at runtime (e.g., rpcgen). (This is a partial list; there are
169 # other directories that should be included and do not yet have the
170 # necessary Makefile support. See 6414855.)
171 #
172 DYNHDRDIRS =    common/idmap \
173                 common/klm \
174                 common/rpcsvc \
175                 common/sys
176 DYNHDRDIRS = common/rpcsvc common/idmap common/sys

174 sparc_HDRDIRS= sun/sys
175 i386_HDRDIRS= i86pc/vm i86xpv/vm

177 HDRDIRS= $(COMMON_HDRDIRS) $($(MACH)_HDRDIRS)
178 install_h check: $(HDRDIRS) $($(MACH)_ARCHITECTURES)

180 $(HDRDIRS): FRC
181     @cd $@; pwd; $(MAKE) $(TARGET)

183 # ensures that headers made by rpcgen and others are available in uts source
184 # for kernel builds to reference without building install_h
185 #
186 all_h: FRC
187     @cd common/sys; pwd; $(MAKE) $@
188     @cd common/rpc; pwd; $(MAKE) $@
189     @cd common/rpcsvc; pwd; $(MAKE) $@

```

new/usr/src/uts/Makefile

4

```

190     @cd common/gssapi; pwd; $(MAKE) $@
191     @cd common/idmap; pwd; $(MAKE) $@
192     @cd common/klm; pwd; $(MAKE) $@

194 clean clobber: $($(MACH)_ARCHITECTURES) $(DYNHDRDIRS)
195     @if [ '$(PATCH_BUILD)' != '#' ] ; then \
196         echo $(RM) $(PMTMO_FILE) ; \
197         $(RM) $(PMTMO_FILE) ; \
198     fi

200 EXTRA_CLOBBER_TARGETS= common/avs/ns/rdc
201 clobber: $(EXTRA_CLOBBER_TARGETS)

204 clean.lint modlist: $($(MACH)_ARCHITECTURES)

206 #
207 # Cross-reference customization: build a cross-reference over all of
208 # the supported architectures. Although there's no correct way to set
209 # the include path (since we don't know what architecture is the one
210 # the user will be interested in), it's historically been set to
211 # mirror the $(XRDIRS) list, and that works kinda sorta okay.
212 #
213 # We need to manually prune usr/closed/uts/{i86xpv|sfmmu|i86pc} since
214 # none of them exist.
215 #
216 SHARED_XRDIRS = $(sparc_ARCHITECTURES) $(i386_ARCHITECTURES) sun4 sfmmu \
217                 sun common
218 CLOSED_XRDIRS = $(SHARED_XRDIRS:%=% ../../closed/uts/%)
219 XRDIRS = $(SHARED_XRDIRS)
220 CLOSED_XRDIRS_XEN = $(CLOSED_XRDIRS:../../closed/uts/i86xpv=)
221 CLOSED_XRDIRS_1 = $(CLOSED_XRDIRS_XEN:../../closed/uts/i86pc=)
222 $(CLOSED_BUILD)XRDIRS = $(CLOSED_XRDIRS_1:../../closed/uts/sfmmu=)

224 XRINCDIRS = $(XRDIRS)

226 cscope.out tags: FRC
227     $(XREF) -x $@

229 FRC:

```

```

*****
43663 Fri Aug 16 09:04:43 2013
new/usr/src/uts/common/Makefile.files
195 Need replacement for nfs/lockd+klm
*****
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    bz2bzlib.o        \
114    bz2crctable.o    \
115    bz2huffman.o      \
116    callb.o           \
117    callout.o         \
118    chdir.o           \
119    chmod.o           \
120    chown.o           \
121    cladm.o           \
122    class.o           \
123    clock.o           \
124    clock_highres.o   \
125    clock_realtime.o \
126    close.o           \
127    compress.o

```


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 ddi_fm.o \
143 ddi_hp_impl.o \
144 ddi_hp_ndi.o \
145 ddi_intr.o \
146 ddi_intr_impl.o \
147 ddi_intr_irm.o \
148 ddi_nodeid.o \
149 ddi_periodic.o \
150 devcfg.o \
151 devcache.o \
152 device.o \
153 devid.o \
154 devid_cache.o \
155 devid_scsi.o \
156 devid_smp.o \
157 devpolicy.o \
158 disp_lock.o \
159 dnlc.o \
160 driver.o \
161 dumpsubr.o \
162 driver_lyr.o \
163 dtrace_subr.o \
164 errorq.o \
165 etheraddr.o \
166 evchannels.o \
167 exacct.o \
168 exacct_core.o \
169 exec.o \
170 exit.o \
171 fbio.o \
172 fcntl.o \
173 fdbuffer.o \
174 fdsync.o \
175 fem.o \
176 ffs.o \
177 fio.o \
178 flock.o \
179 fm.o \
180 fork.o \
181 vpm.o \
182 fs_reparse.o \
183 fs_subr.o \
184 fsflush.o \
185 ftrace.o \
186 getcwd.o \
187 getdents.o \
188 getloadavg.o \
189 getpagesizes.o \
190 getpid.o \
191 gfs.o \
192 rusagesys.o \
193 gid.o \

```

3

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

```

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

```

4

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

```

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

```

5

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

```

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

```

6

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

7

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

406 #
407 #       Stubs for the stand-alone linker/loader
408 #
409 sparc_GENSTUBS_OBJS = \
410       kobj_stubs.o

412 i386_GENSTUBS_OBJS =

414 COMMON_GENSTUBS_OBJS =

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

418 #
419 #       DTrace and DTrace Providers
420 #
421 DTRACE_OBJS += dtrace.o dtrace_isa.o dtrace_asm.o

423 SDT_OBJS += sdt_subr.o

425 PROFILE_OBJS += profile.o

427 SYSTRACE_OBJS += systrace.o

429 LOCKSTAT_OBJS += lockstat.o

431 FASTTRAP_OBJS += fasttrap.o fasttrap_isa.o

433 DCPC_OBJS += dcpc.o

435 #
436 #       Driver (pseudo-driver) Modules
437 #
438 IPP_OBJS += ippctl.o

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

445 AUDIOEMU10K_OBJS += audioemu10k.o

447 AUDIOENS_OBJS += audioens.o

449 AUDIOVIA823X_OBJS += audiovia823x.o

451 AUDIOVIA97_OBJS += audiovia97.o

453 AUDIO1575_OBJS += audio1575.o

455 AUDIO810_OBJS += audio810.o

457 AUDIOCMI_OBJS += audiocmi.o
```

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

8

```
459 AUDIOCMIHD_OBJS += audiocmihd.o

461 AUDIOHD_OBJS += audiohd.o

463 AUDIOIXP_OBJS += audioixp.o

465 AUDIOLS_OBJS += audiols.o

467 AUDIOP16X_OBJS += audiop16x.o

469 AUDIOPCI_OBJS += audiopci.o

471 AUDIOSOLO_OBJS += audiosolo.o

473 AUDIOTS_OBJS += audiots.o

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

477 BLKDEV_OBJS += blkdev.o

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

481 CONSKBD_OBJS += conskbd.o

483 CONSMS_OBJS += consms.o

485 OLDPTY_OBJS += tty_ptyconf.o

487 PTC_OBJS += tty_pty.o

489 PTSL_OBJS += tty_pts.o

491 PTM_OBJS += ptm.o

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

496 PTS_OBJS += pts.o

498 PTY_OBJS += ptms_conf.o

500 SAD_OBJS += sad.o

502 MD4_OBJS += md4.o md4_mod.o

504 MD5_OBJS += md5.o md5_mod.o

506 SHA1_OBJS += sha1.o sha1_mod.o

508 SHA2_OBJS += sha2.o sha2_mod.o

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

513 DSCPMK_OBJS += dscpmk.o dscpmkddi.o

515 DLCOSMK_OBJS += dlcosmk.o dlcosmkddi.o

517 FLOWACCT_OBJS += flowacctddi.o flowacct.o

519 TOKENMT_OBJS += tokenmt.o tokenmtddi.o

521 TSWTCL_OBJS += tswtcl.o tswtclddi.o

523 ARP_OBJS += arpddi.o
```

```

525 ICMP_OBJS +=      icmpddi.o
527 ICMP6_OBJS +=    icmp6ddi.o
529 RTS_OBJS +=      rtsddi.o

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

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

563 IP6_OBJS +=       ip6ddi.o
565 HOOK_OBJS +=      hook.o

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

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

571 IPNET_OBJS +=     ipnet.o ipnet_bpf.o

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

575 IPSECESP_OBJS +=  ipsecespddi.o ipsecesp.o

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

579 SPPP_OBJS +=      sPPP.o sPPP_dlpi.o sPPP_mod.o sPPP_common.o

581 SPPPTUN_OBJS +=   sppptun.o sppptun_mod.o

583 SPPPASYN_OBJS +=  spppasyn.o spppasyn_mod.o

585 SPPPCOMP_OBJS +=  sppppcomp.o sppppcomp_mod.o deflate.o bsd-comp.o vjcompress.o \
586                   zlib.o

588 TCP_OBJS +=       tcpddi.o

```

```

590 TCP6_OBJS +=      tcp6ddi.o
592 NCA_OBJS +=       ncaddi.o

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

596 SCTP SOCK_MOD_OBJS += sockmod_sctp.o sockscctp.o sockscctpsubr.o

598 PFP SOCK_MOD_OBJS += sockmod_pfp.o

600 RDS SOCK_MOD_OBJS += sockmod_rds.o

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

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

607 RDSV3_OBJS +=     af_rds.o rds_v3_ddi.o bind.o loop.o threads.o connection.o \
608                   transport.o cong.o sysctl.o message.o rds_rcv.o send.o \
609                   stats.o info.o page.o rdma_transport.o ib_ring.o ib_rdma.o \
610                   ib_rcv.o ib.o ib_send.o ib_sysctl.o ib_stats.o ib_cm.o \
611                   rds_v3_sc.o rds_v3_debug.o rds_v3_impl.o rdma.o rds_v3_af_thr.o

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

616 UDP_OBJS +=       udpddi.o

618 UDP6_OBJS +=      udp6ddi.o

620 SY_OBJS +=        gentyty.o

622 TCO_OBJS +=       ticots.o

624 TCOO_OBJS +=      ticotsord.o

626 TCL_OBJS +=       ticlts.o

628 TL_OBJS +=        tl.o

630 DUMP_OBJS +=      dump.o

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

634 CLONE_OBJS +=     clone.o

636 CN_OBJS +=        cons.o

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

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

642 GLD_OBJS +=       gld.o gldutil.o

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

648 MAC_6TO4_OBJS +=  mac_6to4.o

650 MAC_ETHER_OBJS += mac_ether.o

652 MAC_IPV4_OBJS +=  mac_ipv4.o

654 MAC_IPV6_OBJS +=  mac_ipv6.o

```

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

11

```

656 MAC_WIFI_OBJS +=      mac_wifi.o
658 MAC_IB_OBJS +=       mac_ib.o
660 IPTUN_OBJS +=       iptun_dev.o iptun_ctl.o iptun.o
662 AGGR_OBJS +=        aggr_dev.o aggr_ctl.o aggr_grp.o aggr_port.o \
663                      aggr_send.o aggr_recv.o aggr_lacp.o
665 SOFTMAC_OBJS +=     softmac_main.o softmac_ctl.o softmac_capab.o \
666                      softmac_dev.o softmac_stat.o softmac_pkt.o softmac_fp.o
668 NET80211_OBJS +=    net80211.o net80211_proto.o net80211_input.o \
669                      net80211_output.o net80211_node.o net80211_crypto.o \
670                      net80211_crypto_none.o net80211_crypto_wep.o net80211_ioctl.o \
671                      net80211_crypto_tkip.o net80211_crypto_ccmp.o \
672                      net80211_ht.o
674 VNIC_OBJS +=       vnic_ctl.o vnic_dev.o
676 SIMNET_OBJS +=     simnet.o
678 IB_OBJS +=         ibnex.o ibnex_ioctl.o ibnex_hca.o
680 IBCM_OBJS +=       ibcm_impl.o ibcm_sm.o ibcm_ti.o ibcm_utils.o ibcm_path.o \
681                      ibcm_arp.o ibcm_arp_link.o
683 IBDM_OBJS +=       ibdm.o
685 IBDMA_OBJS +=      ibdma.o
687 IBMF_OBJS +=       ibmf.o ibmf_impl.o ibmf_dr.o ibmf_wqe.o ibmf_ud_dest.o ibmf_mod.
688                      ibmf_send.o ibmf_recv.o ibmf_handlers.o ibmf_trans.o \
689                      ibmf_timers.o ibmf_msg.o ibmf_utils.o ibmf_rmpp.o \
690                      ibmf_saa.o ibmf_saa_impl.o ibmf_saa_utils.o ibmf_saa_events.o
692 IBTL_OBJS +=       ibtl_impl.o ibtl_util.o ibtl_mem.o ibtl_handlers.o ibtl_qp.o \
693                      ibtl_cq.o ibtl_wr.o ibtl_hca.o ibtl_chan.o ibtl_cm.o \
694                      ibtl_mcg.o ibtl_ibnex.o ibtl_srqp.o ibtl_part.o
696 TAVOR_OBJS +=      tavor.o tavor_agents.o tavor_cfg.o tavor_ci.o tavor_cmd.o \
697                      tavor_cq.o tavor_event.o tavor_ioctl.o tavor_misc.o \
698                      tavor_mr.o tavor_qp.o tavor_qpmod.o tavor_rsrc.o \
699                      tavor_srqp.o tavor_stats.o tavor_umap.o tavor_wr.o
701 HERMON_OBJS +=     hermon.o hermon_agents.o hermon_cfg.o hermon_ci.o hermon_cmd.o \
702                      hermon_cq.o hermon_event.o hermon_ioctl.o hermon_misc.o \
703                      hermon_mr.o hermon_qp.o hermon_qpmod.o hermon_rsrc.o \
704                      hermon_srqp.o hermon_stats.o hermon_umap.o hermon_wr.o \
705                      hermon_fcoib.o hermon_fm.o
707 DAPLT_OBJS +=      daplt.o
709 SOL_OFS_OBJS +=    sol_cma.o sol_ib_cma.o sol_uobj.o \
710                      sol_ofs_debug_util.o sol_ofs_gen_util.o \
711                      sol_kverbs.o
713 SOL_UCMA_OBJS +=   sol_ucma.o
715 SOL_UVERBS_OBJS += sol_uverbs.o sol_uverbs_comp.o sol_uverbs_event.o \
716                      sol_uverbs_hca.o sol_uverbs_qp.o
718 SOL_UMAD_OBJS +=   sol_umad.o
720 KSTAT_OBJS +=     kstat.o

```

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

12

```

722 KSYMS_OBJS +=      ksyms.o
724 INSTANCE_OBJS +=   inst_sync.o
726 IWSCN_OBJS +=      iwscons.o
728 LOFI_OBJS +=       lofi.o LzmaDec.o
730 FSSNAP_OBJS +=     fssnap.o
732 FSSNAPIF_OBJS +=   fssnap_if.o
734 MM_OBJS +=         mem.o
736 PHYSMEM_OBJS +=    physmem.o
738 OPTIONS_OBJS +=    options.o
740 WINLOCK_OBJS +=    winlockio.o
742 PM_OBJS +=         pm.o
743 SRN_OBJS +=        srn.o
745 PSEUDO_OBJS +=     pseudonex.o
747 RAMDISK_OBJS +=    ramdisk.o
749 LLC1_OBJS +=       llc1.o
751 USBKBM_OBJS +=     usbkbm.o
753 USBWCM_OBJS +=     usbwcm.o
755 BOFI_OBJS +=       bofi.o
757 HID_OBJS +=        hid.o
759 HWA_RC_OBJS +=     hwarc.o
761 USBSKEL_OBJS +=    usbskel.o
763 USBVC_OBJS +=       usbvc.o usbvc_v412.o
765 HIDPARSER_OBJS +=  hidparser.o
767 USB_AC_OBJS +=     usb_ac.o
769 USB_AS_OBJS +=     usb_as.o
771 USB_AH_OBJS +=     usb_ah.o
773 USBMS_OBJS +=      usbms.o
775 USBPRN_OBJS +=     usbprn.o
777 UGEN_OBJS +=       ugen.o
779 USBSER_OBJS +=     usbser.o usbser_rseq.o
781 USBSACM_OBJS +=    usbsacm.o
783 USBSER_KEYSPAN_OBJS += usbser_keyspan.o keyspan_dsd.o keyspan_pipe.o
785 USBS49_FW_OBJS +=  keyspan_49fw.o
787 USBSPRL_OBJS +=    usbser_pl2303.o pl2303_dsd.o

```

```

789 WUSB_CA_OBJS += wusb_ca.o
791 USBFTDI_OBJS += usbser_uftdi.o uftdi_dsd.o
793 USBECM_OBJS += usbecm.o
795 WC_OBJS += wscons.o vcons.o
797 VCONS_CONF_OBJS += vcons_conf.o

799 SCSI_OBJS +=      scsi_capabilities.o scsi_confsubr.o scsi_control.o \
800                  scsi_data.o scsi_fm.o scsi_hba.o scsi_reset_notify.o \
801                  scsi_resource.o scsi_subr.o scsi_transport.o scsi_watch.o \
802                  smp_transport.o

804 SCSI_VHCI_OBJS +=      scsi_vhci.o mpapi_impl.o scsi_vhci_tpgs.o

806 SCSI_VHCI_F_SYM_OBJS +=      sym.o

808 SCSI_VHCI_F_TPGS_OBJS +=      tpgs.o

810 SCSI_VHCI_F_ASYM_SUN_OBJS +=  asym_sun.o

812 SCSI_VHCI_F_SYM_HDS_OBJS +=  sym_hds.o

814 SCSI_VHCI_F_TAPE_OBJS +=     tape.o

816 SCSI_VHCI_F_TPGS_TAPE_OBJS += tpgs_tape.o

818 SGEN_OBJS +=      sgen.o

820 SMP_OBJS +=      smp.o

822 SATA_OBJS +=     sata.o

824 USBA_OBJS +=     hcdi.o  usba.o  usbai.o  hubdi.o  parser.o  genconsole.o \
825                  usbai_pipe_mgmt.o  usbai_req.o  usbai_util.o  usbai_register.o \
826                  usba_devdb.o  usba10_calls.o  usba_uugen.o  whcdi.o  wa.o
827 USBA_WITHOUT_WUSB_OBJS +=     hcdi.o  usba.o  usbai.o  hubdi.o  parser.o  gencons
828                  usbai_pipe_mgmt.o  usbai_req.o  usbai_util.o  usbai_register.o \
829                  usba_devdb.o  usba10_calls.o  usba_uugen.o

831 USBA10_OBJS +=   usba10.o

833 RSM_OBJS +=      rsm.o  rsmka_pathmanager.o  rsmka_util.o

835 RSMOPS_OBJS +=   rsmops.o

837 S1394_OBJS +=    t1394.o  t1394_errmsg.o  s1394.o  s1394_addr.o  s1394_async.o \
838                  s1394_bus_reset.o  s1394_cmp.o  s1394_csr.o  s1394_dev_disc.o \
839                  s1394_fa.o  s1394_fcp.o \
840                  s1394_hotplug.o  s1394_isoch.o  s1394_misc.o  h1394.o  nx1394.o

842 HCI1394_OBJS +=  hcil1394.o  hcil1394_async.o  hcil1394_attach.o  hcil1394_buf.o \
843                  hcil1394_csr.o  hcil1394_detach.o  hcil1394_extern.o \
844                  hcil1394_ioctl.o  hcil1394_isoch.o  hcil1394_isr.o \
845                  hcil1394_ixl_comp.o  hcil1394_ixl_isr.o  hcil1394_ixl_misc.o \
846                  hcil1394_ixl_update.o  hcil1394_misc.o  hcil1394_ohci.o \
847                  hcil1394_q.o  hcil1394_s1394if.o  hcil1394_tlabel.o \
848                  hcil1394_tlist.o  hcil1394_vendor.o

850 AV1394_OBJS +=   av1394.o  av1394_as.o  av1394_async.o  av1394_cfgrom.o \
851                  av1394_cmp.o  av1394_fcp.o  av1394_isoch.o  av1394_isoch_chan.o \
852                  av1394_isoch_recv.o  av1394_isoch_xmit.o  av1394_list.o \
853                  av1394_queue.o

```

```

855 DCAM1394_OBJS +=  dcam.o  dcam_frame.o  dcam_param.o  dcam_reg.o \
856                  dcam_ring_buff.o

858 SCSA1394_OBJS +=  hba.o  sbp2_driver.o  sbp2_bus.o

860 SBP2_OBJS +=      cfgrom.o  sbp2.o

862 PMODEM_OBJS +=   pmodem.o  pmodem_cis.o  cis.o  cis_callout.o  cis_handlers.o  cis_para

864 DSW_OBJS +=      dsw.o  dsw_dev.o  ii_tree.o

866 NCALL_OBJS +=   ncall.o \
867                  ncall_stub.o

869 RDC_OBJS +=      rdc.o \
870                  rdc_dev.o \
871                  rdc_io.o \
872                  rdc_clnt.o \
873                  rdc_prot_xdr.o \
874                  rdc_svc.o \
875                  rdc_bitmap.o \
876                  rdc_health.o \
877                  rdc_subr.o \
878                  rdc_diskq.o

880 RDCSRV_OBJS +=  rdcsrv.o

882 RDCSTUB_OBJS += rdc_stub.o

884 SDBC_OBJS +=    sd_bcache.o \
885                  sd_bio.o \
886                  sd_conf.o \
887                  sd_ft.o \
888                  sd_hash.o \
889                  sd_io.o \
890                  sd_misc.o \
891                  sd_pcu.o \
892                  sd_tdaemon.o \
893                  sd_trace.o \
894                  sd_iob_impl0.o \
895                  sd_iob_impl1.o \
896                  sd_iob_impl2.o \
897                  sd_iob_impl3.o \
898                  sd_iob_impl4.o \
899                  sd_iob_impl5.o \
900                  sd_iob_impl6.o \
901                  sd_iob_impl7.o \
902                  safestore.o \
903                  safestore_ram.o

905 NSCTL_OBJS +=   nsctl.o \
906                  nsc_cache.o \
907                  nsc_disk.o \
908                  nsc_dev.o \
909                  nsc_freeze.o \
910                  nsc_gen.o \
911                  nsc_mem.o \
912                  nsc_ncallio.o \
913                  nsc_power.o \
914                  nsc_resv.o \
915                  nsc_rmspin.o \
916                  nsc_solaris.o \
917                  nsc_trap.o \
918                  nsc_list.o
919 UNISTAT_OBJS += spuni.o \

```

```

920             spcs_s_k.o
922 NSKERN_OBJS += nsc_ddi.o \
923                nsc_proc.o \
924                nsc_raw.o \
925                nsc_thread.o \
926                nskernd.o
928 SV_OBJS +=    sv.o
930 PMCS_OBJS +=  pmcs_attach.o pmcs_ds.o pmcs_intr.o pmcs_nvram.o pmcs_sata.o \
931                pmcs_scsa.o pmcs_smhba.o pmcs_subr.o pmcs_fwlog.o
933 PMCS8001FW_C_OBJS +=    pmcs_fw_hdr.o
934 PMCS8001FW_OBJS +=      $(PMCS8001FW_C_OBJS) SPCBoot.o ila.o firmware.o
936 #
937 #      Build up defines and paths.
939 ST_OBJS +=     st.o      st_conf.o
941 EMLXS_OBJS +=  emlxs_clock.o emlxs_dfc.o emlxs_dhchap.o emlxs_diag.o \
942                emlxs_download.o emlxs_dump.o emlxs_els.o emlxs_event.o \
943                emlxs_fcf.o emlxs_fcp.o emlxs_fct.o emlxs_hba.o emlxs_ip.o \
944                emlxs_mbox.o emlxs_mem.o emlxs_msg.o emlxs_node.o \
945                emlxs_pkt.o emlxs_sli3.o emlxs_sli4.o emlxs_solaris.o \
946                emlxs_thread.o
948 EMLXS_FW_OBJS +=    emlxs_fw.o
950 OCE_OBJS +=      oce_buf.o oce_fm.o oce_gld.o oce_hw.o oce_intr.o oce_main.o \
951                oce_mbx.o oce_mq.o oce_queue.o oce_rx.o oce_stat.o oce_tx.o \
952                oce_utils.o
954 FCT_OBJS +=     discovery.o fct.o
956 QLT_OBJS +=     2400.o 2500.o 8100.o qlt.o qlt_dma.o
958 SRPT_OBJS +=    srpt_mod.o srpt_ch.o srpt_cm.o srpt_ioc.o srpt_stp.o
960 FCOE_OBJS +=    fcoe.o fcoe_eth.o fcoe_fc.o
962 FCOET_OBJS +=   fcoet.o fcoet_eth.o fcoet_fc.o
964 FCOEI_OBJS +=   fcoei.o fcoei_eth.o fcoei_lv.o
966 ISCSIT_SHARED_OBJS += \
967                iscsit_common.o
969 ISCSIT_OBJS +=   $(ISCSIT_SHARED_OBJS) \
970                iscsit.o iscsit_tgt.o iscsit_sess.o iscsit_login.o \
971                iscsit_text.o iscsit_isns.o iscsit_radiusauth.o \
972                iscsit_radiuspacket.o iscsit_auth.o iscsit_authclient.o
974 PPPT_OBJS +=    alua_ic_if.o pppt.o pppt_msg.o pppt_tgt.o
976 STMF_OBJS +=    lun_map.o stmf.o
978 STMF_SBD_OBJS += sbd.o sbd_scsi.o sbd_pgr.o sbd_zvol.o
980 SYMSMSG_OBJS += sysmsg.o
982 SES_OBJS +=     ses.o ses_sen.o ses_safte.o ses_ses.o
984 TNF_OBJS +=     tnf_buf.o      tnf_trace.o      tnf_writer.o      trace_init.o \
985                trace_funcs.o  tnf_probe.o      tnf.o

```

```

987 LOGINDMUX_OBJS += loginmux.o
989 DEVINFO_OBJS += devinfo.o
991 DEVPOLL_OBJS += devpoll.o
993 DEVPOOL_OBJS += devpool.o
995 I8042_OBJS +=    i8042.o
997 KB8042_OBJS +=  \
998                at_keyprocess.o \
999                kb8042.o \
1000               kb8042_keytables.o
1002 MOUSE8042_OBJS += mouse8042.o
1004 FDC_OBJS +=     fdc.o
1006 ASY_OBJS +=     asy.o
1008 ECPP_OBJS +=     ecpp.o
1010 VUIDM3P_OBJS +=  vuidmice.o vuidm3p.o
1012 VUIDM4P_OBJS +=  vuidmice.o vuidm4p.o
1014 VUIDM5P_OBJS +=  vuidmice.o vuidm5p.o
1016 VUIDPS2_OBJS +=  vuidmice.o vuidps2.o
1018 HPCSVCS_OBJS +=  hpcsvc.o
1020 PCIE_MISC_OBJS += pcie.o pcie_fault.o pcie_hp.o pciehpc.o pcishpc.o pcie_pwr.o p
1022 PCIHPNEXUS_OBJS += pcihp.o
1024 OPENEEPROM_OBJS += openprom.o
1026 RANDOM_OBJS +=  random.o
1028 PSHOT_OBJS +=    pshot.o
1030 GEN_DRV_OBJS +=  gen_drv.o
1032 TCLIENT_OBJS += tclient.o
1034 TPHCI_OBJS +=    tphci.o
1036 TVHCI_OBJS +=    tvhci.o
1038 EMUL64_OBJS +=   emul64.o emul64_bsd.o
1040 FCP_OBJS +=       fcp.o
1042 FCIP_OBJS +=      fcip.o
1044 FCSM_OBJS +=      fcsm.o
1046 FCTL_OBJS +=      fctl.o
1048 FP_OBJS +=        fp.o
1050 QLC_OBJS +=        ql_api.o ql_debug.o ql_hba_fru.o ql_init.o ql_iocb.o ql_ioctl.o \
1051                ql_isr.o ql_mbx.o ql_nx.o ql_xioctl.o ql_fw_table.o

```

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

17

```

1053 QLC_FW_2200_OBJS += ql_fw_2200.o
1055 QLC_FW_2300_OBJS += ql_fw_2300.o
1057 QLC_FW_2400_OBJS += ql_fw_2400.o
1059 QLC_FW_2500_OBJS += ql_fw_2500.o
1061 QLC_FW_6322_OBJS += ql_fw_6322.o
1063 QLC_FW_8100_OBJS += ql_fw_8100.o
1065 QLGE_OBJS += qlge.o qlge_dbg.o qlge_flash.o qlge_fm.o qlge_gld.o qlge_mpi.o
1067 ZCONS_OBJS += zcons.o
1069 NV_SATA_OBJS += nv_sata.o
1071 SI3124_OBJS += si3124.o
1073 AHCI_OBJS += ahci.o
1075 PCIIDE_OBJS += pci-ide.o
1077 PCEPP_OBJS += pcepp.o
1079 CPC_OBJS += cpc.o
1081 CPUID_OBJS += cpuid_drv.o
1083 SYSEVENT_OBJS += sysevent.o
1085 BL_OBJS += bl.o
1087 DRM_OBJS += drm_sunmod.o drm_kstat.o drm_agpsupport.o \
1088     drm_auth.o drm_bufs.o drm_context.o drm_dma.o \
1089     drm_drawable.o drm_drv.o drm_fops.o drm_ioctl.o drm_irq.o \
1090     drm_lock.o drm_memory.o drm_msg.o drm_pci.o drm_scatter.o \
1091     drm_cache.o drm_gem.o drm_mm.o ati_pcigart.o
1093 FM_OBJS += devfm.o devfm_machdep.o
1095 RTLS_OBJS += rtls.o
1097 #
1098 #           exec modules
1099 #
1100 AOUTEXEC_OBJS += aout.o
1102 ELFEXEC_OBJS += elf.o elf_notes.o old_notes.o
1104 INTPEXEC_OBJS += intp.o
1106 SHBINEXEC_OBJS += shbin.o
1108 JAVAEXEC_OBJS += java.o
1110 #
1111 #           file system modules
1112 #
1113 AUTOFS_OBJS += auto_vfsops.o auto_vnops.o auto_subr.o auto_xdr.o auto_sys.o
1115 CACHEFS_OBJS += cachefs_cnode.o      cachefs_cod.o \
1116     cachefs_dir.o      cachefs_dlog.o  cachefs_filegrp.o \
1117     cachefs_fscache.o  cachefs_ioctl.o cachefs_log.o \

```

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

18

```

1118     cachefs_module.o \
1119     cachefs_noopc.o      cachefs_resource.o \
1120     cachefs_strict.o \
1121     cachefs_subr.o      cachefs_vfsops.o \
1122     cachefs_vnops.o
1124 DCFS_OBJS += dc_vnops.o
1126 DEVFS_OBJS += devfs_subr.o  devfs_vfsops.o  devfs_vnops.o
1128 DEV_OBJS += sdev_subr.o      sdev_vfsops.o  sdev_vnops.o \
1129     sdev_ptsops.o  sdev_zvolops.o  sdev_comm.o \
1130     sdev_profile.o  sdev_ncache.o  sdev_netops.o \
1131     sdev_ipnetops.o \
1132     sdev_vtops.o
1134 CTFS_OBJS += ctfs_all.o ctfs_cdir.o ctfs_ctl.o ctfs_event.o \
1135     ctfs_latest.o ctfs_root.o ctfs_sym.o ctfs_tdir.o ctfs_tmpl.o
1137 OBJFS_OBJS += objfs_vfs.o      objfs_root.o  objfs_common.o \
1138     objfs_odir.o      objfs_data.o
1140 FDFS_OBJS += fdops.o
1142 FIFO_OBJS += fifosubr.o      fifovnops.o
1144 PIPE_OBJS += pipe.o
1146 HSFS_OBJS += hsfs_node.o      hsfs_subr.o  hsfs_vfsops.o  hsfs_vnops.o \
1147     hsfs_susp.o      hsfs_rrip.o  hsfs_susp_subr.o
1149 LOFS_OBJS += lofs_subr.o      lofs_vfsops.o  lofs_vnops.o
1151 NAMEFS_OBJS += namevfs.o      namevno.o
1153 NFS_OBJS += nfs_client.o      nfs_common.o  nfs_dump.o \
1154     nfs_subr.o      nfs_vfsops.o  nfs_vnops.o \
1155     nfs_xdr.o      nfs_sys.o      nfs_strerror.o \
1156     nfs3_vfsops.o  nfs3_vnops.o  nfs3_xdr.o \
1157     nfs_acl_vnops.o  nfs_acl_xdr.o  nfs4_vfsops.o \
1158     nfs4_vnops.o  nfs4_xdr.o  nfs4_idmap.o \
1159     nfs4_shadow.o  nfs4_subr.o \
1160     nfs4_attr.o      nfs4_rnode.o  nfs4_client.o \
1161     nfs4_acache.o  nfs4_common.o  nfs4_client_state.o \
1162     nfs4_callback.o  nfs4_recovery.o  nfs4_client_secinfo.o \
1163     nfs4_client_debug.o  nfs_stats.o \
1164     nfs4_acl.o      nfs4_stub_vnops.o  nfs_cmd.o
1166 NFSSRV_OBJS += nfs_server.o      nfs_srv.o      nfs3_srv.o \
1167     nfs_acl_srv.o  nfs_auth.o      nfs_auth_xdr.o \
1168     nfs_export.o  nfs_log.o      nfs_log_xdr.o \
1169     nfs4_srv.o    nfs4_state.o  nfs4_srv_attr.o \
1170     nfs4_srv_ns.o  nfs4_db.o      nfs4_srv_deleg.o \
1171     nfs4_deleg_ops.o  nfs4_srv_readdir.o  nfs4_dispatch.o
1173 SMBSRV_SHARED_OBJS += \
1174     smb_inet.o \
1175     smb_match.o \
1176     smb_msgbuf.o \
1177     smb_oem.o \
1178     smb_string.o \
1179     smb_utf8.o \
1180     smb_door_legacy.o \
1181     smb_xdr.o \
1182     smb_token.o \
1183     smb_token_xdr.o \

```


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

19

```

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

1188 SMBSRV_OBJS += $(SMBSRV_SHARED_OBJS) \
1189         smb_acl.o \
1190         smb_alloc.o \
1191         smb_close.o \
1192         smb_common_open.o \
1193         smb_common_transact.o \
1194         smb_create.o \
1195         smb_delete.o \
1196         smb_directory.o \
1197         smb_dispatch.o \
1198         smb_echo.o \
1199         smb_fem.o \
1200         smb_find.o \
1201         smb_flush.o \
1202         smb_fsinfo.o \
1203         smb_fsops.o \
1204         smb_init.o \
1205         smb_kdoor.o \
1206         smb_kshare.o \
1207         smb_kutil.o \
1208         smb_lock.o \
1209         smb_lock_byte_range.o \
1210         smb_locking_andx.o \
1211         smb_logoff_andx.o \
1212         smb_mangle_name.o \
1213         smb_mbuf_marshallng.o \
1214         smb_mbuf_util.o \
1215         smb_negotiate.o \
1216         smb_net.o \
1217         smb_node.o \
1218         smb_nt_cancel.o \
1219         smb_nt_create_andx.o \
1220         smb_nt_transact_create.o \
1221         smb_nt_transact_ioctl.o \
1222         smb_nt_transact_notify_change.o \
1223         smb_nt_transact_quota.o \
1224         smb_nt_transact_security.o \
1225         smb_odir.o \
1226         smb_ofile.o \
1227         smb_open_andx.o \
1228         smb_opipe.o \
1229         smb_oplock.o \
1230         smb_pathname.o \
1231         smb_print.o \
1232         smb_process_exit.o \
1233         smb_query_fileinfo.o \
1234         smb_read.o \
1235         smb_rename.o \
1236         smb_sd.o \
1237         smb_seek.o \
1238         smb_server.o \
1239         smb_session.o \
1240         smb_session_setup_andx.o \
1241         smb_set_fileinfo.o \
1242         smb_signing.o \
1243         smb_tree.o \
1244         smb_trans2_create_directory.o \
1245         smb_trans2_dfs.o \
1246         smb_trans2_find.o \
1247         smb_tree_connect.o \
1248         smb_unlock_byte_range.o \
1249         smb_user.o

```

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

20

```

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

1256 PCFS_OBJS += pc_alloc.o pc_dir.o pc_node.o pc_subr.o \
1257         pc_vfsops.o pc_vnops.o

1259 PROC_OBJS += prcontrol.o prioctl.o prsubr.o prusr.io \
1260         prvfsops.o prvnops.o

1262 MNTFS_OBJS += mntvfsops.o mntvnops.o

1264 SHAREFS_OBJS += sharetab.o sharefs_vfsops.o sharefs_vnops.o

1266 SPEC_OBJS += specsubr.o specvops.o specvnops.o

1268 SOCK_OBJS += socksubr.o sockvops.o sockparams.o \
1269         socksyscalls.o socktpi.o sockstr.o \
1270         sockcommon_vnops.o sockcommon_subr.o \
1271         sockcommon_sops.o sockcommon.o \
1272         sock_notsupp.o socknotify.o \
1273         nl7c.o nl7curi.o nl7chttp.o nl7clogd.o \
1274         nl7cnca.o sodirect.o sockfilter.o

1276 TMPFS_OBJS += tmp_dir.o tmp_subr.o tmp_tnode.o tmp_vfsops.o \
1277         tmp_vnops.o

1279 UDFS_OBJS += udf_alloc.o udf_bmap.o udf_dir.o \
1280         udf_inode.o udf_subr.o udf_vfsops.o \
1281         udf_vnops.o

1283 UFS_OBJS += ufs_alloc.o ufs_bmap.o ufs_dir.o ufs_xattr.o \
1284         ufs_inode.o ufs_subr.o ufs_tables.o ufs_vfsops.o \
1285         ufs_vnops.o quota.o quotacalls.o quota_ufs.o \
1286         ufs_filio.o ufs_lockfs.o ufs_thread.o ufs_trans.o \
1287         ufs_acl.o ufs_panic.o ufs_directio.o ufs_log.o \
1288         ufs_extvnops.o ufs_snap.o lufs.o lufs_thread.o \
1289         lufs_log.o lufs_map.o lufs_top.o lufs_debug.o

1290 VSCAN_OBJS += vscan_drv.o vscan_svc.o vscan_door.o

1292 NSMB_OBJS += smb_conn.o smb_dev.o smb_iod.o smb_pass.o \
1293         smb_rq.o smb_sign.o smb_smb.o smb_subrs.o \
1294         smb_time.o smb_tran.o smb_trantcp.o smb_usr.o \
1295         subr_mchain.o

1297 SMBFS_COMMON_OBJS += smbfs_ntacl.o
1298 SMBFS_OBJS += smbfs_vfsops.o smbfs_vnops.o smbfs_node.o \
1299         smbfs_acl.o smbfs_client.o smbfs_smb.o \
1300         smbfs_subr.o smbfs_subr2.o \
1301         smbfs_rwlock.o smbfs_xattr.o \
1302         $(SMBFS_COMMON_OBJS)

1305 #
1306 # LVM modules
1307 #
1308 MD_OBJS += md.o md_error.o md_ioctl.o md_mddb.o md_names.o \
1309         md_med.o md_rename.o md_subr.o

1311 MD_COMMON_OBJS = md_convert.o md_crc.o md_revchk.o

1313 MD_DERIVED_OBJS = metamed_xdr.o meta_basic_xdr.o

1315 SOFTPART_OBJS += sp.o sp_ioctl.o

```

```

1317 STRIPE_OBJS += stripe.o stripe_ioctl.o
1319 HOTSPARES_OBJS += hotspares.o

1321 RAID_OBJS += raid.o raid_ioctl.o raid_replay.o raid_resync.o raid_hotspare.o

1323 MIRROR_OBJS += mirror.o mirror_ioctl.o mirror_resync.o

1325 NOTIFY_OBJS += md_notify.o

1327 TRANS_OBJS += mdtrans.o trans_ioctl.o trans_log.o

1329 ZFS_COMMON_OBJS += \
1330     arc.o \
1331     bplist.o \
1332     bpobj.o \
1333     bptree.o \
1334     dbuf.o \
1335     ddt.o \
1336     ddt_zap.o \
1337     dmuf.o \
1338     dmuf_diff.o \
1339     dmuf_send.o \
1340     dmuf_object.o \
1341     dmuf_objset.o \
1342     dmuf_traverse.o \
1343     dmuf_tx.o \
1344     dnode.o \
1345     dnode_sync.o \
1346     dsl_dir.o \
1347     dsl_dataset.o \
1348     dsl_deadlist.o \
1349     dsl_destroy.o \
1350     dsl_pool.o \
1351     dsl_synctask.o \
1352     dsl_userhold.o \
1353     dmuf_zfetch.o \
1354     dsl_deleg.o \
1355     dsl_prop.o \
1356     dsl_scan.o \
1357     zfeature.o \
1358     gzip.o \
1359     lz4.o \
1360     lzjb.o \
1361     metaslab.o \
1362     refcount.o \
1363     rwlock.o \
1364     sa.o \
1365     sha256.o \
1366     spa.o \
1367     spa_config.o \
1368     spa_errlog.o \
1369     spa_history.o \
1370     spa_misc.o \
1371     space_map.o \
1372     txg.o \
1373     uberblock.o \
1374     unique.o \
1375     vdev.o \
1376     vdev_cache.o \
1377     vdev_file.o \
1378     vdev_label.o \
1379     vdev_mirror.o \
1380     vdev_missing.o \
1381     vdev_queue.o \

```

```

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

1401 ZFS_SHARED_OBJS += \
1402     zfeature_common.o \
1403     zfs_comutil.o \
1404     zfs_deleg.o \
1405     zfs_fletcher.o \
1406     zfs_namecheck.o \
1407     zfs_prop.o \
1408     zpool_prop.o \
1409     zprop_common.o

1411 ZFS_OBJS += \
1412     $(ZFS_COMMON_OBJS) \
1413     $(ZFS_SHARED_OBJS) \
1414     vdev_disk.o \
1415     zfs_acl.o \
1416     zfs_ctldir.o \
1417     zfs_dir.o \
1418     zfs_ioctl.o \
1419     zfs_log.o \
1420     zfs_onexit.o \
1421     zfs_replay.o \
1422     zfs_rlock.o \
1423     zfs_vfsops.o \
1424     zfs_vnops.o \
1425     zvol.o

1427 ZUT_OBJS += \
1428     zut.o

1430 #
1431 # streams modules
1432 #
1433 BUFMOD_OBJS += bufmod.o

1435 CONNLD_OBJS += connld.o

1437 DEDUMP_OBJS += dedump.o

1439 DRCOMPAT_OBJS += drcompat.o

1441 LDLINUX_OBJS += ldlinux.o

1443 LDTERM_OBJS += ldterm.o uwidth.o

1445 PKKT_OBJS += pkt.o

1447 PFMOD_OBJS += pfmod.o

```

```

1449 PTEM_OBJS +=      ptem.o
1451 REDIRMOD_OBJS +=  strredirm.o
1453 TIMOD_OBJS +=    timod.o
1455 TIRDWR_OBJS +=   tirdwr.o
1457 TTCOMPAT_OBJS += ttcompat.o
1459 LOG_OBJS +=      log.o
1461 PIPEMOD_OBJS +=  pipemod.o
1463 RPCMOD_OBJS +=    rpcmod.o      clnt_cots.o      clnt_clts.o \
1464                  clnt_gen.o      clnt_perr.o      mt_rpcinit.o    rpc_calmsg.o \
1465                  rpc_prot.o       rpc_sztypes.o    rpc_subr.o      rpcb_prot.o \
1466                  svc.o            svc_clts.o       svc_gen.o       svc_cots.o \
1467                  rpcsys.o         xdr_sizeof.o    clnt_rdma.o     svc_rdma.o \
1468                  xdr_rdma.o        rdma_subr.o     xdrdma_sizeof.o
1470 KLMOD_OBJS +=     klmmod.o \
1471                  nlm_impl.o \
1472                  nlm_rpc_handle.o \
1473                  nlm_dispatch.o \
1474                  nlm_rpc_svc.o \
1475                  nlm_client.o \
1476                  nlm_service.o \
1477                  nlm_prot_clnt.o \
1478                  nlm_prot_xdr.o \
1479                  nlm_rpc_clnt.o \
1480                  nsm_addr_clnt.o \
1481                  nsm_addr_xdr.o \
1482                  sm_inter_clnt.o \
1483                  sm_inter_xdr.o
1485 KLMOPS_OBJS +=   klmops.o
1487 TLIMOD_OBJS +=   tlimod.o      t_kalloc.o      t_kbind.o      t_kclose.o \
1488                  t_kconnect.o    t_kfree.o      t_kgtstate.o   t_kopen.o \
1489                  t_krcvudat.o     t_ksndudat.o  t_kspoll.o     t_kunbind.o \
1490                  t_kutil.o
1492 RLMOD_OBJS +=    rlmmod.o
1494 TELMOD_OBJS +=   telmod.o
1496 CRYPTMOD_OBJS += cryptmod.o
1498 KB_OBJS +=       kbd.o          keytables.o
1500 #
1501 #                   ID mapping module
1502 #
1503 IDMAP_OBJS +=    idmap_mod.o      idmap_kapi.o    idmap_xdr.o     idmap_cache.o
1505 #
1506 #                   scheduling class modules
1507 #
1508 SDC_OBJS +=      sysdc.o
1510 RT_OBJS +=       rt.o
1511 RT_DPTBL_OBJS += rt_dptbl.o
1513 TS_OBJS +=       ts.o

```

```

1514 TS_DPTBL_OBJS +=  ts_dptbl.o
1516 IA_OBJS +=        ia.o
1518 FSS_OBJS +=       fss.o
1520 FX_OBJS +=        fx.o
1521 FX_DPTBL_OBJS +=  fx_dptbl.o
1523 #
1524 #                   Inter-Process Communication (IPC) modules
1525 #
1526 IPC_OBJS +=        ipc.o
1528 IPCMSG_OBJS +=    msg.o
1530 IPCSEM_OBJS +=    sem.o
1532 IPCSHM_OBJS +=    shm.o
1534 #
1535 #                   bignum module
1536 #
1537 COMMON_BIGNUM_OBJS += bignum_mod.o bignumimpl.o
1539 BIGNUM_OBJS +=    $(COMMON_BIGNUM_OBJS) $(BIGNUM_PSR_OBJS)
1541 #
1542 #                   kernel cryptographic framework
1543 #
1544 KCF_OBJS +=        kcf.o kcf_callprov.o kcf_cbufcall.o kcf_cipher.o kcf_crypto.o \
1545                  kcf_cryptoadm.o kcf_ctxops.o kcf_digest.o kcf_dual.o \
1546                  kcf_keys.o kcf_mac.o kcf_mech_tabs.o kcf_miscapi.o \
1547                  kcf_object.o kcf_policy.o kcf_prov_lib.o kcf_prov_tabs.o \
1548                  kcf_sched.o kcf_session.o kcf_sign.o kcf_spi.o kcf_verify.o \
1549                  kcf_random.o modes.o ecb.o cbc.o ctr.o ccm.o gcm.o \
1550                  fips_random.o
1552 CRYPTOADM_OBJS += cryptoadm.o
1554 CRYPTO_OBJS +=    crypto.o
1556 DPROV_OBJS +=     dprov.o
1558 DCA_OBJS +=        dca.o dca_3des.o dca_debug.o dca_dsa.o dca_kstat.o dca_rng.o \
1559                  dca_rsa.o
1561 AESPROV_OBJS +=    aes.o aes_impl.o aes_modes.o
1563 ARCFOURPROV_OBJS += arcfour.o arcfour_crypt.o
1565 BLOWFISHPROV_OBJS += blowfish.o blowfish_impl.o
1567 ECCPROV_OBJS +=    ecc.o ec.o ec2_163.o ec2_mont.o ecdecode.o ecl_mult.o \
1568                  ecp_384.o ecp_jac.o ec2_193.o ec2_193.o ecl.o ecp_192.o ecp_521.o \
1569                  ecp_jm.o ec2_233.o ecl_curve.o ecp_224.o ecp_aff.o \
1570                  ecp_mont.o ec2_aff.o ec_naf.o ecl_gf.o ecp_256.o mp_gf2m.o \
1571                  mpi.o mplogic.o mpmontg.o mpprime.o oid.o \
1572                  secitem.o ec2_test.o ecp_test.o
1574 RSAPROV_OBJS +=    rsa.o rsa_impl.o pkcs1.o
1576 SWRANDPROV_OBJS += swrand.o
1578 #
1579 #                   kernel SSL

```

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

25

```

1580 #
1581 KSSL_OBJS +=      kssl.o ksslioc1.o

1583 KSSL_SOCKFIL_MOD_OBJS += ksslfilter.o ksslapi.o ksslrec.o

1585 #
1586 #             misc. modules
1587 #

1589 C2AUDIT_OBJS +=  adr.o audit.o audit_event.o audit_io.o \
1590                audit_path.o audit_start.o audit_syscalls.o audit_token.o \
1591                audit_mem.o

1593 PCIC_OBJS +=    pcic.o

1595 RPCSEC_OBJS +=  secmod.o      sec_clnt.o      sec_svc.o      sec_gen.o \
1596                auth_des.o     auth_kern.o     auth_none.o     auth_loopb.o \
1597                authdesprt.o   authdesubr.o   authu_prot.o \
1598                key_call.o     key_prot.o     svc_authu.o     svcauthdes.o

1600 RPCSEC_GSS_OBJS +=      rpcsec_gssmod.o rpcsec_gss.o rpcsec_gss_misc.o \
1601                rpcsec_gss_utils.o svc_rpcsec_gss.o

1603 CONSCONFIG_OBJS +=  consconfig.o

1605 CONSCONFIG_DACF_OBJS += consconfig_dacf.o consplat.o

1607 TEM_OBJS +=  tem.o tem_safe.o 6x10.o 7x14.o 12x22.o

1609 KBTRANS_OBJS +=      \
1610                kbtrans.o \
1611                kbtrans_keytables.o \
1612                kbtrans_polled.o \
1613                kbtrans_streams.o \
1614                usb_keytables.o

1616 KGSSD_OBJS +=  gssd_clnt_stubs.o gssd_handle.o gssd_prot.o \
1617                gss_display_name.o gss_release_name.o gss_import_name.o \
1618                gss_release_buffer.o gss_release_oid_set.o gen_oids.o gssdmod.o

1620 KGSSD_DERIVED_OBJS = gssd_xdr.o

1622 KGSS_DUMMY_OBJS +=  dmech.o

1624 KSOCKET_OBJS +=  ksocket.o ksocket_mod.o

1626 CRYPTO=  cksumtypes.o decrypt.o encrypt.o encrypt_length.o etypes.o \
1627          nfold.o verify_checksum.o prng.o block_size.o make_checksum.o \
1628          checksum_length.o hmac.o default_state.o mandatory_sumtype.o

1630 # crypto/des
1631 CRYPTO_DES= f_cbc.o f_cksum.o f_parity.o weak_key.o d3_cbc.o ef_crypto.o

1633 CRYPTO_DK=  checksum.o derive.o dk_decrypt.o dk_encrypt.o

1635 CRYPTO_ARCFOUR= k5_arcfour.o

1637 # crypto/enc_provider
1638 CRYPTO_ENC=  des.o des3.o arcfour_provider.o aes_provider.o

1640 # crypto/hash_provider
1641 CRYPTO_HASH= hash_kef_generic.o hash_kmd5.o hash_crc32.o hash_kshal.o

1643 # crypto/keyhash_provider
1644 CRYPTO_KEYHASH= descbc.o k5_kmd5des.o k_hmac_md5.o

```

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

26

```

1646 # crypto/crc32
1647 CRYPTO_CRC32=  crc32.o

1649 # crypto/old
1650 CRYPTO_OLD=  old_decrypt.o old_encrypt.o

1652 # crypto/raw
1653 CRYPTO_RAW=  raw_decrypt.o raw_encrypt.o

1655 K5_KRB=  kfree.o copy_key.o \
1656          parse.o init_ctx.o \
1657          ser_adata.o ser_addr.o \
1658          ser_auth.o ser_cksum.o \
1659          ser_key.o ser_princ.o \
1660          serialize.o unparse.o \
1661          ser_actx.o

1663 K5_OS=  timeofday.o toffset.o \
1664          init_os_ctx.o c_ustime.o

1666 SEAL=  seal.o unseal.o

1668 MECH=  delete_sec_context.o \
1669          import_sec_context.o \
1670          gssapi_krb5.o \
1671          k5seal.o k5unseal.o k5sealv3.o \
1672          ser_sctx.o \
1673          sign.o \
1674          util_crypt.o \
1675          util_validate.o util_ordering.o \
1676          util_seqnum.o util_set.o util_seed.o \
1677          wrap_size_limit.o verify.o

1681 MECH_GEN=  util_token.o

1684 KGSS_KRB5_OBJS +=  krb5mech.o \
1685                    $(MECH) $(SEAL) $(MECH_GEN) \
1686                    $(CRYPTO) $(CRYPTO_DES) $(CRYPTO_DK) $(CRYPTO_ARCFOUR) \
1687                    $(CRYPTO_ENC) $(CRYPTO_HASH) \
1688                    $(CRYPTO_KEYHASH) $(CRYPTO_CRC32) \
1689                    $(CRYPTO_OLD) \
1690                    $(CRYPTO_RAW) $(K5_KRB) $(K5_OS)

1692 DES_OBJS +=  des_crypt.o des_impl.o des_ks.o des_soft.o

1694 DLBOOT_OBJS +=  bootparam_xdr.o nfs_dlinet.o scan.o

1696 KRTLD_OBJS +=  kobj_bootflags.o getoptstr.o \
1697                kobj.o kobj_kdi.o kobj_lm.o kobj_subr.o

1699 MOD_OBJS +=  modctl.o modsubr.o modsysfile.o modconf.o modhash.o

1701 STRPLUMB_OBJS +=  strplumb.o

1703 CPR_OBJS +=  cpr_driver.o cpr_dump.o \
1704              cpr_main.o cpr_misc.o cpr_mod.o cpr_stat.o \
1705              cpr_uthread.o

1707 PROF_OBJS +=  prf.o

1709 SE_OBJS +=  se_driver.o

1711 SYSACCT_OBJS +=  acct.o

```

```

1713 ACCTCTL_OBJS += acctctl.o
1715 EXACCTSYS_OBJS += exacctsys.o
1717 KAIO_OBJS += aio.o
1719 PCMCIA_OBJS += pcmcia.o cs.o cis.o cis_callout.o cis_handlers.o cis_params.o
1721 BUSRA_OBJS += busra.o
1723 PCS_OBJS += pcs.o
1725 PCAN_OBJS += pcan.o
1727 PCATA_OBJS += pcide.o pcdisk.o pclabel.o pcata.o
1729 PCSER_OBJS += pcser.o pcser_cis.o
1731 PCWL_OBJS += pcwl.o
1733 PSET_OBJS += pset.o
1735 OHCI_OBJS += ohci.o ohci_hub.o ohci_polled.o
1737 UHCI_OBJS += uhci.o uhciutil.o uhcigt.o uhcihub.o uhcipolled.o
1739 EHCI_OBJS += ehci.o ehci_hub.o ehci_xfer.o ehci_intr.o ehci_util.o ehci_polled.o
1741 HUBD_OBJS += hubd.o
1743 USB_MID_OBJS += usb_mid.o
1745 USB_IA_OBJS += usb_ia.o
1747 UWBA_OBJS += uwba.o uwbai.o
1749 SCSA2USB_OBJS += scsa2usb.o usb_ms_bulkonly.o usb_ms_cbi.o
1751 HWAHC_OBJS += hwahc.o hwahc_util.o
1753 WUSB_DF_OBJS += wusb_df.o
1754 WUSB_FWMOD_OBJS += wusb_fwmod.o
1756 IPF_OBJS += ip_fil_solaris.o fil.o solaris.o ip_state.o ip_frag.o ip_nat.o \
1757 ip_proxy.o ip_auth.o ip_pool.o ip_hstable.o ip_lookup.o \
1758 ip_log.o misc.o ip_compat.o ip_nat6.o drand48.o
1760 IBD_OBJS += ibd.o ibd_cm.o
1762 EIBNX_OBJS += enx_main.o enx_hdlrs.o enx_ibt.o enx_log.o enx_fip.o \
1763 enx_misc.o enx_q.o enx_ctl.o
1765 EOIB_OBJS += eib_adm.o eib_chan.o eib_cmn.o eib_ctl.o eib_data.o \
1766 eib_fip.o eib_ibt.o eib_log.o eib_mac.o eib_main.o \
1767 eib_rsrc.o eib_svc.o eib_vnic.o
1769 DLPSTUB_OBJS += dlpistub.o
1771 SDP_OBJS += sdpddi.o
1773 TRILL_OBJS += trill.o
1775 CTF_OBJS += ctf_create.o ctf_decl.o ctf_error.o ctf_hash.o ctf_labels.o \
1776 ctf_lookup.o ctf_open.o ctf_types.o ctf_util.o ctf_subr.o ctf_mod.o

```

```

1778 SMBIOS_OBJS += smb_error.o smb_info.o smb_open.o smb_subr.o smb_dev.o
1780 RPCIB_OBJS += rpcib.o
1782 KMDB_OBJS += kdrv.o
1784 AFE_OBJS += afe.o
1786 BGE_OBJS += bge_main2.o bge_chip2.o bge_kstats.o bge_log.o bge_ndd.o \
1787 bge_atomic.o bge_mii.o bge_send.o bge_recv2.o bge_mii_5906.o
1789 DMFE_OBJS += dmfe_log.o dmfe_main.o dmfe_mii.o
1791 EFE_OBJS += efe.o
1793 ELXL_OBJS += elxl.o
1795 HME_OBJS += hme.o
1797 IXGB_OBJS += ixgb.o ixgb_atomic.o ixgb_chip.o ixgb_gld.o ixgb_kstats.o \
1798 ixgb_log.o ixgb_ndd.o ixgb_rx.o ixgb_tx.o ixgb_xmii.o
1800 NGE_OBJS += nge_main.o nge_atomic.o nge_chip.o nge_ndd.o nge_kstats.o \
1801 nge_log.o nge_rx.o nge_tx.o nge_xmii.o
1803 PCN_OBJS += pcn.o
1805 RGE_OBJS += rge_main.o rge_chip.o rge_ndd.o rge_kstats.o rge_log.o rge_rxtx.o
1807 URTW_OBJS += urtw.o
1809 ARN_OBJS += arn_hw.o arn_eeeprom.o arn_mac.o arn_calib.o arn_ani.o arn_phy.o arn_
1810 arn_main.o arn_recv.o arn_xmit.o arn_rc.o
1812 ATH_OBJS += ath_aux.o ath_main.o ath_osdep.o ath_rate.o
1814 ATU_OBJS += atu.o
1816 IPW_OBJS += ipw2100_hw.o ipw2100.o
1818 IWI_OBJS += ipw2200_hw.o ipw2200.o
1820 IWH_OBJS += iwh.o
1822 IWK_OBJS += iwk2.o
1824 IWP_OBJS += iwp.o
1826 MWL_OBJS += mwl.o
1828 MWLFW_OBJS += mwlfw_mode.o
1830 WPI_OBJS += wpi.o
1832 RAL_OBJS += rt2560.o ral_rate.o
1834 RUM_OBJS += rum.o
1836 RWD_OBJS += rt2661.o
1838 RWN_OBJS += rt2860.o
1840 UATH_OBJS += uath.o
1842 UATHFW_OBJS += uathfw_mod.o

```

```

1844 URAL_OBJS += ural.o
1846 RTW_OBJS += rtw.o smc93cx6.o rtwphy.o rtwphyio.o
1848 ZYD_OBJS += zyd.o zyd_usb.o zyd_hw.o zyd_fw.o
1850 MXFE_OBJS += mxfe.o
1852 MPTSAS_OBJS += mptsas.o mptsas_impl.o mptsas_init.o mptsas_raid.o mptsas_smhba.o
1854 SFE_OBJS += sfe.o sfe_util.o
1856 BFE_OBJS += bfe.o
1858 BRIDGE_OBJS += bridge.o
1860 IDM_SHARED_OBJS += base64.o
1862 IDM_OBJS += $(IDM_SHARED_OBJS) \
1863         idm.o idm_impl.o idm_text.o idm_conn_sm.o idm_so.o
1865 VR_OBJS += vr.o
1867 ATGE_OBJS += atge_main.o atge_lle.o atge_mii.o atge_ll.o atge_llc.o
1869 YGE_OBJS = yge.o
1871 #
1872 #     Build up defines and paths.
1873 #
1874 LINT_DEFS     += -Dunix
1876 #
1877 #     This duality can be removed when the native and target compilers
1878 #     are the same (or at least recognize the same command line syntax!)
1879 #     It is a bug in the current compilation system that the assembler
1880 #     can't process the -Y I, flag.
1881 #
1882 NATIVE_INC_PATH += $(INC_PATH) $(CCYFLAG)$(UTSBASE)/common
1883 AS_INC_PATH     += $(INC_PATH) -I$(UTSBASE)/common
1884 INCLUDE_PATH   += $(INC_PATH) $(CCYFLAG)$(UTSBASE)/common
1886 PCIEB_OBJS += pcieb.o
1888 #     Chelsio N110 10G NIC driver module
1889 #
1890 CH_OBJS = ch.o glue.o pe.o sge.o
1892 CH_COM_OBJS = ch_mac.o ch_subr.o cspi.o espi.o ixfl1010.o mc3.o mc4.o mc5.o \
1893         mv88elxxx.o mv88x20lx.o my3126.o pm3393.o tp.o ulp.o \
1894         vsc7321.o vsc7326.o xpak.o
1896 #
1897 #     Chelsio Terminator 4 10G NIC nexus driver module
1898 #
1899 CXGBE_FW_OBJS = t4_fw.o t4_cfg.o
1900 CXGBE_COM_OBJS = t4_hw.o common.o
1901 CXGBE_NEX_OBJS = t4_nexus.o t4_sge.o t4_mac.o t4_ioctl.o shared.o \
1902         t4_l2t.o adapter.o osdep.o
1904 #
1905 #     Chelsio Terminator 4 10G NIC driver module
1906 #
1907 CXGBE_OBJS = cxgbe.o
1909 #

```

```

1910 #     PCI strings file
1911 #
1912 PCI_STRING_OBJS = pci_strings.o
1914 NET_DACF_OBJS += net_dacf.o
1916 #
1917 #     Xframe 10G NIC driver module
1918 #
1919 XGE_OBJS = xge.o xgell.o
1921 XGE_HAL_OBJS = xgehal-channel.o xgehal-fifo.o xgehal-ring.o xgehal-config.o \
1922         xgehal-driver.o xgehal-mm.o xgehal-stats.o xgehal-device.o \
1923         xge-queue.o xgehal-mgmt.o xgehal-mgmtaux.o
1925 #
1926 #     e1000g module
1927 #
1928 E1000G_OBJS += e1000_80003es2lan.o e1000_82540.o e1000_82541.o e1000_82542.o \
1929         e1000_82543.o e1000_82571.o e1000_api.o e1000_ich8lan.o \
1930         e1000_mac.o e1000_manage.o e1000_nvmm.o e1000_osdep.o \
1931         e1000_phy.o e1000g_debug.o e1000g_main.o e1000g_alloc.o \
1932         e1000g_tx.o e1000g_rx.o e1000g_stat.o
1934 #
1935 #     Intel 82575 1G NIC driver module
1936 #
1937 IGB_OBJS = igb_82575.o igb_api.o igb_mac.o igb_manage.o \
1938         igb_nvmm.o igb_osdep.o igb_phy.o igb_buf.o \
1939         igb_debug.o igb_gld.o igb_log.o igb_main.o \
1940         igb_rx.o igb_stat.o igb_tx.o
1942 #
1943 #     Intel Pro/100 NIC driver module
1944 #
1945 IPRB_OBJS = iprb.o
1947 #
1948 #     Intel 10GbE PCIE NIC driver module
1949 #
1950 IXGBE_OBJS = ixgbe_82598.o ixgbe_82599.o ixgbe_api.o \
1951         ixgbe_common.o ixgbe_phy.o \
1952         ixgbe_buf.o ixgbe_debug.o ixgbe_gld.o \
1953         ixgbe_log.o ixgbe_main.o \
1954         ixgbe_osdep.o ixgbe_rx.o ixgbe_stat.o \
1955         ixgbe_tx.o ixgbe_x540.o ixgbe_mbx.o
1957 #
1958 #     NIU 10G/1G driver module
1959 #
1960 NXGE_OBJS = nxge_mac.o nxge_ipp.o nxge_rxdma.o \
1961         nxge_txdma.o nxge_txc.o nxge_main.o \
1962         nxge_hw.o nxge_fzc.o nxge_virtual.o \
1963         nxge_send.o nxge_classify.o nxge_fflp.o \
1964         nxge_fflp_hash.o nxge_ndd.o nxge_kstats.o \
1965         nxge_zcp.o nxge_fm.o nxge_espc.o nxge_hv.o \
1966         nxge_hio.o nxge_hio_guest.o nxge_intr.o
1968 NXGE_NPI_OBJS = \
1969         npi.o npi_mac.o npi_ipp.o \
1970         npi_txdma.o npi_rxdma.o npi_txc.o \
1971         npi_zcp.o npi_espc.o npi_fflp.o \
1972         npi_vir.o
1974 NXGE_HCALL_OBJS = \
1975         nxge_hcall.o

```

```

1977 #
1978 # Virtio modules
1979 #

1981 # Virtio core
1982 VIRTIO_OBJS = virtio.o

1984 # Virtio block driver
1985 VIOBLK_OBJS = vioblk.o

1987 #
1988 #     kiconv modules
1989 #
1990 KICONV_EMEA_OBJS += kiconv_emea.o

1992 KICONV_JA_OBJS += kiconv_ja.o

1994 KICONV_KO_OBJS += kiconv_cck_common.o kiconv_ko.o

1996 KICONV_SC_OBJS += kiconv_cck_common.o kiconv_sc.o

1998 KICONV_TC_OBJS += kiconv_cck_common.o kiconv_tc.o

2000 #
2001 #     AAC module
2002 #
2003 AAC_OBJS = aac.o aac_ioctl.o

2005 #
2006 #     sdc card modules
2007 #
2008 SDA_OBJS =     sda_cmd.o sda_host.o sda_init.o sda_mem.o sda_mod.o sda_slot.o
2009 SDHOST_OBJS = sdhost.o

2011 #
2012 #     hxge 10G driver module
2013 #
2014 HXGE_OBJS =     hxge_main.o hxge_vmac.o hxge_send.o           \
2015                hxge_txdma.o hxge_rxdma.o hxge_virtual.o     \
2016                hxge_fm.o hxge_fzc.o hxge_hw.o hxge_kstats.o \
2017                hxge_ndd.o hxge_pfc.o                       \
2018                hpi.o hpi_vmac.o hpi_rxdma.o hpi_txdma.o     \
2019                hpi_vir.o hpi_pfc.o

2021 #
2022 #     MEGARAID_SAS module
2023 #
2024 MEGA_SAS_OBJS = megaraid_sas.o

2026 #
2027 #     MR_SAS module
2028 #
2029 MR_SAS_OBJS = ld_pd_map.o mr_sas.o mr_sas_tbolt.o mr_sas_list.o

2031 #
2032 #     ISCSI_INITIATOR module
2033 #
2034 ISCSI_INITIATOR_OBJS = chap.o iscsi_io.o iscsi_thread.o       \
2035                       iscsi_ioctl.o iscsid.o iscsi.o         \
2036                       iscsi_login.o isns_client.o iscsiAuthClient.o \
2037                       iscsi_lun.o iscsiAuthClientGlue.o     \
2038                       iscsi_net.o nvfile.o iscsi_cmd.o      \
2039                       iscsi_queue.o persistent.o iscsi_conn.o \
2040                       iscsi_sess.o radius_auth.o iscsi_crc.o \
2041                       iscsi_stats.o radius_packet.o iscsi_doorclt.o \

```

```

2042                                     iscsi_targetparam.o utils.o kifconf.o

2044 #
2045 #     ntxn 10Gb/1Gb NIC driver module
2046 #
2047 NTXN_OBJS =     unm_nic_init.o unm_gem.o unm_nic_hw.o unm_ndd.o \
2048                unm_nic_main.o unm_nic_isr.o unm_nic_ctx.o niu.o

2050 #
2051 #     Myricom 10Gb NIC driver module
2052 #
2053 MYRI10GE_OBJS = myril0ge.o myril0ge_lro.o

2055 #     nulldriver module
2056 #
2057 NULLDRIVER_OBJS =     nulldriver.o

2059 TPM_OBJS =     tpm.o tpm_hcall.o

```

```

*****
73698 Fri Aug 16 09:04:45 2013
new/usr/src/uts/common/Makefile.rules
195 Need replacement for nfs/lockd+klm
*****
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 #
26 #
27 #
28 # uts/common/Makefile.rules
29 #
30 # This Makefile defines all the file build rules for the directory
31 # uts/common and its children. These are the source files which may
32 # be considered common to all SunOS systems.
33 #
34 # The following two-level ordering must be maintained in this file.
35 # Lines are sorted first in order of decreasing specificity based on
36 # the first directory component. That is, sun4u rules come before
37 # sparc rules come before common rules.
38 #
39 # Lines whose initial directory components are equal are sorted
40 # alphabetically by the remaining components.
41 #
42 #
43 # Section 1a: C objects build rules
44 #
45 $(OBJSDIR)/%.o: $(COMMONBASE)/crypto/aes/%.c
46 $(COMPILE.c) -o $@ $<
47 $(CTFCONVERT_O)
48 #
49 $(OBJSDIR)/%.o: $(COMMONBASE)/crypto/arcfour/%.c
50 $(COMPILE.c) -o $@ $<
51 $(CTFCONVERT_O)
52 #
53 $(OBJSDIR)/%.o: $(COMMONBASE)/crypto/blowfish/%.c
54 $(COMPILE.c) -o $@ $<
55 $(CTFCONVERT_O)
56 #
57 $(OBJSDIR)/%.o: $(COMMONBASE)/crypto/ecc/%.c
58 $(COMPILE.c) -o $@ $<
59 $(CTFCONVERT_O)
60 #
61 $(OBJSDIR)/%.o: $(COMMONBASE)/crypto/modes/%.c

```

```

62 $(COMPILE.c) -o $@ $<
63 $(CTFCONVERT_O)
64 #
65 $(OBJSDIR)/%.o: $(COMMONBASE)/crypto/padding/%.c
66 $(COMPILE.c) -o $@ $<
67 $(CTFCONVERT_O)
68 #
69 $(OBJSDIR)/%.o: $(COMMONBASE)/crypto/rng/%.c
70 $(COMPILE.c) -o $@ $<
71 $(CTFCONVERT_O)
72 #
73 $(OBJSDIR)/%.o: $(COMMONBASE)/crypto/rsa/%.c
74 $(COMPILE.c) -o $@ $<
75 $(CTFCONVERT_O)
76 #
77 $(OBJSDIR)/%.o: $(COMMONBASE)/bignum/%.c
78 $(COMPILE.c) -o $@ $<
79 $(CTFCONVERT_O)
80 #
81 $(OBJSDIR)/%.o: $(UTSBASE)/common/bignum/%.c
82 $(COMPILE.c) -o $@ $<
83 $(CTFCONVERT_O)
84 #
85 $(OBJSDIR)/%.o: $(COMMONBASE)/mpi/%.c
86 $(COMPILE.c) -o $@ $<
87 $(CTFCONVERT_O)
88 #
89 $(OBJSDIR)/%.o: $(COMMONBASE)/acl/%.c
90 $(COMPILE.c) -o $@ $<
91 $(CTFCONVERT_O)
92 #
93 $(OBJSDIR)/%.o: $(COMMONBASE)/avl/%.c
94 $(COMPILE.c) -o $@ $<
95 $(CTFCONVERT_O)
96 #
97 $(OBJSDIR)/%.o: $(COMMONBASE)/ucode/%.c
98 $(COMPILE.c) -o $@ $<
99 $(CTFCONVERT_O)
100 #
101 $(OBJSDIR)/%.o: $(UTSBASE)/common/brand/sn1/%.c
102 $(COMPILE.c) -o $@ $<
103 $(CTFCONVERT_O)
104 #
105 $(OBJSDIR)/%.o: $(UTSBASE)/common/brand/solaris10/%.c
106 $(COMPILE.c) -o $@ $<
107 $(CTFCONVERT_O)
108 #
109 $(OBJSDIR)/%.o: $(UTSBASE)/common/c2/%.c
110 $(COMPILE.c) -o $@ $<
111 $(CTFCONVERT_O)
112 #
113 $(OBJSDIR)/%.o: $(UTSBASE)/common/conf/%.c
114 $(COMPILE.c) -o $@ $<
115 $(CTFCONVERT_O)
116 #
117 $(OBJSDIR)/%.o: $(UTSBASE)/common/contract/%.c
118 $(COMPILE.c) -o $@ $<
119 $(CTFCONVERT_O)
120 #
121 $(OBJSDIR)/%.o: $(UTSBASE)/common/cpr/%.c
122 $(COMPILE.c) -o $@ $<
123 $(CTFCONVERT_O)
124 #
125 $(OBJSDIR)/%.o: $(UTSBASE)/common/ctf/%.c
126 $(COMPILE.c) -o $@ $<
127 $(CTFCONVERT_O)

```



```

129 $(OBJSDIR)/%.o: $(COMMONBASE)/ctf/%.c
130 $(COMPILE.c) -o $@ $<
131 $(CTFCONVERT_O)

133 $(OBJSDIR)/%.o: $(COMMONBASE)/crypto/des/%.c
134 $(COMPILE.c) -o $@ $<
135 $(CTFCONVERT_O)

137 $(OBJSDIR)/%.o: $(COMMONBASE)/smbios/%.c
138 $(COMPILE.c) -o $@ $<
139 $(CTFCONVERT_O)

141 $(OBJSDIR)/%.o: $(UTSBASE)/common/des/%.c
142 $(COMPILE.c) -o $@ $<
143 $(CTFCONVERT_O)

145 $(OBJSDIR)/%.o: $(UTSBASE)/common/crypto/api/%.c
146 $(COMPILE.c) -o $@ $<
147 $(CTFCONVERT_O)

149 $(OBJSDIR)/%.o: $(UTSBASE)/common/crypto/core/%.c
150 $(COMPILE.c) -o $@ $<
151 $(CTFCONVERT_O)

153 $(OBJSDIR)/%.o: $(UTSBASE)/common/crypto/io/%.c
154 $(COMPILE.c) -o $@ $<
155 $(CTFCONVERT_O)

157 $(OBJSDIR)/%.o: $(UTSBASE)/common/crypto/spi/%.c
158 $(COMPILE.c) -o $@ $<
159 $(CTFCONVERT_O)

161 $(OBJSDIR)/%.o: $(COMMONBASE)/pci/%.c
162 $(COMPILE.c) -o $@ $<
163 $(CTFCONVERT_O)

165 $(OBJSDIR)/%.o: $(COMMONBASE)/devid/%.c
166 $(COMPILE.c) -o $@ $<
167 $(CTFCONVERT_O)

169 $(OBJSDIR)/%.o: $(UTSBASE)/common/disp/%.c
170 $(COMPILE.c) -o $@ $<
171 $(CTFCONVERT_O)

173 $(OBJSDIR)/%.o: $(UTSBASE)/common/dtrace/%.c
174 $(COMPILE.c) -o $@ $<
175 $(CTFCONVERT_O)

177 $(OBJSDIR)/%.o: $(COMMONBASE)/exacct/%.c
178 $(COMPILE.c) -o $@ $<
179 $(CTFCONVERT_O)

181 $(OBJSDIR)/%.o: $(UTSBASE)/common/exec/aout/%.c
182 $(COMPILE.c) -o $@ $<
183 $(CTFCONVERT_O)

185 $(OBJSDIR)/%.o: $(UTSBASE)/common/exec/elf/%.c
186 $(COMPILE.c) -o $@ $<
187 $(CTFCONVERT_O)

189 $(OBJSDIR)/%.o: $(UTSBASE)/common/exec/intp/%.c
190 $(COMPILE.c) -o $@ $<
191 $(CTFCONVERT_O)

193 $(OBJSDIR)/%.o: $(UTSBASE)/common/exec/shbin/%.c

```

```

194 $(COMPILE.c) -o $@ $<
195 $(CTFCONVERT_O)

197 $(OBJSDIR)/%.o: $(UTSBASE)/common/exec/java/%.c
198 $(COMPILE.c) -o $@ $<
199 $(CTFCONVERT_O)

201 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/%.c
202 $(COMPILE.c) -o $@ $<
203 $(CTFCONVERT_O)

205 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/autofs/%.c
206 $(COMPILE.c) -o $@ $<
207 $(CTFCONVERT_O)

209 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/cacheofs/%.c
210 $(COMPILE.c) -o $@ $<
211 $(CTFCONVERT_O)

213 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/dcfis/%.c
214 $(COMPILE.c) -o $@ $<
215 $(CTFCONVERT_O)

217 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/devfs/%.c
218 $(COMPILE.c) -o $@ $<
219 $(CTFCONVERT_O)

221 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/ctfs/%.c
222 $(COMPILE.c) -o $@ $<
223 $(CTFCONVERT_O)

225 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/doorfs/%.c
226 $(COMPILE.c) -o $@ $<
227 $(CTFCONVERT_O)

229 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/dev/%.c
230 $(COMPILE.c) -o $@ $<
231 $(CTFCONVERT_O)

233 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/fd/%.c
234 $(COMPILE.c) -o $@ $<
235 $(CTFCONVERT_O)

237 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/fifofs/%.c
238 $(COMPILE.c) -o $@ $<
239 $(CTFCONVERT_O)

241 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/hsfs/%.c
242 $(COMPILE.c) -o $@ $<
243 $(CTFCONVERT_O)

245 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/lofs/%.c
246 $(COMPILE.c) -o $@ $<
247 $(CTFCONVERT_O)

249 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/mntfs/%.c
250 $(COMPILE.c) -o $@ $<
251 $(CTFCONVERT_O)

253 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/namefs/%.c
254 $(COMPILE.c) -o $@ $<
255 $(CTFCONVERT_O)

257 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/nfs/%.c
258 $(COMPILE.c) -o $@ $<
259 $(CTFCONVERT_O)

```

```

261 $(OBJSDIR)/%.o: $(COMMONBASE)/smbsrv/%.c
262 $(COMPILE.c) -o $@ $<
263 $(CTFCONVERT_O)

265 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/smbsrv/%.c
266 $(COMPILE.c) -o $@ $<
267 $(CTFCONVERT_O)

269 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/objfs/%.c
270 $(COMPILE.c) -o $@ $<
271 $(CTFCONVERT_O)

273 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/pcfefs/%.c
274 $(COMPILE.c) -o $@ $<
275 $(CTFCONVERT_O)

277 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/portfs/%.c
278 $(COMPILE.c) -o $@ $<
279 $(CTFCONVERT_O)

281 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/proc/%.c
282 $(COMPILE.c) -o $@ $<
283 $(CTFCONVERT_O)

285 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/sharefs/%.c
286 $(COMPILE.c) -o $@ $<
287 $(CTFCONVERT_O)

289 $(OBJSDIR)/%.o: $(COMMONBASE)/smbclnt/%.c
290 $(COMPILE.c) -o $@ $<
291 $(CTFCONVERT_O)

293 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/smbclnt/netsmb/%.c
294 $(COMPILE.c) -o $@ $<
295 $(CTFCONVERT_O)

297 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/smbclnt/smbfs/%.c
298 $(COMPILE.c) -o $@ $<
299 $(CTFCONVERT_O)

301 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/sockfs/%.c
302 $(COMPILE.c) -o $@ $<
303 $(CTFCONVERT_O)

305 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/specfs/%.c
306 $(COMPILE.c) -o $@ $<
307 $(CTFCONVERT_O)

309 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/swapfs/%.c
310 $(COMPILE.c) -o $@ $<
311 $(CTFCONVERT_O)

313 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/tmpfs/%.c
314 $(COMPILE.c) -o $@ $<
315 $(CTFCONVERT_O)

317 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/udfs/%.c
318 $(COMPILE.c) -o $@ $<
319 $(CTFCONVERT_O)

321 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/ufs/%.c
322 $(COMPILE.c) -o $@ $<
323 $(CTFCONVERT_O)

325 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/vscan/%.c

```

```

326 $(COMPILE.c) -o $@ $<
327 $(CTFCONVERT_O)

329 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/zfs/%.c
330 $(COMPILE.c) -o $@ $<
331 $(CTFCONVERT_O)

333 $(OBJSDIR)/%.o: $(UTSBASE)/common/fs/zut/%.c
334 $(COMPILE.c) -o $@ $<
335 $(CTFCONVERT_O)

337 $(OBJSDIR)/%.o: $(COMMONBASE)/xattr/%.c
338 $(COMPILE.c) -o $@ $<
339 $(CTFCONVERT_O)

341 $(OBJSDIR)/%.o: $(COMMONBASE)/zfs/%.c
342 $(COMPILE.c) -o $@ $<
343 $(CTFCONVERT_O)

345 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/scsi/adapters/pmcs/%.c
346 $(COMPILE.c) -o $@ $<
347 $(CTFCONVERT_O)

349 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/scsi/adapters/pmcs/%.bin
350 $(COMPILE.b) -o $@ $<
351 $(CTFCONVERT_O)

353 $(OBJSDIR)/%.o: $(COMMONBASE)/fsreparse/%.c
354 $(COMPILE.c) -o $@ $<
355 $(CTFCONVERT_O)

357 KMECHKRB5_BASE=$(UTSBASE)/common/gssapi/mechs/krb5

359 KGSSDFLAGS=-I $(UTSBASE)/common/gssapi/include

361 # Note, KRB5_DEFS can be assigned various preprocessor flags,
362 # typically -D defines on the make invocation. The standard compiler
363 # flags will not be overwritten.
364 KGSSDFLAGS += $(KRB5_DEFS)

366 $(OBJSDIR)/%.o: $(UTSBASE)/common/gssapi/%.c
367 $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
368 $(CTFCONVERT_O)

370 $(OBJSDIR)/%.o: $(UTSBASE)/common/gssapi/mechs/dummy/%.c
371 $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
372 $(CTFCONVERT_O)

374 $(OBJSDIR)/%.o: $(KMECHKRB5_BASE)/%.c
375 $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
376 $(CTFCONVERT_O)

378 $(OBJSDIR)/%.o: $(KMECHKRB5_BASE)/crypto/%.c
379 $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
380 $(CTFCONVERT_O)

382 $(OBJSDIR)/%.o: $(KMECHKRB5_BASE)/crypto/des/%.c
383 $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
384 $(CTFCONVERT_O)

386 $(OBJSDIR)/%.o: $(KMECHKRB5_BASE)/crypto/arcfour/%.c
387 $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
388 $(CTFCONVERT_O)

390 $(OBJSDIR)/%.o: $(KMECHKRB5_BASE)/crypto/dk/%.c
391 $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<

```

```

392      $(CTFCONVERT_O)

394 $(OBJS_DIR)/%.o:                $(KMECHKRB5_BASE)/crypto/enc_provider/%.c
395     $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
396     $(CTFCONVERT_O)

398 $(OBJS_DIR)/%.o:                $(KMECHKRB5_BASE)/crypto/hash_provider/%.c
399     $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
400     $(CTFCONVERT_O)

402 $(OBJS_DIR)/%.o:                $(KMECHKRB5_BASE)/crypto/keyhash_provider/%.c
403     $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
404     $(CTFCONVERT_O)

406 $(OBJS_DIR)/%.o:                $(KMECHKRB5_BASE)/crypto/raw/%.c
407     $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
408     $(CTFCONVERT_O)

410 $(OBJS_DIR)/%.o:                $(KMECHKRB5_BASE)/crypto/old/%.c
411     $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
412     $(CTFCONVERT_O)

414 $(OBJS_DIR)/%.o:                $(KMECHKRB5_BASE)/krb5/krb/%.c
415     $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
416     $(CTFCONVERT_O)

418 $(OBJS_DIR)/%.o:                $(KMECHKRB5_BASE)/krb5/os/%.c
419     $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
420     $(CTFCONVERT_O)

422 $(OBJS_DIR)/ser_sctx.o := CPPFLAGS += -DPROVIDE_KERNEL_IMPORT=1

424 $(OBJS_DIR)/%.o:                $(KMECHKRB5_BASE)/mech/%.c
425     $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
426     $(CTFCONVERT_O)

428 $(OBJS_DIR)/%.o:                $(KMECHKRB5_BASE)/profile/%.c
429     $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
430     $(CTFCONVERT_O)

432 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/avs/ncall/%.c
433     $(COMPILE.c) -o $@ $<
434     $(CTFCONVERT_O)

436 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/avs/ns/dsw/%.c
437     $(COMPILE.c) -o $@ $<
438     $(CTFCONVERT_O)

440 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/avs/ns/nsctl/%.c
441     $(COMPILE.c) -o $@ $<
442     $(CTFCONVERT_O)

444 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/avs/ns/rdc/%.c
445     $(COMPILE.c) -o $@ $<
446     $(CTFCONVERT_O)

448 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/avs/ns/sdbc/%.c
449     $(COMPILE.c) -o $@ $<
450     $(CTFCONVERT_O)

452 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/avs/ns/solaris/%.c
453     $(COMPILE.c) -o $@ $<
454     $(CTFCONVERT_O)

456 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/avs/ns/sv/%.c
457     $(COMPILE.c) -o $@ $<

```

```

458      $(CTFCONVERT_O)

460 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/avs/ns/unistat/%.c
461     $(COMPILE.c) -o $@ $<
462     $(CTFCONVERT_O)

464 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/idmap/%.c
465     $(COMPILE.c) -o $@ $<
466     $(CTFCONVERT_O)

468 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/%.c
469     $(COMPILE.c) -o $@ $<
470     $(CTFCONVERT_O)

472 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/arp/%.c
473     $(COMPILE.c) -o $@ $<
474     $(CTFCONVERT_O)

476 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/ip/%.c
477     $(COMPILE.c) -o $@ $<
478     $(CTFCONVERT_O)

480 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/ipnet/%.c
481     $(COMPILE.c) -o $@ $<
482     $(CTFCONVERT_O)

484 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/iptun/%.c
485     $(COMPILE.c) -o $@ $<
486     $(CTFCONVERT_O)

488 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/kssl/%.c
489     $(COMPILE.c) -o $@ $<
490     $(CTFCONVERT_O)

492 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/sctp/%.c
493     $(COMPILE.c) -o $@ $<
494     $(CTFCONVERT_O)

496 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/tcp/%.c
497     $(COMPILE.c) -o $@ $<
498     $(CTFCONVERT_O)

500 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/ilb/%.c
501     $(COMPILE.c) -o $@ $<
502     $(CTFCONVERT_O)

504 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/ipf/%.c
505     $(COMPILE.c) -o $@ $<
506     $(CTFCONVERT_O)

508 $(OBJS_DIR)/%.o:                $(COMMONBASE)/net/patricia/%.c
509     $(COMPILE.c) -o $@ $<
510     $(CTFCONVERT_O)

512 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/udp/%.c
513     $(COMPILE.c) -o $@ $<
514     $(CTFCONVERT_O)

516 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/nca/%.c
517     $(COMPILE.c) -o $@ $<
518     $(CTFCONVERT_O)

520 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/sockmods/%.c
521     $(COMPILE.c) -o $@ $<
522     $(CTFCONVERT_O)

```

```

524 $(OBJSDIR)/%.o: $(UTSBASE)/common/inet/dlpistub/%.c
525 $(COMPILE.c) -o $@ $<
526 $(CTFCONVERT_O)

528 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/%.c
529 $(COMPILE.c) -o $@ $<
530 $(CTFCONVERT_O)

532 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/l394/%.c
533 $(COMPILE.c) -o $@ $<
534 $(CTFCONVERT_O)

536 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/l394/adapters/%.c
537 $(COMPILE.c) -o $@ $<
538 $(CTFCONVERT_O)

540 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/l394/targets/av1394/%.c
541 $(COMPILE.c) -o $@ $<
542 $(CTFCONVERT_O)

544 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/l394/targets/dcam1394/%.c
545 $(COMPILE.c) -o $@ $<
546 $(CTFCONVERT_O)

548 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/l394/targets/scsal394/%.c
549 $(COMPILE.c) -o $@ $<
550 $(CTFCONVERT_O)

552 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/sbp2/%.c
553 $(COMPILE.c) -o $@ $<
554 $(CTFCONVERT_O)

556 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/aac/%.c
557 $(COMPILE.c) -o $@ $<
558 $(CTFCONVERT_O)

560 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/afe/%.c
561 $(COMPILE.c) -o $@ $<
562 $(CTFCONVERT_O)

564 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/atge/%.c
565 $(COMPILE.c) -o $@ $<
566 $(CTFCONVERT_O)

568 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/arn/%.c
569 $(COMPILE.c) -o $@ $<
570 $(CTFCONVERT_O)

572 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ath/%.c
573 $(COMPILE.c) -o $@ $<
574 $(CTFCONVERT_O)

576 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/atu/%.c
577 $(COMPILE.c) -o $@ $<
578 $(CTFCONVERT_O)

580 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/audio/impl/%.c
581 $(COMPILE.c) -o $@ $<
582 $(CTFCONVERT_O)

584 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/audio/ac97/%.c
585 $(COMPILE.c) -o $@ $<
586 $(CTFCONVERT_O)

588 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/audio/drv/audioens/%.c
589 $(COMPILE.c) -o $@ $<

```

```

590 $(CTFCONVERT_O)

592 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/audio/drv/audioemu10k/%.c
593 $(COMPILE.c) -o $@ $<
594 $(CTFCONVERT_O)

596 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/audio/drv/audio1575/%.c
597 $(COMPILE.c) -o $@ $<
598 $(CTFCONVERT_O)

600 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/audio/drv/audio810/%.c
601 $(COMPILE.c) -o $@ $<
602 $(CTFCONVERT_O)

604 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/audio/drv/audiocmi/%.c
605 $(COMPILE.c) -o $@ $<
606 $(CTFCONVERT_O)

608 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/audio/drv/audiocmihd/%.c
609 $(COMPILE.c) -o $@ $<
610 $(CTFCONVERT_O)

612 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/audio/drv/audiohd/%.c
613 $(COMPILE.c) -o $@ $<
614 $(CTFCONVERT_O)

616 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/audio/drv/audioixp/%.c
617 $(COMPILE.c) -o $@ $<
618 $(CTFCONVERT_O)

620 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/audio/drv/audiols/%.c
621 $(COMPILE.c) -o $@ $<
622 $(CTFCONVERT_O)

624 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/audio/drv/audiopci/%.c
625 $(COMPILE.c) -o $@ $<
626 $(CTFCONVERT_O)

628 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/audio/drv/audiopl6x/%.c
629 $(COMPILE.c) -o $@ $<
630 $(CTFCONVERT_O)

632 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/audio/drv/audiosolo/%.c
633 $(COMPILE.c) -o $@ $<
634 $(CTFCONVERT_O)

636 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/audio/drv/audiots/%.c
637 $(COMPILE.c) -o $@ $<
638 $(CTFCONVERT_O)

640 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/audio/drv/audiovia823x/%.c
641 $(COMPILE.c) -o $@ $<
642 $(CTFCONVERT_O)

644 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/audio/drv/audiovia97/%.c
645 $(COMPILE.c) -o $@ $<
646 $(CTFCONVERT_O)

648 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/bfe/%.c
649 $(COMPILE.c) -o $@ $<
650 $(CTFCONVERT_O)

652 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/bge/%.c
653 $(COMPILE.c) -o $@ $<
654 $(CTFCONVERT_O)

```

```

656 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/blkdev/%.c
657 $(COMPILE.c) -o $@ $<
658 $(CTFCONVERT_O)

660 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/bpf/%.c
661 $(COMPILE.c) -o $@ $<
662 $(CTFCONVERT_O)

664 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/cardbus/%.c
665 $(COMPILE.c) -o $@ $<
666 $(CTFCONVERT_O)

668 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/comstar/stmf/%.c
669 $(COMPILE.c) -o $@ $<
670 $(CTFCONVERT_O)

672 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/comstar/port/fct/%.c
673 $(COMPILE.c) -o $@ $<
674 $(CTFCONVERT_O)

676 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/comstar/port/qlt/%.c
677 $(COMPILE.c) -o $@ $<
678 $(CTFCONVERT_O)

680 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/comstar/port/srpt/%.c
681 $(COMPILE.c) -o $@ $<
682 $(CTFCONVERT_O)

684 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/comstar/port/fcoet/%.c
685 $(COMPILE.c) -o $@ $<
686 $(CTFCONVERT_O)

688 $(OBJSDIR)/%.o: $(COMMONBASE)/iscsit/%.c
689 $(COMPILE.c) -o $@ $<
690 $(CTFCONVERT_O)

692 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/comstar/port/iscsit/%.c
693 $(COMPILE.c) -o $@ $<
694 $(CTFCONVERT_O)

696 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/comstar/port/pppt/%.c
697 $(COMPILE.c) -o $@ $<
698 $(CTFCONVERT_O)

700 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/comstar/lu/stmf_sbd/%.c
701 $(COMPILE.c) -o $@ $<
702 $(CTFCONVERT_O)

704 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/dld/%.c
705 $(COMPILE.c) -o $@ $<
706 $(CTFCONVERT_O)

708 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/dls/%.c
709 $(COMPILE.c) -o $@ $<
710 $(CTFCONVERT_O)

712 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/dmfe/%.c
713 $(COMPILE.c) -o $@ $<
714 $(CTFCONVERT_O)

716 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/drm/%.c
717 $(COMPILE.c) -o $@ $<
718 $(CTFCONVERT_O)

720 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/efe/%.c
721 $(COMPILE.c) -o $@ $<

```

```

722 $(CTFCONVERT_O)

724 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/elxl/%.c
725 $(COMPILE.c) -o $@ $<
726 $(CTFCONVERT_O)

728 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/fcoe/%.c
729 $(COMPILE.c) -o $@ $<
730 $(CTFCONVERT_O)

732 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/hme/%.c
733 $(COMPILE.c) -o $@ $<
734 $(CTFCONVERT_O)

736 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/pciex/%.c
737 $(COMPILE.c) -o $@ $<
738 $(CTFCONVERT_O)

740 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/hotplug/hpcsvc/%.c
741 $(COMPILE.c) -o $@ $<
742 $(CTFCONVERT_O)

744 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/pciex/hotplug/%.c
745 $(COMPILE.c) -o $@ $<
746 $(CTFCONVERT_O)

748 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/hotplug/pcihp/%.c
749 $(COMPILE.c) -o $@ $<
750 $(CTFCONVERT_O)

752 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ib/clients/rds/%.c
753 $(COMPILE.c) -o $@ $<
754 $(CTFCONVERT_O)

756 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ib/clients/rdsv3/%.c
757 $(COMPILE.c) -o $@ $<
758 $(CTFCONVERT_O)

760 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ib/clients/iser/%.c
761 $(COMPILE.c) -o $@ $<
762 $(CTFCONVERT_O)

764 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ib/clients/ibd/%.c
765 $(COMPILE.c) -o $@ $<
766 $(CTFCONVERT_O)

768 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ib/clients/eoib/%.c
769 $(COMPILE.c) -o $@ $<
770 $(CTFCONVERT_O)

772 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ib/clients/of/sol_ofs/%.c
773 $(COMPILE.c) -o $@ $<
774 $(CTFCONVERT_O)

776 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ib/clients/of/sol_ucma/%.c
777 $(COMPILE.c) -o $@ $<
778 $(CTFCONVERT_O)

780 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ib/clients/of/sol_umad/%.c
781 $(COMPILE.c) -o $@ $<
782 $(CTFCONVERT_O)

784 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ib/clients/of/sol_uverbs/%.c
785 $(COMPILE.c) -o $@ $<
786 $(CTFCONVERT_O)

```

```

788 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ib/clients/sdp/%.c
789 $(COMPILE.c) -o $@ $<
790 $(CTFCONVERT_O)

792 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ib/mgt/ibcm/%.c
793 $(COMPILE.c) -o $@ $<
794 $(CTFCONVERT_O)

796 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ib/mgt/ibdm/%.c
797 $(COMPILE.c) -o $@ $<
798 $(CTFCONVERT_O)

800 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ib/mgt/ibdma/%.c
801 $(COMPILE.c) -o $@ $<
802 $(CTFCONVERT_O)

804 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ib/mgt/ibmf/%.c
805 $(COMPILE.c) -o $@ $<
806 $(CTFCONVERT_O)

808 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ib/ibnex/%.c
809 $(COMPILE.c) -o $@ $<
810 $(CTFCONVERT_O)

812 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ib/ibtl/%.c
813 $(COMPILE.c) -o $@ $<
814 $(CTFCONVERT_O)

816 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ib/adapters/tavor/%.c
817 $(COMPILE.c) -o $@ $<
818 $(CTFCONVERT_O)

820 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ib/adapters/hermon/%.c
821 $(COMPILE.c) -o $@ $<
822 $(CTFCONVERT_O)

824 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ib/clients/daplt/%.c
825 $(COMPILE.c) -o $@ $<
826 $(CTFCONVERT_O)

828 $(OBJSDIR)/%.o: $(COMMONBASE)/iscsi/%.c
829 $(COMPILE.c) -o $@ $<
830 $(CTFCONVERT_O)

832 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/idm/%.c
833 $(COMPILE.c) -o $@ $<
834 $(CTFCONVERT_O)

836 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ipw/%.c
837 $(COMPILE.c) -o $@ $<
838 $(CTFCONVERT_O)

840 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/iwh/%.c
841 $(COMPILE.c) -o $@ $<
842 $(CTFCONVERT_O)

844 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/iwi/%.c
845 $(COMPILE.c) -o $@ $<
846 $(CTFCONVERT_O)

848 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/iwk/%.c
849 $(COMPILE.c) -o $@ $<
850 $(CTFCONVERT_O)

852 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/iwp/%.c
853 $(COMPILE.c) -o $@ $<

```

```

854 $(CTFCONVERT_O)

856 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/kb8042/%.c
857 $(COMPILE.c) -o $@ $<
858 $(CTFCONVERT_O)

860 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/kbtrans/%.c
861 $(COMPILE.c) -o $@ $<
862 $(CTFCONVERT_O)

864 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ksocket/%.c
865 $(COMPILE.c) -o $@ $<
866 $(CTFCONVERT_O)

868 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/aggr/%.c
869 $(COMPILE.c) -o $@ $<
870 $(CTFCONVERT_O)

872 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/lp/%.c
873 $(COMPILE.c) -o $@ $<
874 $(CTFCONVERT_O)

876 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/lvm/hotspares/%.c
877 $(COMPILE.c) -o $@ $<
878 $(CTFCONVERT_O)

880 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/lvm/md/%.c
881 $(COMPILE.c) -o $@ $<
882 $(CTFCONVERT_O)

884 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/lvm/mirror/%.c
885 $(COMPILE.c) -o $@ $<
886 $(CTFCONVERT_O)

888 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/lvm/notify/%.c
889 $(COMPILE.c) -o $@ $<
890 $(CTFCONVERT_O)

892 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/lvm/raid/%.c
893 $(COMPILE.c) -o $@ $<
894 $(CTFCONVERT_O)

896 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/lvm/softpart/%.c
897 $(COMPILE.c) -o $@ $<
898 $(CTFCONVERT_O)

900 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/lvm/striped/%.c
901 $(COMPILE.c) -o $@ $<
902 $(CTFCONVERT_O)

904 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/lvm/trans/%.c
905 $(COMPILE.c) -o $@ $<
906 $(CTFCONVERT_O)

908 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/mac/%.c
909 $(COMPILE.c) -o $@ $<
910 $(CTFCONVERT_O)

912 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/mac/plugins/%.c
913 $(COMPILE.c) -o $@ $<
914 $(CTFCONVERT_O)

916 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/mega_sas/%.c
917 $(COMPILE.c) -o $@ $<
918 $(CTFCONVERT_O)

```

```

920 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/mii/%.c
921     $(COMPILE.c) -o $@ $<
922     $(CTFCONVERT_O)

924 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/mr_sas/%.c
925     $(COMPILE.c) -o $@ $<
926     $(CTFCONVERT_O)

928 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/scsi/adapters/mpt_sas/%.c
929     $(COMPILE.c) -o $@ $<
930     $(CTFCONVERT_O)

932 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/mxfe/%.c
933     $(COMPILE.c) -o $@ $<
934     $(CTFCONVERT_O)

936 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/mwl/%.c
937     $(COMPILE.c) -o $@ $<
938     $(CTFCONVERT_O)

940 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/mwl/mwl_fw/%.c
941     $(COMPILE.c) -o $@ $<
942     $(CTFCONVERT_O)

944 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/net80211/%.c
945     $(COMPILE.c) -o $@ $<
946     $(CTFCONVERT_O)

948 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/nge/%.c
949     $(COMPILE.c) -o $@ $<
950     $(CTFCONVERT_O)

952 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/nxge/%.c
953     $(COMPILE.c) -o $@ $<
954     $(CTFCONVERT_O)

956 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/nxge/npi/%.c
957     $(COMPILE.c) -o $@ $<
958     $(CTFCONVERT_O)

960 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/nxge/%.s
961     $(COMPILE.s) -o $@ $<

963 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/pci-ide/%.c
964     $(COMPILE.c) -o $@ $<
965     $(CTFCONVERT_O)

967 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/pcmcia/%.c
968     $(COMPILE.c) -o $@ $<
969     $(CTFCONVERT_O)

971 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/pcan/%.c
972     $(COMPILE.c) -o $@ $<
973     $(CTFCONVERT_O)

975 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/pcn/%.c
976     $(COMPILE.c) -o $@ $<
977     $(CTFCONVERT_O)

979 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/pcwl/%.c
980     $(COMPILE.c) -o $@ $<
981     $(CTFCONVERT_O)

983 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ppp/sppp/%.c
984     $(COMPILE.c) -o $@ $<
985     $(CTFCONVERT_O)

```

```

987 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ppp/sppasyn/%.c
988     $(COMPILE.c) -o $@ $<
989     $(CTFCONVERT_O)

991 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ppp/sppptun/%.c
992     $(COMPILE.c) -o $@ $<
993     $(CTFCONVERT_O)

995 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ral/%.c
996     $(COMPILE.c) -o $@ $<
997     $(CTFCONVERT_O)

999 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/rge/%.c
1000    $(COMPILE.c) -o $@ $<
1001    $(CTFCONVERT_O)

1003 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/rtls/%.c
1004    $(COMPILE.c) -o $@ $<
1005    $(CTFCONVERT_O)

1007 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/rsm/%.c
1008    $(COMPILE.c) -o $@ $<
1009    $(CTFCONVERT_O)

1011 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/rtw/%.c
1012    $(COMPILE.c) -o $@ $<
1013    $(CTFCONVERT_O)

1015 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/rum/%.c
1016    $(COMPILE.c) -o $@ $<
1017    $(CTFCONVERT_O)

1019 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/rwd/%.c
1020    $(COMPILE.c) -o $@ $<
1021    $(CTFCONVERT_O)

1023 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/rwn/%.c
1024    $(COMPILE.c) -o $@ $<
1025    $(CTFCONVERT_O)

1027 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/sata/adapters/ahci/%.c
1028    $(COMPILE.c) -o $@ $<
1029    $(CTFCONVERT_O)

1031 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/sata/adapters/nv_sata/%.c
1032    $(COMPILE.c) -o $@ $<
1033    $(CTFCONVERT_O)

1035 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/sata/adapters/si3124/%.c
1036    $(COMPILE.c) -o $@ $<
1037    $(CTFCONVERT_O)

1039 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/sata/impl/%.c
1040    $(COMPILE.c) -o $@ $<
1041    $(CTFCONVERT_O)

1043 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/scsi/conf/%.c
1044    $(COMPILE.c) -o $@ $<
1045    $(CTFCONVERT_O)

1047 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/scsi/impl/%.c
1048    $(COMPILE.c) -o $@ $<
1049    $(CTFCONVERT_O)

1051 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/scsi/targets/%.c

```

```

1052 $(COMPILE.c) -o $$ $<
1053 $(CTFCONVERT_O)

1055 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/scsi/adapters/%.c
1056 $(COMPILE.c) -o $$ $<
1057 $(CTFCONVERT_O)

1059 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/scsi/adapters/blk2scsa/%.c
1060 $(COMPILE.c) -o $$ $<
1061 $(CTFCONVERT_O)

1063 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/scsi/adapters/scsi_vhci/%.c
1064 $(COMPILE.c) -o $$ $<
1065 $(CTFCONVERT_O)

1067 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/scsi/adapters/scsi_vhci/fop
1068 $(COMPILE.c) -o $$ $<
1069 $(CTFCONVERT_O)

1071 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/fibre-channel/ulp/%.c
1072 $(COMPILE.c) -o $$ $<
1073 $(CTFCONVERT_O)

1075 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/fibre-channel/impl/%.c
1076 $(COMPILE.c) -o $$ $<
1077 $(CTFCONVERT_O)

1079 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/fibre-channel/fca/qlc/%.c
1080 $(COMPILE.c) -o $$ $<
1081 $(CTFCONVERT_O)

1083 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/fibre-channel/fca/qlge/%.c
1084 $(COMPILE.c) -o $$ $<
1085 $(CTFCONVERT_O)

1087 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/fibre-channel/fca/emlxs/%.c
1088 $(COMPILE.c) -o $$ $<
1089 $(CTFCONVERT_O)

1091 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/fibre-channel/fca/oce/%.c
1092 $(COMPILE.c) -o $$ $<
1093 $(CTFCONVERT_O)

1095 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/fibre-channel/fca/fcoei/%.c
1096 $(COMPILE.c) -o $$ $<
1097 $(CTFCONVERT_O)

1099 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/sdcard/adapters/sdhost/%.c
1100 $(COMPILE.c) -o $$ $<
1101 $(CTFCONVERT_O)

1103 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/sdcard/impl/%.c
1104 $(COMPILE.c) -o $$ $<
1105 $(CTFCONVERT_O)

1107 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/sdcard/targets/sdcard/%.c
1108 $(COMPILE.c) -o $$ $<
1109 $(CTFCONVERT_O)

1111 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/sfe/%.c
1112 $(COMPILE.c) -o $$ $<
1113 $(CTFCONVERT_O)

1115 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/simnet/%.c
1116 $(COMPILE.c) -o $$ $<
1117 $(CTFCONVERT_O)

```

```

1119 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/softmac/%.c
1120 $(COMPILE.c) -o $$ $<
1121 $(CTFCONVERT_O)

1123 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/uath/%.c
1124 $(COMPILE.c) -o $$ $<
1125 $(CTFCONVERT_O)

1127 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/uath/uath_fw/%.c
1128 $(COMPILE.c) -o $$ $<
1129 $(CTFCONVERT_O)

1131 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ural/%.c
1132 $(COMPILE.c) -o $$ $<
1133 $(CTFCONVERT_O)

1135 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/urtw/%.c
1136 $(COMPILE.c) -o $$ $<
1137 $(CTFCONVERT_O)

1139 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/audio/usb_ac/%.c
1140 $(COMPILE.c) -o $$ $<
1141 $(CTFCONVERT_O)

1143 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/audio/usb_as/%.c
1144 $(COMPILE.c) -o $$ $<
1145 $(CTFCONVERT_O)

1147 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/audio/usb_ah/%.c
1148 $(COMPILE.c) -o $$ $<
1149 $(CTFCONVERT_O)

1151 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/usbskel/%.c
1152 $(COMPILE.c) -o $$ $<
1153 $(CTFCONVERT_O)

1155 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/video/usbvc/%.c
1156 $(COMPILE.c) -o $$ $<
1157 $(CTFCONVERT_O)

1159 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/hwarc/%.c
1160 $(COMPILE.c) -o $$ $<
1161 $(CTFCONVERT_O)

1163 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/hid/%.c
1164 $(COMPILE.c) -o $$ $<
1165 $(CTFCONVERT_O)

1167 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/hidparser/%.c
1168 $(COMPILE.c) -o $$ $<
1169 $(CTFCONVERT_O)

1171 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/printer/%.c
1172 $(COMPILE.c) -o $$ $<
1173 $(CTFCONVERT_O)

1175 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/usbbkm/%.c
1176 $(COMPILE.c) -o $$ $<
1177 $(CTFCONVERT_O)

1179 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/uslbs/%.c
1180 $(COMPILE.c) -o $$ $<
1181 $(CTFCONVERT_O)

1183 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/usbinp/usbwcm

```



```

1184 $(COMPILE.c) -o $$ $<
1185 $(CTFCONVERT_O)

1187 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/ugen/%.c
1188 $(COMPILE.c) -o $$ $<
1189 $(CTFCONVERT_O)

1191 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/usbser/%.c
1192 $(COMPILE.c) -o $$ $<
1193 $(CTFCONVERT_O)

1195 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/usbser/usbsacm/
1196 $(COMPILE.c) -o $$ $<
1197 $(CTFCONVERT_O)

1199 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/usbser/usbftdi/
1200 $(COMPILE.c) -o $$ $<
1201 $(CTFCONVERT_O)

1203 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/usbser/usbser_k
1204 $(COMPILE.c) -o $$ $<
1205 $(CTFCONVERT_O)

1207 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/usbser/usbsprl/
1208 $(COMPILE.c) -o $$ $<
1209 $(CTFCONVERT_O)

1211 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/wusb_df/%.c
1212 $(COMPILE.c) -o $$ $<
1213 $(CTFCONVERT_O)

1215 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/hwal480_fw/%.c
1216 $(COMPILE.c) -o $$ $<
1217 $(CTFCONVERT_O)

1219 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/wusb_ca/%.c
1220 $(COMPILE.c) -o $$ $<
1221 $(CTFCONVERT_O)

1223 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/usbecm/%.c
1224 $(COMPILE.c) -o $$ $<
1225 $(CTFCONVERT_O)

1227 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/hcd/openhci/%.c
1228 $(COMPILE.c) -o $$ $<
1229 $(CTFCONVERT_O)

1231 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/hcd/ehci/%.c
1232 $(COMPILE.c) -o $$ $<
1233 $(CTFCONVERT_O)

1235 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/hcd/uhci/%.c
1236 $(COMPILE.c) -I../common -o $$ $<
1237 $(CTFCONVERT_O)

1239 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/hubd/%.c
1240 $(COMPILE.c) -o $$ $<
1241 $(CTFCONVERT_O)

1243 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/scsa2usb/%.c
1244 $(COMPILE.c) -o $$ $<
1245 $(CTFCONVERT_O)

1247 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/usb_mid/%.c
1248 $(COMPILE.c) -o $$ $<
1249 $(CTFCONVERT_O)

```

```

1251 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/usb_ia/%.c
1252 $(COMPILE.c) -o $$ $<
1253 $(CTFCONVERT_O)

1255 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/usba/%.c
1256 $(COMPILE.c) -o $$ $<
1257 $(CTFCONVERT_O)

1259 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/usba10/%.c
1260 $(COMPILE.c) -o $$ $<
1261 $(CTFCONVERT_O)

1263 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/hwa/hwahc/%.c
1264 $(COMPILE.c) -o $$ $<
1265 $(CTFCONVERT_O)

1267 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/uwb/uwba/%.c
1268 $(COMPILE.c) -o $$ $<
1269 $(CTFCONVERT_O)

1271 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/vuidmice/%.c
1272 $(COMPILE.c) -o $$ $<
1273 $(CTFCONVERT_O)

1275 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/vnic/%.c
1276 $(COMPILE.c) -o $$ $<
1277 $(CTFCONVERT_O)

1279 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/wpi/%.c
1280 $(COMPILE.c) -o $$ $<
1281 $(CTFCONVERT_O)

1283 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/zyd/%.c
1284 $(COMPILE.c) -o $$ $<
1285 $(CTFCONVERT_O)

1287 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/chxge/com/%.c
1288 $(COMPILE.c) -o $$ $<
1289 $(CTFCONVERT_O)

1291 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/chxge/%.c
1292 $(COMPILE.c) -o $$ $<
1293 $(CTFCONVERT_O)

1295 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/cxgbe/common/%.c
1296 $(COMPILE.c) -o $$ $<
1297 $(CTFCONVERT_O)

1299 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/cxgbe/shared/%.c
1300 $(COMPILE.c) -o $$ $<
1301 $(CTFCONVERT_O)

1303 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/cxgbe/firmware/%.c
1304 $(COMPILE.c) -o $$ $<
1305 $(CTFCONVERT_O)

1307 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/cxgbe/t4nex/%.c
1308 $(COMPILE.c) -o $$ $<
1309 $(CTFCONVERT_O)

1311 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/cxgbe/cxgbe/%.c
1312 $(COMPILE.c) -o $$ $<
1313 $(CTFCONVERT_O)

1315 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ixgb/%.c

```

```

1316 $(COMPILE.c) -o $@ $<
1317 $(CTFCONVERT_O)

1319 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/xge/drv/%.c
1320 $(COMPILE.c) -o $@ $<
1321 $(CTFCONVERT_O)

1323 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/xge/hal/xgehal/%.c
1324 $(COMPILE.c) -o $@ $<
1325 $(CTFCONVERT_O)

1327 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/e1000g/%.c
1328 $(COMPILE.c) -o $@ $<
1329 $(CTFCONVERT_O)

1331 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/igb/%.c
1332 $(COMPILE.c) -o $@ $<
1333 $(CTFCONVERT_O)

1335 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/iprb/%.c
1336 $(COMPILE.c) -o $@ $<
1337 $(CTFCONVERT_O)

1339 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ixgbe/%.c
1340 $(COMPILE.c) -o $@ $<
1341 $(CTFCONVERT_O)

1343 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ntxn/%.c
1344 $(COMPILE.c) -o $@ $<
1345 $(CTFCONVERT_O)

1347 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/myril0ge/drv/%.c
1348 $(COMPILE.c) -o $@ $<
1349 $(CTFCONVERT_O)

1351 $(OBJSDIR)/%.o: $(UTSBASE)/common/ipp/%.c
1352 $(COMPILE.c) -o $@ $<
1353 $(CTFCONVERT_O)

1355 $(OBJSDIR)/%.o: $(UTSBASE)/common/ipp/ipgpc/%.c
1356 $(COMPILE.c) -o $@ $<
1357 $(CTFCONVERT_O)

1359 $(OBJSDIR)/%.o: $(UTSBASE)/common/ipp/dlcosmk/%.c
1360 $(COMPILE.c) -o $@ $<
1361 $(CTFCONVERT_O)

1363 $(OBJSDIR)/%.o: $(UTSBASE)/common/ipp/flowacct/%.c
1364 $(COMPILE.c) -o $@ $<
1365 $(CTFCONVERT_O)

1367 $(OBJSDIR)/%.o: $(UTSBASE)/common/ipp/dscpmk/%.c
1368 $(COMPILE.c) -o $@ $<
1369 $(CTFCONVERT_O)

1371 $(OBJSDIR)/%.o: $(UTSBASE)/common/ipp/meters/%.c
1372 $(COMPILE.c) -o $@ $<
1373 $(CTFCONVERT_O)

1375 $(OBJSDIR)/%.o: $(UTSBASE)/common/kiconv/kiconv_emea/%.c
1376 $(COMPILE.c) -o $@ $<
1377 $(CTFCONVERT_O)

1379 $(OBJSDIR)/%.o: $(UTSBASE)/common/kiconv/kiconv_ja/%.c
1380 $(COMPILE.c) -o $@ $<
1381 $(CTFCONVERT_O)

```

```

1383 $(OBJSDIR)/%.o: $(UTSBASE)/common/kiconv/kiconv_ko/%.c
1384 $(COMPILE.c) -o $@ $<
1385 $(CTFCONVERT_O)

1387 $(OBJSDIR)/%.o: $(UTSBASE)/common/kiconv/kiconv_sc/%.c
1388 $(COMPILE.c) -o $@ $<
1389 $(CTFCONVERT_O)

1391 $(OBJSDIR)/%.o: $(UTSBASE)/common/kiconv/kiconv_tc/%.c
1392 $(COMPILE.c) -o $@ $<
1393 $(CTFCONVERT_O)

1395 $(OBJSDIR)/%.o: $(UTSBASE)/common/klm/%.c
1396 $(COMPILE.c) -o $@ $<
1397 $(CTFCONVERT_O)

1399 $(OBJSDIR)/%.o: $(UTSBASE)/common/kmdb/%.c
1400 $(COMPILE.c) -o $@ $<
1401 $(CTFCONVERT_O)

1403 $(OBJSDIR)/%.o: $(UTSBASE)/common/ktli/%.c
1404 $(COMPILE.c) -o $@ $<
1405 $(CTFCONVERT_O)

1407 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/scsi/adapters/iscsi/%.c
1408 $(COMPILE.c) -o $@ $<
1409 $(CTFCONVERT_O)

1411 $(OBJSDIR)/%.o: $(COMMONBASE)/iscsi/%.c
1412 $(COMPILE.c) -o $@ $<
1413 $(CTFCONVERT_O)

1415 $(OBJSDIR)/%.o: $(UTSBASE)/common/inet/kifconf/%.c
1416 $(COMPILE.c) -o $@ $<
1417 $(CTFCONVERT_O)

1419 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/vr/%.c
1420 $(COMPILE.c) -o $@ $<
1421 $(CTFCONVERT_O)

1423 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/yge/%.c
1424 $(COMPILE.c) -o $@ $<
1425 $(CTFCONVERT_O)

1427 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/virtio/%.c
1428 $(COMPILE.c) -o $@ $<
1429 $(CTFCONVERT_O)

1431 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/vioblk/%.c
1432 $(COMPILE.c) -o $@ $<
1433 $(CTFCONVERT_O)

1435 #
1436 # krtld must refer to its own bzero/bcopy until the kernel is fully linked
1437 #
1438 $(OBJSDIR)/bootrd.o := CPPFLAGS += -DKOBJ_OVERRIDES
1439 $(OBJSDIR)/doreloc.o := CPPFLAGS += -DKOBJ_OVERRIDES
1440 $(OBJSDIR)/kobj.o := CPPFLAGS += -DKOBJ_OVERRIDES
1441 $(OBJSDIR)/kobj_boot.o := CPPFLAGS += -DKOBJ_OVERRIDES
1442 $(OBJSDIR)/kobj_bootflags.o := CPPFLAGS += -DKOBJ_OVERRIDES
1443 $(OBJSDIR)/kobj_convrelstr.o := CPPFLAGS += -DKOBJ_OVERRIDES
1444 $(OBJSDIR)/kobj_isa.o := CPPFLAGS += -DKOBJ_OVERRIDES
1445 $(OBJSDIR)/kobj_kdi.o := CPPFLAGS += -DKOBJ_OVERRIDES
1446 $(OBJSDIR)/kobj_lm.o := CPPFLAGS += -DKOBJ_OVERRIDES
1447 $(OBJSDIR)/kobj_reloc.o := CPPFLAGS += -DKOBJ_OVERRIDES

```

```

1448 $(OBJSDIR)/kobj_stubs.o      := CPPFLAGS += -DKOBJ_OVERRIDES
1449 $(OBJSDIR)/kobj_subr.o       := CPPFLAGS += -DKOBJ_OVERRIDES

1451 $(OBJSDIR)/%.o:              $(UTSBASE)/common/krtld/%.c
1452     $(COMPILE.c) -o $@ $<
1453     $(CTFCONVERT_O)

1455 $(OBJSDIR)/%.o:              $(COMMONBASE)/list/%.c
1456     $(COMPILE.c) -o $@ $<
1457     $(CTFCONVERT_O)

1459 $(OBJSDIR)/%.o:              $(COMMONBASE)/lvm/%.c
1460     $(COMPILE.c) -o $@ $<
1461     $(CTFCONVERT_O)

1463 $(OBJSDIR)/%.o:              $(COMMONBASE)/lzma/%.c
1464     $(COMPILE.c) -o $@ $<
1465     $(CTFCONVERT_O)

1467 $(OBJSDIR)/%.o:              $(COMMONBASE)/crypto/md4/%.c
1468     $(COMPILE.c) -o $@ $<
1469     $(CTFCONVERT_O)

1471 $(OBJSDIR)/%.o:              $(COMMONBASE)/crypto/md5/%.c
1472     $(COMPILE.c) -o $@ $<
1473     $(CTFCONVERT_O)

1475 $(OBJSDIR)/%.o:              $(COMMONBASE)/net/dhbj/%.c
1476     $(COMPILE.c) -o $@ $<
1477     $(CTFCONVERT_O)

1479 $(OBJSDIR)/%.o:              $(COMMONBASE)/nvpair/%.c
1480     $(COMPILE.c) -o $@ $<
1481     $(CTFCONVERT_O)

1483 $(OBJSDIR)/%.o:              $(UTSBASE)/common/os/%.c
1484     $(COMPILE.c) -o $@ $<
1485     $(CTFCONVERT_O)

1487 $(OBJSDIR)/%.o:              $(UTSBASE)/common/pcmcia/cis/%.c
1488     $(COMPILE.c) -o $@ $<
1489     $(CTFCONVERT_O)

1491 $(OBJSDIR)/%.o:              $(UTSBASE)/common/pcmcia/cs/%.c
1492     $(COMPILE.c) -o $@ $<
1493     $(CTFCONVERT_O)

1495 $(OBJSDIR)/%.o:              $(UTSBASE)/common/pcmcia/nexus/%.c
1496     $(COMPILE.c) -o $@ $<
1497     $(CTFCONVERT_O)

1499 $(OBJSDIR)/%.o:              $(UTSBASE)/common/pcmcia/pcs/%.c
1500     $(COMPILE.c) -o $@ $<
1501     $(CTFCONVERT_O)

1503 $(OBJSDIR)/%.o:              $(UTSBASE)/common/rpc/%.c
1504     $(COMPILE.c) -o $@ $<
1505     $(CTFCONVERT_O)

1507 $(OBJSDIR)/%.o:              $(UTSBASE)/common/rpc/sec/%.c
1508     $(COMPILE.c) -o $@ $<
1509     $(CTFCONVERT_O)

1511 $(OBJSDIR)/%.o:              $(UTSBASE)/common/rpc/sec_gss/%.c
1512     $(COMPILE.c) -o $@ $<
1513     $(CTFCONVERT_O)

```

```

1515 $(OBJSDIR)/%.o:              $(COMMONBASE)/crypto/shal/%.c
1516     $(COMPILE.c) -o $@ $<
1517     $(CTFCONVERT_O)

1519 $(OBJSDIR)/%.o:              $(COMMONBASE)/crypto/sha2/%.c
1520     $(COMPILE.c) -o $@ $<
1521     $(CTFCONVERT_O)

1523 $(OBJSDIR)/%.o:              $(UTSBASE)/common/syscall/%.c
1524     $(COMPILE.c) -o $@ $<
1525     $(CTFCONVERT_O)

1527 $(OBJSDIR)/%.o:              $(UTSBASE)/common/tnf/%.c
1528     $(COMPILE.c) -o $@ $<
1529     $(CTFCONVERT_O)

1531 $(OBJSDIR)/%.o:              $(COMMONBASE)/tsol/%.c
1532     $(COMPILE.c) -o $@ $<
1533     $(CTFCONVERT_O)

1535 $(OBJSDIR)/%.o:              $(COMMONBASE)/util/%.c
1536     $(COMPILE.c) -o $@ $<
1537     $(CTFCONVERT_O)

1539 $(OBJSDIR)/%.o:              $(COMMONBASE)/unicode/%.c
1540     $(COMPILE.c) -o $@ $<
1541     $(CTFCONVERT_O)

1543 $(OBJSDIR)/%.o:              $(UTSBASE)/common/vm/%.c
1544     $(COMPILE.c) -o $@ $<
1545     $(CTFCONVERT_O)

1547 $(OBJSDIR)/%.o:              $(UTSBASE)/common/zmod/%.c
1548     $(COMPILE.c) -o $@ $<
1549     $(CTFCONVERT_O)

1551 $(OBJSDIR)/zlib_obj.o:        $(ZLIB_OBJS:%=$(OBJSDIR)/%)
1552     $(LD) -r -Breduce -M$(UTSBASE)/common/zmod/mapfile -o $@ \
1553         $(ZLIB_OBJS:%=$(OBJSDIR)/%)
1554     $(CTFMERGE) -t -f -L VERSION -o $@ $(ZLIB_OBJS:%=$(OBJSDIR)/%)

1556 $(OBJSDIR)/%.o:              $(UTSBASE)/common/io/hxge/%.c
1557     $(COMPILE.c) -o $@ $<
1558     $(CTFCONVERT_O)

1560 $(OBJSDIR)/%.o:              $(UTSBASE)/common/io/tpm/%.c
1561     $(COMPILE.c) -o $@ $<
1562     $(CTFCONVERT_O)

1564 $(OBJSDIR)/%.o:              $(UTSBASE)/common/io/tpm/%.s
1565     $(COMPILE.s) -o $@ $<

1567 $(OBJSDIR)/bz2%.o:           $(COMMONBASE)/bzip2/%.c
1568     $(COMPILE.c) -o $@ -I$(COMMONBASE)/bzip2 $<
1569     $(CTFCONVERT_O)

1571 BZ2LINT = -erroff=%all -I$(UTSBASE)/common/bzip2

1573 $(LINTSDIR)/bz2%.ln:          $(COMMONBASE)/bzip2/%.c
1574     @$(LHEAD) $(LINT.c) -C $(LINTSDIR)/`basename $@ .ln` $(BZ2LINT) $< $(

1576 #
1577 # SVM
1578 #

```



```

1712      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1714 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/disp/%.c
1715      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1717 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/dtrace/%.c
1718      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1720 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/exacct/%.c
1721      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1723 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/exec/aout/%.c
1724      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1726 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/exec/elf/%.c
1727      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1729 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/exec/intp/%.c
1730      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1732 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/exec/shbin/%.c
1733      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1735 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/exec/java/%.c
1736      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1738 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/%.c
1739      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1741 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/autofs/%.c
1742      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1744 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/cacheufs/%.c
1745      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1747 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/ctfs/%.c
1748      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1750 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/doorfs/%.c
1751      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1753 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/dcfis/%.c
1754      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1756 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/devfs/%.c
1757      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1759 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/dev/%.c
1760      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1762 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/fd/%.c
1763      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1765 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/fifofs/%.c
1766      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1768 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/hsfs/%.c
1769      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1771 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/lofs/%.c
1772      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1774 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/mntfs/%.c
1775      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1777 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/namefs/%.c

```

```

1778      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1780 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/smbsrv/%.c
1781      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1783 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/smbsrv/%.c
1784      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1786 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/nfs/%.c
1787      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1789 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/objfs/%.c
1790      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1792 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/pcfs/%.c
1793      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1795 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/portfs/%.c
1796      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1798 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/proc/%.c
1799      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1801 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/sharefs/%.c
1802      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1804 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/smbclnt/%.c
1805      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1807 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/smbclnt/net smb/%.c
1808      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1810 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/smbclnt/smbfs/%.c
1811      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1813 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/sockfs/%.c
1814      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1816 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/specfs/%.c
1817      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1819 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/swapfs/%.c
1820      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1822 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/tmpfs/%.c
1823      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1825 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/udfs/%.c
1826      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1828 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/ufs/%.c
1829      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1831 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/ufs_log/%.c
1832      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1834 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/vscan/%.c
1835      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1837 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/zfs/%.c
1838      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1840 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/zut/%.c
1841      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1843 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/xattr/%.c

```

```

1844      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1846 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/zfs/%.c
1847      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1849 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/gssapi/%.c
1850      @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))
1852 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/gssapi/mechs/dummy/%.c
1853      @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))
1855 $(LINTS_DIR)/%.ln:      $(KMECHKRB5_BASE)/%.c
1856      @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))
1858 $(LINTS_DIR)/%.ln:      $(KMECHKRB5_BASE)/crypto/%.c
1859      @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))
1861 $(LINTS_DIR)/%.ln:      $(KMECHKRB5_BASE)/crypto/des/%.c
1862      @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))
1864 $(LINTS_DIR)/%.ln:      $(KMECHKRB5_BASE)/crypto/dk/%.c
1865      @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))
1867 $(LINTS_DIR)/%.ln:      $(KMECHKRB5_BASE)/crypto/os/%.c
1868      @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))
1870 $(LINTS_DIR)/%.ln:      $(KMECHKRB5_BASE)/crypto/arcfour/%.c
1871      @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))
1873 $(LINTS_DIR)/%.ln:      $(KMECHKRB5_BASE)/crypto/enc_provider/%.c
1874      @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))
1876 $(LINTS_DIR)/%.ln:      $(KMECHKRB5_BASE)/crypto/hash_provider/%.c
1877      @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))
1879 $(LINTS_DIR)/%.ln:      $(KMECHKRB5_BASE)/crypto/keyhash_provider/%.c
1880      @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))
1882 $(LINTS_DIR)/%.ln:      $(KMECHKRB5_BASE)/crypto/raw/%.c
1883      @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))
1885 $(LINTS_DIR)/%.ln:      $(KMECHKRB5_BASE)/crypto/old/%.c
1886      @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))
1888 $(LINTS_DIR)/%.ln:      $(KMECHKRB5_BASE)/krb5/krb/%.c
1889      @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))
1891 $(LINTS_DIR)/%.ln:      $(KMECHKRB5_BASE)/krb5/os/%.c
1892      @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))
1894 $(LINTS_DIR)/%.ln:      $(KMECHKRB5_BASE)/mech/%.c
1895      @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))
1897 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/idmap/%.c
1898      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1900 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/%.c
1901      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1903 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/sockmods/%.c
1904      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1906 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/arp/%.c
1907      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1909 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/ip/%.c

```

```

1910      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1912 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/ipnet/%.c
1913      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1915 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/iptun/%.c
1916      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1918 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/ipf/%.c
1919      @$(LHEAD) $(LINT.c) $(IPFFLAGS) $< $(LTAIL))
1921 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/kssl/%.c
1922      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1924 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/net/patricia/%.c
1925      @$(LHEAD) $(LINT.c) $(IPFFLAGS) $< $(LTAIL))
1927 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/udp/%.c
1928      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1930 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/sctp/%.c
1931      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1933 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/tcp/%.c
1934      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1936 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/ilb/%.c
1937      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1939 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/nca/%.c
1940      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1942 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/dlpistub/%.c
1943      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1945 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/%.c
1946      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1948 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/1394/%.c
1949      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1951 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/1394/adapters/%.c
1952      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1954 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/1394/targets/av1394/%.c
1955      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1957 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/1394/targets/dcam1394/%.c
1958      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1960 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/1394/targets/scsal394/%.c
1961      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1963 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/sbp2/%.c
1964      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1966 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/aac/%.c
1967      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1969 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/afe/%.c
1970      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1972 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/atge/%.c
1973      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1975 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/arn/%.c

```

```

1976      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1978 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ath/%.c
1979      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1981 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/atu/%.c
1982      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1984 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/impl/%.c
1985      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1987 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/ac97/%.c
1988      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1990 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/audio1575/%.c
1991      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1993 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/audio810/%.c
1994      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1996 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/audiocmi/%.c
1997      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1999 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/audiocmihd/%.c
2000      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2002 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/audioens/%.c
2003      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2005 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/audioemu10k/%.c
2006      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2008 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/audiohd/%.c
2009      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2011 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/audioixp/%.c
2012      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2014 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/audiols/%.c
2015      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2017 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/audiopci/%.c
2018      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2020 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/audiopl6x/%.c
2021      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2023 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/audiosolo/%.c
2024      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2026 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/audiots/%.c
2027      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2029 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/audiovia823x/%.c
2030      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2032 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/audiovia97/%.c
2033      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2035 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/bfe/%.c
2036      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2038 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/bpf/%.c
2039      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2041 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/bge/%.c

```

```

2042      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2044 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/blkdev/%.c
2045      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2047 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/cardbus/%.c
2048      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2050 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/comstar/lu/stmf_sbd/%.c
2051      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2053 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/comstar/port/fct/%.c
2054      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2056 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/comstar/port/qlt/%.c
2057      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2059 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/comstar/port/srpt/%.c
2060      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2062 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/iscsit/%.c
2063      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2065 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/comstar/port/fcoet/%.c
2066      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2068 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/comstar/port/iscsit/%.c
2069      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2071 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/comstar/port/pppt/%.c
2072      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2074 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/comstar/stmf/%.c
2075      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2077 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/dld/%.c
2078      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2080 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/dls/%.c
2081      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2083 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/dmfe/%.c
2084      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2086 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/drm/%.c
2087      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2089 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/efe/%.c
2090      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2092 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/elxl/%.c
2093      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2095 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/fcoe/%.c
2096      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2098 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/hme/%.c
2099      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2101 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/pciex/%.c
2102      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2104 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/hotplug/hpcsvc/%.c
2105      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2107 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/pciex/hotplug/%.c

```

```

2108      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2110 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/hotplug/pcihp/%.c
2111      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2113 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/clients/rds/%.c
2114      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2116 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/clients/rds3/%.c
2117      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2119 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/clients/iser/%.c
2120      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2122 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/clients/ibd/%.c
2123      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2125 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/clients/eoib/%.c
2126      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2128 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/clients/of/sol_ofs/%.c
2129      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2131 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/clients/of/sol_ucma/%.c
2132      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2134 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/clients/of/sol_umad/%.c
2135      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2137 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/clients/of/sol_uverbs/%.
2138      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2140 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/clients/sdp/%.c
2141      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2143 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/mgt/ibcm/%.c
2144      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2146 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/mgt/ibdm/%.c
2147      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2149 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/mgt/ibdma/%.c
2150      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2152 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/mgt/ibmf/%.c
2153      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2155 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/ibnex/%.c
2156      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2158 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/ibtl/%.c
2159      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2161 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/adapters/tavor/%.c
2162      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2164 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/adapters/hermon/%.c
2165      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2167 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/clients/daplt/%.c
2168      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2170 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/iscsi/%.c
2171      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2173 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/idm/%.c

```

```

2174      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2176 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ipw/%.c
2177      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2179 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/iwh/%.c
2180      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2182 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/iwi/%.c
2183      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2185 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/iwk/%.c
2186      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2188 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/iwp/%.c
2189      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2191 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/kb8042/%.c
2192      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2194 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/kbtrans/%.c
2195      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2197 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ksocket/%.c
2198      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2200 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/aggr/%.c
2201      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2203 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/lp/%.c
2204      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2206 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/lvm/hotspares/%.c
2207      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2209 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/lvm/md/%.c
2210      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2212 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/lvm/mirror/%.c
2213      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2215 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/lvm/raid/%.c
2216      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2218 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/lvm/softpart/%.c
2219      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2221 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/lvm/stripe/%.c
2222      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2224 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/lvm/notify/%.c
2225      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2227 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/lvm/trans/%.c
2228      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2230 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/mac/%.c
2231      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2233 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/mac/plugins/%.c
2234      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2236 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/mega_sas/%.c
2237      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2239 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/mii/%.c

```



```

2240      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2242 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/mr_sas/%.c
2243      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2245 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/scsi/adapters/mpt_sas/%.c
2246      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2248 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/mxfe/%.c
2249      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2251 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/mwl/%.c
2252      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2254 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/mwl/mwl_fw/%.c
2255      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2257 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/net80211/%.c
2258      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2260 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/nge/%.c
2261      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2263 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/nxge/%.c
2264      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2266 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/nxge/%.s
2267      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2269 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/nxge/npi/%.c
2270      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2272 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/pci-ide/%.c
2273      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2275 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/pcmcia/%.c
2276      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2278 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/pcan/%.c
2279      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2281 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/pcn/%.c
2282      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2284 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/pcwl/%.c
2285      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2287 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ppp/sppp/%.c
2288      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2290 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ppp/spppasyn/%.c
2291      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2293 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ppp/spptun/%.c
2294      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2296 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ral/%.c
2297      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2299 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/rge/%.c
2300      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2302 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/rtls/%.c
2303      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2305 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/rsm/%.c

```

```

2306      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2308 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/rtw/%.c
2309      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2311 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/rum/%.c
2312      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2314 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/rwd/%.c
2315      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2317 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/rwn/%.c
2318      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2320 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/sata/adapters/ahci/%.c
2321      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2323 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/sata/adapters/nv_sata/%.c
2324      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2326 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/sata/adapters/si3124/%.c
2327      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2329 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/sata/impl/%.c
2330      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2332 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/scsi/adapters/%.c
2333      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2335 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/scsi/adapters/blk2scsa/%.c
2336      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2338 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/scsi/adapters/pmcs/%.c
2339      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2341 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/scsi/adapters/scsi_vhci/%.c
2342      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2344 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/scsi/adapters/scsi_vhci/fop
2345      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2347 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/fibre-channel/ulp/%.c
2348      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2350 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/fibre-channel/impl/%.c
2351      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2353 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/fibre-channel/fca/qlc/%.c
2354      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2356 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/fibre-channel/fca/qlge/%.c
2357      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2359 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/fibre-channel/fca/emlxs/%.c
2360      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2362 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/fibre-channel/fca/oce/%.c
2363      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2365 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/fibre-channel/fca/fcoei/%.c
2366      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2368 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/scsi/conf/%.c
2369      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2371 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/scsi/impl/%.c

```

```

2372      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2374 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/scsi/targets/%.c
2375      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2377 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/sdcard/adapters/sdhost/%.c
2378      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2380 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/sdcard/impl/%.c
2381      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2383 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/sdcard/targets/sdcard/%.c
2384      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2386 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/sfe/%.c
2387      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2389 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/simnet/%.c
2390      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2392 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/softmac/%.c
2393      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2395 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/uath/%.c
2396      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2398 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/uath/uath_fw/%.c
2399      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2401 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ural/%.c
2402      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2404 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/urtw/%.c
2405      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2407 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/clients/audio/usb_ac/%.
2408      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2410 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/clients/audio/usb_as/%.
2411      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2413 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/clients/audio/usb_ah/%.
2414      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2416 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/clients/usbskel/%.c
2417      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2419 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/clients/video/usbvc/%.c
2420      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2422 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/clients/hwarc/%.c
2423      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2425 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/clients/hid/%.c
2426      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2428 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/clients/hidparser/%.c
2429      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2431 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/clients/usbkbm/%.c
2432      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2434 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/clients/usbms/%.c
2435      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2437 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/clients/usbinp/usbwcm

```

```

2438      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2440 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/clients/ugen/%.c
2441      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2443 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/clients/printer/%.c
2444      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2446 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/clients/usbser/%.c
2447      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2449 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/clients/usbser/usbsacm/
2450      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2452 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/clients/usbser/usbftdi/
2453      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2455 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/clients/usbser/usbser_k
2456      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2458 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/clients/usbser/usbsprl/
2459      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2461 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/clients/wusb_df/%.c
2462      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2464 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/clients/hwal480_fw/%.c
2465      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2467 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/clients/wusb_ca/%.c
2468      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2470 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/clients/usbecm/%.c
2471      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2473 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/hcd/openhci/%.c
2474      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2476 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/hcd/ehci/%.c
2477      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2479 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/hcd/uhci/%.c
2480      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2482 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/hubd/%.c
2483      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2485 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/scsa2usb/%.c
2486      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2488 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/usb_mid/%.c
2489      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2491 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/usb_ia/%.c
2492      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2494 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/usba/%.c
2495      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2497 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/usba10/%.c
2498      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2500 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/uwb/uwba/%.c
2501      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2503 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/usb/hwa/hwahc/%.c

```

```

2504      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2506 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/vuidmice/%.c
2507      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2509 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/vnic/%.c
2510      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2512 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/wpi/%.c
2513      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2515 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/zyd/%.c
2516      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2518 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/chxge/com/%.c
2519      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2521 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/chxge/%.c
2522      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2524 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/cxgbe/common/%.c
2525      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2527 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/cxgbe/shared/%.c
2528      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2530 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/cxgbe/firmware/%.c
2531      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2533 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/cxgbe/t4nex/%.c
2534      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2536 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/cxgbe/cxgbe/%.c
2537      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2539 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ixgb/%.c
2540      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2542 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/xge/drv/%.c
2543      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2545 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/xge/hal/xgehal/%.c
2546      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2548 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/e1000g/%.c
2549      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2551 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/igb/%.c
2552      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2554 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/iprb/%.c
2555      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2557 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ixgbe/%.c
2558      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2560 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ntxn/%.c
2561      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2563 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/myril0ge/drv/%.c
2564      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2566 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/ipp/%.c
2567      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2569 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/ipp/ipgpc/%.c

```

```

2570      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2572 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/ipp/dlcosmk/%.c
2573      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2575 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/ipp/flowacct/%.c
2576      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2578 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/ipp/dscpmk/%.c
2579      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2581 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/ipp/meters/%.c
2582      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2584 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/kiconv/kiconv_emea/%.c
2585      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2587 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/kiconv/kiconv_ja/%.c
2588      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2590 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/kiconv/kiconv_ko/%.c
2591      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2593 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/kiconv/kiconv_sc/%.c
2594      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2596 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/kiconv/kiconv_tc/%.c
2597      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2599 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/klm/%.c
2600      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2602 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/kmdb/%.c
2603      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2605 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/krtld/%.c
2606      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2608 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/ktli/%.c
2609      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2611 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/list/%.c
2612      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2614 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/lvm/%.c
2615      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2617 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/lzma/%.c
2618      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2620 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/crypto/md4/%.c
2621      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2623 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/crypto/md5/%.c
2624      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2626 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/net/dhcp/%.c
2627      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2629 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/nvpair/%.c
2630      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2632 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/os/%.c
2633      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2635 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/rpc/%.c

```

```

2636      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2638 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/pcmcia/cs/%.c
2639      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2641 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/pcmcia/cis/%.c
2642      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2644 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/pcmcia/nexus/%.c
2645      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2647 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/pcmcia/pcs/%.c
2648      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2650 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/rpc/%.c
2651      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2653 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/rpc/sec/%.c
2654      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2656 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/rpc/sec_gss/%.c
2657      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2659 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/crypto/shal/%.c
2660      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2662 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/crypto/sha2/%.c
2663      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2665 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/syscall/%.c
2666      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2668 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/tnf/%.c
2669      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2671 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/tsol/%.c
2672      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2674 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/util/%.c
2675      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2677 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/unicode/%.c
2678      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2680 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/vm/%.c
2681      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2683 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/scsi/adapters/iscsi/%.c
2684      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2686 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/iscsi/%.c
2687      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2689 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/kifconf/%.c
2690      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2692 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/virtio/%.c
2693      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2695 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/vioblk/%.c
2696      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2698 ZMODLINTFLAGS = -erroff=E_CONSTANT_CONDITION
2700 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/zmod/%.c
2701      @$(LHEAD) $(LINT.c) $(ZMODLINTFLAGS) $< $(LTAIL))

```

```

2703 $(LINTS_DIR)/zlib_obj.ln:      $(ZLIB_OBJS:%.o=$(LINTS_DIR)/%.ln) \
2704      $(UTSBASE)/common/zmod/zlib_lint.c
2705      @$(LHEAD) $(LINT.c) -C $(LINTS_DIR)/zlib_obj \
2706      $(UTSBASE)/common/zmod/zlib_lint.c $(LTAIL))
2708 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/hxge/%.c
2709      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2711 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/tpm/%.c
2712      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2714 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/tpm/%.s
2715      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2717 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/vr/%.c
2718      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2720 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/yge/%.c
2721      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2723 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/fsreparse/%.c
2724      @$(LHEAD) $(LINT.c) $< $(LTAIL))

```

```

*****
161686 Fri Aug 16 09:04:46 2013
new/usr/src/uts/common/io/tl.c
195 Need replacement for nfs/lockd+klm
*****
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 2010 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */
25 /*
26 * Copyright 2011 Nexenta Systems, Inc. All rights reserved.
27 * Copyright (c) 2012 by Delphix. All rights reserved.
28 */

30 /*
31 * Multithreaded STREAMS Local Transport Provider.
32 *
33 * OVERVIEW
34 * =====
35 *
36 * This driver provides TLI as well as socket semantics. It provides
37 * connectionless, connection oriented, and connection oriented with orderly
38 * release transports for TLI and sockets. Each transport type has separate name
39 * spaces (i.e. it is not possible to connect from a socket to a TLI endpoint) -
40 * this removes any name space conflicts when binding to socket style transport
41 * addresses.
42 *
43 * NOTE: There is one exception: Socket ticots and ticotsord transports share
44 * the same namespace. In fact, sockets always use ticotsord type transport.
45 *
46 * The driver mode is specified during open() by the minor number used for
47 * open.
48 *
49 * The sockets in addition have the following semantic differences:
50 * No support for passing up credentials (TL_SET[U]CRED).
51 *
52 * Options are passed through transparently on T_CONN_REQ to T_CONN_IND,
53 * from T_UNITDATA_REQ to T_UNIDATA_IND, and from T_OPTDATA_REQ to
54 * T_OPTDATA_IND.
55 *
56 * The T_CONN_CON is generated when processing the T_CONN_REQ i.e. before
57 * a T_CONN_RES is received from the acceptor. This means that a socket
58 * connect will complete before the peer has called accept.
59 *
60 *
61 * MULTITHREADING

```

```

62 * =====
63 *
64 * The driver does not use STREAMS protection mechanisms. Instead it uses a
65 * generic "serializer" abstraction. Most of the operations are executed behind
66 * the serializer and are, essentially single-threaded. All functions executed
67 * behind the same serializer are strictly serialized. So if one thread calls
68 * serializer_enter(serializer, foo, mp1, arg1); and another thread calls
69 * serializer_enter(serializer, bar, mp2, arg1); then (depending on which one
70 * was called) the actual sequence will be foo(mp1, arg1); bar(mp1, arg2) or
71 * bar(mp1, arg2); foo(mp1, arg1); But foo() and bar() will never run at the
72 * same time.
73 *
74 * Connectionless transport use a single serializer per transport type (one for
75 * TLI and one for sockets. Connection-oriented transports use finer-grained
76 * serializers.
77 *
78 * All COTS-type endpoints start their life with private serializers. During
79 * connection request processing the endpoint serializer is switched to the
80 * listener's serializer and the rest of T_CONN_REQ processing is done on the
81 * listener serializer. During T_CONN_RES processing the eager serializer is
82 * switched from listener to acceptor serializer and after that point all
83 * processing for eager and acceptor happens on this serializer. To avoid races
84 * with endpoint closes while its serializer may be changing closes are blocked
85 * while serializers are manipulated.
86 *
87 * References accounting
88 * -----
89 *
90 * Endpoints are reference counted and freed when the last reference is
91 * dropped. Functions within the serializer may access an endpoint state even
92 * after an endpoint closed. The te_closing being set on the endpoint indicates
93 * that the endpoint entered its close routine.
94 *
95 * One reference is held for each opened endpoint instance. The reference
96 * counter is incremented when the endpoint is linked to another endpoint and
97 * decremented when the link disappears. It is also incremented when the
98 * endpoint is found by the hash table lookup. This increment is atomic with the
99 * lookup itself and happens while the hash table read lock is held.
100 *
101 * Close synchronization
102 * -----
103 *
104 * During close the endpoint as marked as closing using te_closing flag. It is
105 * usually enough to check for te_closing flag since all other state changes
106 * happen after this flag is set and the close entered serializer. Immediately
107 * after setting te_closing flag tl_close() enters serializer and waits until
108 * the callback finishes. This allows all functions called within serializer to
109 * simply check te_closing without any locks.
110 *
111 * Serializer management.
112 * -----
113 *
114 * For COTS transports serializers are created when the endpoint is constructed
115 * and destroyed when the endpoint is destructed. CLTS transports use global
116 * serializers - one for sockets and one for TLI.
117 *
118 * COTS serializers have separate reference counts to deal with several
119 * endpoints sharing the same serializer. There is a subtle problem related to
120 * the serializer destruction. The serializer should never be destroyed by any
121 * function executed inside serializer. This means that close has to wait till
122 * all serializer activity for this endpoint is finished before it can drop the
123 * last reference on the endpoint (which may as well free the serializer). This
124 * is only relevant for COTS transports which manage serializers
125 * dynamically. For CLTS transports close may complete without waiting for all
126 * serializer activity to finish since serializer is only destroyed at driver
127 * detach time.

```

```

128 *
129 * COTS endpoints keep track of the number of outstanding requests on the
130 * serializer for the endpoint. The code handling accept() avoids changing
131 * client serializer if it has any pending messages on the serializer and
132 * instead moves acceptor to listener's serializer.
133 *
134 *
135 * Use of hash tables
136 * -----
137 *
138 * The driver uses modhash hash table implementation. Each transport uses two
139 * hash tables - one for finding endpoints by acceptor ID and another one for
140 * finding endpoints by address. For sockets TICOTS and TICOTSORD share the same
141 * pair of hash tables since sockets only use TICOTSORD.
142 *
143 * All hash tables lookups increment a reference count for returned endpoints,
144 * so we may safely check the endpoint state even when the endpoint is removed
145 * from the hash by another thread immediately after it is found.
146 *
147 *
148 * CLOSE processing
149 * =====
150 *
151 * The driver enters serializer twice on close(). The close sequence is the
152 * following:
153 *
154 * 1) Wait until closing is safe (te_closewait becomes zero)
155 *    This step is needed to prevent close during serializer switches. In most
156 *    cases (close happening after connection establishment) te_closewait is
157 *    zero.
158 * 1) Set te_closing.
159 * 2) Call tl_close_ser() within serializer and wait for it to complete.
160 *
161 *    te_close_ser simply marks endpoint and wakes up waiting tl_close().
162 *    It also needs to clear write-side q_next pointers - this should be done
163 *    before qprocsoff().
164 *
165 * This synchronous serializer entry during close is needed to ensure that
166 * the queue is valid everywhere inside the serializer.
167 *
168 * Note that in many cases close will execute tl_close_ser() synchronously,
169 * so it will not wait at all.
170 *
171 * 3) Calls qprocsoff().
172 * 4) Calls tl_close_finish_ser() within the serializer and waits for it to
173 *    complete (for COTS transports). For CLTS transport there is no wait.
174 *
175 *    tl_close_finish_ser() Finishes the close process and wakes up waiting
176 *    close if there is any.
177 *
178 * Note that in most cases close will enter te_close_ser_finish()
179 * synchronously and will not wait at all.
180 *
181 *
182 * Flow Control
183 * =====
184 *
185 * The driver implements both read and write side service routines. No one calls
186 * putq() on the read queue. The read side service routine tl_rsrv() is called
187 * when the read side stream is back-enabled. It enters serializer synchronously
188 * (waits till serializer processing is complete). Within serializer it
189 * back-enables all endpoints blocked by the queue for connection-less
190 * transports and enables write side service processing for the peer for
191 * connection-oriented transports.
192 *
193 * Read and write side service routines use special mblk_sized space in the

```

```

194 * endpoint structure to enter perimeter.
195 *
196 * Write-side flow control
197 * -----
198 *
199 * Write side flow control is a bit tricky. The driver needs to deal with two
200 * message queues - the explicit STREAMS message queue maintained by
201 * putq()/getq()/putbq() and the implicit queue within the serializer. These two
202 * queues should be synchronized to preserve message ordering and should
203 * maintain a single order determined by the order in which messages enter
204 * tl_wput(). In order to maintain the ordering between these two queues the
205 * STREAMS queue is only manipulated within the serializer, so the ordering is
206 * provided by the serializer.
207 *
208 * Functions called from the tl_wsrv() sometimes may call putbq(). To
209 * immediately stop any further processing of the STREAMS message queues the
210 * code calling putbq() also sets the te_nowsrv flag in the endpoint. The write
211 * side service processing stops when the flag is set.
212 *
213 * The tl_wsrv() function enters serializer synchronously and waits for it to
214 * complete. The serializer call-back tl_wsrv_ser() either drains all messages
215 * on the STREAMS queue or terminates when it notices the te_nowsrv flag
216 * set. Note that the maximum amount of messages processed by tl_wput_ser() is
217 * always bounded by the amount of messages on the STREAMS queue at the time
218 * tl_wsrv_ser() is entered. Any new messages may only appear on the STREAMS
219 * queue from another serialized entry which can't happen in parallel. This
220 * guarantees that tl_wput_ser() is complete in bounded time (there is no risk
221 * of it draining forever while writer places new messages on the STREAMS
222 * queue).
223 *
224 * Note that a closing endpoint never sets te_nowsrv and never calls putbq().
225 *
226 *
227 * Unix Domain Sockets
228 * =====
229 *
230 * The driver knows the structure of Unix Domain sockets addresses and treats
231 * them differently from generic TLI addresses. For sockets implicit binds are
232 * requested by setting SOU_MAGIC_IMPLICIT in the sou_magic part of the address
233 * instead of using address length of zero. Explicit binds specify
234 * SOU_MAGIC_EXPLICIT as magic.
235 *
236 * For implicit binds we always use minor number as soua_vp part of the address
237 * and avoid any hash table lookups. This saves two hash tables lookups per
238 * anonymous bind.
239 *
240 * For explicit address we hash the vnode pointer instead of hashing the
241 * full-scale address+zone+length. Hashing by pointer is more efficient than
242 * hashing by the full address.
243 *
244 * For unix domain sockets the te_ap is always pointing to te_uaddr part of the
245 * tep structure, so it should be never freed.
246 *
247 * Also for sockets the driver always uses minor number as acceptor id.
248 *
249 * TPI VIOLATIONS
250 * -----
251 *
252 * This driver violates TPI in several respects for Unix Domain Sockets:
253 *
254 * 1) It treats O_T_BIND_REQ as T_BIND_REQ and refuses bind if an explicit bind
255 *    is requested and the endpoint is already in use. There is no point in
256 *    generating an unused address since this address will be rejected by
257 *    sockfs anyway. For implicit binds it always generates a new address
258 *    (sets soua_vp to its minor number).
259 *

```

```

260 * 2) It always uses minor number as acceptor ID and never uses queue
261 * pointer. It is ok since sockets get acceptor ID from T_CAPABILITY_REQ
262 * message and they do not use the queue pointer.
263 *
264 * 3) For Listener sockets the usual sequence is to issue bind() zero backlog
265 * followed by listen(). The listen() should be issued with non-zero
266 * backlog, so sotpi_listen() issues unbind request followed by bind
267 * request to the same address but with a non-zero qlen value. Both
268 * tl_bind() and tl_unbind() require write lock on the hash table to
269 * insert/remove the address. The driver does not remove the address from
270 * the hash for endpoints that are bound to the explicit address and have
271 * backlog of zero. During T_BIND_REQ processing if the address requested
272 * is equal to the address the endpoint already has it updates the backlog
273 * without reinserting the address in the hash table. This optimization
274 * avoids two hash table updates for each listener created. It always
275 * avoids the problem of a "stolen" address when another listener may use
276 * the same address between the unbind and bind and suddenly listen() fails
277 * because address is in use even though the bind() succeeded.
278 *
279 *
280 * CONNECTIONLESS TRANSPORTS
281 * =====
282 *
283 * Connectionless transports all share the same serializer (one for TLI and one
284 * for Sockets). Functions executing behind serializer can check or modify state
285 * of any endpoint.
286 *
287 * When endpoint X talks to another endpoint Y it caches the pointer to Y in the
288 * te_lastep field. The next time X talks to some address A it checks whether A
289 * is the same as Y's address and if it is there is no need to lookup Y. If the
290 * address is different or the state of Y is not appropriate (e.g. closed or not
291 * idle) X does a lookup using tl_find_peer() and caches the new address.
292 * NOTE: tl_find_peer() never returns closing endpoint and it places a refhold
293 * on the endpoint found.
294 *
295 * During close of endpoint Y it doesn't try to remove itself from other
296 * endpoints caches. They will detect that Y is gone and will search the peer
297 * endpoint again.
298 *
299 * Flow Control Handling.
300 * -----
301 *
302 * Each connectionless endpoint keeps a list of endpoints which are
303 * flow-controlled by its queue. It also keeps a pointer to the queue which
304 * flow-controls itself. Whenever flow control releases for endpoint X it
305 * enables all queues from the list. During close it also back-enables everyone
306 * in the list. If X is flow-controlled when it is closing it removes it from
307 * the peers list.
308 *
309 * DATA STRUCTURES
310 * =====
311 *
312 * Each endpoint is represented by the tl_endpt_t structure which keeps all the
313 * endpoint state. For connection-oriented transports it has a keeps a list
314 * of pending connections (tl_icon_t). For connectionless transports it keeps a
315 * list of endpoints flow controlled by this one.
316 *
317 * Each transport type is represented by a per-transport data structure
318 * tl_transport_state_t. It contains a pointer to an acceptor ID hash and the
319 * endpoint address hash tables for each transport. It also contains pointer to
320 * transport serializer for connectionless transports.
321 *
322 * Each endpoint keeps a link to its transport structure, so the code can find
323 * all per-transport information quickly.
324 */

```

```

326 #include <sys/types.h>
327 #include <sys/inttypes.h>
328 #include <sys/stream.h>
329 #include <sys/stropts.h>
330 #define _SUN_TPI_VERSION 2
331 #include <sys/tihdr.h>
332 #include <sys/strlog.h>
333 #include <sys/debug.h>
334 #include <sys/cred.h>
335 #include <sys/errno.h>
336 #include <sys/kmem.h>
337 #include <sys/id_space.h>
338 #include <sys/modhash.h>
339 #include <sys/mkdev.h>
340 #include <sys/tl.h>
341 #include <sys/stat.h>
342 #include <sys/conf.h>
343 #include <sys/modctl.h>
344 #include <sys/strsun.h>
345 #include <sys/socket.h>
346 #include <sys/socketvar.h>
347 #include <sys/sysmacros.h>
348 #include <sys/xti_xtiopt.h>
349 #include <sys/ddi.h>
350 #include <sys/sunddi.h>
351 #include <sys/zone.h>
352 #include <inet/common.h> /* typedef int (*pfi_t)() for inet/optcom.h */
353 #include <inet/optcom.h>
354 #include <sys/strsubr.h>
355 #include <sys/ucred.h>
356 #include <sys/suntpi.h>
357 #include <sys/list.h>
358 #include <sys/serializer.h>
359
360 /*
361 * TBD List
362 * 14 Eliminate state changes through table
363 * 16. AF_UNIX socket options
364 * 17. connect() for ticlts
365 * 18. support for "netstat" to show AF_UNIX plus TLI local
366 * transport connections
367 * 21. sanity check to flushing on sending M_ERROR
368 */
369
370 /*
371 * CONSTANT DECLARATIONS
372 * -----
373 */
374
375 /*
376 * Local declarations
377 */
378 #define NEXTSTATE(EV, ST) ti_statetbl[EV][ST]
379
380 #define BADSEQNUM (-1) /* initial seq number used by T_DISCON_IND */
381 #define TL_BUFWAIT (10000) /* usecs to wait for allocb buffer timeout */
382 #define TL_TIDUSZ (64*1024) /* tidu size when "strmsgz" is unlimited (0) */
383 /*
384 * Hash tables size.
385 */
386 #define TL_HASH_SIZE 311
387
388 /*
389 * Definitions for module_info
390 */
391 #define TL_ID (104) /* module ID number */

```

```

392 #define TL_NAME "tl" /* module name */
393 #define TL_MINPSZ (0) /* min packet size */
394 #define TL_MAXPSZ INFPSZ /* max packet size ZZZ */
395 #define TL_HIWAT (16*1024) /* hi water mark */
396 #define TL_LOWAT (256) /* lo water mark */
397 /*
398 * Definition of minor numbers/modes for new transport provider modes.
399 * We view the socket use as a separate mode to get a separate name space.
400 */
401 #define TL_TICOTS 0 /* connection oriented transport */
402 #define TL_TICOTSORD 1 /* COTS w/ orderly release */
403 #define TL_TICLTS 2 /* connectionless transport */
404 #define TL_UNUSED 3
405 #define TL_SOCKET 4 /* Socket */
406 #define TL SOCK_COTS (TL_SOCKET|TL_TICOTS)
407 #define TL SOCK_COTSORD (TL_SOCKET|TL_TICOTSORD)
408 #define TL SOCK_CLTS (TL_SOCKET|TL_TICLTS)

410 #define TL_MINOR_MASK 0x7
411 #define TL_MINOR_START (TL_TICLTS + 1)

413 /*
414 * LOCAL MACROS
415 */
416 #define T_ALIGN(p) P2ROUNDUP((p), sizeof (t_scalar_t))

418 /*
419 * EXTERNAL VARIABLE DECLARATIONS
420 * -----
421 */
422 /*
423 * state table defined in the OS space.c
424 */
425 extern char ti_statetbl[TE_NOEVENTS][TS_NOSTATES];

427 /*
428 * STREAMS DRIVER ENTRY POINTS PROTOTYPES
429 */
430 static int tl_open(queue_t *, dev_t *, int, int, cred_t *);
431 static int tl_close(queue_t *, int, cred_t *);
432 static void tl_wput(queue_t *, mblk_t *);
433 static void tl_wsrv(queue_t *);
434 static void tl_rsrv(queue_t *);

436 static int tl_attach(dev_info_t *, ddi_attach_cmd_t);
437 static int tl_detach(dev_info_t *, ddi_detach_cmd_t);
438 static int tl_info(dev_info_t *, ddi_info_cmd_t, void *, void **);

441 /*
442 * GLOBAL DATA STRUCTURES AND VARIABLES
443 * -----
444 */

446 /*
447 * Table representing database of all options managed by T_SVR4_OPTMGMT_REQ
448 * For now, we only manage the SO_RECVUCRED option but we also have
449 * harmless dummy options to make things work with some common code we access.
450 */
451 opdes_t tl_opt_arr[] = {
452 /* The SO_TYPE is needed for the hack below */
453 {
454 SO_TYPE,
455 SOL_SOCKET,
456 OA_R,
457 OA_R,

```

```

458 OP_NP,
459 0,
460 sizeof (t_scalar_t),
461 0
462 },
463 {
464 SO_RECVUCRED,
465 SOL_SOCKET,
466 OA_RW,
467 OA_RW,
468 OP_NP,
469 0,
470 sizeof (int),
471 0
472 }
473 };
unchanged_portion_omitted

4253 static void
4254 tl_connected_cots_addr_req(mblk_t *mp, tl_endpt_t *tep)
4255 {
4256     tl_endpt_t *peer_tep = tep->te_conp;
4257     tl_endpt_t *peer_tep;
4258     size_t ack_sz;
4259     mblk_t *ackmp;
4260     struct T_addr_ack *taa;
4261     uchar_t *addr_startp;

4262     if (tep->te_closing) {
4263         freemsg(mp);
4264         return;
4265     }

4267     if (peer_tep == NULL || peer_tep->te_closing) {
4268         tl_error_ack(tep->te_wq, mp, TSYSERR, ECONNRESET, T_ADDR_REQ);
4269         return;
4270     }

4272     ASSERT(tep->te_state >= TS_IDLE);

4274     ack_sz = sizeof (struct T_addr_ack);
4275     ack_sz += T_ALIGN(tep->te_alen);
4276     peer_tep = tep->te_conp;
4277     ack_sz += peer_tep->te_alen;

4278     ackmp = tpi_ack_alloc(mp, ack_sz, M_PCPROTO, T_ADDR_ACK);
4279     if (ackmp == NULL) {
4280         (void) (STRLOG(TL_ID, tep->te_minor, 1, SL_TRACE|SL_ERROR,
4281             "tl_connected_cots_addr_req: realloc failed"));
4282         tl_memrecover(tep->te_wq, mp, ack_sz);
4283         return;
4284     }

4286     taa = (struct T_addr_ack *)ackmp->b_rptr;

4288     /* endpoint is bound */
4289     taa->LOCADDR_length = tep->te_alen;
4290     taa->LOCADDR_offset = (t_scalar_t)sizeof (*taa);

4292     addr_startp = (uchar_t *)&taa[1];

4294     bcopy(tep->te_abuf, addr_startp,
4295         tep->te_alen);

4297     taa->REMADDR_length = peer_tep->te_alen;

```



```
4298     taa->REMADDR_offset = (t_scalar_t)T_ALIGN(taa->LOCADDR_offset +
4299         taa->LOCADDR_length);
4300     addr_startp = ackmp->b_rptr + taa->REMADDR_offset;
4301     bcopy(peer_tep->te_abuf, addr_startp,
4302         peer_tep->te_alen);
4303     ackmp->b_wptr = (uchar_t *)ackmp->b_rptr +
4304         taa->REMADDR_offset + peer_tep->te_alen;
4305     ASSERT(ackmp->b_wptr <= ackmp->b_datap->db_lim);

4307     putnext(tep->te_rq, ackmp);
4308 }
```

unchanged portion omitted

```
*****
```

```
1859 Fri Aug 16 09:04:48 2013
```

```
new/usr/src/uts/common/klm/Makefile
```

```
195 Need replacement for nfs/lockd+klm
```

```
*****
```

```
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 is of the CDDL is also available via the Internet
9 # at http://www.illumos.org/license/CDDL.
10 #
```

```
12 #
13 # Copyright 2010 Nexenta Systems, Inc. All rights reserved.
14 # Copyright (c) 2012 by Delphix. All rights reserved.
15 #
```

```
17 include ../../../Makefile.master
```

```
19 NLM_PROT_X= ../rpcsvc/nlm_prot.x
20 SM_INTER_X= ../rpcsvc/sm_inter.x
21 NSM_ADDR_X= ../rpcsvc/nsm_addr.x
```

```
23 RPCGENFLAGS = -C -M -i 0
24 SED_INCL='/^\.include/s:\.\.\*/rpcsvc:rpcsvc:'
```

```
26 DERIVED_FILES= nlm_prot_clnt.c nlm_prot_xdr.c \
27                 sm_inter_clnt.c sm_inter_xdr.c \
28                 nsm_addr_clnt.c nsm_addr_xdr.c
```

```
30 install_h: all_h
```

```
32 all_h: $(DERIVED_FILES)
```

```
34 nlm_prot_clnt.c : $(NLM_PROT_X) nlm_prot_clnt.sed
35     $(RPCGEN) $(RPCGENFLAGS) -l -o $@.tmp $(NLM_PROT_X)
36     sed -f nlm_prot_clnt.sed < $@.tmp > $@
37     $(RM) -f $@.tmp
```

```
39 nlm_prot_xdr.c : $(NLM_PROT_X)
40     $(RPCGEN) $(RPCGENFLAGS) -c -o $@.tmp $(NLM_PROT_X)
41     sed -e $(SED_INCL) < $@.tmp > $@
42     $(RM) -f $@.tmp
```

```
44 sm_inter_clnt.c : $(SM_INTER_X) sm_inter_clnt.sed
45     $(RPCGEN) $(RPCGENFLAGS) -l -o $@.tmp $(SM_INTER_X)
46     sed -f sm_inter_clnt.sed < $@.tmp > $@
47     $(RM) -f $@.tmp
```

```
49 sm_inter_xdr.c : $(SM_INTER_X)
50     $(RPCGEN) $(RPCGENFLAGS) -c -o $@.tmp $(SM_INTER_X)
51     sed -e $(SED_INCL) < $@.tmp > $@
52     $(RM) -f $@.tmp
```

```
54 nsm_addr_clnt.c : $(NSM_ADDR_X) nsm_addr_clnt.sed
55     $(RPCGEN) $(RPCGENFLAGS) -l -o $@.tmp $(NSM_ADDR_X)
56     sed -f nsm_addr_clnt.sed < $@.tmp > $@
57     $(RM) -f $@.tmp
```

```
59 nsm_addr_xdr.c : $(NSM_ADDR_X)
60     $(RPCGEN) $(RPCGENFLAGS) -c -o $@.tmp $(NSM_ADDR_X)
61     sed -e $(SED_INCL) < $@.tmp > $@
```

```
62     $(RM) -f $@.tmp
64 check:
66 clean:
67     $(RM) $(DERIVED_FILES)
69 clobber:      clean
71 lint:
73 .KEEP_STATE:
```

```

*****
11343 Fri Aug 16 09:04:48 2013
new/usr/src/uts/common/klm/klmmod.c
195 Need replacement for nfs/lockd+klm
*****
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 is of the CDDL is also available via the Internet
9  * at http://www.illumos.org/license/CDDL.
10 */

12 /*
13  * Copyright 2011 Nexenta Systems, Inc. All rights reserved.
14  * Copyright (c) 2012 by Delphix. All rights reserved.
15 */

17 /*
18  * NFS Lock Manager, server-side and common.
19  *
20  * This file contains all the external entry points of klmmod.
21  * Basically, this is the "glue" to the BSD nlm code.
22 */

24 #include <sys/types.h>
25 #include <sys/errno.h>
26 #include <sys/modctl.h>
27 #include <sys/flock.h>

29 #include <nfs/nfs.h>
30 #include <nfs/nfssys.h>
31 #include <nfs/lm.h>
32 #include <rpcsvc/nlm_prot.h>
33 #include "nlm_impl.h"

35 static struct modlmisc modlmisc = {
36     &mod_miscops, "lock mgr common module"
37 };

39 static struct modlinkage modlinkage = {
40     MODREV_1, &modlmisc, NULL
41 };

43 /*
44  * Cluster node ID. Zero unless we're part of a cluster.
45  * Set by lm_set_nlmid flk. Pass to lm_set_nlm_status.
46  * We're not yet doing "clustered" NLM stuff.
47 */
48 int lm_global_nlmid = 0;

50 /*
51  * Call-back hook for clusters: Set lock manager status.
52  * If this hook is set, call this instead of the usual
53  * flk_set_lockmgr_status(FLK_LOCKMGR_UP / DOWN);
54 */
55 void (*lm_set_nlm_status)(int nlm_id, flk_nlm_status_t) = NULL;

57 /*
58  * Call-back hook for clusters: Delete all locks held by sysid.
59  * Call from code that drops all client locks (for which we're
60  * the server) i.e. after the SM tells us a client has crashed.
61 */

```

```

62 void (*lm_remove_file_locks)(int) = NULL;

64 krwlock_t          lm_lck;
65 zone_key_t         nlm_zone_key;

67 /*
68  * Init/fini per-zone stuff for klm
69 */
70 /* ARGSUSED */
71 void *
72 lm_zone_init(zoneid_t zoneid)
73 {
74     struct nlm_globals *g;

76     g = kmem_zalloc(sizeof(*g), KM_SLEEP);

78     avl_create(&g->nlm_hosts_tree, nlm_host_cmp,
79              sizeof(struct nlm_host),
80              offsetof(struct nlm_host, nh_by_addr));

82     g->nlm_hosts_hash = mod_hash_create_idhash("nlm_host_by_sysid",
83                                               64, mod_hash_null_valdtor);

85     TAILQ_INIT(&g->nlm_idle_hosts);
86     TAILQ_INIT(&g->nlm_slocks);

88     mutex_init(&g->lock, NULL, MUTEX_DEFAULT, NULL);
89     cv_init(&g->nlm_gc_sched_cv, NULL, CV_DEFAULT, NULL);
90     cv_init(&g->nlm_gc_finish_cv, NULL, CV_DEFAULT, NULL);
91     mutex_init(&g->clean_lock, NULL, MUTEX_DEFAULT, NULL);

93     g->lockd_pid = 0;
94     g->run_status = NLM_ST_DOWN;

96     nlm_globals_register(g);
97     return (g);
98 }

100 /* ARGSUSED */
101 void
102 lm_zone_fini(zoneid_t zoneid, void *data)
103 {
104     struct nlm_globals *g = data;

106     ASSERT(avl_is_empty(&g->nlm_hosts_tree));
107     avl_destroy(&g->nlm_hosts_tree);
108     mod_hash_destroy_idhash(g->nlm_hosts_hash);

110     ASSERT(g->nlm_gc_thread == NULL);
111     mutex_destroy(&g->lock);
112     cv_destroy(&g->nlm_gc_sched_cv);
113     cv_destroy(&g->nlm_gc_finish_cv);
114     mutex_destroy(&g->clean_lock);

116     nlm_globals_unregister(g);
117     kmem_free(g, sizeof(*g));
118 }

122 /*
123  * *****
124  * module init, fini, info
125 */
126 int
127 _init()

```

```

128 {
129     int retval;

131     rw_init(&lm_lck, NULL, RW_DEFAULT, NULL);
132     nlm_init();

134     zone_key_create(&nlm_zone_key, lm_zone_init, NULL, lm_zone_fini);
135     /* Per-zone lockmgr data. See: os/flock.c */
136     zone_key_create(&flock_zone_key, flk_zone_init, NULL, flk_zone_fini);

138     retval = mod_install(&modlinkage);
139     if (retval == 0)
140         return (0);

142     /*
143      * mod_install failed! undo above, reverse order
144      */

146     (void) zone_key_delete(flock_zone_key);
147     flock_zone_key = ZONE_KEY_UNINITIALIZED;
148     (void) zone_key_delete(nlm_zone_key);
149     rw_destroy(&lm_lck);

151     return (retval);
152 }

154 int
155 _fini()
156 {
157     /* Don't unload. */
158     return (EBUSY);
159 }

161 int
162 _info(struct modinfo *modinfo)
163 {
164     return (mod_info(&modlinkage, modinfo));
165 }

169 /*
170 * *****
171 * Stubs listed in modstubs.s
172 */

174 /*
175 * klm system calls. Start service on some endpoint.
176 * Called by nfssys() LM_SVC, from lockd.
177 */
178 int
179 lm_svc(struct lm_svc_args *args)
180 {
181     struct knetconfig knc;
182     const char *netid;
183     struct nlm_globals *g;
184     struct file *fp = NULL;
185     int err = 0;

187     /* Get our "globals" */
188     g = zone_getspecific(nlm_zone_key, curzone);

190     /*
191      * Check version of lockd calling.
192      */
193     if (args->version != LM_SVC_CUR_VERS) {

```

```

194         NLM_ERR("lm_svc: Version mismatch "
195                "(given 0x%x, expected 0x%x)\n",
196                args->version, LM_SVC_CUR_VERS);
197         return (EINVAL);
198     }

200     /*
201      * Build knetconfig, checking arg values.
202      * Also come up with the "netid" string.
203      * (With some knowledge of /etc/netconfig)
204      */
205     bzero(&knc, sizeof (knc));
206     switch (args->n_proto) {
207     case LM_TCP:
208         knc.knc_semantics = NC_TPI_COTS_ORD;
209         knc.knc_proto = NC_TCP;
210         break;
211     case LM_UDP:
212         knc.knc_semantics = NC_TPI_CLTS;
213         knc.knc_proto = NC_UDP;
214         break;
215     default:
216         NLM_ERR("nlm_build_knetconfig: Unknown "
217                "lm_proto=0x%x\n", args->n_proto);
218         return (EINVAL);
219     }

221     switch (args->n_fmly) {
222     case LM_INET:
223         knc.knc_protofmly = NC_INET;
224         break;
225     case LM_INET6:
226         knc.knc_protofmly = NC_INET6;
227         break;
228     case LM_LOOPBACK:
229         knc.knc_protofmly = NC_LOOPBACK;
230         /* Override what we set above. */
231         knc.knc_proto = NC_NOPROTO;
232         break;
233     default:
234         NLM_ERR("nlm_build_knetconfig: Unknown "
235                "lm_fmly=0x%x\n", args->n_fmly);
236         return (EINVAL);
237     }

239     knc.knc_rdev = args->n_rdev;
240     netid = nlm_knc_to_netid(&knc);
241     if (!netid)
242         return (EINVAL);

244     /*
245      * Setup service on the passed transport.
246      * NB: must release(fp) after this.
247      */
248     if ((fp = getf(args->fd)) == NULL)
249         return (EBADF);

251     mutex_enter(&g->lock);
252     /*
253      * Don't try to start while still shutting down,
254      * or lots of things will fail...
255      */
256     if (g->run_status == NLM_ST_STOPPING) {
257         err = EAGAIN;
258         goto out;
259     }

```

```

261  /*
262  * There is no separate "initialize" sub-call for nfssys,
263  * and we want to do some one-time work when the first
264  * binding comes in from lockd.
265  */
266  if (g->run_status == NLM_ST_DOWN) {
267      g->run_status = NLM_ST_STARTING;
268      g->lockd_pid = curproc->p_pid;

270      /* Save the options. */
271      g->cn_idle_tmo = args->timeout;
272      g->grace_period = args->grace;
273      g->retrans_tmo = args->retransmittimeout;

275      /* See nfs_sys.c (not yet per-zone) */
276      if (INGLOBALZONE(curproc)) {
277          rfs4_grace_period = args->grace;
278          rfs4_lease_time = args->grace;
279      }

281      mutex_exit(&g->lock);
282      err = nlm_svc_starting(g, fp, netid, &knc);
283      mutex_enter(&g->lock);
284  } else {
285      /*
286      * If KLM is not started and the very first endpoint lockd
287      * tries to add is not a loopback device, report an error.
288      */
289      if (g->run_status != NLM_ST_UP) {
290          err = ENOTACTIVE;
291          goto out;
292      }
293      if (g->lockd_pid != curproc->p_pid) {
294          /* Check if caller has the same PID lockd does */
295          err = EPERM;
296          goto out;
297      }

299      err = nlm_svc_add_ep(fp, netid, &knc);
300  }

302 out:
303     mutex_exit(&g->lock);
304     if (fp != NULL)
305         releasef(args->fd);

307     return (err);
308 }

310 /*
311 * klm system calls. Kill the lock manager.
312 * Called by nfssys() KILL_LOCKMGR,
313 * liblm:lm_shutdown() <- unused?
314 */
315 int
316 lm_shutdown(void)
317 {
318     struct nlm_globals *g;
319     proc_t *p;
320     pid_t pid;

322     /* Get our "globals" */
323     g = zone_getspecific(nlm_zone_key, curzone);

325     mutex_enter(&g->lock);

```

```

326     if (g->run_status != NLM_ST_UP) {
327         mutex_exit(&g->lock);
328         return (EBUSY);
329     }

331     g->run_status = NLM_ST_STOPPING;
332     pid = g->lockd_pid;
333     mutex_exit(&g->lock);
334     nlm_svc_stopping(g);

336     mutex_enter(&pidlock);
337     p = prfind(pid);
338     if (p != NULL)
339         psignal(p, SIGTERM);

341     mutex_exit(&pidlock);
342     return (0);
343 }

345 /*
346 * Cleanup remote locks on FS un-export.
347 *
348 * NOTE: called from nfs_export.c:unexport()
349 * right before the share is going to
350 * be unexported.
351 */
352 void
353 lm_unexport(struct exportinfo *exi)
354 {
355     nlm_unexport(exi);
356 }

358 /*
359 * CPR suspend/resume hooks.
360 * See: cpr_suspend, cpr_resume
361 *
362 * Before suspend, get current state from "statd" on
363 * all remote systems for which we have locks.
364 *
365 * After resume, check with those systems again,
366 * and either reclaim locks, or do SIGLOST.
367 */
368 void
369 lm_cprsuspend(void)
370 {
371     nlm_cprsuspend();
372 }

374 void
375 lm_cprresume(void)
376 {
377     nlm_cprresume();
378 }

380 /*
381 * Add the nlm_id bits to the sysid (by ref).
382 */
383 void
384 lm_set_nlmid_flk(int *new_sysid)
385 {
386     if (lm_global_nlmid != 0)
387         *new_sysid |= (lm_global_nlmid << BITS_IN_SYSID);
388 }

390 /*
391 * It seems that closed source klmmod used

```

```

392 * this function to release knetconfig stored
393 * in mntinfo structure (see mntinfo's mi_klmconfig
394 * field).
395 * We store knetconfigs differently, thus we don't
396 * need this function.
397 */
398 void
399 lm_free_config(struct knetconfig *knc)
400 {
401     _NOTE(ARGUNUSED(knc));
402 }
403
404 /*
405 * Called by NFS4 delegation code to check if there are any
406 * NFSv2/v3 locks for the file, so it should not delegate.
407 *
408 * NOTE: called from NFSv4 code
409 * (see nfs4_srv_deleg.c:rfs4_bgrant_delegation())
410 */
411 int
412 lm_vp_active(const vnode_t *vp)
413 {
414     return (nlm_vp_active(vp));
415 }
416
417 /*
418 * Find or create a "sysid" for given knc+addr.
419 * name is optional. Sets nc_changed if the
420 * found knc_proto is different from passed.
421 * Increments the reference count.
422 *
423 * Called internally, and in nfs4_find_sysid()
424 */
425 struct lm_sysid *
426 lm_get_sysid(struct knetconfig *knc, struct netbuf *addr,
427             char *name, bool_t *nc_changed)
428 {
429     struct nlm_globals *g;
430     const char *netid;
431     struct nlm_host *hostp;
432
433     _NOTE(ARGUNUSED(nc_changed));
434     netid = nlm_knc_to_netid(knc);
435     if (netid == NULL)
436         return (NULL);
437
438     g = zone_getspecific(nlm_zone_key, curzone);
439
440     hostp = nlm_host_findcreate(g, name, netid, addr);
441     if (hostp == NULL)
442         return (NULL);
443
444     return ((struct lm_sysid *)hostp);
445 }
446
447 /*
448 * Release a reference on a "sysid".
449 */
450 void
451 lm_rel_sysid(struct lm_sysid *sysid)
452 {
453     struct nlm_globals *g;
454
455     g = zone_getspecific(nlm_zone_key, curzone);
456     nlm_host_release(g, (struct nlm_host *)sysid);
457 }

```

```

459 /*
460 * Alloc/free a sysid_t (a unique number between
461 * LM_SYSID and LM_SYSID_MAX).
462 *
463 * Used by NFSv4 rfs4_op_lockt and smbfs/smb_fsop_frlock,
464 * both to represent non-local locks outside of klm.
465 *
466 * NOTE: called from NFSv4 and SMBFS to allocate unique
467 * sysid.
468 */
469 sysid_t
470 lm_alloc_sysid(void)
471 {
472     return (nlm_sysid_alloc());
473 }
474
475 void
476 lm_free_sysid(sysid_t sysid)
477 {
478     nlm_sysid_free(sysid);
479 }
480
481 /* Access private member lms->sysid */
482 sysid_t
483 lm_sysid(struct lm_sysid *lms)
484 {
485     return (((struct nlm_host *)lms)->nh_sysid);
486 }
487
488 /*
489 * Called by nfs_frlock to check lock constraints.
490 * Return non-zero if the lock request is "safe", i.e.
491 * the range is not mapped, not MANDLOCK, etc.
492 *
493 * NOTE: called from NFSv3/NFSv2 frlock() functions to
494 * determine whether it's safe to add new lock.
495 */
496 int
497 lm_safelock(vnode_t *vp, const struct flock64 *fl, cred_t *cr)
498 {
499     return (nlm_safelock(vp, fl, cr));
500 }
501
502 /*
503 * Called by nfs_lockcompletion to check whether it's "safe"
504 * to map the file (and cache it's data). Walks the list of
505 * file locks looking for any that are not "whole file".
506 *
507 * NOTE: called from nfs_client.c:nfs_lockcompletion()
508 */
509 int
510 lm_safemap(const vnode_t *vp)
511 {
512     return (nlm_safemap(vp));
513 }
514
515 /*
516 * Called by nfs_map() for the MANDLOCK case.
517 * Return non-zero if the file has any locks with a
518 * blocked request (sleep).
519 *
520 * NOTE: called from NFSv3/NFSv2 map() functions in
521 * order to determine whether it's safe to add new
522 * mapping.
523 */

```

```
524 int
525 lm_has_sleep(const vnode_t *vp)
526 {
527     return (nlm_has_sleep(vp));
528 }
529
530 /*
531  * *****
532  * Stuff needed by klmops?
533  */
```

```

*****
3641 Fri Aug 16 09:04:49 2013
new/usr/src/uts/common/klm/klmops.c
195 Need replacement for nfs/lockd+klm
*****
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 is of the CDDL is also available via the Internet
9  * at http://www.illumos.org/license/CDDL.
10 */

12 /*
13  * Copyright 2011 Nexenta Systems, Inc. All rights reserved.
14 */

16 /*
17  * NFS Lock Manager, client-side
18  * Note: depends on (links with) klmmmod
19  *
20  * This file contains all the external entry points of klmops.
21  * Basically, this is the "glue" to the BSD nlm code.
22 */

24 #include <sys/types.h>
25 #include <sys/errno.h>
26 #include <sys/modctl.h>
27 #include <sys/flock.h>

29 #include <nfs/lm.h>
30 #include <rpcsvc/nlm_prot.h>
31 #include "nlm_impl.h"

34 static struct modlmisc modlmisc = {
35     &mod_miscops, "lock mgr calls"
36 };

38 static struct modlinkage modlinkage = {
39     MODREV_1, &modlmisc, NULL
40 };

44 /*
45  * *****
46  * module init, fini, info
47  */
48 int
49 _init()
50 {
51     return (mod_install(&modlinkage));
52 }

54 int
55 _fini()
56 {
57     /* Don't unload. */
58     return (EBUSY);
59 }

61 int

```

```

62 _info(struct modinfo *modinfop)
63 {
64     return (mod_info(&modlinkage, modinfop));
65 }

69 /*
70  * *****
71  * Stubs listed in modstubs.s
72  * These are called from fs/nfs
73 */

75 /*
76  * NFSv2 lock/unlock. Called by nfs_frlock()
77  * Uses NLM version 1 (NLM_VERS)
78 */
79 int
80 lm_frlock(struct vnode *vp, int cmd, struct flock64 *flk, int flags,
81     u_offset_t off, struct cred *cr, struct netobj *fh,
82     struct flk_callback *flcb)
83 {
84     return (nlm_frlock(vp, cmd, flk, flags, off,
85         cr, fh, flcb, NLM_VERS));
86 }

88 /*
89  * NFSv3 lock/unlock. Called by nfs3_frlock()
90  * Uses NLM version 4 (NLM4_VERS)
91 */
92 int
93 lm4_frlock(struct vnode *vp, int cmd, struct flock64 *flk, int flags,
94     u_offset_t off, struct cred *cr, struct netobj *fh,
95     struct flk_callback *flcb)
96 {
97     int err;
98     err = nlm_frlock(vp, cmd, flk, flags, off,
99         cr, fh, flcb, NLM4_VERS);
100     return (err);
101 }

103 /*
104  * NFSv2 shrlk/unshrlk. See nfs_shrlock
105  * Uses NLM version 3 (NLM_VERSX)
106 */
107 int
108 lm_shrlock(struct vnode *vp, int cmd,
109     struct shrlock *shr, int flags, struct netobj *fh)
110 {
111     return (nlm_shrlock(vp, cmd, shr, flags, fh, NLM_VERSX));
112 }

114 /*
115  * NFSv3 shrlk/unshrlk. See nfs3_shrlock
116  * Uses NLM version 4 (NLM4_VERS)
117 */
118 int
119 lm4_shrlock(struct vnode *vp, int cmd,
120     struct shrlock *shr, int flags, struct netobj *fh)
121 {
122     return (nlm_shrlock(vp, cmd, shr, flags, fh, NLM4_VERS));
123 }

125 /*
126  * Helper for lm_frlock, lm4_frlock, nfs_lockrelease
127  * After getting a lock from a remote lock manager,

```



```
128 * register the lock locally.
129 */
130 void
131 lm_register_lock_locally(struct vnode *vp, struct lm_sysid *ls,
132     struct flock64 *flk, int flags, u_offset_t offset)
133 {
134     nlm_register_lock_locally(vp, (struct nlm_host *)ls,
135     flk, flags, offset);
136 }
137
138 /*
139 * Old RPC service dispatch functions, no longer used.
140 * Here only to satisfy modstubs.s references.
141 */
142 void
143 lm_nlm_dispatch(struct svc_req *req, SVCXPRT *xprt)
144 {
145     _NOTE(ARGUNUSED(req, xprt))
146 }
147
148 void
149 lm_nlm4_dispatch(struct svc_req *req, SVCXPRT *xprt)
150 {
151     _NOTE(ARGUNUSED(req, xprt))
152 }
153
154 /*
155 * Old internal functions used for reclaiming locks
156 * our NFS client holds after some server restarts.
157 * The new NLM code does this differently, so these
158 * are here only to satisfy modstubs.s references.
159 */
160 void
161 lm_nlm_reclaim(struct vnode *vp, struct flock64 *flkp)
162 {
163     _NOTE(ARGUNUSED(vp, flkp))
164 }
165
166 void
167 lm_nlm4_reclaim(struct vnode *vp, struct flock64 *flkp)
168 {
169     _NOTE(ARGUNUSED(vp, flkp))
170 }
```

1260 Fri Aug 16 09:04:49 2013

new/usr/src/uts/common/klm/mapfile-mod

195 Need replacement for nfs/lockd+klm

```
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 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 is of the CDDL is also available via the Internet
9 # at http://www.illumos.org/license/CDDL.
10 #
11 #
12 #
13 # Copyright 2011 Nexenta Systems, Inc. All rights reserved.
14 #
```

17 \$mapfile_version 2

```
19 SYMBOL_SCOPE {
20     global:
21     # loadable module linkage
22         _fini;
23         _info;
24         _init;
25     # These are all the symbols referenced in ml/modstubs.s
26     # If we want to remain a drop-in replacment for the old
27     # (closed source) klm, we need to define all of these.
28         lm_alloc_sysidt;
29         lm_cprresume;
30         lm_cprsuspend;
31         lm_free_config;
32         lm_free_sysidt;
33         lm_get_sysidt;
34         lm_global_nlmid;
35         lm_has_sleep;
36         lm_rel_sysidt;
37         lm_remove_file_locks;
38         lm_safelock;
39         lm_safemap;
40         lm_set_nlmid_flk;
41         lm_shutdown;
42         lm_svc;
43         lm_sysidt;
44         lm_unexport;
45         lm_vp_active;
46     # The following three functions are not mentioned in modstubs.s
47     # files, because they are not an entry points to KLM. They
48     # are called from klmops only.
49         nlm_frlock;
50         nlm_register_lock_locally;
51         nlm_shrlock;
52
53     local:
54         *;
55 };
```

new/usr/src/uts/common/klm/mapfile-ops

1

898 Fri Aug 16 09:04:50 2013

new/usr/src/uts/common/klm/mapfile-ops

195 Need replacement for nfs/lockd+klm

```
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 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 is of the CDDL is also available via the Internet
9 # at http://www.illumos.org/license/CDDL.
10 #
11 #
12 #
13 # Copyright 2011 Nexenta Systems, Inc. All rights reserved.
14 #
```

```
17 $mapfile_version 2
```

```
19 SYMBOL_SCOPE {
20     global:
```

```
21 # loadable module linkage
```

```
22     _fini;
```

```
23     _info;
```

```
24     _init;
```

```
25 # These are all the symbols referenced in ml/modstubs.s
```

```
26 # If we want to remain a drop-in replacment for the old
```

```
27 # (closed source) klm, we need to define all of these.
```

```
29         lm4_frlock;
```

```
30         lm4_shrlock;
```

```
31         lm_frlock;
```

```
32         lm_nlm4_dispatch;
```

```
33         lm_nlm4_reclaim;
```

```
34         lm_nlm_dispatch;
```

```
35         lm_nlm_reclaim;
```

```
36         lm_register_lock_locally;
```

```
38     local:
```

```
39         *;
```

```
40 };
```

```

*****
38688 Fri Aug 16 09:04:51 2013
new/usr/src/uts/common/klm/nlm_client.c
195 Need replacement for nfs/lockd+klm
*****
1 /*
2  * Copyright (c) 2008 Isilon Inc http://www.isilon.com/
3  * Authors: Doug Rabson <df@rabson.org>
4  * Developed with Red Inc: Alfred Perlstein <alfred@freebsd.org>
5  *
6  * Redistribution and use in source and binary forms, with or without
7  * modification, are permitted provided that the following conditions
8  * are met:
9  * 1. Redistributions of source code must retain the above copyright
10 * notice, this list of conditions and the following disclaimer.
11 * 2. Redistributions in binary form must reproduce the above copyright
12 * notice, this list of conditions and the following disclaimer in the
13 * documentation and/or other materials provided with the distribution.
14 *
15 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
16 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
17 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
18 * ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
19 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
20 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
21 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
22 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
23 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
24 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
25 * SUCH DAMAGE.
26 */

28 /*
29  * Copyright 2011 Nexenta Systems, Inc. All rights reserved.
30  * Copyright (c) 2012 by Delphix. All rights reserved.
31 */

33 /*
34  * Client-side support for (NFS) VOP_FRLOCK, VOP_SHRLOCK.
35  * (called via klmops.c: lm_frlock, lm4_frlock)
36  *
37  * Source code derived from FreeBSD nlm_advlock.c
38 */

40 #include <sys/param.h>
41 #include <sys/fcntl.h>
42 #include <sys/lock.h>
43 #include <sys/flock.h>
44 #include <sys/mount.h>
45 #include <sys/mutex.h>
46 #include <sys/proc.h>
47 #include <sys/share.h>
48 #include <sys/syslog.h>
49 #include <sys/system.h>
50 #include <sys/unistd.h>
51 #include <sys/vnode.h>
52 #include <sys/queue.h>
53 #include <sys/sdt.h>
54 #include <netinet/in.h>

56 #include <fs/fs_subr.h>
57 #include <rpcsvc/nlm_prot.h>

59 #include <nfs/nfs.h>
60 #include <nfs/nfs_clnt.h>
61 #include <nfs/export.h>

```

```

62 #include <nfs/rnode.h>
63 #include <nfs/lm.h>

65 #include "nlm_impl.h"

67 /* Extra flags for nlm_call_lock() - xflags */
68 #define NLM_X_RECLAIM 1
69 #define NLM_X_BLOCKING 2

71 /*
72  * Max. number of retries nlm_call_cancel() does
73  * when NLM server is in grace period or doesn't
74  * respond correctly.
75 */
76 #define NLM_CANCEL_NRETRS 5

78 /*
79  * Determines whether given lock "flp" is safe.
80  * The lock is considered to be safe when it
81  * acquires the whole file (i.e. its start
82  * and len are zeroes).
83 */
84 #define NLM_FLOCK_IS_SAFE(fl) \
85     ((fl)->l_start == 0 && (fl)->l_len == 0)

87 static volatile uint32_t nlm_xid = 1;

89 static int nlm_init_fh_by_vp(vnode_t *, struct netobj *, rpcvers_t *);
90 static int nlm_map_status(nlm4_stats);
91 static int nlm_map_clnt_stat(enum clnt_stat);
92 static void nlm_send_siglost(pid_t);

94 static int nlm_frlock_getlk(struct nlm_host *, vnode_t *,
95     struct flock64 *, int, u_offset_t, struct netobj *, int);

97 static int nlm_frlock_setlk(struct nlm_host *, vnode_t *,
98     struct flock64 *, int, u_offset_t, struct netobj *,
99     struct flk_callback *, int, bool_t);

101 static int nlm_reclaim_lock(struct nlm_host *, vnode_t *,
102     struct flock64 *, int32_t);

104 static void nlm_init_lock(struct nlm4_lock *,
105     const struct flock64 *, struct netobj *,
106     struct nlm_owner_handle *);

108 static int nlm_call_lock(vnode_t *, struct flock64 *,
109     struct nlm_host *, struct netobj *,
110     struct flk_callback *, int, int);
111 static int nlm_call_unlock(struct flock64 *, struct nlm_host *,
112     struct netobj *, int);
113 static int nlm_call_test(struct flock64 *, struct nlm_host *,
114     struct netobj *, int);
115 static int nlm_call_cancel(struct nlm4_lockargs *,
116     struct nlm_host *, int);

118 static int nlm_local_getlk(vnode_t *, struct flock64 *, int);
119 static int nlm_local_setlk(vnode_t *, struct flock64 *, int);
120 static void nlm_local_cancel(vnode_t *, struct flock64 *);

122 static void nlm_init_share(struct nlm4_share *,
123     const struct shrlock *, struct netobj *);

125 static int nlm_call_share(struct shrlock *, struct nlm_host *,
126     struct netobj *, int, int);
127 static int nlm_call_unshare(struct shrlock *, struct nlm_host *,

```

```

128     struct netobj *, int);
129 static int nlm_reclaim_share(struct nlm_host *, vnode_t *,
130     struct shrlock *, uint32_t);
131 static int nlm_local_shrlock(vnode_t *, struct shrlock *, int, int);
132 static void nlm_local_shrcancel(vnode_t *, struct shrlock *);

134 /*
135  * Reclaim locks/shares acquired by the client side
136  * on the given server represented by hostp.
137  * The function is called from a dedicated thread
138  * when server reports us that it's entered grace
139  * period.
140  */
141 void
142 nlm_reclaim_client(struct nlm_globals *g, struct nlm_host *hostp)
143 {
144     int32_t state;
145     int error, sysid;
146     struct locklist *llp_head, *llp;
147     struct nlm_shres *nsp_head, *nsp;
148     bool_t restart;

150     sysid = hostp->nh_sysid | LM_SYSID_CLIENT;
151     do {
152         error = 0;
153         restart = FALSE;
154         state = nlm_host_get_state(hostp);

156         DTRACE_PROBE3(reclaim_iter, struct nlm_globals *, g,
157             struct nlm_host *, hostp, int, state);

159         /*
160          * We cancel all sleeping locks that were
161          * done by the host, because we don't allow
162          * reclamation of sleeping locks. The reason
163          * we do this is that allowing of sleeping locks
164          * reclamation can potentially break locks recovery
165          * order.
166          *
167          * Imagine that we have two client machines A and B
168          * and an NLM server machine. A adds a non sleeping
169          * lock to the file F and acquires this file. Machine
170          * B in its turn adds sleeping lock to the file
171          * F and blocks because F is already acquired by
172          * the machine A. Then server crashes and after the
173          * reboot it notifies its clients about the crash.
174          * If we would allow sleeping locks reclamation,
175          * there would be possible that machine B recovers
176          * its lock faster than machine A (by some reason).
177          * So that B acquires the file F after server crash and
178          * machine A (that by some reason recovers slower) fails
179          * to recover its non sleeping lock. Thus the original
180          * locks order becomes broken.
181          */
182         nlm_host_cancel_slocks(g, hostp);

184         /*
185          * Try to reclaim all active locks we have
186          */
187         llp_head = llp = flk_get_active_locks(sysid, NOPID);
188         while (llp != NULL) {
189             error = nlm_reclaim_lock(hostp, llp->ll_vp,
190                 &llp->ll_flock, state);

192             if (error == 0) {
193                 llp = llp->ll_next;

```

```

194         continue;
195     } else if (error == ERESTART) {
196         restart = TRUE;
197         break;
198     } else {
199         /*
200          * Critical error occurred, the lock
201          * can not be recovered, just take it away.
202          */
203         nlm_local_cancelk(llp->ll_vp, &llp->ll_flock);
204     }

206     llp = llp->ll_next;
207 }

209 flk_free_locklist(llp_head);
210 if (restart) {
211     /*
212      * Lock reclamation function reported us that
213      * the server state was changed (again), so
214      * try to repeat the whole reclamation process.
215      */
216     continue;
217 }

219 nsp_head = nsp = nlm_get_active_shres(hostp);
220 while (nsp != NULL) {
221     error = nlm_reclaim_share(hostp, nsp->ns_vp,
222         nsp->ns_shr, state);

224     if (error == 0) {
225         nsp = nsp->ns_next;
226         continue;
227     } else if (error == ERESTART) {
228         break;
229     } else {
230         /* Failed to reclaim share */
231         nlm_shres_untrack(hostp, nsp->ns_vp,
232             nsp->ns_shr);
233         nlm_local_shrcancel(nsp->ns_vp,
234             nsp->ns_shr);
235     }

237     nsp = nsp->ns_next;
238 }

240     nlm_free_shrlist(nsp_head);
241 } while (state != nlm_host_get_state(hostp));
242 }

244 /*
245  * nlm_frlock --
246  *     NFS advisory byte-range locks.
247  *     Called in klmops.c
248  *
249  * Note that the local locking code (os/flock.c) is used to
250  * keep track of remote locks granted by some server, so we
251  * can reclaim those locks after a server restarts. We can
252  * also sometimes use this as a cache of lock information.
253  *
254  * Was: nlm_advlock()
255  */
256 /* ARGSUSED */
257 int
258 nlm_frlock(struct vnode *vp, int cmd, struct flock64 *flkp,
259     int flags, u_offset_t offset, struct cred *crp,

```

```

260     struct netobj *fhp, struct flk_callback *flcb, int vers)
261 {
262     mntinfo_t *mi;
263     servinfo_t *sv;
264     const char *netid;
265     struct nlm_host *hostp;
266     int error;
267     struct nlm_globals *g;
268
269     mi = VTOMI(vp);
270     sv = mi->mi_curr_serv;
271
272     netid = nlm_knc_to_netid(sv->sv_knconf);
273     if (netid == NULL) {
274         NLM_ERR("nlm_frlock: unknown NFS netid");
275         return (ENOSYS);
276     }
277
278     g = zone_getspecific(nlm_zone_key, curzone);
279     hostp = nlm_host_findcreate(g, sv->sv_hostname, netid, &sv->sv_addr);
280     if (hostp == NULL)
281         return (ENOSYS);
282
283     /*
284      * Purge cached attributes in order to make sure that
285      * future calls of convoff()/VOP_GETATTR() will get the
286      * latest data.
287      */
288     if (flkp->l_whence == SEEK_END)
289         PURGE_ATTRCACHE(vp);
290
291     /* Now flk0 is the zero-based lock request. */
292     switch (cmd) {
293     case F_GETLK:
294         error = nlm_frlock_getlk(hostp, vp, flkp, flags,
295                                 offset, fhp, vers);
296         break;
297
298     case F_SETLK:
299     case F_SETLKW:
300         error = nlm_frlock_setlk(hostp, vp, flkp, flags,
301                                 offset, fhp, flcb, vers, (cmd == F_SETLKW));
302         if (error == 0)
303             nlm_host_monitor(g, hostp, 0);
304         break;
305
306     default:
307         error = EINVAL;
308         break;
309     }
310
311     nlm_host_release(g, hostp);
312     return (error);
313 }
314
315 static int
316 nlm_frlock_getlk(struct nlm_host *hostp, vnode_t *vp,
317                 struct flock64 *flkp, int flags, u_offset_t offset,
318                 struct netobj *fhp, int vers)
319 {
320     struct flock64 flk0;
321     int error;
322
323     /*
324      * Check local (cached) locks first.
325      * If we find one, no need for RPC.

```

```

326     /*
327     flk0 = *flkp;
328     flk0.l_pid = curproc->p_pid;
329     error = nlm_local_getlk(vp, &flk0, flags);
330     if (error != 0)
331         return (error);
332     if (flk0.l_type != F_UNLCK) {
333         *flkp = flk0;
334         return (0);
335     }
336
337     /* Not found locally. Try remote. */
338     flk0 = *flkp;
339     flk0.l_pid = curproc->p_pid;
340     error = convoff(vp, &flk0, 0, (offset_t)offset);
341     if (error != 0)
342         return (error);
343
344     error = nlm_call_test(&flk0, hostp, fhp, vers);
345     if (error != 0)
346         return (error);
347
348     if (flk0.l_type == F_UNLCK) {
349         /*
350          * Update the caller's *flkp with information
351          * on the conflicting lock (or lack thereof).
352          */
353         flkp->l_type = F_UNLCK;
354     } else {
355         /*
356          * Found a conflicting lock. Set the
357          * caller's *flkp with the info, first
358          * converting to the caller's whence.
359          */
360         (void) convoff(vp, &flk0, flkp->l_whence, (offset_t)offset);
361         *flkp = flk0;
362     }
363
364     return (0);
365 }
366
367 static int
368 nlm_frlock_setlk(struct nlm_host *hostp, vnode_t *vp,
369                 struct flock64 *flkp, int flags, u_offset_t offset,
370                 struct netobj *fhp, struct flk_callback *flcb,
371                 int vers, bool_t do_block)
372 {
373     int error, xflags;
374
375     error = convoff(vp, flkp, 0, (offset_t)offset);
376     if (error != 0)
377         return (error);
378
379     /*
380     * NFS v2 clients should not request locks where any part
381     * of the lock range is beyond 0xffffffff. The NFS code
382     * checks that (see nfs_frlock, flk_check_lock_data), but
383     * as that's outside this module, let's check here too.
384     * This check ensures that we will be able to convert this
385     * lock request into 32-bit form without change, and that
386     * (more importantly) when the granted call back arrives,
387     * it's unchanged when converted back into 64-bit form.
388     * If this lock range were to change in any way during
389     * either of those conversions, the "granted" call back
390     * from the NLM server would not find our sleeping lock.
391     */

```

```

392     if (vers < NLM4_VERS) {
393         if (flkp->l_start > MAX_UOFF32 ||
394             flkp->l_start + flkp->l_len > MAX_UOFF32 + 1)
395             return (EINVAL);
396     }
397
398     /*
399     * Fill in l_sysid for the local locking calls.
400     * Also, let's not trust the caller's l_pid.
401     */
402     flkp->l_sysid = hostp->nh_sysid | LM_SYSID_CLIENT;
403     flkp->l_pid = curproc->p_pid;
404
405     if (flkp->l_type == F_UNLCK) {
406         /*
407         * Purge local (cached) lock information first,
408         * then clear the remote lock.
409         */
410         (void) nlm_local_setlk(vp, flkp, flags);
411         error = nlm_call_unlock(flkp, hostp, fhp, vers);
412
413         return (error);
414     }
415
416     if (!do_block) {
417         /*
418         * This is a non-blocking "set" request,
419         * so we can check locally first, and
420         * sometimes avoid an RPC call.
421         */
422         struct flock64 flk0;
423
424         flk0 = *flkp;
425         error = nlm_local_getlk(vp, &flk0, flags);
426         if (error != 0 && flk0.l_type != F_UNLCK) {
427             /* Found a conflicting lock. */
428             return (EAGAIN);
429         }
430
431         xflags = 0;
432     } else {
433         xflags = NLM_X_BLOCKING;
434     }
435
436     nfs_add_locking_id(vp, curproc->p_pid, RLMPID_PID,
437         (char *)&curproc->p_pid, sizeof(pid_t));
438
439     error = nlm_call_lock(vp, flkp, hostp, fhp, flcb, vers, xflags);
440     if (error != 0)
441         return (error);
442
443     /*
444     * Save the lock locally. This should not fail,
445     * because the server is authoritative about locks
446     * and it just told us we have the lock!
447     */
448     error = nlm_local_setlk(vp, flkp, flags);
449     if (error != 0) {
450         /*
451         * That's unexpected situation. Just ignore the error.
452         */
453         NLM_WARN("nlm_frlock_setlk: Failed to set local lock. "
454             "[err=%d]\n", error);
455         error = 0;
456     }

```

```

458         return (error);
459     }
460
461     /*
462     * Cancel all client side remote locks/shares on the
463     * given host. Report to the processes that own
464     * cancelled locks that they are removed by force
465     * by sending SIGLOST.
466     */
467     void
468     nlm_client_cancel_all(struct nlm_globals *g, struct nlm_host *hostp)
469     {
470         struct locklist *llp_head, *llp;
471         struct nlm_shres *nsp_head, *nsp;
472         struct netobj lm_fh;
473         rpcvers_t vers;
474         int error, sysid;
475
476         sysid = hostp->nh_sysid | LM_SYSID_CLIENT;
477         nlm_host_cancel_slocks(g, hostp);
478
479         /*
480         * Destroy all active locks
481         */
482         llp_head = llp = flk_get_active_locks(sysid, NOPID);
483         while (llp != NULL) {
484             llp->ll_flock.l_type = F_UNLCK;
485
486             error = nlm_init_fh_by_vp(llp->ll_vp, &lm_fh, &vers);
487             if (error == 0)
488                 (void) nlm_call_unlock(&llp->ll_flock, hostp,
489                     &lm_fh, vers);
490
491             nlm_local_cancelk(llp->ll_vp, &llp->ll_flock);
492             llp = llp->ll_next;
493         }
494
495         flk_free_locklist(llp_head);
496
497         /*
498         * Destroy all active share reservations
499         */
500         nsp_head = nsp = nlm_get_active_shres(hostp);
501         while (nsp != NULL) {
502             error = nlm_init_fh_by_vp(nsp->ns_vp, &lm_fh, &vers);
503             if (error == 0)
504                 (void) nlm_call_unshare(nsp->ns_shr, hostp,
505                     &lm_fh, vers);
506
507             nlm_local_shrcancel(nsp->ns_vp, nsp->ns_shr);
508             nlm_shres_untrack(hostp, nsp->ns_vp, nsp->ns_shr);
509             nsp = nsp->ns_next;
510         }
511
512         nlm_free_shrlist(nsp_head);
513     }
514
515     /*
516     * The function determines whether the lock "fl" can
517     * be safely applied to the file vnode "vp" corresponds to.
518     * The lock can be "safely" applied if all the conditions
519     * above are held:
520     * - It's not a mandatory lock
521     * - The vnode wasn't mapped by anyone
522     * - The vnode was mapped, but it hasn't any locks on it.
523     * - The vnode was mapped and all locks it has occupies

```

```

524 *   the whole file.
525 */
526 int
527 nlm_safelock(vnode_t *vp, const struct flock64 *fl, cred_t *cr)
528 {
529     rnode_t *rp = VTOR(vp);
530     struct vattr va;
531     int err;
532
533     if ((rp->r_mapcnt > 0) && (fl->l_start != 0 || fl->l_len != 0))
534         return (0);
535
536     va.va_mask = AT_MODE;
537     err = VOP_GETATTR(vp, &va, 0, cr, NULL);
538     if (err != 0)
539         return (0);
540
541     /* NLM4 doesn't allow mandatory file locking */
542     if (MANDLOCK(vp, va.va_mode))
543         return (0);
544
545     return (1);
546 }
547
548 /*
549 * The function determines whether it's safe to map
550 * a file corresponding to vnode vp.
551 * The mapping is considered to be "safe" if file
552 * either has no any locks on it or all locks it
553 * has occupy the whole file.
554 */
555 int
556 nlm_safemap(const vnode_t *vp)
557 {
558     struct locklist *llp, *llp_next;
559     struct nlm_slock *nslp;
560     struct nlm_globals *g;
561     int safe = 1;
562
563     /* Check active locks at first */
564     llp = flk_active_locks_for_vp(vp);
565     while (llp != NULL) {
566         if ((llp->ll_vp == vp) &&
567             !NLM_FLOCK_IS_SAFE(&llp->ll_flock))
568             safe = 0;
569
570         llp_next = llp->ll_next;
571         VN_RELE(llp->ll_vp);
572         kmem_free(llp, sizeof (*llp));
573         llp = llp_next;
574     }
575     if (!safe)
576         return (safe);
577
578     /* Then check sleeping locks if any */
579     g = zone_getspecific(nlm_zone_key, curzone);
580     mutex_enter(&g->lock);
581     TAILQ_FOREACH(nslp, &g->nlm_slocks, nsl_link) {
582         if (nslp->nsl_state == NLM_SL_BLOCKED &&
583             nslp->nsl_vp == vp &&
584             (nslp->nsl_lock.l_offset != 0 ||
585              nslp->nsl_lock.l_len != 0)) {
586             safe = 0;
587             break;
588         }
589     }

```

```

591     mutex_exit(&g->lock);
592     return (safe);
593 }
594
595 int
596 nlm_has_sleep(const vnode_t *vp)
597 {
598     struct nlm_globals *g;
599     struct nlm_slock *nslp;
600     int has_slocks = FALSE;
601
602     g = zone_getspecific(nlm_zone_key, curzone);
603     mutex_enter(&g->lock);
604     TAILQ_FOREACH(nslp, &g->nlm_slocks, nsl_link) {
605         if (nslp->nsl_state == NLM_SL_BLOCKED &&
606             nslp->nsl_vp == vp) {
607             has_slocks = TRUE;
608             break;
609         }
610     }
611
612     mutex_exit(&g->lock);
613     return (has_slocks);
614 }
615
616 void
617 nlm_register_lock_locally(struct vnode *vp, struct nlm_host *hostp,
618     struct flock64 *flk, int flags, u_offset_t offset)
619 {
620     int sysid = 0;
621
622     if (hostp != NULL) {
623         sysid = hostp->nh_sysid | LM_SYSID_CLIENT;
624     }
625
626     flk->l_sysid = sysid;
627     (void) convoff(vp, flk, 0, (offset_t)offset);
628     (void) nlm_local_setlk(vp, flk, flags);
629 }
630
631 /*
632 * The BSD code had functions here to "reclaim" (destroy)
633 * remote locks when a vnode is being forcibly destroyed.
634 * We just keep vnodes around until statd tells us the
635 * client has gone away.
636 */
637
638 static int
639 nlm_reclaim_lock(struct nlm_host *hostp, vnode_t *vp,
640     struct flock64 *flp, int32_t orig_state)
641 {
642     struct netobj lm_fh;
643     int error, state;
644     rpcvers_t vers;
645
646     /*
647      * If the remote NSM state changes during recovery, the host
648      * must have rebooted a second time. In that case, we must
649      * restart the recovery.
650      */
651     state = nlm_host_get_state(hostp);
652     if (state != orig_state)
653         return (ERESTART);

```



```

656     error = nlm_init_fh_by_vp(vp, &lm_fh, &vers);
657     if (error != 0)
658         return (error);

660     return (nlm_call_lock(vp, flp, hostp, &lm_fh,
661         NULL, vers, NLM_X_RECLAIM));
662 }

664 /*
665  * Get local lock information for some NFS server.
666  *
667  * This gets (checks for) a local conflicting lock.
668  * Note: Modifies passed flock, if a conflict is found,
669  * but the caller expects that.
670  */
671 static int
672 nlm_local_getlk(vnode_t *vp, struct flock64 *fl, int flags)
673 {
674     VERIFY(fl->l_whence == SEEK_SET);
675     return (relock(vp, fl, 0, flags, 0, NULL));
676 }

678 /*
679  * Set local lock information for some NFS server.
680  *
681  * Called after a lock request (set or clear) succeeded. We record the
682  * details in the local lock manager. Note that since the remote
683  * server has granted the lock, we can be sure that it doesn't
684  * conflict with any other locks we have in the local lock manager.
685  *
686  * Since it is possible that host may also make NLM client requests to
687  * our NLM server, we use a different sysid value to record our own
688  * client locks.
689  *
690  * Note that since it is possible for us to receive replies from the
691  * server in a different order than the locks were granted (e.g. if
692  * many local threads are contending for the same lock), we must use a
693  * blocking operation when registering with the local lock manager.
694  * We expect that any actual wait will be rare and short hence we
695  * ignore signals for this.
696  */
697 static int
698 nlm_local_setlk(vnode_t *vp, struct flock64 *fl, int flags)
699 {
700     VERIFY(fl->l_whence == SEEK_SET);
701     return (relock(vp, fl, SETFLCK, flags, 0, NULL));
702 }

704 /*
705  * Cancel local lock and send send SIGLOST signal
706  * to the lock owner.
707  *
708  * NOTE: modifies flp
709  */
710 static void
711 nlm_local_cancelk(vnode_t *vp, struct flock64 *flp)
712 {
713     flp->l_type = F_UNLCK;
714     (void) nlm_local_setlk(vp, flp, FREAD | FWRITE);
715     nlm_send_siglost(fl->l_pid);
716 }

718 /*
719  * Do NLM LOCK call.
720  * Was: nlm_setlock()
721  */

```

```

722  * NOTE: nlm_call_lock() function should care about locking/unlocking
723  * of rnode->r_lkserlock which should be released before nlm_call_lock()
724  * sleeps on waiting lock and acquired when it wakes up.
725  */
726 static int
727 nlm_call_lock(vnode_t *vp, struct flock64 *flp,
728     struct nlm_host *hostp, struct netobj *fhp,
729     struct flk_callback *flcb, int vers, int xflags)
730 {
731     struct nlm4_lockargs args;
732     struct nlm_owner_handle oh;
733     struct nlm_globals *g;
734     rnode_t *rnp = VTOR(vp);
735     struct nlm_slock *nslp = NULL;
736     uint32_t xid;
737     int error = 0;

739     bzero(&args, sizeof (args));
740     g = zone_getspecific(nlm_zone_key, curzone);
741     nlm_init_lock(&args.alock, flp, fhp, &oh);

743     args.exclusive = (flp->l_type == F_WRLCK);
744     args.reclaim = xflags & NLM_X_RECLAIM;
745     args.state = g->nsm_state;
746     args.cookie.n_len = sizeof (xid);
747     args.cookie.n_bytes = (char *)&xid;

749     oh.oh_sysid = hostp->nh_sysid;
750     xid = atomic_inc_32_nv(&nmlm_xid);

752     if (xflags & NLM_X_BLOCKING) {
753         args.block = TRUE;
754         nslp = nlm_slock_register(g, hostp, &args.alock, vp);
755     }

757     for (;;) {
758         nlm_rpc_t *rpcp;
759         enum clnt_stat stat;
760         struct nlm4_res res;
761         enum nlm4_stats nlm_err;

763         error = nlm_host_get_rpc(hostp, vers, &rpcp);
764         if (error != 0) {
765             error = ENOLCK;
766             goto out;
767         }

769         bzero(&res, sizeof (res));
770         stat = nlm_lock_rpc(&args, &res, rpcp->nr_handle, vers);
771         nlm_host_rele_rpc(hostp, rpcp);

773         error = nlm_map_clnt_stat(stat);
774         if (error != 0) {
775             if (error == EAGAIN)
776                 continue;

778             goto out;
779         }

781         DTRACE_PROBE1(lock_res, enum nlm4_stats, res.stat.stat);
782         nlm_err = res.stat.stat;
783         xdr_free((xdrproc_t)xdr_nlm4_res, (void *)&res);
784         if (nlm_err == nlm4_denied_grace_period) {
785             if (args.reclaim) {
786                 error = ENOLCK;
787                 goto out;

```

```

788     }
790     error = nlm_host_wait_grace(hostp);
791     if (error != 0)
792         goto out;
794     continue;
795 }
797 switch (nlm_err) {
798 case nlm4_granted:
799 case nlm4_blocked:
800     error = 0;
801     break;
803 case nlm4_denied:
804     if (nslp != NULL) {
805         NLM_WARN("nlm_call_lock: got nlm4_denied for "
806                "blocking lock\n");
807     }
809     error = EAGAIN;
810     break;
812 default:
813     error = nlm_map_status(nlm_err);
814 }
816 /*
817  * If we deal with either non-blocking lock or
818  * with a blocking locks that wasn't blocked on
819  * the server side (by some reason), our work
820  * is finished.
821  */
822 if (nslp == NULL
823     || nlm_err != nlm4_blocked
824     || error != 0)
825     goto out;
827 /*
828  * Before releasing the r_lkserlock of rnode, we should
829  * check whether the new lock is "safe". If it's not
830  * safe, disable caching for the given vnode. That is done
831  * for sleeping locks only that are waiting for a GRANT reply
832  * from the NLM server.
833  *
834  * NOTE: the vnode cache can be enabled back later if an
835  * unsafe lock will be merged with existent locks so that
836  * it will become safe. This condition is checked in the
837  * NFSv3 code (see nfs_lockcompletion).
838  */
839 if (!NLM_FLOCK_IS_SAFE(flp)) {
840     mutex_enter(&vp->v_lock);
841     vp->v_flag &= ~VNOCACHE;
842     mutex_exit(&vp->v_lock);
843 }
845 /*
846  * The server should call us back with a
847  * granted message when the lock succeeds.
848  * In order to deal with broken servers,
849  * lost granted messages, or server reboots,
850  * we will also re-try every few seconds.
851  *
852  * Note: We're supposed to call these
853  * flk_invoke_callbacks when blocking.

```

```

854     * Take care on rnode->r_lkserlock, we should
855     * release it before going to sleep.
856     */
857     (void) flk_invoke_callbacks(flcb, FLK_BEFORE_SLEEP);
858     nfs_rw_exit(&rnp->r_lkserlock);
860     error = nlm_slock_wait(g, nslp, g->retrans_tmo);
862     /*
863     * NFS expects that we return with rnode->r_lkserlock
864     * locked on write, lock it back.
865     *
866     * NOTE: nfs_rw_enter_sig() can be either interruptible
867     * or not. It depends on options of NFS mount. Here
868     * we're _always_ uninterruptible (independently of mount
869     * options), because nfs_frlock/nfs3_frlock expects that
870     * we return with rnode->r_lkserlock acquired. So we don't
871     * want our lock attempt to be interrupted by a signal.
872     */
873     (void) nfs_rw_enter_sig(&rnp->r_lkserlock, RW_WRITER, 0);
874     (void) flk_invoke_callbacks(flcb, FLK_AFTER_SLEEP);
876     if (error == 0) {
877         break;
878     } else if (error == EINTR) {
879         /*
880         * We need to call the server to cancel our
881         * lock request.
882         */
883         DTRACE_PROBE1(cancel_lock, int, error);
884         (void) nlm_call_cancel(&args, hostp, vers);
885         break;
886     } else {
887         /*
888         * Timeout happened, resend the lock request to
889         * the server. Well, we're a bit paranoid here,
890         * but keep in mind previous request could lost
891         * (especially with connectionless transport).
892         */
894         ASSERT(error == ETIMEDOUT);
895         continue;
896     }
897 }
899 /*
900  * We could disable the vnode cache for the given _sleeping_
901  * (condition: nslp != NULL) lock if it was unsafe. Normally,
902  * nfs_lockcompletion() function can enable the vnode cache
903  * back if the lock becomes safe after activation. But it
904  * will not happen if any error occurs on the locking path.
905  *
906  * Here we enable the vnode cache back if the error occurred
907  * and if there aren't any unsafe locks on the given vnode.
908  * Note that if error happened, sleeping lock was deregistered.
909  */
910 if (error != 0 && nslp != NULL && nlm_safemap(vp)) {
911     mutex_enter(&vp->v_lock);
912     vp->v_flag |= VNOCACHE;
913     mutex_exit(&vp->v_lock);
914 }
916 out:
917 if (nslp != NULL)
918     nlm_slock_unregister(g, nslp);

```

```

920     return (error);
921 }

923 /*
924  * Do NLM_CANCEL call.
925  * Helper for nlm_call_lock() error recovery.
926  */
927 static int
928 nlm_call_cancel(struct nlm4_lockargs *largs,
929                struct nlm_host *hostp, int vers)
930 {
931     nlm4_cancargs cargs;
932     uint32_t xid;
933     int error, retries;

935     bzero(&cargs, sizeof (cargs));

937     xid = atomic_inc_32_nv(&nlm_xid);
938     cargs.cookie.n_len = sizeof (xid);
939     cargs.cookie.n_bytes = (char *)&xid;
940     cargs.block = largs->block;
941     cargs.exclusive = largs->exclusive;
942     cargs.alock = largs->alock;

944     /*
945      * Unlike all other nlm_call_* functions, nlm_call_cancel
946      * doesn't spin forever until it gets reasonable response
947      * from NLM server. It makes limited number of retries and
948      * if server doesn't send a reasonable reply, it returns an
949      * error. It behaves like that because it's called from nlm_call_lock
950      * with blocked signals and thus it can not be interrupted from
951      * user space.
952      */
953     for (retries = 0; retries < NLM_CANCEL_NRETRS; retries++) {
954         nlm_rpc_t *rpcp;
955         enum clnt_stat stat;
956         struct nlm4_res res;

958         error = nlm_host_get_rpc(hostp, vers, &rpcp);
959         if (error != 0)
960             return (ENOLCK);

962         bzero(&res, sizeof (res));
963         stat = nlm_cancel_rpc(&cargs, &res, rpcp->nr_handle, vers);
964         nlm_host_rele_rpc(hostp, rpcp);

966         DTRACE_PROBE1(cancel__rloop_end, enum clnt_stat, stat);
967         error = nlm_map_clnt_stat(stat);
968         if (error != 0) {
969             if (error == EAGAIN)
970                 continue;

972             return (error);
973         }

975         DTRACE_PROBE1(cancel__res, enum nlm4_stats, res.stat.stat);
976         switch (res.stat.stat) {
977             /*
978              * There was nothing to cancel. We are going to go ahead
979              * and assume we got the lock.
980              */
981             case nlm_denied:
982                 /*
983                  * The server has recently rebooted. Treat this as a
984                  * successful cancellation.
985                  */

```

```

986         case nlm4_denied_grace_period:
987             /*
988              * We managed to cancel.
989              */
990             case nlm4_granted:
991                 error = 0;
992                 break;

994         default:
995             /*
996              * Broken server implementation. Can't really do
997              * anything here.
998              */
999             error = EIO;
1000             break;
1001     }

1003     xdr_free((xdrproc_t)xdr_nlm4_res, (void *)&res);
1004     break;
1005 }

1007     return (error);
1008 }

1010 /*
1011  * Do NLM_UNLOCK call.
1012  * Was: nlm_clearlock
1013  */
1014 static int
1015 nlm_call_unlock(struct flock64 *flp, struct nlm_host *hostp,
1016                struct netobj *fhp, int vers)
1017 {
1018     struct nlm4_unlockargs args;
1019     struct nlm_owner_handle oh;
1020     enum nlm4_stats nlm_err;
1021     uint32_t xid;
1022     int error;

1024     bzero(&args, sizeof (args));
1025     nlm_init_lock(&args.alock, flp, fhp, &oh);

1027     oh.oh_sysid = hostp->nh_sysid;
1028     xid = atomic_inc_32_nv(&nlm_xid);
1029     args.cookie.n_len = sizeof (xid);
1030     args.cookie.n_bytes = (char *)&xid;

1032     for (;;) {
1033         nlm_rpc_t *rpcp;
1034         struct nlm4_res res;
1035         enum clnt_stat stat;

1037         error = nlm_host_get_rpc(hostp, vers, &rpcp);
1038         if (error != 0)
1039             return (ENOLCK);

1041         bzero(&res, sizeof (res));
1042         stat = nlm_unlock_rpc(&args, &res, rpcp->nr_handle, vers);
1043         nlm_host_rele_rpc(hostp, rpcp);

1045         error = nlm_map_clnt_stat(stat);
1046         if (error != 0) {
1047             if (error == EAGAIN)
1048                 continue;

1050             return (error);
1051         }

```

```

1053         DTRACE_PROBE1(unlock_res, enum nlm4_stats, res.stat.stat);
1054         nlm_err = res.stat.stat;
1055         xdr_free((xdrproc_t)xdr_nlm4_res, (void *)&res);
1056         if (nlm_err == nlm4_denied_grace_period) {
1057             error = nlm_host_wait_grace(hostp);
1058             if (error != 0)
1059                 return (error);
1061         continue;
1062     }
1064     break;
1065 }
1067 /* special cases */
1068 switch (nlm_err) {
1069 case nlm4_denied:
1070     error = EINVAL;
1071     break;
1072 default:
1073     error = nlm_map_status(nlm_err);
1074     break;
1075 }
1077 return (error);
1078 }
1080 /*
1081  * Do NLM_TEST call.
1082  * Was: nlm_getlock()
1083  */
1084 static int
1085 nlm_call_test(struct flock64 *flp, struct nlm_host *hostp,
1086              struct netobj *fhp, int vers)
1087 {
1088     struct nlm4_testargs args;
1089     struct nlm4_holder h;
1090     struct nlm_owner_handle oh;
1091     enum nlm4_stats nlm_err;
1092     uint32_t xid;
1093     int error;
1095     bzero(&args, sizeof (args));
1096     nlm_init_lock(&args.alock, flp, fhp, &oh);
1098     args.exclusive = (flp->l_type == F_WRLCK);
1099     oh.oh_sysid = hostp->nh_sysid;
1100     xid = atomic_inc_32_nv(&nlm_xid);
1101     args.cookie.n_len = sizeof (xid);
1102     args.cookie.n_bytes = (char *)&xid;
1104     for (;;) {
1105         nlm_rpc_t *rpcp;
1106         struct nlm4_testres res;
1107         enum clnt_stat stat;
1109         error = nlm_host_get_rpc(hostp, vers, &rpcp);
1110         if (error != 0)
1111             return (ENOLCK);
1113         bzero(&res, sizeof (res));
1114         stat = nlm_test_rpc(&args, &res, rpcp->nr_handle, vers);
1115         nlm_host_rele_rpc(hostp, rpcp);
1117         error = nlm_map_clnt_stat(stat);

```

```

1118         if (error != 0) {
1119             if (error == EAGAIN)
1120                 continue;
1122             return (error);
1123         }
1125         DTRACE_PROBE1(test_res, enum nlm4_stats, res.stat.stat);
1126         nlm_err = res.stat.stat;
1127         bcopy(&res.stat.nlm4_testrply_u_holder, &h, sizeof (h));
1128         xdr_free((xdrproc_t)xdr_nlm4_testres, (void *)&res);
1129         if (nlm_err == nlm4_denied_grace_period) {
1130             error = nlm_host_wait_grace(hostp);
1131             if (error != 0)
1132                 return (error);
1134         continue;
1135     }
1137     break;
1138 }
1140 switch (nlm_err) {
1141 case nlm4_granted:
1142     flp->l_type = F_UNLCK;
1143     error = 0;
1144     break;
1146 case nlm4_denied:
1147     flp->l_start = h.l_offset;
1148     flp->l_len = h.l_len;
1149     flp->l_pid = h.svid;
1150     flp->l_type = (h.exclusive) ? F_WRLCK : F_RDLCK;
1151     flp->l_whence = SEEK_SET;
1152     flp->l_sysid = 0;
1153     error = 0;
1154     break;
1156 default:
1157     error = nlm_map_status(nlm_err);
1158     break;
1159 }
1161 return (error);
1162 }
1165 static void
1166 nlm_init_lock(struct nlm4_lock *lock,
1167              const struct flock64 *fl, struct netobj *fh,
1168              struct nlm_owner_handle *oh)
1169 {
1171     /* Caller converts to zero-base. */
1172     VERIFY(fl->l_whence == SEEK_SET);
1173     bzero(lock, sizeof (*lock));
1174     bzero(oh, sizeof (*oh));
1176     lock->caller_name = uts_nodename();
1177     lock->fh.n_len = fh->n_len;
1178     lock->fh.n_bytes = fh->n_bytes;
1179     lock->oh.n_len = sizeof (*oh);
1180     lock->oh.n_bytes = (void *)oh;
1181     lock->svid = fl->l_pid;
1182     lock->l_offset = fl->l_start;
1183     lock->l_len = fl->l_len;

```

```

1184 }
1186 /* ***** */
1188 int
1189 nlm_shrlock(struct vnode *vp, int cmd, struct shrlock *shr,
1190            int flags, struct netobj *fh, int vers)
1191 {
1192     struct shrlock shlk;
1193     mntinfo_t *mi;
1194     servinfo_t *sv;
1195     const char *netid;
1196     struct nlm_host *host = NULL;
1197     int error;
1198     struct nlm_globals *g;
1200     mi = VTOMI(vp);
1201     sv = mi->mi_curr_serv;
1203     netid = nlm_knc_to_netid(sv->sv_knconf);
1204     if (netid == NULL) {
1205         NLM_ERR("nlm_shrlock: unknown NFS netid\n");
1206         return (ENOSYS);
1207     }
1209     g = zone_getspecific(nlm_zone_key, curzone);
1210     host = nlm_host_findcreate(g, sv->sv_hostname, netid, &sv->sv_addr);
1211     if (host == NULL)
1212         return (ENOSYS);
1214     /*
1215      * Fill in s_sysid for the local locking calls.
1216      * Also, let's not trust the caller's l_pid.
1217      */
1218     shlk = *shr;
1219     shlk.s_sysid = host->nh_sysid | LM_SYSID_CLIENT;
1220     shlk.s_pid = curproc->p_pid;
1222     if (cmd == F_UNSHARE) {
1223         /*
1224          * Purge local (cached) share information first,
1225          * then clear the remote share.
1226          */
1227         (void) nlm_local_shrlock(vp, &shlk, cmd, flags);
1228         nlm_shres_untrack(host, vp, &shlk);
1229         error = nlm_call_unshare(&shlk, host, fh, vers);
1230         goto out;
1231     }
1233     nfs_add_locking_id(vp, curproc->p_pid, RLMLPL_OWNER,
1234                      shr->s_owner, shr->s_own_len);
1236     error = nlm_call_share(&shlk, host, fh, vers, FALSE);
1237     if (error != 0)
1238         goto out;
1240     /*
1241      * Save the share locally. This should not fail,
1242      * because the server is authoritative about shares
1243      * and it just told us we have the share reservation!
1244      */
1245     error = nlm_local_shrlock(vp, shr, cmd, flags);
1246     if (error != 0) {
1247         /*
1248          * Oh oh, we really don't expect an error here.
1249          */

```

```

1250         NLM_WARN("nlm_shrlock: set locally, err %d\n", error);
1251         error = 0;
1252     }
1254     nlm_shres_track(host, vp, &shlk);
1255     nlm_host_monitor(g, host, 0);
1257 out:
1258     nlm_host_release(g, host);
1260     return (error);
1261 }
1263 static int
1264 nlm_reclaim_share(struct nlm_host *hostp, vnode_t *vp,
1265                  struct shrlock *shr, uint32_t orig_state)
1266 {
1267     struct netobj lm_fh;
1268     int error, state;
1269     rpcvers_t vers;
1271     state = nlm_host_get_state(hostp);
1272     if (state != orig_state) {
1273         /*
1274          * It seems that NLN server rebooted while
1275          * we were busy with recovery.
1276          */
1277         return (ERESTART);
1278     }
1280     error = nlm_init_fh_by_vp(vp, &lm_fh, &vers);
1281     if (error != 0)
1282         return (error);
1284     return (nlm_call_share(shr, hostp, &lm_fh, vers, 1));
1285 }
1287 /*
1288  * Set local share information for some NFS server.
1289  *
1290  * Called after a share request (set or clear) succeeded. We record
1291  * the details in the local lock manager. Note that since the remote
1292  * server has granted the share, we can be sure that it doesn't
1293  * conflict with any other shares we have in the local lock manager.
1294  *
1295  * Since it is possible that host may also make NLN client requests to
1296  * our NLN server, we use a different sysid value to record our own
1297  * client shares.
1298  */
1299 int
1300 nlm_local_shrlock(vnode_t *vp, struct shrlock *shr, int cmd, int flags)
1301 {
1302     return (fs_shrlock(vp, cmd, shr, flags, CRED(), NULL));
1303 }
1305 static void
1306 nlm_local_shrcancel(vnode_t *vp, struct shrlock *shr)
1307 {
1308     (void) nlm_local_shrlock(vp, shr, F_UNSHARE, FREAD | FWRITE);
1309     nlm_send_siglost(shr->s_pid);
1310 }
1312 /*
1313  * Do NLN SHARE call.
1314  * Was: nlm_setshare()
1315  */

```

```

1316 static int
1317 nlm_call_share(struct shrlock *shr, struct nlm_host *host,
1318               struct netobj *fh, int vers, int reclaim)
1319 {
1320     struct nlm4_shareargs args;
1321     enum nlm4_stats nlm_err;
1322     uint32_t xid;
1323     int error;
1324
1325     bzero(&args, sizeof (args));
1326     nlm_init_share(&args.share, shr, fh);
1327
1328     args.reclaim = reclaim;
1329     xid = atomic_inc_32_nv(&nlm_xid);
1330     args.cookie.n_len = sizeof (xid);
1331     args.cookie.n_bytes = (char *)&xid;
1332
1333     for (;;) {
1334         nlm_rpc_t *rpcp;
1335         struct nlm4_sharereres res;
1336         enum clnt_stat stat;
1337
1338         error = nlm_host_get_rpc(host, vers, &rpcp);
1339         if (error != 0)
1340             return (ENOLCK);
1341
1342         bzero(&res, sizeof (res));
1343         stat = nlm_share_rpc(&args, &res, rpcp->nr_handle, vers);
1344         nlm_host_rele_rpc(host, rpcp);
1345
1346         error = nlm_map_clnt_stat(stat);
1347         if (error != 0) {
1348             if (error == EAGAIN)
1349                 continue;
1350
1351             return (error);
1352         }
1353
1354         DTRACE_PROBE1(share_res, enum nlm4_stats, res.stat);
1355         nlm_err = res.stat;
1356         xdr_free((xdrproc_t)xdr_nlm4_sharereres, (void *)&res);
1357         if (nlm_err == nlm4_denied_grace_period) {
1358             if (args.reclaim)
1359                 return (ENOLCK);
1360
1361             error = nlm_host_wait_grace(host);
1362             if (error != 0)
1363                 return (error);
1364
1365             continue;
1366         }
1367     }
1368     break;
1369 }
1370
1371 switch (nlm_err) {
1372 case nlm4_granted:
1373     error = 0;
1374     break;
1375 case nlm4_blocked:
1376 case nlm4_denied:
1377     error = EAGAIN;
1378     break;
1379 case nlm4_denied_nolocks:
1380 case nlm4_deadlck:

```

```

1382         error = ENOLCK;
1383         break;
1384     default:
1385         error = EINVAL;
1386         break;
1387     }
1388
1389     return (error);
1390 }
1391
1392 /*
1393  * Do NLM_UNSHARE call.
1394  */
1395 static int
1396 nlm_call_unshare(struct shrlock *shr, struct nlm_host *host,
1397                 struct netobj *fh, int vers)
1398 {
1399     struct nlm4_shareargs args;
1400     enum nlm4_stats nlm_err;
1401     uint32_t xid;
1402     int error;
1403
1404     bzero(&args, sizeof (args));
1405     nlm_init_share(&args.share, shr, fh);
1406
1407     xid = atomic_inc_32_nv(&nlm_xid);
1408     args.cookie.n_len = sizeof (xid);
1409     args.cookie.n_bytes = (char *)&xid;
1410
1411     for (;;) {
1412         nlm_rpc_t *rpcp;
1413         struct nlm4_sharereres res;
1414         enum clnt_stat stat;
1415
1416         error = nlm_host_get_rpc(host, vers, &rpcp);
1417         if (error != 0)
1418             return (ENOLCK);
1419
1420         bzero(&res, sizeof (res));
1421         stat = nlm_unshare_rpc(&args, &res, rpcp->nr_handle, vers);
1422         nlm_host_rele_rpc(host, rpcp);
1423
1424         error = nlm_map_clnt_stat(stat);
1425         if (error != 0) {
1426             if (error == EAGAIN)
1427                 continue;
1428
1429             return (error);
1430         }
1431
1432         DTRACE_PROBE1(unshare_res, enum nlm4_stats, res.stat);
1433         nlm_err = res.stat;
1434         xdr_free((xdrproc_t)xdr_nlm4_res, (void *)&res);
1435         if (nlm_err == nlm4_denied_grace_period) {
1436             error = nlm_host_wait_grace(host);
1437             if (error != 0)
1438                 return (error);
1439
1440             continue;
1441         }
1442     }
1443     break;
1444 }
1445
1446 switch (nlm_err) {
1447 case nlm4_granted:

```

```

1448         error = 0;
1449         break;
1450     case nlm4_denied:
1451         error = EAGAIN;
1452         break;
1453     case nlm4_denied_nolocks:
1454         error = ENOLCK;
1455         break;
1456     default:
1457         error = EINVAL;
1458         break;
1459     }
1461     return (error);
1462 }

1464 static void
1465 nlm_init_share(struct nlm4_share *args,
1466               const struct shrlock *shr, struct netobj *fh)
1467 {
1469     bzero(args, sizeof (*args));

1471     args->caller_name = uts_nodename();
1472     args->fh.n_len = fh->n_len;
1473     args->fh.n_bytes = fh->n_bytes;
1474     args->oh.n_len = shr->s_own_len;
1475     args->oh.n_bytes = (void *)shr->s_owner;

1477     switch (shr->s_deny) {
1478     default:
1479     case F_NODNY:
1480         args->mode = fsm_DN;
1481         break;
1482     case F_RDDNY:
1483         args->mode = fsm_DR;
1484         break;
1485     case F_WRDNY:
1486         args->mode = fsm_DW;
1487         break;
1488     case F_RWDNY:
1489         args->mode = fsm_DRW;
1490         break;
1491     }

1493     switch (shr->s_access) {
1494     default:
1495     case 0: /* seen with F_UNSHARE */
1496         args->access = fsa_NONE;
1497         break;
1498     case F_RDACC:
1499         args->access = fsa_R;
1500         break;
1501     case F_WRACC:
1502         args->access = fsa_W;
1503         break;
1504     case F_RWACC:
1505         args->access = fsa_RW;
1506         break;
1507     }
1508 }

1510 /*
1511  * Initialize filehandle according to the version
1512  * of NFS vnode was created on. The version of
1513  * NLM that can be used with given NFS version

```

```

1514  * is saved to lm_vers.
1515  */
1516 static int
1517 nlm_init_fh_by_vp(vnode_t *vp, struct netobj *fh, rpcvers_t *lm_vers)
1518 {
1519     mntinfo_t *mi = VTOMI(vp);

1521     /*
1522      * Too bad the NFS code doesn't just carry the FH
1523      * in a netobj or a netbuf.
1524      */
1525     switch (mi->mi_vers) {
1526     case NFS_V3:
1527         /* See nfs3_frlock() */
1528         *lm_vers = NLM4_VERS;
1529         fh->n_len = VTOFH3(vp)->fh3_length;
1530         fh->n_bytes = (char *)&(VTOFH3(vp)->fh3_u.data);
1531         break;

1533     case NFS_VERSION:
1534         /* See nfs_frlock() */
1535         *lm_vers = NLM_VERS;
1536         fh->n_len = sizeof (fhandle_t);
1537         /* LINTED E_BAD_PTR_CAST_ALIGN */
1538         fh->n_bytes = (char *)VTOFH(vp);
1539         break;
1540     default:
1541         return (ENOSYS);
1542     }

1544     return (0);
1545 }

1547 /*
1548  * Send SIGLOST to the process identified by pid.
1549  * NOTE: called when NLM decides to remove lock
1550  * or share reservation owner by the process
1551  * by force.
1552  */
1553 static void
1554 nlm_send_siglost(pid_t pid)
1555 {
1556     proc_t *p;

1558     mutex_enter(&pidlock);
1559     p = prfind(pid);
1560     if (p != NULL)
1561         psignal(p, SIGLOST);

1563     mutex_exit(&pidlock);
1564 }

1566 static int
1567 nlm_map_clnt_stat(enum clnt_stat stat)
1568 {
1569     switch (stat) {
1570     case RPC_SUCCESS:
1571         return (0);

1573     case RPC_TIMEDOUT:
1574     case RPC_PROGUNAVAIL:
1575         return (EAGAIN);

1577     case RPC_INTR:
1578         return (EINTR);

```

```
1580     default:
1581         return (EINVAL);
1582     }
1583 }

1585 static int
1586 nlm_map_status(enum nlm4_stats stat)
1587 {
1588     switch (stat) {
1589     case nlm4_granted:
1590         return (0);

1592     case nlm4_denied:
1593         return (EAGAIN);

1595     case nlm4_denied_nolocks:
1596         return (ENOLCK);

1598     case nlm4_blocked:
1599         return (EAGAIN);

1601     case nlm4_denied_grace_period:
1602         return (EAGAIN);

1604     case nlm4_deadlck:
1605         return (EDEADLK);

1607     case nlm4_rofs:
1608         return (EROFS);

1610     case nlm4_stale_fh:
1611         return (ESTALE);

1613     case nlm4_fbig:
1614         return (EFBIG);

1616     case nlm4_failed:
1617         return (EACCES);

1619     default:
1620         return (EINVAL);
1621     }
1622 }
```



```

*****
15379 Fri Aug 16 09:04:51 2013
new/usr/src/uts/common/klm/nlm_dispatch.c
195 Need replacement for nfs/lockd+klm
*****
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 is of the CDDL is also available via the Internet
9  * at http://www.illumos.org/license/CDDL.
10 */

12 /*
13  * Copyright 2013 Nexenta Systems, Inc. All rights reserved.
14 */

16 /*
17  * NFS Lock Manager, server-side dispatch tables and
18  * dispatch programs: nlm_prog_3, nlm_prog4
19  *
20  * These are called by RPC framework after the RPC service
21  * endpoints setup done in nlm_impl.c: nlm_svc_add_ep().
22  *
23  * Originally from rpcgen, then reduced.
24 */

26 #include <sys/param.h>
27 #include <sys/system.h>
28 #include <sys/sdt.h>
29 #include <rpcsvc/nlm_prot.h>
30 #include "nlm_impl.h"

32 /*
33  * Dispatch entry function pointers.
34 */
35 typedef bool_t (*nlm_svc_func_t)(void *, void *, struct svc_req *);
36 typedef void (*nlm_freeres_func_t)(void *);

38 /*
39  * Entries in the dispatch tables below.
40 */
41 struct dispatch_entry {
42     nlm_svc_func_t      de_svc;          /* service routine function */
43     xdrproc_t           de_xargs;        /* XDR args decode function */
44     xdrproc_t           de_xres;        /* XDR res encode function */
45     nlm_freeres_func_t de_resfree;      /* free res function */
46     int                 de_ressize;     /* size of result */
47     uint_t              de_flags;       /* flags */
48 };

50 /* Flag bits in de_flags */
51 #define NLM_DISP_NOREMOTE 1 /* Local calls only */

53 /*
54  * Cast macros for dispatch table function pointers.
55 */
56 #define NLM_SVC_FUNC(func) (nlm_svc_func_t)func
57 #define NLM_FREERES_FUNC(func) (nlm_freeres_func_t)func

59 /* ARGSUSED */
60 static bool_t
61 nlm_null_svc(void *args, void *resp, struct svc_req *sr)

```

```

62 {
63     return (TRUE);
64 }

66 /*
67  * The common NLM service dispatch function, used by
68  * both: nlm_prog_3, nlm_prog_4
69 */
70 void
71 nlm_dispatch(
72     struct svc_req *rqstp,
73     SVCXPRT *transp,
74     const struct dispatch_entry *de)
75 {
76     union {
77         /* All the arg types */
78         nlm_cancargs  au_cancargs;
79         nlm_lockargs  au_lockargs;
80         nlm_notify    au_notify;
81         nlm_res        au_res;
82         nlm_shareargs au_shareargs;
83         nlm_sm_status au_sm_status;
84         nlm_testargs  au_testargs;
85         nlm_testres   au_testres;
86         nlm_unlockargs au_unlockargs;
87         nlm4_cancargs au_cancargs4;
88         nlm4_lockargs au_lockargs4;
89         nlm4_notify   au_notify4;
90         nlm4_res       au_res4;
91         nlm4_shareargs au_shareargs4;
92         nlm4_testargs au_testargs4;
93         nlm4_testres  au_testres4;
94         nlm4_unlockargs au_unlockargs4;
95     } argu;
96     void *args = &argu;
97     union {
98         /* All the ret types */
99         int          ru_int;
100        nlm_res       ru_res;
101        nlm_sharer    ru_sharer;
102        nlm_testres  ru_testres;
103        nlm4_res     ru_res4;
104        nlm4_sharer  ru_sharer4;
105        nlm4_testres ru_testres4;
106    } resu;
107     void *res = &resu;
108     nlm_svc_func_t func;
109     bool_t do_reply = FALSE;
110     bool_t dupcached = FALSE;
111     struct dupreq *dr;
112     int dupstat;

115     if ((func = de->de_svc) == NULL) {
116         svcerr_noproc(transp);
117         return;
118     }

120     if ((de->de_flags & NLM_DISP_NOREMOTE) &&
121         !nlm_caller_is_local(transp)) {
122         svcerr_noproc(transp);
123         return;
124     }

126     /*
127      * This section from rpcgen, and then modified slightly.

```

```

128  *
129  * Dispatch entries that should never send a response
130  * (i.e. all the MSG and RES entries) put NULL in the
131  * de_xres field to indicate that. For such entries, we
132  * will NOT call svc_sendreply nor xdr free(). Normal
133  * dispatch entries skip svc_sendreply if the dispatch
134  * function returns zero, but always call xdr_free().
135  *
136  * There are more complex cases where some dispatch
137  * functions need to send their own reply. We chose
138  * to indicate those by returning false from the
139  * service routine.
140  */
141  bzero(&argu, sizeof (argu));
142  if (!SVC_GETARGS(transp, de->de_xargs, args)) {
143      svcerr_decode(transp);
144      return;
145  }
146
147  /*
148  * Duplicate request cache.
149  *
150  * Since none of the NLM replies are very large we have simplified the
151  * DRC by not distinguishing between idempotent and non-idempotent
152  * requests.
153  */
154  dupstat = SVC_DUP_EXT(transp, rqstp, res, de->de_ressz, &dr,
155                      &dupcached);
156
157  switch (dupstat) {
158  case DUP_ERROR:
159      svcerr_systemerr(transp);
160      break;
161  case DUP_INPROGRESS:
162      break;
163  case DUP_NEW:
164  case DUP_DROP:
165      /*
166       * When UFS is quiescing it uses lockfs to block vnode
167       * operations until it has finished quiescing. Set the
168       * thread's T_DONTPEND flag to prevent the service routine
169       * from blocking due to a lockfs lock. (See ufs_check_lockfs)
170       */
171      curthread->t_flag |= T_DONTPEND;
172
173      bzero(&resu, sizeof (resu));
174      do_reply = (*func)(args, res, rqstp);
175
176      curthread->t_flag &= ~T_DONTPEND;
177      if (curthread->t_flag & T_WOULDBLOCK) {
178          curthread->t_flag &= ~T_WOULDBLOCK;
179          SVC_DUPDONE_EXT(transp, dr, res, NULL,
180                        de->de_ressz, DUP_DROP);
181          do_reply = FALSE;
182          break;
183      }
184      SVC_DUPDONE_EXT(transp, dr, res, de->de_resfree,
185                    de->de_ressz, DUP_DONE);
186      dupcached = TRUE;
187      break;
188  case DUP_DONE:
189      /*
190       * The service routine may have been responsible for sending
191       * the reply for the original request but for a re-xmitted
192       * request we don't invoke the service routine so we must
193       * re-xmit the reply from the dispatch function.

```

```

194  *
195  * If de_xres is NULL this is a one-way message so no reply is
196  * needed.
197  */
198  if (de->de_xres != NULL_xdrproc_t) {
199      do_reply = TRUE;
200  }
201  break;
202  }
203
204  if (do_reply) {
205      ASSERT(de->de_xres != NULL_xdrproc_t);
206      DTRACE_PROBE3(sendreply, struct svc_req *, rqstp,
207                  SVCXPRT *, transp, struct dispatch_entry *, de);
208
209      if (!svc_sendreply(transp, de->de_xres, res)) {
210          svcerr_systemerr(transp);
211          NLM_ERR("nlm_dispatch(): svc_sendreply() failed!\n");
212      }
213
214      if (!dupcached) {
215          xdr_free(de->de_xres, res);
216      }
217  }
218
219  if (!SVC_FREEARGS(transp, de->de_xargs, args))
220      NLM_WARN("nlm_dispatch(): unable to free arguments");
221  }
222
223  /*
224  * Result free functions. The functions are called by the RPC duplicate
225  * request cache code when an entry is being evicted from the cache.
226  */
227  static void
228  nlm_res_free(nlm_res *resp)
229  {
230      xdr_free(xdr_nlm_res, (char *)resp);
231  }
232
233  static void
234  nlm_sharereres_free(nlm_sharereres *resp)
235  {
236      xdr_free(xdr_nlm_sharereres, (char *)resp);
237  }
238
239  static void
240  nlm_testres_free(nlm_testres *resp)
241  {
242      xdr_free(xdr_nlm_testres, (char *)resp);
243  }
244
245  static void
246  nlm4_res_free(nlm4_res *resp)
247  {
248      xdr_free(xdr_nlm4_res, (char *)resp);
249  }
250
251  static void
252  nlm4_sharereres_free(nlm4_sharereres *resp)
253  {
254      xdr_free(xdr_nlm4_sharereres, (char *)resp);
255  }
256
257  static void
258  nlm4_testres_free(nlm4_testres *resp)
259  {

```

```

260     xdr_free(xdr_nlm4_testres, (char *)resp);
261 }

263 /*
264 * Dispatch tables for each program version.
265 *
266 * The tables here were all originally from rpcgen,
267 * but then arg/resp sizes removed, flags added.
268 */

270 /*
271 * Dispatch table for versions 1, 2, 3
272 * (NLM_VERS, NLM_SM, NLM_VERSX)
273 */
274 static const struct dispatch_entry
275 nlm_prog_3_dtable[] = {

277     /*
278      * Version 1 (NLM_VERS) entries.
279      */

281     { /* 0: NULLPROC */
282       NLM_SVC_FUNC(nlm_null_svc),
283       (xdrproc_t)xdr_void,
284       (xdrproc_t)xdr_void,
285       NULL,
286       0,
287       0 },

289     { /* 1: NLM_TEST */
290       NLM_SVC_FUNC(nlm_test_1_svc),
291       (xdrproc_t)xdr_nlm_testargs,
292       (xdrproc_t)xdr_nlm_testres,
293       NLM_FREERES_FUNC(nlm_testres_free),
294       sizeof (nlm_testres),
295       0 },

297     { /* 2: NLM_LOCK */
298       NLM_SVC_FUNC(nlm_lock_1_svc),
299       (xdrproc_t)xdr_nlm_lockargs,
300       (xdrproc_t)xdr_nlm_res,
301       NLM_FREERES_FUNC(nlm_res_free),
302       sizeof (nlm_res),
303       0 },

305     { /* 3: NLM_CANCEL */
306       NLM_SVC_FUNC(nlm_cancel_1_svc),
307       (xdrproc_t)xdr_nlm_cancargs,
308       (xdrproc_t)xdr_nlm_res,
309       NLM_FREERES_FUNC(nlm_res_free),
310       sizeof (nlm_res),
311       0 },

313     { /* 4: NLM_UNLOCK */
314       NLM_SVC_FUNC(nlm_unlock_1_svc),
315       (xdrproc_t)xdr_nlm_unlockargs,
316       (xdrproc_t)xdr_nlm_res,
317       NLM_FREERES_FUNC(nlm_res_free),
318       sizeof (nlm_res),
319       0 },

321     { /* 5: NLM_GRANTED */
322       NLM_SVC_FUNC(nlm_granted_1_svc),
323       (xdrproc_t)xdr_nlm_testargs,
324       (xdrproc_t)xdr_nlm_res,
325       NLM_FREERES_FUNC(nlm_res_free),

```

```

326     sizeof (nlm_res),
327     0 },

329     /*
330      * All the _MSG and _RES entries are "one way" calls that
331      * skip the usual RPC reply. We give them a null xdr_res
332      * function so the dispatcher will not send a reply.
333      */

335     { /* 6: NLM_TEST_MSG */
336       NLM_SVC_FUNC(nlm_test_msg_1_svc),
337       (xdrproc_t)xdr_nlm_testargs,
338       (xdrproc_t)0,
339       NULL,
340       0,
341       0 },

343     { /* 7: NLM_LOCK_MSG */
344       NLM_SVC_FUNC(nlm_lock_msg_1_svc),
345       (xdrproc_t)xdr_nlm_lockargs,
346       (xdrproc_t)0,
347       NULL,
348       0,
349       0 },

351     { /* 8: NLM_CANCEL_MSG */
352       NLM_SVC_FUNC(nlm_cancel_msg_1_svc),
353       (xdrproc_t)xdr_nlm_cancargs,
354       (xdrproc_t)0,
355       NULL,
356       0,
357       0 },

359     { /* 9: NLM_UNLOCK_MSG */
360       NLM_SVC_FUNC(nlm_unlock_msg_1_svc),
361       (xdrproc_t)xdr_nlm_unlockargs,
362       (xdrproc_t)0,
363       NULL,
364       0,
365       0 },

367     { /* 10: NLM_GRANTED_MSG */
368       NLM_SVC_FUNC(nlm_granted_msg_1_svc),
369       (xdrproc_t)xdr_nlm_testargs,
370       (xdrproc_t)0,
371       NULL,
372       0,
373       0 },

375     { /* 11: NLM_TEST_RES */
376       NLM_SVC_FUNC(nlm_test_res_1_svc),
377       (xdrproc_t)xdr_nlm_testres,
378       (xdrproc_t)0,
379       NULL,
380       0,
381       0 },

383     { /* 12: NLM_LOCK_RES */
384       NLM_SVC_FUNC(nlm_lock_res_1_svc),
385       (xdrproc_t)xdr_nlm_res,
386       (xdrproc_t)0,
387       NULL,
388       0,
389       0 },

391     { /* 13: NLM_CANCEL_RES */

```

```

392     NLM_SVC_FUNC(nlm_cancel_res_1_svc),
393     (xdrproc_t)xdr_nlm_res,
394     (xdrproc_t)0,
395     NULL,
396     0,
397     0 },

399     { /* 14: NLM_UNLOCK_RES */
400     NLM_SVC_FUNC(nlm_unlock_res_1_svc),
401     (xdrproc_t)xdr_nlm_res,
402     (xdrproc_t)0,
403     NULL,
404     0,
405     0 },

407     { /* 15: NLM_GRANTED_RES */
408     NLM_SVC_FUNC(nlm_granted_res_1_svc),
409     (xdrproc_t)xdr_nlm_res,
410     (xdrproc_t)0,
411     NULL,
412     0,
413     0 },

415     { /* 16: not used */
416     NLM_SVC_FUNC(0),
417     (xdrproc_t)0,
418     (xdrproc_t)0,
419     NULL,
420     0,
421     0 },

423     { /* 17: NLM_SM_NOTIFY1 */
424     NLM_SVC_FUNC(nlm_sm_notify1_2_svc),
425     (xdrproc_t)xdr_nlm_sm_status,
426     (xdrproc_t)xdr_void,
427     NULL,
428     0,
429     NLM_DISP_NOREMOTE },

431     { /* 18: NLM_SM_NOTIFY2 */
432     NLM_SVC_FUNC(nlm_sm_notify2_2_svc),
433     (xdrproc_t)xdr_nlm_sm_status,
434     (xdrproc_t)xdr_void,
435     NULL,
436     0,
437     NLM_DISP_NOREMOTE },

439     /*
440     * Version 3 (NLM_VERSX) entries.
441     */

443     { /* 19: not used */
444     NLM_SVC_FUNC(0),
445     (xdrproc_t)0,
446     (xdrproc_t)0,
447     NULL,
448     0,
449     0 },

451     { /* 20: NLM_SHARE */
452     NLM_SVC_FUNC(nlm_share_3_svc),
453     (xdrproc_t)xdr_nlm_shareargs,
454     (xdrproc_t)xdr_nlm_sharereres,
455     NLM_FREERES_FUNC(nlm_sharereres_free),
456     sizeof (nlm_sharereres),
457     0 },

```

```

459     { /* 21: NLM_UNSHARE */
460     NLM_SVC_FUNC(nlm_unshare_3_svc),
461     (xdrproc_t)xdr_nlm_shareargs,
462     (xdrproc_t)xdr_nlm_sharereres,
463     NLM_FREERES_FUNC(nlm_sharereres_free),
464     sizeof (nlm_sharereres),
465     0 },

467     { /* 22: NLM_NM_LOCK */
468     NLM_SVC_FUNC(nlm_nm_lock_3_svc),
469     (xdrproc_t)xdr_nlm_lockargs,
470     (xdrproc_t)xdr_nlm_res,
471     NLM_FREERES_FUNC(nlm_res_free),
472     sizeof (nlm_res),
473     0 },

475     { /* 23: NLM_FREE_ALL */
476     NLM_SVC_FUNC(nlm_free_all_3_svc),
477     (xdrproc_t)xdr_nlm_notify,
478     (xdrproc_t)xdr_void,
479     NULL,
480     0,
481     0 },
482 };
483 static int nlm_prog_3_dtsize =
484     sizeof (nlm_prog_3_dtable) /
485     sizeof (nlm_prog_3_dtable[0]);

487 /*
488  * RPC dispatch function for nlm_prot versions: 1,2,3
489  */
490 void
491 nlm_prog_3(struct svc_req *rqstp, register SVCXPRT *transp)
492 {
493     const struct dispatch_entry *de;
494     rpcproc_t max_proc;

496     switch (rqstp->rq_vers) {
497     case NLM_VERS:
498         max_proc = NLM_GRANTED_RES;
499         break;
500     case NLM_SM:
501         max_proc = NLM_SM_NOTIFY2;
502         break;
503     case NLM_VERSX:
504         max_proc = NLM_FREE_ALL;
505         break;
506     default:
507         /* Our svc registration should prevent this. */
508         ASSERT(0); /* paranoid */
509         svcerr_noprogram(transp);
510         return;
511     }
512     ASSERT(max_proc < nlm_prog_3_dtsize);

514     if (rqstp->rq_proc > max_proc) {
515         svcerr_noprogram(transp);
516         return;
517     }

519     de = &nlm_prog_3_dtable[rqstp->rq_proc];

521     nlm_dispatch(rqstp, transp, de);
522 }

```

```

524 /*
525  * Dispatch table for version 4 (NLM4_VERS)
526  */
527 static const struct dispatch_entry
528 nlm_prog_4_dtable[] = {
530     { /* 0: NULLPROC */
531       NLM_SVC_FUNC(nlm_null_svc),
532       (xdrproc_t)xdr_void,
533       (xdrproc_t)xdr_void,
534       NULL,
535       0,
536     },
538     { /* 1: NLM4_TEST */
539       NLM_SVC_FUNC(nlm4_test_4_svc),
540       (xdrproc_t)xdr_nlm4_testargs,
541       (xdrproc_t)xdr_nlm4_testres,
542       NLM_FREERES_FUNC(nlm4_testres_free),
543       sizeof (nlm4_testres),
544       0 },
546     { /* 2: NLM4_LOCK */
547       NLM_SVC_FUNC(nlm4_lock_4_svc),
548       (xdrproc_t)xdr_nlm4_lockargs,
549       (xdrproc_t)xdr_nlm4_res,
550       NLM_FREERES_FUNC(nlm4_res_free),
551       sizeof (nlm4_res),
552       0 },
554     { /* 3: NLM4_CANCEL */
555       NLM_SVC_FUNC(nlm4_cancel_4_svc),
556       (xdrproc_t)xdr_nlm4_cancargs,
557       (xdrproc_t)xdr_nlm4_res,
558       NLM_FREERES_FUNC(nlm4_res_free),
559       sizeof (nlm4_res),
560       0 },
562     { /* 4: NLM4_UNLOCK */
563       NLM_SVC_FUNC(nlm4_unlock_4_svc),
564       (xdrproc_t)xdr_nlm4_unlockargs,
565       (xdrproc_t)xdr_nlm4_res,
566       NLM_FREERES_FUNC(nlm4_res_free),
567       sizeof (nlm4_res),
568       0 },
570     { /* 5: NLM4_GRANTED */
571       NLM_SVC_FUNC(nlm4_granted_4_svc),
572       (xdrproc_t)xdr_nlm4_testargs,
573       (xdrproc_t)xdr_nlm4_res,
574       NLM_FREERES_FUNC(nlm4_res_free),
575       sizeof (nlm4_res),
576       0 },
578     /*
579     * All the MSG and RES entries are "one way" calls that
580     * skip the usual RPC reply. We give them a null xdr_res
581     * function so the dispatcher will not send a reply.
582     */
584     { /* 6: NLM4_TEST_MSG */
585       NLM_SVC_FUNC(nlm4_test_msg_4_svc),
586       (xdrproc_t)xdr_nlm4_testargs,
587       (xdrproc_t)0,
588       NULL,
589       0,

```

```

590     0 },
592     { /* 7: NLM4_LOCK_MSG */
593       NLM_SVC_FUNC(nlm4_lock_msg_4_svc),
594       (xdrproc_t)xdr_nlm4_lockargs,
595       (xdrproc_t)0,
596       NULL,
597       0,
598     },
600     { /* 8: NLM4_CANCEL_MSG */
601       NLM_SVC_FUNC(nlm4_cancel_msg_4_svc),
602       (xdrproc_t)xdr_nlm4_cancargs,
603       (xdrproc_t)0,
604       NULL,
605       0,
606     },
608     { /* 9: NLM4_UNLOCK_MSG */
609       NLM_SVC_FUNC(nlm4_unlock_msg_4_svc),
610       (xdrproc_t)xdr_nlm4_unlockargs,
611       (xdrproc_t)0,
612       NULL,
613       0,
614     },
616     { /* 10: NLM4_GRANTED_MSG */
617       NLM_SVC_FUNC(nlm4_granted_msg_4_svc),
618       (xdrproc_t)xdr_nlm4_testargs,
619       (xdrproc_t)0,
620       NULL,
621       0,
622     },
624     { /* 11: NLM4_TEST_RES */
625       NLM_SVC_FUNC(nlm4_test_res_4_svc),
626       (xdrproc_t)xdr_nlm4_testres,
627       (xdrproc_t)0,
628       NULL,
629       0,
630     },
632     { /* 12: NLM4_LOCK_RES */
633       NLM_SVC_FUNC(nlm4_lock_res_4_svc),
634       (xdrproc_t)xdr_nlm4_res,
635       (xdrproc_t)0,
636       NULL,
637       0,
638     },
640     { /* 13: NLM4_CANCEL_RES */
641       NLM_SVC_FUNC(nlm4_cancel_res_4_svc),
642       (xdrproc_t)xdr_nlm4_res,
643       (xdrproc_t)0,
644       NULL,
645       0,
646     },
648     { /* 14: NLM4_UNLOCK_RES */
649       NLM_SVC_FUNC(nlm4_unlock_res_4_svc),
650       (xdrproc_t)xdr_nlm4_res,
651       (xdrproc_t)0,
652       NULL,
653       0,
654     },

```

```

656     { /* 15: NLM4_GRANTED_RES */
657     NLM_SVC_FUNC(nlm4_granted_res_4_svc),
658     (xdrproc_t)xdr_nlm4_res,
659     (xdrproc_t)0,
660     NULL,
661     0,
662     0 },

664     { /* 16: not used */
665     NLM_SVC_FUNC(0),
666     (xdrproc_t)0,
667     (xdrproc_t)0,
668     NULL,
669     0,
670     0 },

672     { /* 17: NLM_SM_NOTIFY1 (not in v4) */
673     NLM_SVC_FUNC(0),
674     (xdrproc_t)0,
675     (xdrproc_t)0,
676     NULL,
677     0,
678     0 },

680     { /* 18: NLM_SM_NOTIFY2 (not in v4) */
681     NLM_SVC_FUNC(0),
682     (xdrproc_t)0,
683     (xdrproc_t)0,
684     NULL,
685     0,
686     0 },

688     { /* 19: not used */
689     NLM_SVC_FUNC(0),
690     (xdrproc_t)0,
691     (xdrproc_t)0,
692     NULL,
693     0,
694     0 },

696     { /* 20: NLM4_SHARE */
697     NLM_SVC_FUNC(nlm4_share_4_svc),
698     (xdrproc_t)xdr_nlm4_shareargs,
699     (xdrproc_t)xdr_nlm4_sharereres,
700     NLM_FREERES_FUNC(nlm4_sharereres_free),
701     sizeof (nlm4_sharereres),
702     0 },

704     { /* 21: NLM4_UNSHARE */
705     NLM_SVC_FUNC(nlm4_unshare_4_svc),
706     (xdrproc_t)xdr_nlm4_shareargs,
707     (xdrproc_t)xdr_nlm4_sharereres,
708     NLM_FREERES_FUNC(nlm4_sharereres_free),
709     sizeof (nlm4_sharereres),
710     0 },

712     { /* 22: NLM4_NM_LOCK */
713     NLM_SVC_FUNC(nlm4_nm_lock_4_svc),
714     (xdrproc_t)xdr_nlm4_lockargs,
715     (xdrproc_t)xdr_nlm4_res,
716     NLM_FREERES_FUNC(nlm4_res_free),
717     sizeof (nlm4_res),
718     0 },

720     { /* 23: NLM4_FREE_ALL */
721     NLM_SVC_FUNC(nlm4_free_all_4_svc),

```

```

722     (xdrproc_t)xdr_nlm4_notify,
723     (xdrproc_t)xdr_void,
724     NULL,
725     0,
726     0 },
727 };
728 static int nlm_prog_4_dtsize =
729     sizeof (nlm_prog_4_dtable) /
730     sizeof (nlm_prog_4_dtable[0]);

732 /*
733  * RPC dispatch function for nlm_prot version 4.
734  */
735 void
736 nlm_prog_4(struct svc_req *rqstp, register SVCXPRT *transp)
737 {
738     const struct dispatch_entry *de;

740     if (rqstp->rq_vers != NLM4_VERS) {
741         /* Our svc registration should prevent this. */
742         ASSERT(0); /* paranoid */
743         svcerr_noprogram(transp);
744         return;
745     }

747     if (rqstp->rq_proc >= nlm_prog_4_dtsize) {
748         svcerr_noprogram(transp);
749         return;
750     }

752     de = &nlm_prog_4_dtable[rqstp->rq_proc];

754     nlm_dispatch(rqstp, transp, de);
755 }

```

```

*****
65841 Fri Aug 16 09:04:52 2013
new/usr/src/uts/common/klm/nlm_impl.c
195 Need replacement for nfs/lockd+klm
*****
1 /*
2  * Copyright (c) 2008 Isilon Inc http://www.isilon.com/
3  * Authors: Doug Rabson <dfr@rabson.org>
4  * Developed with Red Inc: Alfred Perlstein <alfred@freebsd.org>
5  *
6  * Redistribution and use in source and binary forms, with or without
7  * modification, are permitted provided that the following conditions
8  * are met:
9  * 1. Redistributions of source code must retain the above copyright
10 * notice, this list of conditions and the following disclaimer.
11 * 2. Redistributions in binary form must reproduce the above copyright
12 * notice, this list of conditions and the following disclaimer in the
13 * documentation and/or other materials provided with the distribution.
14 *
15 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
16 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
17 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
18 * ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
19 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
20 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
21 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
22 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
23 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
24 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
25 * SUCH DAMAGE.
26 */

28 /*
29  * Copyright 2013 Nexenta Systems, Inc. All rights reserved.
30  * Copyright (c) 2012 by Delphix. All rights reserved.
31 */

33 /*
34  * NFS LockManager, start/stop, support functions, etc.
35  * Most of the interesting code is here.
36  *
37  * Source code derived from FreeBSD nlm_prot_impl.c
38 */

40 #include <sys/param.h>
41 #include <sys/system.h>
42 #include <sys/thread.h>
43 #include <sys/fcntl.h>
44 #include <sys/flock.h>
45 #include <sys/mount.h>
46 #include <sys/priv.h>
47 #include <sys/proc.h>
48 #include <sys/share.h>
49 #include <sys/socket.h>
50 #include <sys/syscall.h>
51 #include <sys/syslog.h>
52 #include <sys/system.h>
53 #include <sys/class.h>
54 #include <sys/unistd.h>
55 #include <sys/vnode.h>
56 #include <sys/vfs.h>
57 #include <sys/queue.h>
58 #include <sys/bitmap.h>
59 #include <sys/sdt.h>
60 #include <netinet/in.h>

```

```

62 #include <rpc/rpc.h>
63 #include <rpc/xdr.h>
64 #include <rpc/pmap_prot.h>
65 #include <rpc/pmap_clnt.h>
66 #include <rpc/rpcb_prot.h>

68 #include <rpcsvc/nlm_prot.h>
69 #include <rpcsvc/sm_inter.h>
70 #include <rpcsvc/nsm_addr.h>

72 #include <nfs/nfs.h>
73 #include <nfs/nfs_clnt.h>
74 #include <nfs/export.h>
75 #include <nfs/rnode.h>
76 #include <nfs/lm.h>

78 #include "nlm_impl.h"

80 struct nlm_knc {
81     struct knetconfig      n_knc;
82     const char             *n_netid;
83 };

85 /*
86  * Number of attempts NLM tries to obtain RPC binding
87  * of local statd.
88 */
89 #define NLM_NSM_RPCBIND_RETRIES 10

91 /*
92  * Timeout (in seconds) NLM waits before making another
93  * attempt to obtain RPC binding of local statd.
94 */
95 #define NLM_NSM_RPCBIND_TIMEOUT 5

97 /*
98  * Total number of sysids in NLM sysid bitmap
99 */
100 #define NLM_BMAP_NITEMS (LM_SYSID_MAX + 1)

102 /*
103  * Number of ulong_t words in bitmap that is used
104  * for allocation of sysid numbers.
105 */
106 #define NLM_BMAP_WORDS (NLM_BMAP_NITEMS / BT_NBIPUL)

108 /*
109  * Given an integer x, the macro returns
110  * -1 if x is negative,
111  * 0 if x is zero
112  * 1 if x is positive
113 */
114 #define SIGN(x) (((x) > 0) - ((x) < 0))

116 #define ARRSIZE(arr) (sizeof (arr) / sizeof ((arr)[0]))
117 #define NLM_KNCS ARRSIZE(nlm_netconfigs)

119 krwlock_t lm_lck;

121 /*
122  * Zero timeout for asynchronous NLM RPC operations
123 */
124 static const struct timeval nlm_rpctv_zero = { 0, 0 };

126 /*
127  * List of all Zone globals nlm_globals instances

```

```

128 * linked together.
129 */
130 static struct nlm_globals_list nlm_zones_list; /* (g) */

132 /*
133 * NLM kmem caches
134 */
135 static struct kmem_cache *nlm_hosts_cache = NULL;
136 static struct kmem_cache *nlm_vhold_cache = NULL;

138 /*
139 * A bitmap for allocation of new sysids.
140 * Sysid is a unique number between LM_SYSID
141 * and LM_SYSID_MAX. Sysid represents unique remote
142 * host that does file locks on the given host.
143 */
144 static ulong_t nlm_sysid_bmap[NLM_BMAP_WORDS]; /* (g) */
145 static int nlm_sysid_nidx; /* (g) */

147 /*
148 * RPC service registration for all transports
149 */
150 static SVC_CALLOUT nlm_svcs[] = {
151     { NLM_PROG, 4, 4, nlm_prog_4 }, /* NLM4_VERS */
152     { NLM_PROG, 1, 3, nlm_prog_3 } /* NLM_VERS - NLM_VERSX */
153 };

155 static SVC_CALLOUT_TABLE nlm_sct = {
156     ARRSIZE(nlm_svcs),
157     FALSE,
158     nlm_svcs
159 };

161 /*
162 * Static table of all netid/knetconfig network
163 * lock manager can work with. nlm_netconfigs table
164 * is used when we need to get valid knetconfig by
165 * netid and vice versa.
166 *
167 * Knetconfigs are activated either by the call from
168 * user-space lockd daemon (server side) or by taking
169 * knetconfig from NFS mountinfo (client side)
170 */
171 static struct nlm_knc nlm_netconfigs[] = { /* (g) */
172     /* UDP */
173     {
174         { NC_TPI_CLTS, NC_INET, NC_UDP, NODEV },
175         "udp",
176     },
177     /* TCP */
178     {
179         { NC_TPI_COTS_ORD, NC_INET, NC_TCP, NODEV },
180         "tcp",
181     },
182     /* UDP over IPv6 */
183     {
184         { NC_TPI_CLTS, NC_INET6, NC_UDP, NODEV },
185         "udp6",
186     },
187     /* TCP over IPv6 */
188     {
189         { NC_TPI_COTS_ORD, NC_INET6, NC_TCP, NODEV },
190         "tcp6",
191     },
192     /* ticlts (loopback over UDP) */
193     {

```

```

194         { NC_TPI_CLTS, NC_LOOPBACK, NC_NOPROTO, NODEV },
195         "ticlts",
196     },
197     /* ticotsord (loopback over TCP) */
198     {
199         { NC_TPI_COTS_ORD, NC_LOOPBACK, NC_NOPROTO, NODEV },
200         "ticotsord",
201     },
202 };

204 /*
205 * NLM misc. function
206 */
207 static void nlm_copy_netbuf(struct netbuf *, struct netbuf *);
208 static int nlm_netbuf_addrs_cmp(struct netbuf *, struct netbuf *);
209 static void nlm_kmem_reclaim(void *);
210 static void nlm_pool_shutdown(void);
211 static void nlm_suspend_zone(struct nlm_globals *);
212 static void nlm_resume_zone(struct nlm_globals *);
213 static void nlm_nsm_clnt_init(CLIENT *, struct nlm_nsm *);
214 static void nlm_netbuf_to_netobj(struct netbuf *, int *, netobj *);

216 /*
217 * NLM thread functions
218 */
219 static void nlm_gc(struct nlm_globals *);
220 static void nlm_reclaimer(struct nlm_host *);

222 /*
223 * NLM NSM functions
224 */
225 static int nlm_init_local_knc(struct knetconfig *);
226 static int nlm_nsm_init_local(struct nlm_nsm *);
227 static int nlm_nsm_init(struct nlm_nsm *, struct knetconfig *, struct netbuf *);
228 static void nlm_nsm_fini(struct nlm_nsm *);
229 static enum clnt_stat nlm_nsm_simu_crash(struct nlm_nsm *);
230 static enum clnt_stat nlm_nsm_stat(struct nlm_nsm *, int32_t *);
231 static enum clnt_stat nlm_nsm_mon(struct nlm_nsm *, char *, uint16_t);
232 static enum clnt_stat nlm_nsm_unmon(struct nlm_nsm *, char *);

234 /*
235 * NLM host functions
236 */
237 static int nlm_host_ctor(void *, void *, int);
238 static void nlm_host_dtor(void *, void *);
239 static void nlm_host_destroy(struct nlm_host *);
240 static struct nlm_host *nlm_host_create(char *, const char *,
241     struct knetconfig *, struct netbuf *);
242 static struct nlm_host *nlm_host_find_locked(struct nlm_globals *,
243     const char *, struct netbuf *, avl_index_t *);
244 static void nlm_host_unregister(struct nlm_globals *, struct nlm_host *);
245 static void nlm_host_gc_vholds(struct nlm_host *);
246 static bool_t nlm_host_has_srv_locks(struct nlm_host *);
247 static bool_t nlm_host_has_cli_locks(struct nlm_host *);
248 static bool_t nlm_host_has_locks(struct nlm_host *);

250 /*
251 * NLM vhold functions
252 */
253 static int nlm_vhold_ctor(void *, void *, int);
254 static void nlm_vhold_dtor(void *, void *);
255 static void nlm_vhold_destroy(struct nlm_host *,
256     struct nlm_vhold *);
257 static bool_t nlm_vhold_busy(struct nlm_host *, struct nlm_vhold *);
258 static void nlm_vhold_clean(struct nlm_vhold *, int);

```



```

260 /*
261  * NLM client/server sleeping locks/share reservation functions
262  */
263 struct nlm_slreq *nlm_slreq_find_locked(struct nlm_host *,
264     struct nlm_vhold *, struct flock64 *);
265 static struct nlm_shres *nlm_shres_create_item(struct shrlock *, vnode_t *);
266 static void nlm_shres_destroy_item(struct nlm_shres *);
267 static bool_t nlm_shres_equal(struct shrlock *, struct shrlock *);

269 /*
270  * NLM initialization functions.
271  */
272 void
273 nlm_init(void)
274 {
275     nlm_hosts_cache = kmem_cache_create("nlm_host_cache",
276     sizeof (struct nlm_host), 0, nlm_host_ctor, nlm_host_dtor,
277     nlm_kmem_reclaim, NULL, NULL, 0);

279     nlm_vhold_cache = kmem_cache_create("nlm_vhold_cache",
280     sizeof (struct nlm_vhold), 0, nlm_vhold_ctor, nlm_vhold_dtor,
281     NULL, NULL, NULL, 0);

283     nlm_rpc_init();
284     TAILQ_INIT(&nlm_zones_list);

286     /* initialize sysids bitmap */
287     bzero(nlm_sysid_bmap, sizeof (nlm_sysid_bmap));
288     nlm_sysid_nidx = 1;

290     /*
291      * Reserv the sysid #0, because it's associated
292      * with local locks only. Don't let to allocate
293      * it for remote locks.
294      */
295     BT_SET(nlm_sysid_bmap, 0);
296 }

298 void
299 nlm_globals_register(struct nlm_globals *g)
300 {
301     rw_enter(&lm_lck, RW_WRITER);
302     TAILQ_INSERT_TAIL(&nlm_zones_list, g, nlm_link);
303     rw_exit(&lm_lck);
304 }

306 void
307 nlm_globals_unregister(struct nlm_globals *g)
308 {
309     rw_enter(&lm_lck, RW_WRITER);
310     TAILQ_REMOVE(&nlm_zones_list, g, nlm_link);
311     rw_exit(&lm_lck);
312 }

314 /* ARGSUSED */
315 static void
316 nlm_kmem_reclaim(void *cdrarg)
317 {
318     struct nlm_globals *g;

320     rw_enter(&lm_lck, RW_READER);
321     TAILQ_FOREACH(g, &nlm_zones_list, nlm_link)
322         cv_broadcast(&g->nlm_gc_sched_cv);

324     rw_exit(&lm_lck);
325 }

```

```

327 /*
328  * NLM garbage collector thread (GC).
329  */
330 * NLM GC periodically checks whether there're any host objects
331 * that can be cleaned up. It also releases stale vnodes that
332 * live on the server side (under protection of vhold objects).
333 *
334 * NLM host objects are cleaned up from GC thread because
335 * operations helping us to determine whether given host has
336 * any locks can be quite expensive and it's not good to call
337 * them every time the very last reference to the host is dropped.
338 * Thus we use "lazy" approach for hosts cleanup.
339 *
340 * The work of GC is to release stale vnodes on the server side
341 * and destroy hosts that haven't any locks and any activity for
342 * some time (i.e. idle hosts).
343 */
344 static void
345 nlm_gc(struct nlm_globals *g)
346 {
347     struct nlm_host *hostp;
348     clock_t now, idle_period;

350     idle_period = SEC_TO_TICK(g->cn_idle_tmo);
351     mutex_enter(&g->lock);
352     for (;;) {
353         /*
354          * GC thread can be explicitly scheduled from
355          * memory reclamation function.
356          */
357         (void) cv_timedwait(&g->nlm_gc_sched_cv, &g->lock,
358             ddi_get_lbolt() + idle_period);

360         /*
361          * NLM is shutting down, time to die.
362          */
363         if (g->run_status == NLM_ST_STOPPING)
364             break;

366         now = ddi_get_lbolt();
367         DTRACE_PROBE2(gc__start, struct nlm_globals *, g,
368             clock_t, now);

370         /*
371          * Handle all hosts that are unused at the moment
372          * until we meet one with idle timeout in future.
373          */
374         while ((hostp = TAILQ_FIRST(&g->nlm_idle_hosts)) != NULL) {
375             bool_t has_locks = FALSE;

377             if (hostp->nh_idle_timeout > now)
378                 break;

380             /*
381              * Drop global lock while doing expensive work
382              * on this host. We'll re-check any conditions
383              * that might change after retaking the global
384              * lock.
385              */
386             mutex_exit(&g->lock);
387             mutex_enter(&hostp->nh_lock);

389             /*
390              * nlm_globals lock was dropped earlier because
391              * garbage collecting of vholds and checking whether

```

```

392     * host has any locks/shares are expensive operations.
393     */
394     nlm_host_gc_vholds(hostp);
395     has_locks = nlm_host_has_locks(hostp);

397     mutex_exit(&hostp->nh_lock);
398     mutex_enter(&g->lock);

400     /*
401     * While we were doing expensive operations outside of
402     * nlm_globals critical section, somebody could
403     * take the host, add lock/share to one of its vnodes
404     * and release the host back. If so, host's idle timeout
405     * is renewed and our information about locks on the
406     * given host is outdated.
407     */
408     if (hostp->nh_idle_timeout > now)
409         continue;

411     /*
412     * If either host has locks or somebody has began to
413     * use it while we were outside the nlm_globals critical
414     * section. In both cases we have to renew host's
415     * timeout and put it to the end of LRU list.
416     */
417     if (has_locks || hostp->nh_refs > 0) {
418         TAILQ_REMOVE(&g->nlm_idle_hosts,
419             hostp, nh_link);
420         hostp->nh_idle_timeout = now + idle_period;
421         TAILQ_INSERT_TAIL(&g->nlm_idle_hosts,
422             hostp, nh_link);
423         continue;
424     }

426     /*
427     * We're here if all the following conditions hold:
428     * 1) Host hasn't any locks or share reservations
429     * 2) Host is unused
430     * 3) Host wasn't touched by anyone at least for
431     *    g->cn_idle_tmo seconds.
432     *
433     * So, now we can destroy it.
434     */
435     nlm_host_unregister(g, hostp);
436     mutex_exit(&g->lock);

438     nlm_host_unmonitor(g, hostp);
439     nlm_host_destroy(hostp);
440     mutex_enter(&g->lock);
441     if (g->run_status == NLM_ST_STOPPING)
442         break;

444     }

446     DTRACE_PROBE(gc__end);
447 }

449     DTRACE_PROBE1(gc__exit, struct nlm_globals *, g);

451     /* Let others know that GC has died */
452     g->nlm_gc_thread = NULL;
453     mutex_exit(&g->lock);

455     cv_broadcast(&g->nlm_gc_finish_cv);
456     zthread_exit();
457 }

```

```

459 /*
460 * Thread reclaim locks/shares acquired by the client side
461 * on the given server represented by hostp.
462 */
463 static void
464 nlm_reclaimer(struct nlm_host *hostp)
465 {
466     struct nlm_globals *g;

468     mutex_enter(&hostp->nh_lock);
469     hostp->nh_reclaimer = curthread;
470     mutex_exit(&hostp->nh_lock);

472     g = zone_getspecific(nlm_zone_key, curzone);
473     nlm_reclaim_client(g, hostp);

475     mutex_enter(&hostp->nh_lock);
476     hostp->nh_flags &= ~NLM_NH_RECLAIM;
477     hostp->nh_reclaimer = NULL;
478     cv_broadcast(&hostp->nh_recl_cv);
479     mutex_exit(&hostp->nh_lock);

481     /*
482     * Host was explicitly referenced before
483     * nlm_reclaim() was called, release it
484     * here.
485     */
486     nlm_host_release(g, hostp);
487     zthread_exit();
488 }

490 /*
491 * Copy a struct netobj. (see xdr.h)
492 */
493 void
494 nlm_copy_netobj(struct netobj *dst, struct netobj *src)
495 {
496     dst->n_len = src->n_len;
497     dst->n_bytes = kmem_alloc(src->n_len, KM_SLEEP);
498     bcopy(src->n_bytes, dst->n_bytes, src->n_len);
499 }

501 /*
502 * An NLM specificw replacement for clnt_call().
503 * nlm_clnt_call() is used by all RPC functions generated
504 * from nlm_prot.x specification. The function is aware
505 * about some pitfalls of NLM RPC procedures and has a logic
506 * that handles them properly.
507 */
508 enum clnt_stat
509 nlm_clnt_call(CLIENT *clnt, rpcproc_t procnum, xdrproc_t xdr_args,
510     caddr_t argsp, xdrproc_t xdr_result, caddr_t resultp, struct timeval wait)
511 {
512     k_sigset_t oldmask;
513     enum clnt_stat stat;
514     bool_t sig_blocked = FALSE;

516     /*
517     * If NLM RPC procnum is one of the NLM_RES procedures
518     * that are used to reply to asynchronous NLM RPC
519     * (MSG calls), explicitly set RPC timeout to zero.
520     * Client doesn't send a reply to RES procedures, so
521     * we don't need to wait anything.
522     *
523     * NOTE: we ignore NLM4*_RES procnums because they are

```

```

524     * equal to NLM_*_RES numbers.
525     */
526     if (procnum >= NLM_TEST_RES && procnum <= NLM_GRANTED_RES)
527         wait = nlm_rpctv_zero;

529     /*
530     * We need to block signals in case of NLM_CANCEL RPC
531     * in order to prevent interruption of network RPC
532     * calls.
533     */
534     if (procnum == NLM_CANCEL) {
535         k_sigset_t newmask;

537         sigfillset(&newmask);
538         sigreplace(&newmask, &oldmask);
539         sig_blocked = TRUE;
540     }

542     stat = clnt_call(clnt, procnum, xdr_args,
543                   argsp, xdr_result, resultp, wait);

545     /*
546     * Restore signal mask back if signals were blocked
547     */
548     if (sig_blocked)
549         sigreplace(&oldmask, (k_sigset_t *)NULL);

551     return (stat);
552 }

554 /*
555 * Suspend NLM client/server in the given zone.
556 *
557 * During suspend operation we mark those hosts
558 * that have any locks with NLM_NH_SUSPEND flags,
559 * so that they can be checked later, when resume
560 * operation occurs.
561 */
562 static void
563 nlm_suspend_zone(struct nlm_globals *g)
564 {
565     struct nlm_host *hostp;
566     struct nlm_host_list all_hosts;

568     /*
569     * Note that while we're doing suspend, GC thread is active
570     * and it can destroy some hosts while we're walking through
571     * the hosts tree. To prevent that and make suspend logic
572     * a bit more simple we put all hosts to local "all_hosts"
573     * list and increment reference counter of each host.
574     * This guaranties that no hosts will be released while
575     * we're doing suspend.
576     * NOTE: reference of each host must be dropped during
577     * resume operation.
578     */
579     TAILQ_INIT(&all_hosts);
580     mutex_enter(&g->lock);
581     for (hostp = avl_first(&g->nlm_hosts_tree); hostp != NULL;
582         hostp = AVL_NEXT(&g->nlm_hosts_tree, hostp)) {
583         /*
584         * If host is idle, remove it from idle list and
585         * clear idle flag. That is done to prevent GC
586         * from touching this host.
587         */
588         if (hostp->nh_flags & NLM_NH_INIDLE) {
589             TAILQ_REMOVE(&g->nlm_idle_hosts, hostp, nh_link);

```

```

590         hostp->nh_flags &= ~NLM_NH_INIDLE;
591     }

593     hostp->nh_refs++;
594     TAILQ_INSERT_TAIL(&all_hosts, hostp, nh_link);
595 }

597     /*
598     * Now we can walk through all hosts on the system
599     * with zone globals lock released. The fact the
600     * we have taken a reference to each host guaranties
601     * that no hosts can be destroyed during that process.
602     */
603     mutex_exit(&g->lock);
604     while ((hostp = TAILQ_FIRST(&all_hosts)) != NULL) {
605         mutex_enter(&hostp->nh_lock);
606         if (nlm_host_has_locks(hostp))
607             hostp->nh_flags |= NLM_NH_SUSPEND;

609         mutex_exit(&hostp->nh_lock);
610         TAILQ_REMOVE(&all_hosts, hostp, nh_link);
611     }
612 }

614 /*
615 * Resume NLM hosts for the given zone.
616 *
617 * nlm_resume_zone() is called after hosts were suspended
618 * (see nlm_suspend_zone) and its main purpose to check
619 * whether remote locks owned by hosts are still in consistent
620 * state. If they aren't, resume function tries to reclaim
621 * reclaim locks (for client side hosts) and clean locks (for
622 * server side hosts).
623 */
624 static void
625 nlm_resume_zone(struct nlm_globals *g)
626 {
627     struct nlm_host *hostp, *h_next;

629     mutex_enter(&g->lock);
630     hostp = avl_first(&g->nlm_hosts_tree);

632     /*
633     * In nlm_suspend_zone() the reference counter of each
634     * host was incremented, so we can safely iterate through
635     * all hosts without worrying that any host we touch will
636     * be removed at the moment.
637     */
638     while (hostp != NULL) {
639         struct nlm_nsm nsm;
640         enum clnt_stat stat;
641         int32_t sm_state;
642         int error;
643         bool_t resume_failed = FALSE;

645         h_next = AVL_NEXT(&g->nlm_hosts_tree, hostp);
646         mutex_exit(&g->lock);

648         DTRACE_PROBE1(resume_host, struct nlm_host *, hostp);

650         /*
651         * Suspend operation marked that the host doesn't
652         * have any locks. Skip it.
653         */
654         if (!(hostp->nh_flags & NLM_NH_SUSPEND))
655             goto cycle_end;

```

```

657     error = nlm_nsm_init(&nsm, &hostp->nh_knc, &hostp->nh_addr);
658     if (error != 0) {
659         NLM_ERR("Resume: Failed to contact to NSM of host %s "
660             "[error=%d]\n", hostp->nh_name, error);
661         resume_failed = TRUE;
662         goto cycle_end;
663     }
664
665     stat = nlm_nsm_stat(&nsm, &sm_state);
666     if (stat != RPC_SUCCESS) {
667         NLM_ERR("Resume: Failed to call SM_STAT operation for "
668             "host %s [stat=%d]\n", hostp->nh_name, stat);
669         resume_failed = TRUE;
670         nlm_nsm_fini(&nsm);
671         goto cycle_end;
672     }
673
674     if (sm_state != hostp->nh_state) {
675         /*
676          * Current SM state of the host isn't equal
677          * to the one host had when it was suspended.
678          * Probably it was rebooted. Try to reclaim
679          * locks if the host has any on its client side.
680          * Also try to clean up its server side locks
681          * (if the host has any).
682          */
683         nlm_host_notify_client(hostp, sm_state);
684         nlm_host_notify_server(hostp, sm_state);
685     }
686
687     nlm_nsm_fini(&nsm);
688
689 cycle_end:
690     if (resume_failed) {
691         /*
692          * Resume failed for the given host.
693          * Just clean up all resources it owns.
694          */
695         nlm_host_notify_server(hostp, 0);
696         nlm_client_cancel_all(g, hostp);
697     }
698
699     hostp->nh_flags &= ~NLM_NH_SUSPEND;
700     nlm_host_release(g, hostp);
701     hostp = h_next;
702     mutex_enter(&g->lock);
703 }
704
705     mutex_exit(&g->lock);
706 }
707
708 /*
709  * NLM functions responsible for operations on NSM handle.
710  */
711
712 /*
713  * Initialize knetconfig that is used for communication
714  * with local statd via loopback interface.
715  */
716 static int
717 nlm_init_local_knc(struct knetconfig *knc)
718 {
719     int error;
720     vnode_t *vp;

```

```

722     bzero(knc, sizeof (*knc));
723     error = lookupname("/dev/tcp", UIO_SYSSPACE,
724         FOLLOW, NULLVPP, &vp);
725     if (error != 0)
726         return (error);
727
728     knc->knc_semantics = NC_TPI_COTS;
729     knc->knc_protofmly = NC_INET;
730     knc->knc_proto = NC_TCP;
731     knc->knc_rdev = vp->v_rdev;
732     VN_RELE(vp);
733
734     return (0);
735 }
736
737 /*
738  * Initialize NSM handle that will be used to talk
739  * to local statd via loopback interface.
740  */
741 static int
742 nlm_nsm_init_local(struct nlm_nsm *nsm)
743 {
744     int error;
745     struct knetconfig knc;
746     struct sockaddr_in sin;
747     struct netbuf nb;
748
749     error = nlm_init_local_knc(&knc);
750     if (error != 0)
751         return (error);
752
753     bzero(&sin, sizeof (sin));
754     sin.sin_addr.s_addr = htonl(INADDR_LOOPBACK);
755     sin.sin_family = AF_INET;
756
757     nb.buf = (char *)&sin;
758     nb.len = nb.maxlen = sizeof (sin);
759
760     return (nlm_nsm_init(nsm, &knc, &nb));
761 }
762
763 /*
764  * Initialize NSM handle used for talking to statd
765  */
766 static int
767 nlm_nsm_init(struct nlm_nsm *nsm, struct knetconfig *knc, struct netbuf *nb)
768 {
769     enum clnt_stat stat;
770     int error, retries;
771
772     bzero(nsm, sizeof (*nsm));
773     nsm->ns_knc = *knc;
774     nlm_copy_netbuf(&nsm->ns_addr, nb);
775
776     /*
777      * Try several times to get the port of statd service,
778      * If rpcbind_getaddr returns RPC_PROGNOTREGISTERED,
779      * retry an attempt, but wait for NLM_NSM_RPCBIND_TIMEOUT
780      * seconds before.
781      */
782     for (retries = 0; retries < NLM_NSM_RPCBIND_RETRIES; retries++) {
783         stat = rpcbind_getaddr(&nsm->ns_knc, SM_PROG,
784             SM_VERS, &nsm->ns_addr);
785         if (stat != RPC_SUCCESS) {
786             if (stat == RPC_PROGNOTREGISTERED) {

```

```

788         delay(SEC_TO_TICK(NLM_NSM_RPCBIND_TIMEOUT));
789         continue;
790     }
791 }
792
793     break;
794 }
795
796 if (stat != RPC_SUCCESS) {
797     DTRACE_PROBE2(rpcbind_error, enum clnt_stat, stat,
798                 int, retries);
799     error = ENOENT;
800     goto error;
801 }
802
803 /*
804  * Create an RPC handle that'll be used for communication with local
805  * statd using the status monitor protocol.
806  */
807 error = clnt_tli_kcreate(&nsm->ns_knc, &nsm->ns_addr, SM_PROG, SM_VERS,
808                        0, NLM_RPC_RETRIES, kcred, &nsm->ns_handle);
809 if (error != 0)
810     goto error;
811
812 /*
813  * Create an RPC handle that'll be used for communication with the
814  * local statd using the address registration protocol.
815  */
816 error = clnt_tli_kcreate(&nsm->ns_knc, &nsm->ns_addr, NSM_ADDR_PROGRAM,
817                        NSM_ADDR_V1, 0, NLM_RPC_RETRIES, kcred, &nsm->ns_addr_handle);
818 if (error != 0)
819     goto error;
820
821     sema_init(&nsm->ns_sem, 1, NULL, SEMA_DEFAULT, NULL);
822     return (0);
823
824 error:
825     kmem_free(nsm->ns_addr.buf, nsm->ns_addr.maxlen);
826     if (nsm->ns_handle)
827         CLNT_DESTROY(nsm->ns_handle);
828
829     return (error);
830 }
831
832 static void
833 nlm_nsm_fini(struct nlm_nsm *nsm)
834 {
835     kmem_free(nsm->ns_addr.buf, nsm->ns_addr.maxlen);
836     CLNT_DESTROY(nsm->ns_addr_handle);
837     nsm->ns_addr_handle = NULL;
838     CLNT_DESTROY(nsm->ns_handle);
839     nsm->ns_handle = NULL;
840     sema_destroy(&nsm->ns_sem);
841 }
842
843 static enum clnt_stat
844 nlm_nsm_simu_crash(struct nlm_nsm *nsm)
845 {
846     enum clnt_stat stat;
847
848     sema_p(&nsm->ns_sem);
849     nlm_nsm_clnt_init(nsm->ns_handle, nsm);
850     stat = sm_simu_crash_1(NULL, NULL, nsm->ns_handle);
851     sema_v(&nsm->ns_sem);
852
853     return (stat);

```

```

854 }
855
856 static enum clnt_stat
857 nlm_nsm_stat(struct nlm_nsm *nsm, int32_t *out_stat)
858 {
859     struct sm_name args;
860     struct sm_stat_res res;
861     enum clnt_stat stat;
862
863     args.mon_name = uts_nodename();
864     bzero(&res, sizeof (res));
865
866     sema_p(&nsm->ns_sem);
867     nlm_nsm_clnt_init(nsm->ns_handle, nsm);
868     stat = sm_stat_1(&args, &res, nsm->ns_handle);
869     sema_v(&nsm->ns_sem);
870
871     if (stat == RPC_SUCCESS)
872         *out_stat = res.state;
873
874     return (stat);
875 }
876
877 static enum clnt_stat
878 nlm_nsm_mon(struct nlm_nsm *nsm, char *hostname, uint16_t priv)
879 {
880     struct mon_args;
881     struct sm_stat_res res;
882     enum clnt_stat stat;
883
884     bzero(&args, sizeof (args));
885     bzero(&res, sizeof (res));
886
887     args.mon_id.mon_name = hostname;
888     args.mon_id.my_id.my_name = uts_nodename();
889     args.mon_id.my_id.my_prog = NLM_PROG;
890     args.mon_id.my_id.my_vers = NLM_SM;
891     args.mon_id.my_id.my_proc = NLM_SM_NOTIFY1;
892     bcopy(&priv, args.priv, sizeof (priv));
893
894     sema_p(&nsm->ns_sem);
895     nlm_nsm_clnt_init(nsm->ns_handle, nsm);
896     stat = sm_mon_1(&args, &res, nsm->ns_handle);
897     sema_v(&nsm->ns_sem);
898
899     return (stat);
900 }
901
902 static enum clnt_stat
903 nlm_nsm_unmon(struct nlm_nsm *nsm, char *hostname)
904 {
905     struct mon_id args;
906     struct sm_stat res;
907     enum clnt_stat stat;
908
909     bzero(&args, sizeof (args));
910     bzero(&res, sizeof (res));
911
912     args.mon_name = hostname;
913     args.my_id.my_name = uts_nodename();
914     args.my_id.my_prog = NLM_PROG;
915     args.my_id.my_vers = NLM_SM;
916     args.my_id.my_proc = NLM_SM_NOTIFY1;
917
918     sema_p(&nsm->ns_sem);
919     nlm_nsm_clnt_init(nsm->ns_handle, nsm);

```

```

920     stat = sm_unmon_l(&args, &res, nsm->ns_handle);
921     sema_v(&nsm->ns_sem);

923     return (stat);
924 }

926 static enum clnt_stat
927 nlm_nsmaddr_reg(struct nlm_nsm *nsm, char *name, int family, netobj *address)
928 {
929     struct reglargs args = { 0 };
930     struct reglres res = { 0 };
931     enum clnt_stat stat;

933     args.family = family;
934     args.name = name;
935     args.address = *address;

937     sema_p(&nsm->ns_sem);
938     nlm_nsm_clnt_init(nsm->ns_addr_handle, nsm);
939     stat = nsmaddrprocl_reg_l(&args, &res, nsm->ns_addr_handle);
940     sema_v(&nsm->ns_sem);

942     return (stat);
943 }

945 /*
946  * Get NLM vhold object corresponding to vnode "vp".
947  * If no such object was found, create a new one.
948  *
949  * The purpose of this function is to associate vhold
950  * object with given vnode, so that:
951  * 1) vnode is hold (VN_HOLD) while vhold object is alive.
952  * 2) host has a track of all vnodes it touched by lock
953  *    or share operations. These vnodes are accessible
954  *    via collection of vhold objects.
955  */
956 struct nlm_vhold *
957 nlm_vhold_get(struct nlm_host *hostp, vnode_t *vp)
958 {
959     struct nlm_vhold *nvp, *new_nvp = NULL;

961     mutex_enter(&hostp->nh_lock);
962     nvp = nlm_vhold_find_locked(hostp, vp);
963     if (nvp != NULL)
964         goto out;

966     /* nlm_vhold wasn't found, then create a new one */
967     mutex_exit(&hostp->nh_lock);
968     new_nvp = kmem_cache_alloc(nlm_vhold_cache, KM_SLEEP);

970     /*
971     * Check if another thread has already
972     * created the same nlm_vhold.
973     */
974     mutex_enter(&hostp->nh_lock);
975     nvp = nlm_vhold_find_locked(hostp, vp);
976     if (nvp == NULL) {
977         nvp = new_nvp;
978         new_nvp = NULL;

980         TAILQ_INIT(&nvp->nv_slreqs);
981         nvp->nv_vp = vp;
982         nvp->nv_refcnt = 1;
983         VN_HOLD(nvp->nv_vp);

985         VERIFY(mod_hash_insert(hostp->nh_vholds_by_vp,

```

```

986         (mod_hash_key_t)vp, (mod_hash_val_t)nvp) == 0);
987         TAILQ_INSERT_TAIL(&hostp->nh_vholds_list, nvp, nv_link);
988     }

990 out:
991     mutex_exit(&hostp->nh_lock);
992     if (new_nvp != NULL)
993         kmem_cache_free(nlm_vhold_cache, new_nvp);

995     return (nvp);
996 }

998 /*
999  * Drop a reference to vhold object nvp.
1000  */
1001 void
1002 nlm_vhold_release(struct nlm_host *hostp, struct nlm_vhold *nvp)
1003 {
1004     if (nvp == NULL)
1005         return;

1007     mutex_enter(&hostp->nh_lock);
1008     ASSERT(nvp->nv_refcnt > 0);
1009     nvp->nv_refcnt--;
1010     mutex_exit(&hostp->nh_lock);
1011 }

1013 /*
1014  * Clean all locks and share reservations on the
1015  * given vhold object that were acquired by the
1016  * given sysid
1017  */
1018 static void
1019 nlm_vhold_clean(struct nlm_vhold *nvp, int sysid)
1020 {
1021     cleanlocks(nvp->nv_vp, IGN_PID, sysid);
1022     cleanshares_by_sysid(nvp->nv_vp, sysid);
1023 }

1025 static void
1026 nlm_vhold_destroy(struct nlm_host *hostp, struct nlm_vhold *nvp)
1027 {
1028     ASSERT(MUTEX_HELD(&hostp->nh_lock));

1030     VERIFY(mod_hash_remove(hostp->nh_vholds_by_vp,
1031         (mod_hash_key_t)nvp->nv_vp,
1032         (mod_hash_val_t)&nvp) == 0);

1034     TAILQ_REMOVE(&hostp->nh_vholds_list, nvp, nv_link);
1035     VN_RELE(nvp->nv_vp);
1036     nvp->nv_vp = NULL;

1038     kmem_cache_free(nlm_vhold_cache, nvp);
1039 }

1041 /*
1042  * Return TRUE if the given vhold is busy.
1043  * Vhold object is considered to be "busy" when
1044  * all the following conditions hold:
1045  * 1) No one uses it at the moment;
1046  * 2) It hasn't any locks;
1047  * 3) It hasn't any share reservations;
1048  */
1049 static bool_t
1050 nlm_vhold_busy(struct nlm_host *hostp, struct nlm_vhold *nvp)
1051 {

```

```

1052     vnode_t *vp;
1053     int sysid;

1055     ASSERT(MUTEX_HELD(&hostp->nh_lock));

1057     if (nvp->nv_refcnt > 0)
1058         return (TRUE);

1060     vp = nvp->nv_vp;
1061     sysid = hostp->nh_sysid;
1062     if (flk_has_remote_locks_for_sysid(vp, sysid) ||
1063         shr_has_remote_shares(vp, sysid))
1064         return (TRUE);

1066     return (FALSE);
1067 }

1069 /* ARGSUSED */
1070 static int
1071 nlm_vhold_ctor(void *datap, void *cdrarg, int kmflags)
1072 {
1073     struct nlm_vhold *nvp = (struct nlm_vhold *)datap;

1075     bzero(nvp, sizeof (*nvp));
1076     return (0);
1077 }

1079 /* ARGSUSED */
1080 static void
1081 nlm_vhold_dtor(void *datap, void *cdrarg)
1082 {
1083     struct nlm_vhold *nvp = (struct nlm_vhold *)datap;

1085     ASSERT(nvp->nv_refcnt == 0);
1086     ASSERT(TAILQ_EMPTY(&nvp->nv_slreqs));
1087     ASSERT(nvp->nv_vp == NULL);
1088 }

1090 struct nlm_vhold *
1091 nlm_vhold_find_locked(struct nlm_host *hostp, const vnode_t *vp)
1092 {
1093     struct nlm_vhold *nvp = NULL;

1095     ASSERT(MUTEX_HELD(&hostp->nh_lock));
1096     (void) mod_hash_find(hostp->nh_vholds_by_vp,
1097         (mod_hash_key_t)vp,
1098         (mod_hash_val_t)&nvp);

1100     if (nvp != NULL)
1101         nvp->nv_refcnt++;

1103     return (nvp);
1104 }

1106 /*
1107  * NLM host functions
1108  */
1109 static void
1110 nlm_copy_netbuf(struct netbuf *dst, struct netbuf *src)
1111 {
1112     ASSERT(src->len <= src->maxlen);

1114     dst->maxlen = src->maxlen;
1115     dst->len = src->len;
1116     dst->buf = kmem_zalloc(src->maxlen, KM_SLEEP);
1117     bcopy(src->buf, dst->buf, src->len);

```

```

1118 }

1120 /* ARGSUSED */
1121 static int
1122 nlm_host_ctor(void *datap, void *cdrarg, int kmflags)
1123 {
1124     struct nlm_host *hostp = (struct nlm_host *)datap;

1126     bzero(hostp, sizeof (*hostp));
1127     return (0);
1128 }

1130 /* ARGSUSED */
1131 static void
1132 nlm_host_dtor(void *datap, void *cdrarg)
1133 {
1134     struct nlm_host *hostp = (struct nlm_host *)datap;
1135     ASSERT(hostp->nh_refs == 0);
1136 }

1138 static void
1139 nlm_host_unregister(struct nlm_globals *g, struct nlm_host *hostp)
1140 {
1141     ASSERT(hostp->nh_refs == 0);

1143     avl_remove(&g->nlm_hosts_tree, hostp);
1144     VERIFY(mod_hash_remove(g->nlm_hosts_hash,
1145         (mod_hash_key_t)(uintptr_t)hostp->nh_sysid,
1146         (mod_hash_val_t)&hostp) == 0);
1147     TAILQ_REMOVE(&g->nlm_idle_hosts, hostp, nh_link);
1148     hostp->nh_flags &= ~NLM_NH_INIDLE;
1149 }

1151 /*
1152  * Free resources used by a host. This is called after the reference
1153  * count has reached zero so it doesn't need to worry about locks.
1154  */
1155 static void
1156 nlm_host_destroy(struct nlm_host *hostp)
1157 {
1158     ASSERT(hostp->nh_name != NULL);
1159     ASSERT(hostp->nh_netid != NULL);
1160     ASSERT(TAILQ_EMPTY(&hostp->nh_vholds_list));

1162     strfree(hostp->nh_name);
1163     strfree(hostp->nh_netid);
1164     kmem_free(hostp->nh_addr.buf, hostp->nh_addr.maxlen);

1166     if (hostp->nh_sysid != LM_NOSYSID)
1167         nlm_sysid_free(hostp->nh_sysid);

1169     nlm_rpc_cache_destroy(hostp);

1171     ASSERT(TAILQ_EMPTY(&hostp->nh_vholds_list));
1172     mod_hash_destroy_ptrhash(hostp->nh_vholds_by_vp);

1174     mutex_destroy(&hostp->nh_lock);
1175     cv_destroy(&hostp->nh_rpcb_cv);
1176     cv_destroy(&hostp->nh_recl_cv);

1178     kmem_cache_free(nlm_hosts_cache, hostp);
1179 }

1181 /*
1182  * Cleanup SERVER-side state after a client restarts,
1183  * or becomes unresponsive, or whatever.

```

```

1184 *
1185 * We unlock any active locks owned by the host.
1186 * When rpc.lockd is shutting down,
1187 * this function is called with newstate set to zero
1188 * which allows us to cancel any pending async locks
1189 * and clear the locking state.
1190 *
1191 * When "state" is 0, we don't update host's state,
1192 * but cleanup all remote locks on the host.
1193 * It's useful to call this function for resources
1194 * cleanup.
1195 */
1196 void
1197 nlm_host_notify_server(struct nlm_host *hostp, int32_t state)
1198 {
1199     struct nlm_vhold *nvp;
1200     struct nlm_slreq *slr;
1201     struct nlm_slreq_list slreqs2free;
1202
1203     TAILQ_INIT(&slreqs2free);
1204     mutex_enter(&hostp->nh_lock);
1205     if (state != 0)
1206         hostp->nh_state = state;
1207
1208     TAILQ_FOREACH(nvp, &hostp->nh_vholds_list, nv_link) {
1209         /* cleanup sleeping requests at first */
1210         while ((slr = TAILQ_FIRST(&nvp->nv_slreqs)) != NULL) {
1211             TAILQ_REMOVE(&nvp->nv_slreqs, slr, nsr_link);
1212
1213             /*
1214              * Instead of freeing cancelled sleeping request
1215              * here, we add it to the linked list created
1216              * on the stack in order to do all frees outside
1217              * the critical section.
1218              */
1219             TAILQ_INSERT_TAIL(&slreqs2free, slr, nsr_link);
1220         }
1221
1222         nvp->nv_refcnt++;
1223         mutex_exit(&hostp->nh_lock);
1224
1225         nlm_vhold_clean(nvp, hostp->nh_sysid);
1226
1227         mutex_enter(&hostp->nh_lock);
1228         nvp->nv_refcnt--;
1229     }
1230
1231     mutex_exit(&hostp->nh_lock);
1232     while ((slr = TAILQ_FIRST(&slreqs2free)) != NULL) {
1233         TAILQ_REMOVE(&slreqs2free, slr, nsr_link);
1234         kmem_free(slr, sizeof(*slr));
1235     }
1236 }
1237
1238 /*
1239 * Cleanup CLIENT-side state after a server restarts,
1240 * or becomes unresponsive, or whatever.
1241 *
1242 * This is called by the local NFS statd when we receive a
1243 * host state change notification. (also nlm_svc_stopping)
1244 *
1245 * Deal with a server restart. If we are stopping the
1246 * NLM service, we'll have newstate == 0, and will just
1247 * cancel all our client-side lock requests. Otherwise,
1248 * start the "recovery" process to reclaim any locks
1249 */

```

```

1250 * we hold on this server.
1251 */
1252 void
1253 nlm_host_notify_client(struct nlm_host *hostp, int32_t state)
1254 {
1255     mutex_enter(&hostp->nh_lock);
1256     hostp->nh_state = state;
1257     if (hostp->nh_flags & NLM_NH_RECLAIM) {
1258         /*
1259          * Either host's state is up to date or
1260          * host is already in recovery.
1261          */
1262         mutex_exit(&hostp->nh_lock);
1263         return;
1264     }
1265
1266     hostp->nh_flags |= NLM_NH_RECLAIM;
1267
1268     /*
1269      * Host will be released by the recovery thread,
1270      * thus we need to increment refcount.
1271      */
1272     hostp->nh_refs++;
1273     mutex_exit(&hostp->nh_lock);
1274
1275     (void) zthread_create(NULL, 0, nlm_reclaimer,
1276                          hostp, 0, minclsyspri);
1277 }
1278
1279 /*
1280 * The function is called when NLM client detects that
1281 * server has entered in grace period and client needs
1282 * to wait until reclamation process (if any) does
1283 * its job.
1284 */
1285 int
1286 nlm_host_wait_grace(struct nlm_host *hostp)
1287 {
1288     struct nlm_globals *g;
1289     int error = 0;
1290
1291     g = zone_getspecific(nlm_zone_key, curzone);
1292     mutex_enter(&hostp->nh_lock);
1293
1294     do {
1295         int rc;
1296
1297         rc = cv_timedwait_sig(&hostp->nh_recl_cv,
1298                             &hostp->nh_lock, ddi_get_lbolt() +
1299                             SEC_TO_TICK(g->retrans_tmo));
1300
1301         if (rc == 0) {
1302             error = EINTR;
1303             break;
1304         }
1305     } while (hostp->nh_flags & NLM_NH_RECLAIM);
1306
1307     mutex_exit(&hostp->nh_lock);
1308     return (error);
1309 }
1310
1311 /*
1312 * Create a new NLM host.
1313 *
1314 * NOTE: The in-kernel RPC (kRPC) subsystem uses TLI/XTI,
1315 * which needs both a knetconfig and an address when creating

```



```

1316 * endpoints. Thus host object stores both knetconfig and
1317 * netid.
1318 */
1319 static struct nlm_host *
1320 nlm_host_create(char *name, const char *netid,
1321                struct knetconfig *knc, struct netbuf *naddr)
1322 {
1323     struct nlm_host *host;
1324
1325     host = kmem_cache_alloc(nlm_hosts_cache, KM_SLEEP);
1326
1327     mutex_init(&host->nh_lock, NULL, MUTEX_DEFAULT, NULL);
1328     cv_init(&host->nh_rpcb_cv, NULL, CV_DEFAULT, NULL);
1329     cv_init(&host->nh_recl_cv, NULL, CV_DEFAULT, NULL);
1330
1331     host->nh_sysid = LM_NOSYSID;
1332     host->nh_refs = 1;
1333     host->nh_name = strdup(name);
1334     host->nh_netid = strdup(netid);
1335     host->nh_knc = *knc;
1336     nlm_copy_netbuf(&host->nh_addr, naddr);
1337
1338     host->nh_state = 0;
1339     host->nh_rpcb_state = NRPCB_NEED_UPDATE;
1340     host->nh_flags = 0;
1341
1342     host->nh_vholds_by_vp = mod_hash_create_ptrhash("nlm vholds hash",
1343            32, mod_hash_null_valdtor, sizeof(vnode_t));
1344
1345     TAILQ_INIT(&host->nh_vholds_list);
1346     TAILQ_INIT(&host->nh_rpcbc);
1347
1348     return (host);
1349 }
1350
1351 /*
1352 * Cancel all client side sleeping locks owned by given host.
1353 */
1354 void
1355 nlm_host_cancel_slocks(struct nlm_globals *g, struct nlm_host *hostp)
1356 {
1357     struct nlm_slock *nslp;
1358
1359     mutex_enter(&g->lock);
1360     TAILQ_FOREACH(nslp, &g->nlm_slocks, nsl_link) {
1361         if (nslp->nsl_host == hostp) {
1362             nslp->nsl_state = NLM_SL_CANCELLED;
1363             cv_broadcast(&nslp->nsl_cond);
1364         }
1365     }
1366
1367     mutex_exit(&g->lock);
1368 }
1369
1370 /*
1371 * Garbage collect stale vhold objects.
1372 *
1373 * In other words check whether vnodes that are
1374 * held by vhold objects still have any locks
1375 * or shares or still in use. If they aren't,
1376 * just destroy them.
1377 */
1378 static void
1379 nlm_host_gc_vholds(struct nlm_host *hostp)
1380 {
1381     struct nlm_vhold *nvp;

```

```

1383     ASSERT(MUTEX_HELD(&hostp->nh_lock));
1384
1385     nvp = TAILQ_FIRST(&hostp->nh_vholds_list);
1386     while (nvp != NULL) {
1387         struct nlm_vhold *nvp_tmp;
1388
1389         if (nlm_vhold_busy(hostp, nvp)) {
1390             nvp = TAILQ_NEXT(nvp, nv_link);
1391             continue;
1392         }
1393
1394         nvp_tmp = TAILQ_NEXT(nvp, nv_link);
1395         nlm_vhold_destroy(hostp, nvp);
1396         nvp = nvp_tmp;
1397     }
1398 }
1399
1400 /*
1401 * Check whether the given host has any
1402 * server side locks or share reservations.
1403 */
1404 static bool_t
1405 nlm_host_has_srv_locks(struct nlm_host *hostp)
1406 {
1407     /*
1408      * It's cheap and simple: if server has
1409      * any locks/shares there must be vhold
1410      * object storing the affected vnode.
1411      *
1412      * NOTE: We don't need to check sleeping
1413      * locks on the server side, because if
1414      * server side sleeping lock is alive,
1415      * there must be a vhold object corresponding
1416      * to target vnode.
1417      */
1418     ASSERT(MUTEX_HELD(&hostp->nh_lock));
1419     if (!TAILQ_EMPTY(&hostp->nh_vholds_list))
1420         return (TRUE);
1421
1422     return (FALSE);
1423 }
1424
1425 /*
1426 * Check whether the given host has any client side
1427 * locks or share reservations.
1428 */
1429 static bool_t
1430 nlm_host_has_cli_locks(struct nlm_host *hostp)
1431 {
1432     ASSERT(MUTEX_HELD(&hostp->nh_lock));
1433
1434     /*
1435      * XXX: It's not the way I'd like to do the check,
1436      * because flk_sysid_has_locks() can be very
1437      * expensive by design. Unfortunately it iterates
1438      * through all locks on the system, doesn't matter
1439      * were they made on remote system via NLM or
1440      * on local system via relock. To understand the
1441      * problem, consider that there're dozens of thousands
1442      * of locks that are made on some ZFS dataset. And there's
1443      * another dataset shared by NFS where NLM client had locks
1444      * some time ago, but doesn't have them now.
1445      * In this case flk_sysid_has_locks() will iterate
1446      * thru dozens of thousands locks until it returns us
1447      * FALSE.

```

```

1448 * Oh, I hope that in shiny future somebody will make
1449 * local lock manager (os/flock.c) better, so that
1450 * it'd be more friendly to remote locks and
1451 * flk_sysid_has_locks() wouldn't be so expensive.
1452 */
1453 if (flk_sysid_has_locks(hostp->nh_sysid |
1454     LM_SYSID_CLIENT, FLK_QUERY_ACTIVE))
1455     return (TRUE);

1457 /*
1458 * Check whether host has any share reservations
1459 * registered on the client side.
1460 */
1461 if (hostp->nh_shrlist != NULL)
1462     return (TRUE);

1464 return (FALSE);
1465 }

1467 /*
1468 * Determine whether the given host owns any
1469 * locks or share reservations.
1470 */
1471 static bool_t
1472 nlm_host_has_locks(struct nlm_host *hostp)
1473 {
1474     if (nlm_host_has_srv_locks(hostp))
1475         return (TRUE);

1477     return (nlm_host_has_cli_locks(hostp));
1478 }

1480 /*
1481 * This function compares only addresses of two netbufs
1482 * that belong to NC_TCP[6] or NC_UDP[6] protofamily.
1483 * Port part of netbuf is ignored.
1484 *
1485 * Return values:
1486 * -1: nbl's address is "smaller" than nb2's
1487 * 0: addresses are equal
1488 * 1: nbl's address is "greater" than nb2's
1489 */
1490 static int
1491 nlm_netbuf_addrs_cmp(struct netbuf *nb1, struct netbuf *nb2)
1492 {
1493     union nlm_addr {
1494         struct sockaddr sa;
1495         struct sockaddr_in sin;
1496         struct sockaddr_in6 sin6;
1497     } *nal, *na2;
1498     int res;

1500     /* LINTED E_BAD_PTR_CAST_ALIGN */
1501     nal = (union nlm_addr *)nb1->buf;
1502     /* LINTED E_BAD_PTR_CAST_ALIGN */
1503     na2 = (union nlm_addr *)nb2->buf;

1505     if (nal->sa.sa_family < na2->sa.sa_family)
1506         return (-1);
1507     if (nal->sa.sa_family > na2->sa.sa_family)
1508         return (1);

1510     switch (nal->sa.sa_family) {
1511     case AF_INET:
1512         res = memcmp(&nal->sin.sin_addr, &na2->sin.sin_addr,
1513             sizeof (nal->sin.sin_addr));

```

```

1514         break;
1515     case AF_INET6:
1516         res = memcmp(&nal->sin6.sin6_addr, &na2->sin6.sin6_addr,
1517             sizeof (nal->sin6.sin6_addr));
1518         break;
1519     default:
1520         VERIFY(0);
1521         return (0);
1522     }

1524     return (SIGN(res));
1525 }

1527 /*
1528 * Compare two nlm hosts.
1529 * Return values:
1530 * -1: host1 is "smaller" than host2
1531 * 0: host1 is equal to host2
1532 * 1: host1 is "greater" than host2
1533 */
1534 int
1535 nlm_host_cmp(const void *p1, const void *p2)
1536 {
1537     struct nlm_host *h1 = (struct nlm_host *)p1;
1538     struct nlm_host *h2 = (struct nlm_host *)p2;
1539     int res;

1541     res = strcmp(h1->nh_netid, h2->nh_netid);
1542     if (res != 0)
1543         return (SIGN(res));

1545     res = nlm_netbuf_addrs_cmp(&h1->nh_addr, &h2->nh_addr);
1546     return (res);
1547 }

1549 /*
1550 * Find the host specified by... (see below)
1551 * If found, increment the ref count.
1552 */
1553 static struct nlm_host *
1554 nlm_host_find_locked(struct nlm_globals *g, const char *netid,
1555     struct netbuf *naddr, avl_index_t *wherep)
1556 {
1557     struct nlm_host *hostp, key;
1558     avl_index_t pos;

1560     ASSERT(MUTEX_HELD(&g->lock));

1562     key.nh_netid = (char *)netid;
1563     key.nh_addr.buf = naddr->buf;
1564     key.nh_addr.len = naddr->len;
1565     key.nh_addr.maxlen = naddr->maxlen;

1567     hostp = avl_find(&g->nlm_hosts_tree, &key, &pos);

1569     if (hostp != NULL) {
1570         /*
1571          * Host is in use now. Remove it from idle
1572          * hosts list if needed.
1573          */
1574         if (hostp->nh_flags & NLM_NH_IDLE) {
1575             TAILQ_REMOVE(&g->nlm_idle_hosts, hostp, nh_link);
1576             hostp->nh_flags &= ~NLM_NH_IDLE;
1577         }

1579         hostp->nh_refs++;

```

```

1580     }
1581     if (wherep != NULL)
1582         *wherep = pos;
1584     return (hostp);
1585 }
1587 /*
1588  * Find NLM host for the given name and address.
1589 */
1590 struct nlm_host *
1591 nlm_host_find(struct nlm_globals *g, const char *netid,
1592             struct netbuf *addr)
1593 {
1594     struct nlm_host *hostp = NULL;
1596     mutex_enter(&g->lock);
1597     if (g->run_status != NLM_ST_UP)
1598         goto out;
1600     hostp = nlm_host_find_locked(g, netid, addr, NULL);
1602 out:
1603     mutex_exit(&g->lock);
1604     return (hostp);
1605 }
1608 /*
1609  * Find or create an NLM host for the given name and address.
1610  *
1611  * The remote host is determined by all of: name, netid, address.
1612  * Note that the netid is whatever nlm_svc_add_ep() gave to
1613  * svc_tli_kcreate() for the service binding. If any of these
1614  * are different, allocate a new host (new sysid).
1615 */
1616 struct nlm_host *
1617 nlm_host_findcreate(struct nlm_globals *g, char *name,
1618                  const char *netid, struct netbuf *addr)
1619 {
1620     int err;
1621     struct nlm_host *host, *newhost = NULL;
1622     struct knetconfig knc;
1623     avl_index_t where;
1625     mutex_enter(&g->lock);
1626     if (g->run_status != NLM_ST_UP) {
1627         mutex_exit(&g->lock);
1628         return (NULL);
1629     }
1631     host = nlm_host_find_locked(g, netid, addr, NULL);
1632     mutex_exit(&g->lock);
1633     if (host != NULL)
1634         return (host);
1636     err = nlm_knc_from_netid(netid, &knc);
1637     if (err != 0)
1638         return (NULL);
1639     /*
1640      * Do allocations (etc.) outside of mutex,
1641      * and then check again before inserting.
1642      */
1643     newhost = nlm_host_create(name, netid, &knc, addr);
1644     newhost->nh_sysid = nlm_sysid_alloc();
1645     if (newhost->nh_sysid == LM_NOSYSID)

```

```

1646         goto out;
1648     mutex_enter(&g->lock);
1649     host = nlm_host_find_locked(g, netid, addr, &where);
1650     if (host == NULL) {
1651         host = newhost;
1652         newhost = NULL;
1654     /*
1655      * Insert host to the hosts AVL tree that is
1656      * used to lookup by <netid, address> pair.
1657      */
1658     avl_insert(&g->nlm_hosts_tree, host, where);
1660     /*
1661      * Insert host ot the hosts hash table that is
1662      * used to lookup host by sysid.
1663      */
1664     VERIFY(mod_hash_insert(g->nlm_hosts_hash,
1665                          (mod_hash_key_t)(uintptr_t)host->nh_sysid,
1666                          (mod_hash_val_t)host) == 0);
1667     }
1669     mutex_exit(&g->lock);
1671 out:
1672     if (newhost != NULL)
1673         nlm_host_destroy(newhost);
1675     return (host);
1676 }
1678 /*
1679  * Find the NLM host that matches the value of 'sysid'.
1680  * If found, return it with a new ref,
1681  * else return NULL.
1682 */
1683 struct nlm_host *
1684 nlm_host_find_by_sysid(struct nlm_globals *g, sysid_t sysid)
1685 {
1686     struct nlm_host *hostp = NULL;
1688     mutex_enter(&g->lock);
1689     if (g->run_status != NLM_ST_UP)
1690         goto out;
1692     (void) mod_hash_find(g->nlm_hosts_hash,
1693                       (mod_hash_key_t)(uintptr_t)sysid,
1694                       (mod_hash_val_t)&hostp);
1696     if (hostp == NULL)
1697         goto out;
1699     /*
1700      * Host is inuse now. Remove it
1701      * from idle hosts list if needed.
1702      */
1703     if (hostp->nh_flags & NLM_NH_IDLE) {
1704         TAILQ_REMOVE(&g->nlm_idle_hosts, hostp, nh_link);
1705         hostp->nh_flags &= ~NLM_NH_IDLE;
1706     }
1708     hostp->nh_refs++;
1710 out:
1711     mutex_exit(&g->lock);

```

```

1712     return (hostp);
1713 }

1715 /*
1716 * Release the given host.
1717 * I.e. drop a reference that was taken earlier by one of
1718 * the following functions: nlm_host_findcreate(), nlm_host_find(),
1719 * nlm_host_find_by_sysid().
1720 *
1721 * When the very last reference is dropped, host is moved to
1722 * so-called "idle state". All hosts that are in idle state
1723 * have an idle timeout. If timeout is expired, GC thread
1724 * checks whether hosts have any locks and if they haven't
1725 * any, it removes them.
1726 * NOTE: only unused hosts can be in idle state.
1727 */
1728 void
1729 nlm_host_release(struct nlm_globals *g, struct nlm_host *hostp)
1730 {
1731     if (hostp == NULL)
1732         return;

1734     mutex_enter(&g->lock);
1735     ASSERT(hostp->nh_refs > 0);

1737     hostp->nh_refs--;
1738     if (hostp->nh_refs != 0) {
1739         mutex_exit(&g->lock);
1740         return;
1741     }

1743     /*
1744     * The very last reference to the host was dropped,
1745     * thus host is unused now. Set its idle timeout
1746     * and move it to the idle hosts LRU list.
1747     */
1748     hostp->nh_idle_timeout = ddi_get_lbolt() +
1749         SEC_TO_TICK(g->cn_idle_tmo);

1751     ASSERT((hostp->nh_flags & NLM_NH_INIDLE) == 0);
1752     TAILQ_INSERT_TAIL(&g->nlm_idle_hosts, hostp, nh_link);
1753     hostp->nh_flags |= NLM_NH_INIDLE;
1754     mutex_exit(&g->lock);
1755 }

1757 /*
1758 * Unregister this NLM host (NFS client) with the local statd
1759 * due to idleness (no locks held for a while).
1760 */
1761 void
1762 nlm_host_unmonitor(struct nlm_globals *g, struct nlm_host *host)
1763 {
1764     enum clnt_stat stat;

1766     VERIFY(host->nh_refs == 0);
1767     if (!(host->nh_flags & NLM_NH_MONITORED))
1768         return;

1770     host->nh_flags &= ~NLM_NH_MONITORED;
1771     stat = nlm_nsm_unmon(&g->nlm_nsm, host->nh_name);
1772     if (stat != RPC_SUCCESS) {
1773         NLM_WARN("NLM: Failed to contact statd, stat=%d\n", stat);
1774         return;
1775     }
1776 }

```

```

1778 /*
1779 * Ask the local NFS statd to begin monitoring this host.
1780 * It will call us back when that host restarts, using the
1781 * prog,vers,proc specified below, i.e. NLM_SM_NOTIFY1,
1782 * which is handled in nlm_do_notify1().
1783 */
1784 void
1785 nlm_host_monitor(struct nlm_globals *g, struct nlm_host *host, int state)
1786 {
1787     int family;
1788     netobj obj;
1789     enum clnt_stat stat;

1791     if (state != 0 && host->nh_state == 0) {
1792         /*
1793         * This is the first time we have seen an NSM state
1794         * Value for this host. We record it here to help
1795         * detect host reboots.
1796         */
1797         host->nh_state = state;
1798     }

1800     mutex_enter(&host->nh_lock);
1801     if (host->nh_flags & NLM_NH_MONITORED) {
1802         mutex_exit(&host->nh_lock);
1803         return;
1804     }

1806     host->nh_flags |= NLM_NH_MONITORED;
1807     mutex_exit(&host->nh_lock);

1809     /*
1810     * Before we begin monitoring the host register the network address
1811     * associated with this hostname.
1812     */
1813     nlm_netbuf_to_netobj(&host->nh_addr, &family, &obj);
1814     stat = nlm_nsmaddr_reg(&g->nlm_nsm, host->nh_name, family, &obj);
1815     if (stat != RPC_SUCCESS) {
1816         NLM_WARN("Failed to register address, stat=%d\n", stat);
1817         mutex_enter(&g->lock);
1818         host->nh_flags &= ~NLM_NH_MONITORED;
1819         mutex_exit(&g->lock);

1821         return;
1822     }

1824     /*
1825     * Tell statd how to call us with status updates for
1826     * this host. Updates arrive via nlm_do_notify1().
1827     *
1828     * We put our assigned system ID value in the priv field to
1829     * make it simpler to find the host if we are notified of a
1830     * host restart.
1831     */
1832     stat = nlm_nsm_mon(&g->nlm_nsm, host->nh_name, host->nh_sysid);
1833     if (stat != RPC_SUCCESS) {
1834         NLM_WARN("Failed to contact local NSM, stat=%d\n", stat);
1835         mutex_enter(&g->lock);
1836         host->nh_flags &= ~NLM_NH_MONITORED;
1837         mutex_exit(&g->lock);

1839         return;
1840     }
1841 }

1843 int

```

```

1844 nlm_host_get_state(struct nlm_host *hostp)
1845 {
1846
1847     return (hostp->nh_state);
1848 }
1849
1850 /*
1851  * NLM client/server sleeping locks
1852  */
1853
1854 /*
1855  * Register client side sleeping lock.
1856  *
1857  * Our client code calls this to keep information
1858  * about sleeping lock somewhere. When it receives
1859  * grant callback from server or when it just
1860  * needs to remove all sleeping locks from vnode,
1861  * it uses this information for remove/apply lock
1862  * properly.
1863  */
1864 struct nlm_slock *
1865 nlm_slock_register(
1866     struct nlm_globals *g,
1867     struct nlm_host *host,
1868     struct nlm4_lock *lock,
1869     struct vnode *vp)
1870 {
1871     struct nlm_slock *nslp;
1872
1873     nslp = kmem_zalloc(sizeof (*nslp), KM_SLEEP);
1874     cv_init(&nslp->nsl_cond, NULL, CV_DEFAULT, NULL);
1875     nslp->nsl_lock = *lock;
1876     nlm_copy_netobj(&nslp->nsl_fh, &nslp->nsl_lock.fh);
1877     nslp->nsl_state = NLM_SL_BLOCKED;
1878     nslp->nsl_host = host;
1879     nslp->nsl_vp = vp;
1880
1881     mutex_enter(&g->lock);
1882     TAILQ_INSERT_TAIL(&g->nlm_slocks, nslp, nsl_link);
1883     mutex_exit(&g->lock);
1884
1885     return (nslp);
1886 }
1887
1888 /*
1889  * Remove this lock from the wait list and destroy it.
1890  */
1891 void
1892 nlm_slock_unregister(struct nlm_globals *g, struct nlm_slock *nslp)
1893 {
1894     mutex_enter(&g->lock);
1895     TAILQ_REMOVE(&g->nlm_slocks, nslp, nsl_link);
1896     mutex_exit(&g->lock);
1897
1898     kmem_free(nslp->nsl_fh.n_bytes, nslp->nsl_fh.n_len);
1899     cv_destroy(&nslp->nsl_cond);
1900     kmem_free(nslp, sizeof (*nslp));
1901 }
1902
1903 /*
1904  * Wait for a granted callback or cancellation event
1905  * for a sleeping lock.
1906  *
1907  * If a signal interrupted the wait or if the lock
1908  * was cancelled, return EINTR - the caller must arrange to send
1909  * a cancellation to the server.

```

```

1910  *
1911  * If timeout occurred, return ETIMEDOUT - the caller must
1912  * resend the lock request to the server.
1913  *
1914  * On success return 0.
1915  */
1916 int
1917 nlm_slock_wait(struct nlm_globals *g,
1918     struct nlm_slock *nslp, uint_t timeo_secs)
1919 {
1920     clock_t timeo_ticks;
1921     int cv_res, error;
1922
1923     /*
1924      * If the granted message arrived before we got here,
1925      * nw->nw_state will be GRANTED - in that case, don't sleep.
1926      */
1927     cv_res = 1;
1928     timeo_ticks = ddi_get_lbolt() + SEC_TO_TICK(timeo_secs);
1929
1930     mutex_enter(&g->lock);
1931     if (nslp->nsl_state == NLM_SL_BLOCKED) {
1932         cv_res = cv_timedwait_sig(&nslp->nsl_cond,
1933             &g->lock, timeo_ticks);
1934     }
1935
1936     /*
1937      * No matter why we wake up, if the lock was
1938      * cancelled, let the function caller to know
1939      * about it by returning EINTR.
1940      */
1941     if (nslp->nsl_state == NLM_SL_CANCELLED) {
1942         error = EINTR;
1943         goto out;
1944     }
1945
1946     if (cv_res <= 0) {
1947         /* We was woken up either by timeout or interrupt */
1948         error = (cv_res < 0) ? ETIMEDOUT : EINTR;
1949
1950         /*
1951          * The granted message may arrive after the
1952          * interrupt/timeout but before we manage to lock the
1953          * mutex. Detect this by examining nslp.
1954          */
1955         if (nslp->nsl_state == NLM_SL_GRANTED)
1956             error = 0;
1957     } else { /* awoken via cv_signal or didn't block */
1958         error = 0;
1959         VERIFY(nslp->nsl_state == NLM_SL_GRANTED);
1960     }
1961
1962 out:
1963     mutex_exit(&g->lock);
1964     return (error);
1965 }
1966
1967 /*
1968  * Mark client side sleeping lock as granted
1969  * and wake up a process blocked on the lock.
1970  * Called from server side NLM_GRANT handler.
1971  *
1972  * If sleeping lock is found return 0, otherwise
1973  * return ENOENT.
1974  */
1975 int

```

```

1976 nlm_slock_grant(struct nlm_globals *g,
1977     struct nlm_host *hostp, struct nlm4_lock *alock)
1978 {
1979     struct nlm_slock *nslp;
1980     int error = EWOULDBLOCK;

1982     mutex_enter(&g->lock);
1983     TAILQ_FOREACH(nslp, &g->nlm_slocks, nsl_link) {
1984         if ((nslp->nsl_state != NLM_SL_BLOCKED) ||
1985             (nslp->nsl_host != hostp))
1986             continue;

1988         if (alock->svid == nslp->nsl_lock.svid &&
1989             alock->l_offset == nslp->nsl_lock.l_offset &&
1990             alock->l_len == nslp->nsl_lock.l_len &&
1991             alock->fh.n_len == nslp->nsl_lock.fh.n_len &&
1992             bcmp(alock->fh.n_bytes, nslp->nsl_lock.fh.n_bytes,
1993                 nslp->nsl_lock.fh.n_len) == 0) {
1994             nslp->nsl_state = NLM_SL_GRANTED;
1995             cv_broadcast(&nslp->nsl_cond);
1996             error = 0;
1997             break;
1998         }
1999     }

2001     mutex_exit(&g->lock);
2002     return (error);
2003 }

2005 /*
2006  * Register sleeping lock request corresponding to
2007  * flp on the given vhold object.
2008  * On success function returns 0, otherwise (if
2009  * lock request with the same flp is already
2010  * registered) function returns EEXIST.
2011  */
2012 int
2013 nlm_slreq_register(struct nlm_host *hostp, struct nlm_vhold *nvp,
2014     struct flock64 *flp)
2015 {
2016     struct nlm_slreq *slr, *new_slr = NULL;
2017     int ret = EEXIST;

2019     mutex_enter(&hostp->nh_lock);
2020     slr = nlm_slreq_find_locked(hostp, nvp, flp);
2021     if (slr != NULL)
2022         goto out;

2024     mutex_exit(&hostp->nh_lock);
2025     new_slr = kmem_zalloc(sizeof (*slr), KM_SLEEP);
2026     bcopy(flp, &new_slr->nsr_fl, sizeof (*flp));

2028     mutex_enter(&hostp->nh_lock);
2029     slr = nlm_slreq_find_locked(hostp, nvp, flp);
2030     if (slr == NULL) {
2031         slr = new_slr;
2032         new_slr = NULL;
2033         ret = 0;
2035     }
2036     TAILQ_INSERT_TAIL(&nvp->nv_slreqs, slr, nsr_link);

2038 out:
2039     mutex_exit(&hostp->nh_lock);
2040     if (new_slr != NULL)
2041         kmem_free(new_slr, sizeof (*new_slr));

```

```

2043     return (ret);
2044 }

2046 /*
2047  * Unregister sleeping lock request corresponding
2048  * to flp from the given vhold object.
2049  * On success function returns 0, otherwise (if
2050  * lock request corresponding to flp isn't found
2051  * on the given vhold) function returns EWOULDBLOCK.
2052  */
2053 int
2054 nlm_slreq_unregister(struct nlm_host *hostp, struct nlm_vhold *nvp,
2055     struct flock64 *flp)
2056 {
2057     struct nlm_slreq *slr;

2059     mutex_enter(&hostp->nh_lock);
2060     slr = nlm_slreq_find_locked(hostp, nvp, flp);
2061     if (slr == NULL) {
2062         mutex_exit(&hostp->nh_lock);
2063         return (EWOULDBLOCK);
2064     }

2066     TAILQ_REMOVE(&nvp->nv_slreqs, slr, nsr_link);
2067     mutex_exit(&hostp->nh_lock);

2069     kmem_free(slr, sizeof (*slr));
2070     return (0);
2071 }

2073 /*
2074  * Find sleeping lock request on the given vhold object by flp.
2075  */
2076 struct nlm_slreq *
2077 nlm_slreq_find_locked(struct nlm_host *hostp, struct nlm_vhold *nvp,
2078     struct flock64 *flp)
2079 {
2080     struct nlm_slreq *slr = NULL;

2082     ASSERT(MUTEX_HELD(&hostp->nh_lock));
2083     TAILQ_FOREACH(slr, &nvp->nv_slreqs, nsr_link) {
2084         if (slr->nsr_fl.l_start == flp->l_start &&
2085             slr->nsr_fl.l_len == flp->l_len &&
2086             slr->nsr_fl.l_pid == flp->l_pid &&
2087             slr->nsr_fl.l_type == flp->l_type)
2088             break;
2089     }

2091     return (slr);
2092 }

2094 /*
2095  * NLM tracks active share reservations made on the client side.
2096  * It needs to have a track of share reservations for two purposes
2097  * 1) to determine if nlm_host is busy (if it has active locks and/or
2098  * share reservations, it is)
2099  * 2) to recover active share reservations when NLM server reports
2100  * that it has rebooted.
2101  *
2102  * Unfortunately Illumos local share reservations manager (see os/share.c)
2103  * doesn't have an ability to lookup all reservations on the system
2104  * by sysid (like local lock manager) or get all reservations by sysid.
2105  * It tracks reservations per vnode and is able to get/lookup them
2106  * on particular vnode. It's not what NLM needs. Thus it has that ugly
2107  * share reservations tracking scheme.

```

```

2108 */
2110 void
2111 nlm_shres_track(struct nlm_host *hostp, vnode_t *vp, struct shrlock *shrp)
2112 {
2113     struct nlm_shres *nsp, *nsp_new;
2114
2115     /*
2116      * NFS code must fill the s_owner, so that
2117      * s_own_len is never 0.
2118      */
2119     ASSERT(shrp->s_own_len > 0);
2120     nsp_new = nlm_shres_create_item(shrp, vp);
2121
2122     mutex_enter(&hostp->nh_lock);
2123     for (nsp = hostp->nh_shrlist; nsp != NULL; nsp = nsp->ns_next)
2124         if (nsp->ns_vp == vp && nlm_shres_equal(shrp, nsp->ns_shr))
2125             break;
2126
2127     if (nsp != NULL) {
2128         /*
2129          * Found a duplicate. Do nothing.
2130          */
2131         goto out;
2132     }
2133
2134     nsp = nsp_new;
2135     nsp_new = NULL;
2136     nsp->ns_next = hostp->nh_shrlist;
2137     hostp->nh_shrlist = nsp;
2138
2139 out:
2140     mutex_exit(&hostp->nh_lock);
2141     if (nsp_new != NULL)
2142         nlm_shres_destroy_item(nsp_new);
2143 }
2144
2145 void
2146 nlm_shres_untrack(struct nlm_host *hostp, vnode_t *vp, struct shrlock *shrp)
2147 {
2148     struct nlm_shres *nsp, *nsp_prev = NULL;
2149
2150     mutex_enter(&hostp->nh_lock);
2151     nsp = hostp->nh_shrlist;
2152     while (nsp != NULL) {
2153         if (nsp->ns_vp == vp && nlm_shres_equal(shrp, nsp->ns_shr)) {
2154             struct nlm_shres *nsp_del;
2155
2156             nsp_del = nsp;
2157             nsp = nsp->ns_next;
2158             if (nsp_prev != NULL)
2159                 nsp_prev->ns_next = nsp;
2160             else
2161                 hostp->nh_shrlist = nsp;
2162
2163             nlm_shres_destroy_item(nsp_del);
2164             continue;
2165         }
2166         nsp_prev = nsp;
2167     }
2168     nsp_prev = nsp;
2169     nsp = nsp->ns_next;
2170 }
2171
2172 mutex_exit(&hostp->nh_lock);
2173 }

```

```

2175 /*
2176  * Get a _copy_ of the list of all active share reservations
2177  * made by the given host.
2178  * NOTE: the list function returns _must_ be released using
2179  *       nlm_free_shrlist().
2180  */
2181 struct nlm_shres *
2182 nlm_get_active_shres(struct nlm_host *hostp)
2183 {
2184     struct nlm_shres *nsp, *nslst = NULL;
2185
2186     mutex_enter(&hostp->nh_lock);
2187     for (nsp = hostp->nh_shrlist; nsp != NULL; nsp = nsp->ns_next) {
2188         struct nlm_shres *nsp_new;
2189
2190         nsp_new = nlm_shres_create_item(nsp->ns_shr, nsp->ns_vp);
2191         nsp_new->ns_next = nslst;
2192         nslst = nsp_new;
2193     }
2194
2195     mutex_exit(&hostp->nh_lock);
2196     return (nslst);
2197 }
2198
2199 /*
2200  * Free memory allocated for the active share reservations
2201  * list created by nlm_get_active_shres() function.
2202  */
2203 void
2204 nlm_free_shrlist(struct nlm_shres *nslst)
2205 {
2206     struct nlm_shres *nsp;
2207
2208     while (nslst != NULL) {
2209         nsp = nslst;
2210         nslst = nslst->ns_next;
2211
2212         nlm_shres_destroy_item(nsp);
2213     }
2214 }
2215
2216 static bool_t
2217 nlm_shres_equal(struct shrlock *shrpl, struct shrlock *shrp2)
2218 {
2219     if (shrpl->s_sysid == shrp2->s_sysid &&
2220         shrpl->s_pid == shrp2->s_pid &&
2221         shrpl->s_own_len == shrp2->s_own_len &&
2222         bcmp(shrpl->s_owner, shrp2->s_owner,
2223             shrpl->s_own_len) == 0)
2224         return (TRUE);
2225
2226     return (FALSE);
2227 }
2228
2229 static struct nlm_shres *
2230 nlm_shres_create_item(struct shrlock *shrp, vnode_t *vp)
2231 {
2232     struct nlm_shres *nsp;
2233
2234     nsp = kmem_alloc(sizeof (*nsp), KM_SLEEP);
2235     nsp->ns_shr = kmem_alloc(sizeof (*shrp), KM_SLEEP);
2236     bcopy(shrp, nsp->ns_shr, sizeof (*shrp));
2237     nsp->ns_shr->s_owner = kmem_alloc(shrp->s_own_len, KM_SLEEP);
2238     bcopy(shrp->s_owner, nsp->ns_shr->s_owner, shrp->s_own_len);
2239     nsp->ns_vp = vp;

```

```

2241     return (nsp);
2242 }

2244 static void
2245 nlm_shres_destroy_item(struct nlm_shres *nsp)
2246 {
2247     kmem_free(nsp->ns_shr->s_owner,
2248             nsp->ns_shr->s_own_len);
2249     kmem_free(nsp->ns_shr, sizeof (struct shrlock));
2250     kmem_free(nsp, sizeof (*nsp));
2251 }

2253 /*
2254  * Called by klmmod.c when lockd adds a network endpoint
2255  * on which we should begin RPC services.
2256  */
2257 int
2258 nlm_svc_add_ep(struct file *fp, const char *netid, struct knetconfig *knc)
2259 {
2260     SVCMASTERXPRT *xpirt = NULL;
2261     int error;

2263     error = svc_tli_kcreate(fp, 0, (char *)netid, NULL, &xpirt,
2264             &nsm_sct, NULL, NLM_SVCPPOOL_ID, FALSE);
2265     if (error != 0)
2266         return (error);

2268     (void) nlm_knc_to_netid(knc);
2269     return (0);
2270 }

2272 /*
2273  * Start NLM service.
2274  */
2275 int
2276 nlm_svc_starting(struct nlm_globals *g, struct file *fp,
2277     const char *netid, struct knetconfig *knc)
2278 {
2279     int error;
2280     enum clnt_stat stat;

2282     VERIFY(g->run_status == NLM_ST_STARTING);
2283     VERIFY(g->nsm_gc_thread == NULL);

2285     error = nlm_nsm_init_local(&g->nsm_nsm);
2286     if (error != 0) {
2287         NLM_ERR("Failed to initialize NSM handler "
2288             "(error=%d)\n", error);
2289         g->run_status = NLM_ST_DOWN;
2290         return (error);
2291     }

2293     error = EIO;

2295     /*
2296     * Create an NLM garbage collector thread that will
2297     * clean up stale vholds and hosts objects.
2298     */
2299     g->nsm_gc_thread = zthread_create(NULL, 0, nlm_gc,
2300         g, 0, minclsyspri);

2302     /*
2303     * Send SIMU_CRASH to local statd to report that
2304     * NLM started, so that statd can report other hosts
2305     * about NLM state change.

```

```

2306     */

2308     stat = nlm_nsm_simu_crash(&g->nsm_nsm);
2309     if (stat != RPC_SUCCESS) {
2310         NLM_ERR("Failed to connect to local statd "
2311             "(rpcerr=%d)\n", stat);
2312         goto shutdown_lm;
2313     }

2315     stat = nlm_nsm_stat(&g->nsm_nsm, &g->nsm_state);
2316     if (stat != RPC_SUCCESS) {
2317         NLM_ERR("Failed to get the status of local statd "
2318             "(rpcerr=%d)\n", stat);
2319         goto shutdown_lm;
2320     }

2322     g->grace_threshold = ddi_get_lbolt() +
2323         SEC_TO_TICK(g->grace_period);

2325     /* Register endpoint used for communications with local NLM */
2326     error = nlm_svc_add_ep(fp, netid, knc);
2327     if (error != 0)
2328         goto shutdown_lm;

2330     (void) svc_pool_control(NLM_SVCPPOOL_ID,
2331         SVCPSET_SHUTDOWN_PROC, (void *)nlm_pool_shutdown);
2332     g->run_status = NLM_ST_UP;
2333     return (0);

2335 shutdown_lm:
2336     mutex_enter(&g->lock);
2337     g->run_status = NLM_ST_STOPPING;
2338     mutex_exit(&g->lock);

2340     nlm_svc_stopping(g);
2341     return (error);
2342 }

2344 /*
2345  * Called when the server pool is destroyed, so that
2346  * all transports are closed and no any server threads
2347  * exist.
2348  *
2349  * Just call lm_shutdown() to shut NLM down properly.
2350  */
2351 static void
2352 nlm_pool_shutdown(void)
2353 {
2354     (void) lm_shutdown();
2355 }

2357 /*
2358  * Stop NLM service, cleanup all resources
2359  * NLM owns at the moment.
2360  *
2361  * NOTE: NFS code can call NLM while it's
2362  * stopping or even if it's shut down. Any attempt
2363  * to lock file either on client or on the server
2364  * will fail if NLM isn't in NLM_ST_UP state.
2365  */
2366 void
2367 nlm_svc_stopping(struct nlm_globals *g)
2368 {
2369     mutex_enter(&g->lock);
2370     ASSERT(g->run_status == NLM_ST_STOPPING);

```



```

2372 /*
2373  * Ask NLM GC thread to exit and wait until it dies.
2374  */
2375 cv_signal(&g->nlm_gc_sched_cv);
2376 while (g->nlm_gc_thread != NULL)
2377     cv_wait(&g->nlm_gc_finish_cv, &g->lock);
2379 mutex_exit(&g->lock);
2381 /*
2382  * Cleanup locks owned by NLM hosts.
2383  * NOTE: New hosts won't be created while
2384  * NLM is stopping.
2385  */
2386 while (!avl_is_empty(&g->nlm_hosts_tree)) {
2387     struct nlm_host *hostp;
2388     int busy_hosts = 0;
2390     /*
2391      * Iterate through all NLM hosts in the system
2392      * and drop the locks they own by force.
2393      */
2394     hostp = avl_first(&g->nlm_hosts_tree);
2395     while (hostp != NULL) {
2396         /* Cleanup all client and server side locks */
2397         nlm_client_cancel_all(g, hostp);
2398         nlm_host_notify_server(hostp, 0);
2400         mutex_enter(&hostp->nh_lock);
2401         nlm_host_gc_vholds(hostp);
2402         if (hostp->nh_refs > 0 || nlm_host_has_locks(hostp)) {
2403             /*
2404              * Oh, it seems the host is still busy, let
2405              * it some time to release and go to the
2406              * next one.
2407              */
2409             mutex_exit(&hostp->nh_lock);
2410             hostp = AVL_NEXT(&g->nlm_hosts_tree, hostp);
2411             busy_hosts++;
2412             continue;
2413         }
2415         mutex_exit(&hostp->nh_lock);
2416         hostp = AVL_NEXT(&g->nlm_hosts_tree, hostp);
2417     }
2419     /*
2420      * All hosts go to nlm_idle_hosts list after
2421      * all locks they own are cleaned up and last refernces
2422      * were dropped. Just destroy all hosts in nlm_idle_hosts
2423      * list, they can not be removed from there while we're
2424      * in stopping state.
2425      */
2426     while ((hostp = TAILQ_FIRST(&g->nlm_idle_hosts)) != NULL) {
2427         nlm_host_unregister(g, hostp);
2428         nlm_host_destroy(hostp);
2429     }
2431     if (busy_hosts > 0) {
2432         /*
2433          * There're some hosts that weren't cleaned
2434          * up. Probably they're in resource cleanup
2435          * process. Give them some time to do drop
2436          * references.
2437          */

```

```

2438         delay(MSEC_TO_TICK(500));
2439     }
2440 }
2442 ASSERT(TAILQ_EMPTY(&g->nlm_slocks));
2444 nlm_nsm_fini(&g->nlm_nsm);
2445 g->lockd_pid = 0;
2446 g->run_status = NLM_ST_DOWN;
2447 }
2449 /*
2450  * Returns TRUE if the given vnode has
2451  * any active or sleeping locks.
2452  */
2453 int
2454 nlm_vp_active(const vnode_t *vp)
2455 {
2456     struct nlm_globals *g;
2457     struct nlm_host *hostp;
2458     struct nlm_vhold *nvp;
2459     int active = 0;
2461     g = zone_getspecific(nlm_zone_key, curzone);
2463     /*
2464      * Server side NLM has locks on the given vnode
2465      * if there exist a vhold object that holds
2466      * the given vnode "vp" in one of NLM hosts.
2467      */
2468     mutex_enter(&g->lock);
2469     hostp = avl_first(&g->nlm_hosts_tree);
2470     while (hostp != NULL) {
2471         mutex_enter(&hostp->nh_lock);
2472         nvp = nlm_vhold_find_locked(hostp, vp);
2473         mutex_exit(&hostp->nh_lock);
2474         if (nvp != NULL) {
2475             active = 1;
2476             break;
2477         }
2479         hostp = AVL_NEXT(&g->nlm_hosts_tree, hostp);
2480     }
2482     mutex_exit(&g->lock);
2483     return (active);
2484 }
2486 /*
2487  * Called right before NFS export is going to
2488  * dissappear. The function finds all vnodes
2489  * belonging to the given export and cleans
2490  * all remote locks and share reservations
2491  * on them.
2492  */
2493 void
2494 nlm_unexport(struct exportinfo *exi)
2495 {
2496     struct nlm_globals *g;
2497     struct nlm_host *hostp;
2499     g = zone_getspecific(nlm_zone_key, curzone);
2501     mutex_enter(&g->lock);
2502     hostp = avl_first(&g->nlm_hosts_tree);
2503     while (hostp != NULL) {

```

```

2504     struct nlm_vhold *nvp;

2506     mutex_enter(&hostp->nh_lock);
2507     TAILQ_FOREACH(nvp, &hostp->nh_vholds_list, nv_link) {
2508         vnode_t *vp;

2510         nvp->nv_refcnt++;
2511         mutex_exit(&hostp->nh_lock);

2513         vp = nvp->nv_vp;

2515         if (!EQFSID(&exi->exi_fsid, &vp->v_vfsp->vfs_fsid))
2516             goto next_iter;

2518         /*
2519          * Ok, it we found out that vnode vp is under
2520          * control by the exportinfo exi, now we need
2521          * to drop all locks from this vnode, let's
2522          * do it.
2523          */
2524         nlm_vhold_clean(nvp, hostp->nh_sysid);

2526     next_iter:
2527         mutex_enter(&hostp->nh_lock);
2528         nvp->nv_refcnt--;
2529     }

2531     mutex_exit(&hostp->nh_lock);
2532     hostp = AVL_NEXT(&g->nlm_hosts_tree, hostp);
2533 }

2535     mutex_exit(&g->lock);
2536 }

2538 /*
2539  * Allocate new unique sysid.
2540  * In case of failure (no available sysids)
2541  * return LM_NOSYSID.
2542  */
2543 sysid_t
2544 nlm_sysid_alloc(void)
2545 {
2546     sysid_t ret_sysid = LM_NOSYSID;

2548     rw_enter(&lm_lck, RW_WRITER);
2549     if (nlm_sysid_nidx > LM_SYSID_MAX)
2550         nlm_sysid_nidx = LM_SYSID;

2552     if (!BT_TEST(nlm_sysid_bmap, nlm_sysid_nidx)) {
2553         BT_SET(nlm_sysid_bmap, nlm_sysid_nidx);
2554         ret_sysid = nlm_sysid_nidx++;
2555     } else {
2556         index_t id;

2558         id = bt_avalbit(nlm_sysid_bmap, NLM_BMAP_NITEMS);
2559         if (id > 0) {
2560             nlm_sysid_nidx = id + 1;
2561             ret_sysid = id;
2562             BT_SET(nlm_sysid_bmap, id);
2563         }
2564     }

2566     rw_exit(&lm_lck);
2567     return (ret_sysid);
2568 }

```

```

2570 void
2571 nlm_sysid_free(sysid_t sysid)
2572 {
2573     ASSERT(sysid >= LM_SYSID && sysid <= LM_SYSID_MAX);

2575     rw_enter(&lm_lck, RW_WRITER);
2576     ASSERT(BT_TEST(nlm_sysid_bmap, sysid));
2577     BT_CLEAR(nlm_sysid_bmap, sysid);
2578     rw_exit(&lm_lck);
2579 }

2581 /*
2582  * Return true if the request came from a local caller.
2583  * By necessity, this "knows" the netid names invented
2584  * in lm_svc() and nlm_netid_from_knetconfig().
2585  */
2586 bool_t
2587 nlm_caller_is_local(SVCXPRT *transp)
2588 {
2589     char *netid;
2590     struct netbuf *rtaddr;

2592     netid = svc_getnetid(transp);
2593     rtaddr = svc_getrpccaller(transp);

2595     if (netid == NULL)
2596         return (FALSE);

2598     if (strcmp(netid, "ticlts") == 0 ||
2599         strcmp(netid, "ticotsord") == 0)
2600         return (TRUE);

2602     if (strcmp(netid, "tcp") == 0 || strcmp(netid, "udp") == 0) {
2603         struct sockaddr_in *sin = (void *)rtaddr->buf;
2604         if (sin->sin_addr.s_addr == htonl(INADDR_LOOPBACK))
2605             return (TRUE);
2606     }
2607     if (strcmp(netid, "tcp6") == 0 || strcmp(netid, "udp6") == 0) {
2608         struct sockaddr_in6 *sin6 = (void *)rtaddr->buf;
2609         if (IN6_IS_ADDR_LOOPBACK(&sin6->sin6_addr))
2610             return (TRUE);
2611     }

2613     return (FALSE); /* unknown transport */
2614 }

2616 /*
2617  * Get netid string correspondig to the given knetconfig.
2618  * If not done already, save knc->knc_rdev in our table.
2619  */
2620 const char *
2621 nlm_knc_to_netid(struct knetconfig *knc)
2622 {
2623     int i;
2624     dev_t rdev;
2625     struct nlm_knc *knc;
2626     const char *netid = NULL;

2628     rw_enter(&lm_lck, RW_READER);
2629     for (i = 0; i < NLM_KNCS; i++) {
2630         nc = &nlm_netconfigs[i];

2632         if (nc->n_knc.knc_semantics == knc->knc_semantics &&
2633             strcmp(nc->n_knc.knc_protomfly,
2634                 knc->knc_protomfly) == 0) {
2635             netid = nc->n_netid;

```

```

2636         rdev = nc->n_knc.knc_rdev;
2637         break;
2638     }
2639 }
2640 rw_exit(&lm_lck);

2642 if (netid != NULL && rdev == NODEV) {
2643     rw_enter(&lm_lck, RW_WRITER);
2644     if (nc->n_knc.knc_rdev == NODEV)
2645         nc->n_knc.knc_rdev = knc->knc_rdev;
2646     rw_exit(&lm_lck);
2647 }

2649 return (netid);
2650 }

2652 /*
2653  * Get a knetconfig corresponding to the given netid.
2654  * If there's no knetconfig for this netid, ENOENT
2655  * is returned.
2656  */
2657 int
2658 nlm_knc_from_netid(const char *netid, struct knetconfig *knc)
2659 {
2660     int i, ret;

2662     ret = ENOENT;
2663     for (i = 0; i < NLM_KNCS; i++) {
2664         struct nlm_knc *nknc;

2666         nknc = &nlm_netconfigs[i];
2667         if (strcmp(netid, nknc->n_netid) == 0 &&
2668             nknc->n_knc.knc_rdev != NODEV) {
2669             *knc = nknc->n_knc;
2670             ret = 0;
2671             break;
2672         }
2673     }

2675     return (ret);
2676 }

2678 void
2679 nlm_cprsuspend(void)
2680 {
2681     struct nlm_globals *g;

2683     rw_enter(&lm_lck, RW_READER);
2684     TAILQ_FOREACH(g, &nlm_zones_list, nlm_link)
2685         nlm_suspend_zone(g);

2687     rw_exit(&lm_lck);
2688 }

2690 void
2691 nlm_cprresume(void)
2692 {
2693     struct nlm_globals *g;

2695     rw_enter(&lm_lck, RW_READER);
2696     TAILQ_FOREACH(g, &nlm_zones_list, nlm_link)
2697         nlm_resume_zone(g);

2699     rw_exit(&lm_lck);
2700 }

```

```

2702 static void
2703 nlm_nsm_clnt_init(CLIENT *clnt, struct nlm_nsm *nsm)
2704 {
2705     (void) clnt_tli_kinit(clnt, &nsm->ns_knc, &nsm->ns_addr, 0,
2706         NLM_RPC_RETRIES, kcred);
2707 }

2709 static void
2710 nlm_netbuf_to_netobj(struct netbuf *addr, int *family, netobj *obj)
2711 {
2712     /* LINTED pointer alignment */
2713     struct sockaddr *sa = (struct sockaddr *)addr->buf;

2715     *family = sa->sa_family;

2717     switch (sa->sa_family) {
2718     case AF_INET: {
2719         /* LINTED pointer alignment */
2720         struct sockaddr_in *sin = (struct sockaddr_in *)sa;

2722         obj->n_len = sizeof (sin->sin_addr);
2723         obj->n_bytes = (char *)&sin->sin_addr;
2724         break;
2725     }

2727     case AF_INET6: {
2728         /* LINTED pointer alignment */
2729         struct sockaddr_in6 *sin6 = (struct sockaddr_in6 *)sa;

2731         obj->n_len = sizeof (sin6->sin6_addr);
2732         obj->n_bytes = (char *)&sin6->sin6_addr;
2733         break;
2734     }

2736     default:
2737         VERIFY(0);
2738         break;
2739     }
2740 }

```

```

*****
23395 Fri Aug 16 09:04:52 2013
new/usr/src/uts/common/klm/nlm_impl.h
195 Need replacement for nfs/lockd+klm
*****
1 /*
2  * Copyright (c) 2008 Isilon Inc http://www.isilon.com/
3  * Authors: Doug Rabson <dfr@rabson.org>
4  * Developed with Red Inc: Alfred Perlstein <alfred@freebsd.org>
5  *
6  * Redistribution and use in source and binary forms, with or without
7  * modification, are permitted provided that the following conditions
8  * are met:
9  * 1. Redistributions of source code must retain the above copyright
10 * notice, this list of conditions and the following disclaimer.
11 * 2. Redistributions in binary form must reproduce the above copyright
12 * notice, this list of conditions and the following disclaimer in the
13 * documentation and/or other materials provided with the distribution.
14 *
15 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
16 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
17 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
18 * ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
19 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
20 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
21 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
22 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
23 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
24 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
25 * SUCH DAMAGE.
26 *
27 * $FreeBSD$
28 */

30 /*
31  * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
32  * Copyright (c) 2012 by Delphix. All rights reserved.
33  */

35 /*
36  * NFS Lock Manager (NLM) private declarations, etc.
37  *
38  * Source code derived from FreeBSD nlm.h
39  */

41 #ifndef _NLM_NLM_H
42 #define _NLM_NLM_H

44 #include <sys/cmn_err.h>
45 #include <sys/queue.h>
46 #include <sys/modhash.h>
47 #include <sys/avl.h>

49 #define RPC_MSGOUT(args...) cmn_err(CE_NOTE, args)
50 #define NLM_ERR(...) cmn_err(CE_NOTE, __VA_ARGS__)
51 #define NLM_WARN(...) cmn_err(CE_WARN, __VA_ARGS__)

53 #ifndef SEEK_SET
54 #define SEEK_SET 0
55 #endif
56 #ifndef SEEK_CUR
57 #define SEEK_CUR 1
58 #endif
59 #ifndef SEEK_END
60 #define SEEK_END 2
61 #endif

```

```

63 /*
64  * Maximum offset supported by NLM calls using the older
65  * (32-bit) versions of the protocol.
66  */
67 #define MAX_UOFF32 0xffffffffULL

69 struct nlm_host;
70 struct vnode;
71 struct exportinfo;
72 struct shrlock;
73 struct _kthread;

75 /*
76  * How to read the code: probably the best point to start
77  * is the nlm_host structure that is sort of most major
78  * structure in klmmod. nlm_host is closely tied with all
79  * other NLM structures.
80  *
81  * There're three major locks we use inside NLM:
82  * 1) Global read-write lock (lm_lck) that is used to
83  * protect operations with sysid allocation and
84  * management of zone globals structures for each
85  * zone.
86  * 2) Zone global lock: (nlm_globals->lock) is a mutex
87  * used to protect all operations inside particular
88  * zone.
89  * 3) Host's lock: (nlm_host->nh_lock) is per-host mutex
90  * used to protect host's internal fields and all
91  * operations with the given host.
92  *
93  * Locks order _must_ obey the following scheme:
94  * lm_lck then nlm_globals->lock then nlm_host->nh_lock
95  *
96  * Locks:
97  * (g) locked by lm_lck
98  * (z) locked by nlm_globals->lock
99  * (l) locked by host->nh_lock
100 * (c) const until freeing
101 */

103 /*
104  * Callback functions for nlm_do_lock() and others.
105  *
106  * Calls to nlm_do_lock are unusual, because it needs to handle
107  * the reply itself, instead of letting it happen the normal way.
108  * It also needs to make an RPC call _back_ to the client when a
109  * blocked lock request completes.
110  *
111  * We pass three callback functions to nlm_do_lock:
112  * nlm_reply_cb: send a normal RPC reply
113  * nlm_res_cb: do a _res (message style) RPC (call)
114  * nlm_testargs_cb: do a "granted" RPC call (after blocking)
115  * Only one of the 1st or 2nd is used.
116  * The 3rd is used only for blocking
117  *
118  * We also use callback functions for all the _msg variants
119  * of the NLM svc calls, where the reply is a reverse call.
120  * The nlm_testres_cb is used by the _test_msg svc calls.
121  * The nlm_res_cb type is used by the other _msg calls.
122  */
123 typedef bool_t (*nlm_reply_cb)(SVCKPRT *, nlm4_res *);
124 typedef enum clnt_stat (*nlm_res_cb)(nlm4_res *, void *, CLIENT *);
125 typedef enum clnt_stat (*nlm_testargs_cb)(nlm4_testargs *, void *, CLIENT *);
126 typedef enum clnt_stat (*nlm_testres_cb)(nlm4_testres *, void *, CLIENT *);

```

```

128 /*
129  * NLM sleeping lock request.
130  *
131  * Sleeping lock requests are server side only objects
132  * that are created when client asks server to add new
133  * sleeping lock and when this lock needs to block.
134  * Server keeps a track of these requests in order to be
135  * able to cancel them or clean them up.
136  *
137  * Sleeping lock requests are closely tiled with particular
138  * vnode or, strictly speaking, NLM vhold object that holds
139  * the vnode.
140  *
141  * struct nlm_slreq:
142  *   nsr_fl: an information about file lock
143  *   nsr_link: a list node to store lock requests
144  *           in vhold object.
145  */
146 struct nlm_slreq {
147     struct flock64      nsr_fl;
148     TAILQ_ENTRY(nlm_slreq) nsr_link;
149 };
150 TAILQ_HEAD(nlm_slreq_list, nlm_slreq);

152 /*
153  * NLM vhold object is a sort of wrapper on vnodes remote
154  * clients have locked (or added share reservation)
155  * on NLM server. Vhold keeps vnode held (by VN_HOLD())
156  * while vnode has any locks or shares made by parent host.
157  * Vholds are used for two purposes:
158  * 1) Hold vnode (with VN_HOLD) while it has any locks;
159  * 2) Keep a track of all vnodes remote host touched
160  *    with lock/share operations on NLM server, so that NLM
161  *    can know what vnodes are potentially locked;
162  *
163  * Vholds are used on server side only. For server side it's really
164  * important to keep vnodes held while they potentially have
165  * any locks/shares. In contrast, it's not important for client
166  * side at all. When particular vnode comes to the NLM client side
167  * code, it's already held (VN_HOLD) by the process calling
168  * lock/share function (it's referenced because client calls open())
169  * before making locks or shares).
170  *
171  * Each NLM host object has a collection of vholds associated
172  * with vnodes host touched earlier by adding locks or shares.
173  * Having this collection allows us to decide if host is still
174  * in use. When it has any vhold objects it's considered to be
175  * in use. Otherwise we're free to destroy it.
176  *
177  * Vholds are destroyed by the NLM garbage collector thread that
178  * periodically checks whether they have any locks or shares.
179  * Checking occurs when parent host is untouched by client
180  * or server for some period of time.
181  *
182  * struct nlm_vhold:
183  *   nv_vp: a pointer to vnode that is hold by given nlm_vhold
184  *   nv_refcnt: reference counter (non zero when vhold is inuse)
185  *   nv_slreqs: sleeping lock requests that were made on the nv_vp
186  *   nv_link: list node to store vholds in host's nh_vnodes_list
187  */
188 struct nlm_vhold {
189     vnode_t      *nv_vp; /* (c) */
190     int          nv_refcnt; /* (1) */
191     struct nlm_slreq_list nv_slreqs; /* (1) */
192     TAILQ_ENTRY(nlm_vhold) nv_link; /* (1) */
193 };

```

```

194 TAILQ_HEAD(nlm_vhold_list, nlm_vhold);

196 /*
197  * Client side sleeping lock state.
198  * - NLM_SL_BLOCKED: some thread is blocked on this lock
199  * - NLM_SL_GRANTED: server granted us the lock
200  * - NLM_SL_CANCELLED: the lock is cancelled (i.e. invalid/inactive)
201  */
202 typedef enum nlm_slock_state {
203     NLM_SL_UNKNOWN = 0,
204     NLM_SL_BLOCKED,
205     NLM_SL_GRANTED,
206     NLM_SL_CANCELLED
207 } nlm_slock_state_t;

209 /*
210  * A client side sleeping lock request (set by F_SETLKW)
211  * stored in nlm_slocks collection of nlm_globals.
212  *
213  * struct nlm_slock
214  *   nsl_state: Sleeping lock state.
215  *           (see nlm_slock_state for more information)
216  *   nsl_cond: Condvar that is used when sleeping lock
217  *           needs to wait for a GRANT callback
218  *           or cancellation event.
219  *   nsl_lock: nlm4_lock structure that is sent to the server
220  *   nsl_fh: Filehandle that corresponds to nw_vp
221  *   nsl_host: A host owning this sleeping lock
222  *   nsl_vp: A vnode sleeping lock is waiting on.
223  *   nsl_link: A list node for nlm_globals->nlm_slocks list.
224  */
225 struct nlm_slock {
226     nlm_slock_state_t      nsl_state; /* (z) */
227     kcondvar_t             nsl_cond; /* (z) */
228     nlm4_lock               nsl_lock; /* (c) */
229     struct netobj           nsl_fh; /* (c) */
230     struct nlm_host         *nsl_host; /* (c) */
231     struct vnode            *nsl_vp; /* (c) */
232     TAILQ_ENTRY(nlm_slock) nsl_link; /* (z) */
233 };
234 TAILQ_HEAD(nlm_slock_list, nlm_slock);

236 /*
237  * Share reservation description. NLM tracks all active
238  * share reservations made by the client side, so that
239  * they can be easily recovered if remote NLM server
240  * reboots. Share reservations tracking is also useful
241  * when NLM needs to determine whether host owns any
242  * resources on the system and can't be destroyed.
243  *
244  * nlm_shres:
245  *   ns_shr: share reservation description
246  *   ns_vp: a pointer to vnode where share reservation is located
247  *   ns_next: next nlm_shres instance (or NULL if next item isn't
248  *           present).
249  */
250 struct nlm_shres {
251     struct shrlock          *ns_shr;
252     vnode_t                 *ns_vp;
253     struct nlm_shres        *ns_next;
254 };

256 /*
257  * NLM RPC handle object.
258  *
259  * In kRPC subsystem it's unsafe to use one RPC handle by

```

```

260 * several threads simultaneously. It was designed so that
261 * each thread has to create an RPC handle that it'll use.
262 * RPC handle creation can be quite expensive operation, especially
263 * with session oriented protocols (such as TCP) that need to
264 * establish session at first. NLM RPC handle object is a sort of
265 * wrapper on kRPC handle object that can be cached and used in
266 * future. We store all created RPC handles for given host in a
267 * host's RPC handles cache, so that to make new requests threads
268 * can simply take ready objects from the cache. That improves
269 * NLM performance.
270 *
271 * nlm_rpc_t:
272 *   nr_handle: a kRPC handle itself.
273 *   nr_vers: a version of NLM protocol kRPC handle was
274 *   created for.
275 *   nr_link: a list node to store NLM RPC handles in the host
276 *   RPC handles cache.
277 */
278 typedef struct nlm_rpc {
279     CLIENT      *nr_handle;          /* (l) */
280     rpcvers_t  nr_vers;             /* (c) */
281     TAILQ_ENTRY(nlm_rpc) nr_link;    /* (l) */
282 } nlm_rpc_t;
283 TAILQ_HEAD(nlm_rpc_list, nlm_rpc);
284
285 /*
286 * Describes the state of NLM host's RPC binding.
287 * RPC binding can be in one of three states:
288 * 1) NRPCB_NEED_UPDATE:
289 *   Binding is either not initialized or stale.
290 * 2) NRPCB_UPDATE_INPROGRESS:
291 *   When some thread updates host's RPC binding,
292 *   it sets binding's state to NRPCB_UPDATE_INPROGRESS
293 *   which denotes that other threads must wait until
294 *   update process is finished.
295 * 3) NRPCB_UPDATED:
296 *   Denotes that host's RPC binding is both initialized
297 *   and fresh.
298 */
299 enum nlm_rpcb_state {
300     NRPCB_NEED_UPDATE = 0,
301     NRPCB_UPDATE_INPROGRESS,
302     NRPCB_UPDATED
303 };
304
305 /*
306 * NLM host flags
307 */
308 #define NLM_NH_MONITORED 0x01
309 #define NLM_NH_RECLAIM  0x02
310 #define NLM_NH_INIDLE   0x04
311 #define NLM_NH_SUSPEND  0x08
312
313 /*
314 * NLM host object is the most major structure in NLM.
315 * It identifies remote client or remote server or both.
316 * NLM host object keep a track of all vnodes client/server
317 * locked and all sleeping locks it has. All lock/unlock
318 * operations are done using host object.
319 *
320 * nlm_host:
321 *   nh_lock: a mutex protecting host object fields
322 *   nh_refs: reference counter. Identifies how many threads
323 *   uses this host object.
324 *   nh_link: a list node for keeping host in zone-global list.
325 *   nh_by_addr: an AVL tree node for keeping host in zone-global tree.

```

```

326 *   Host can be looked up in the tree by <netid, address>
327 *   pair.
328 *   nh_name: host name.
329 *   nh_netid: netid string identifying type of transport host uses.
330 *   nh_knc: host's knetconfig (used by kRPC subsystem).
331 *   nh_addr: host's address (either IPv4 or IPv6).
332 *   nh_sysid: unique sysid associated with this host.
333 *   nh_state: last seen host's state reported by NSM.
334 *   nh_flags: ORED host flags.
335 *   nh_idle_timeout: host idle timeout. When expired host is freed.
336 *   nh_recl_cv: condition variable used for reporting that reclamation
337 *   process is finished.
338 *   nh_rpcb_cv: condition variable that is used to make sure
339 *   that only one thread renews host's RPC binding.
340 *   nh_rpcb_ustat: error code returned by RPC binding update operation.
341 *   nh_rpcb_state: host's RPC binding state (see enum nlm_rpcb_state
342 *   for more details).
343 *   nh_rpcbc: host's RPC handles cache.
344 *   nh_vholds_by_vp: a hash table of all vholds host owns. (used for lookup)
345 *   nh_vholds_list: a linked list of all vholds host owns. (used for iteration)
346 *   nh_shrlist: a list of all active share reasevations on the client side.
347 *   nh_reclaimer: a pointer to reclamation thread (kthread_t)
348 *   NULL if reclamation thread doesn't exist
349 */
350 struct nlm_host {
351     kmutex_t          nh_lock;          /* (c) */
352     volatile uint_t  nh_refs;          /* (z) */
353     TAILQ_ENTRY(nlm_host) nh_link;     /* (z) */
354     avl_node_t        nh_by_addr;      /* (z) */
355     char              *nh_name;        /* (c) */
356     char              *nh_netid;       /* (c) */
357     struct knetconfig nh_knc;          /* (c) */
358     struct netbuf      nh_addr;        /* (c) */
359     sysid_t           nh_sysid;        /* (c) */
360     int32_t           nh_state;        /* (z) */
361     clock_t           nh_idle_timeout; /* (z) */
362     uint8_t           nh_flags;        /* (z) */
363     kcondvar_t        nh_recl_cv;      /* (z) */
364     kcondvar_t        nh_rpcb_cv;      /* (l) */
365     enum clnt_stat     nh_rpcb_ustat;   /* (l) */
366     enum nlm_rpcb_state nh_rpcb_state; /* (l) */
367     struct nlm_rpc_list nh_rpcbc;      /* (l) */
368     mod_hash_t        *nh_vholds_by_vp; /* (l) */
369     struct nlm_vhold_list nh_vholds_list; /* (l) */
370     struct nlm_shres    *nh_shrlist;   /* (l) */
371     kthread_t          *nh_reclaimer;   /* (l) */
372 };
373 TAILQ_HEAD(nlm_host_list, nlm_host);
374
375 /*
376 * nlm_nsm structure describes RPC client handle that can be
377 * used to communicate with local NSM via kRPC.
378 *
379 * We need to wrap handle with nlm_nsm structure because kRPC
380 * can not share one handle between several threads. It's assumed
381 * that NLM uses only one NSM handle per zone, thus all RPC operations
382 * on NSM's handle are serialized using nlm_nsm->sem semaphore.
383 *
384 * nlm_nsm also contains refcnt field used for reference counting.
385 * It's used because there exist a possibility of simultaneous
386 * execution of NLM shutdown operation and host monitor/unmonitor
387 * operations.
388 *
389 * struct nlm_nsm:
390 *   ns_sem: a semaphore for serialization network operations to statd
391 *   ns_knc: a kneconfig describing transport that is used for communication

```

```

392 * ns_addr: an address of local statd we're talking to
393 * ns_handle: an RPC handle used for talking to local statd using the status
394 * monitor protocol (SM_PROG)
395 * ns_addr_handle: an RPC handle used for talking to local statd using the
396 * address registration protocol (NSM_ADDR_PROGRAM)
397 */
398 struct nlm_nsm {
399     ksema_t          ns_sema;
400     struct knetconfig ns_knc;          /* (c) */
401     struct netbuf    ns_addr;         /* (c) */
402     CLIENT          *ns_handle;      /* (c) */
403     CLIENT          *ns_addr_handle; /* (c) */
404 };

406 /*
407 * Could use flock.h flk_nlm_status_t instead, but
408 * prefer our own enum with initial zero...
409 */
410 typedef enum {
411     NLM_ST_DOWN = 0,
412     NLM_ST_STOPPING,
413     NLM_ST_UP,
414     NLM_ST_STARTING
415 } nlm_run_status_t;

417 /*
418 * nlm_globals structure allows NLM be zone aware. The structure
419 * collects all "global variables" NLM has for each zone.
420 *
421 * struct nlm_globals:
422 * lock: mutex protecting all operations inside given zone
423 * grace_threshold: grace period expiration time (in ticks)
424 * lockd_pid: PID of lockd user space daemon
425 * run_status: run status of klmmod inside given zone
426 * nsm_state: state obtained from local statd during klmmod startup
427 * nlm_gc_thread: garbage collector thread
428 * nlm_gc_sched_cv: condvar that can be signalled to wakeup GC
429 * nlm_gc_finish_cv: condvar that is signalled just before GC thread exits
430 * nlm_nsm: an object describing RPC handle used for talking to local statd
431 * nlm_hosts_tree: an AVL tree of all hosts in the given zone
432 * (used for hosts lookup by <netid, address> pair)
433 * nlm_hosts_hash: a hash table of all hosts in the given zone
434 * (used for hosts lookup by sysid)
435 * nlm_idle_hosts: a list of all hosts that are idle state (i.e. unused)
436 * nlm_slocks: a list of all client-side sleeping locks in the zone
437 * cn_idle_tmo: a value of idle timeout (in seconds) obtained from lockd
438 * grace_period: a value of grace period (in seconds) obtained from lockd
439 * retrans_tmo: a value of retransmission timeout (in seconds) obtained
440 * from lockd.
441 * clean_lock: mutex used to serialize clear_locks calls.
442 * nlm_link: a list node used for keeping all nlm_globals objects
443 * in one global linked list.
444 */
445 struct nlm_globals {
446     kmutex_t          lock;
447     clock_t          grace_threshold; /* (z) */
448     pid_t            lockd_pid;      /* (z) */
449     nlm_run_status_t run_status;     /* (z) */
450     int32_t          nsm_state;      /* (z) */
451     kthread_t        *nlm_gc_thread; /* (z) */
452     kcondvar_t       nlm_gc_sched_cv; /* (z) */
453     kcondvar_t       nlm_gc_finish_cv; /* (z) */
454     struct nlm_nsm    nlm_nsm;       /* (z) */
455     avl_tree_t        nlm_hosts_tree; /* (z) */
456     mod_hash_t       *nlm_hosts_hash; /* (z) */
457     struct nlm_host_list nlm_idle_hosts; /* (z) */

```

```

458     struct nlm_slock_list nlm_slocks; /* (z) */
459     int                   cn_idle_tmo; /* (z) */
460     int                   grace_period; /* (z) */
461     int                   retrans_tmo; /* (z) */
462     kmutex_t              clean_lock; /* (c) */
463     TAILQ_ENTRY(nlm_globals) nlm_link; /* (g) */
464 };
465 TAILQ_HEAD(nlm_globals_list, nlm_globals);

468 /*
469 * This is what we pass as the "owner handle" for NLM_LOCK.
470 * This lets us find the blocked lock in NLM_GRANTED.
471 * It also exposes on the wire what we're using as the
472 * sysid for any server, which can be very helpful for
473 * problem diagnosis. (Observability is good).
474 */
475 struct nlm_owner_handle {
476     sysid_t oh_sysid; /* of remote host */
477 };

479 /*
480 * Number retries NLM RPC call is repeated in case of failure.
481 * (used in case of connectionless transport).
482 */
483 #define NLM_RPC_RETRIES 5

485 /*
486 * Klmmod global variables
487 */
488 extern krwlock_t lm_lck;
489 extern zone_key_t nlm_zone_key;

491 /*
492 * NLM interface functions (called directly by
493 * either klmmod or klmpos)
494 */
495 extern int nlm_frlock(struct vnode *, int, struct flock64 *, int, u_offset_t,
496     struct cred *, struct netobj *, struct flk_callback *, int);
497 extern int nlm_shrlock(struct vnode *, int, struct shrlock *, int,
498     struct netobj *, int);
499 extern int nlm_safemap(const vnode_t *);
500 extern int nlm_safelock(vnode_t *, const struct flock64 *, cred_t *);
501 extern int nlm_has_sleep(const vnode_t *);
502 extern void nlm_register_lock_locally(struct vnode *, struct nlm_host *,
503     struct flock64 *, int, u_offset_t);
504 int nlm_vp_active(const vnode_t *vp);
505 void nlm_sysid_free(sysid_t);
506 int nlm_vp_active(const vnode_t *);
507 void nlm_unexport(struct exportinfo *);

509 /*
510 * NLM startup/shutdown
511 */
512 int nlm_svc_starting(struct nlm_globals *, struct file *,
513     const char *, struct knetconfig *);
514 void nlm_svc_stopping(struct nlm_globals *);
515 int nlm_svc_add_ep(struct file *, const char *, struct knetconfig *);

517 /*
518 * NLM suspend/resume
519 */
520 void nlm_cprsuspend(void);
521 void nlm_cprresume(void);

523 /*

```

```

524 * NLM internal functions for initialization.
525 */
526 void nlm_init(void);
527 void nlm_rpc_init(void);
528 void nlm_rpc_cache_destroy(struct nlm_host *);
529 void nlm_globals_register(struct nlm_globals *);
530 void nlm_globals_unregister(struct nlm_globals *);
531 sysid_t nlm_sysid_alloc(void);

533 /*
534 * Client reclamation/cancelation
535 */
536 void nlm_reclaim_client(struct nlm_globals *, struct nlm_host *);
537 void nlm_client_cancel_all(struct nlm_globals *, struct nlm_host *);

539 /* (nlm_rpc_clnt.c) */
540 enum clnt_stat nlm_null_rpc(CLIENT *, rpcvers_t);
541 enum clnt_stat nlm_test_rpc(nlm4_testargs *, nlm4_testres *,
542     CLIENT *, rpcvers_t);
543 enum clnt_stat nlm_lock_rpc(nlm4_lockargs *, nlm4_res *,
544     CLIENT *, rpcvers_t);
545 enum clnt_stat nlm_cancel_rpc(nlm4_cancargs *, nlm4_res *,
546     CLIENT *, rpcvers_t);
547 enum clnt_stat nlm_unlock_rpc(nlm4_unlockargs *, nlm4_res *,
548     CLIENT *, rpcvers_t);
549 enum clnt_stat nlm_share_rpc(nlm4_shareargs *, nlm4_sharereres *,
550     CLIENT *, rpcvers_t);
551 enum clnt_stat nlm_unshare_rpc(nlm4_shareargs *, nlm4_sharereres *,
552     CLIENT *, rpcvers_t);

555 /*
556 * RPC service functions.
557 * nlm_dispatch.c
558 */
559 void nlm_prog_3(struct svc_req *rqstp, SVCXPRT *transp);
560 void nlm_prog_4(struct svc_req *rqstp, SVCXPRT *transp);

562 /*
563 * Functions for working with knetconfigs (nlm_netconfig.c)
564 */
565 const char *nlm_knc_to_netid(struct knetconfig *);
566 int nlm_knc_from_netid(const char *, struct knetconfig *);

568 /*
569 * NLM host functions (nlm_impl.c)
570 */
571 struct nlm_host *nlm_host_findcreate(struct nlm_globals *, char *,
572     const char *, struct netbuf *);
573 struct nlm_host *nlm_host_find(struct nlm_globals *,
574     const char *, struct netbuf *);
575 struct nlm_host *nlm_host_find_by_sysid(struct nlm_globals *, sysid_t);
576 void nlm_host_release(struct nlm_globals *, struct nlm_host *);

578 void nlm_host_monitor(struct nlm_globals *, struct nlm_host *, int);
579 void nlm_host_unmonitor(struct nlm_globals *, struct nlm_host *);

581 void nlm_host_notify_server(struct nlm_host *, int32_t);
582 void nlm_host_notify_client(struct nlm_host *, int32_t);

584 int nlm_host_get_state(struct nlm_host *);

586 struct nlm_vhold *nlm_vhold_get(struct nlm_host *, vnode_t *);
587 void nlm_vhold_release(struct nlm_host *, struct nlm_vhold *);
588 struct nlm_vhold *nlm_vhold_find_locked(struct nlm_host *, const vnode_t *);

```

```

590 struct nlm_slock *nlm_slock_register(struct nlm_globals *,
591     struct nlm_host *, struct nlm4_lock *, struct vnode *);
592 void nlm_slock_unregister(struct nlm_globals *, struct nlm_slock *);
593 int nlm_slock_wait(struct nlm_globals *, struct nlm_slock *, uint_t);
594 int nlm_slock_grant(struct nlm_globals *,
595     struct nlm_host *, struct nlm4_lock *);
596 void nlm_host_cancel_slocks(struct nlm_globals *, struct nlm_host *);

598 int nlm_slreq_register(struct nlm_host *,
599     struct nlm_vhold *, struct flock64 *);
600 int nlm_slreq_unregister(struct nlm_host *,
601     struct nlm_vhold *, struct flock64 *);

603 void nlm_shres_track(struct nlm_host *, vnode_t *, struct shrlock *);
604 void nlm_shres_untrack(struct nlm_host *, vnode_t *, struct shrlock *);
605 struct nlm_shres *nlm_get_active_shres(struct nlm_host *);
606 void nlm_free_shrlist(struct nlm_shres *);

608 int nlm_host_wait_grace(struct nlm_host *);
609 int nlm_host_cmp(const void *, const void *);
610 void nlm_copy_netobj(struct netobj *, struct netobj *);

612 int nlm_host_get_rpc(struct nlm_host *, int, nlm_rpc_t **);
613 void nlm_host_rele_rpc(struct nlm_host *, nlm_rpc_t *);

615 /*
616 * NLM server functions (nlm_service.c)
617 */
618 int nlm_vp_active(const vnode_t *vp);
619 void nlm_do_notify1(nlm_sm_status *, void *, struct svc_req *);
620 void nlm_do_notify2(nlm_sm_status *, void *, struct svc_req *);
621 void nlm_do_test(nlm4_testargs *, nlm4_testres *,
622     struct svc_req *, nlm_testres_cb);
623 void nlm_do_lock(nlm4_lockargs *, nlm4_res *, struct svc_req *,
624     nlm_reply_cb, nlm_res_cb, nlm_testargs_cb);
625 void nlm_do_cancel(nlm4_cancargs *, nlm4_res *,
626     struct svc_req *, nlm_res_cb);
627 void nlm_do_unlock(nlm4_unlockargs *, nlm4_res *,
628     struct svc_req *, nlm_res_cb);
629 void nlm_do_granted(nlm4_testargs *, nlm4_res *,
630     struct svc_req *, nlm_res_cb);
631 void nlm_do_share(nlm4_shareargs *, nlm4_sharereres *, struct svc_req *);
632 void nlm_do_unshare(nlm4_shareargs *, nlm4_sharereres *, struct svc_req *);
633 void nlm_do_free_all(nlm4_notify *, void *, struct svc_req *);

635 /*
636 * NLM RPC functions
637 */
638 enum clnt_stat nlm_clnt_call(CLIENT *, rpcproc_t, xdrproc_t,
639     caddr_t, xdrproc_t, caddr_t, struct timeval);
640 bool_t nlm_caller_is_local(SVCXPRT *);

642 #endif /* _NLM_NLM_H */

```


new/usr/src/uts/common/klm/nlm_prot_clnt.sed

1

746 Fri Aug 16 09:04:53 2013

new/usr/src/uts/common/klm/nlm_prot_clnt.sed

195 Need replacement for nfs/lockd+klm

```
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 is of the CDDL is also available via the Internet
9 # at http://www.illumos.org/license/CDDL.
10 #
11 #
12 # Copyright 2011 Nexenta Systems, Inc. All rights reserved.
13 #
14 #
15 #
16 # This sed script is run on the client code generated by rpcgen
17 # from nlm_prot.x before it is compiled.
18 #
19 #
20 6{
21 i\
22 #include <sys/param.h>
23 i\
24 #include <sys/system.h>
25 i\
26 #include <rpcsvc/nlm_prot.h>
27 i\
28 #include "nlm_impl.h"
29 }
30 /^.include/,/^endif/d
31 s/clnt_call/nlm_clnt_call/g
```

```

*****
6993 Fri Aug 16 09:04:53 2013
new/usr/src/uts/common/klm/nlm_rpc_clnt.c
195 Need replacement for nfs/lockd+klm
*****
1 /*
2  * Copyright (c) 2008 Isilon Inc http://www.isilon.com/
3  * Authors: Doug Rabson <df@rabson.org>
4  * Developed with Red Inc: Alfred Perlstein <alfred@freebsd.org>
5  *
6  * Redistribution and use in source and binary forms, with or without
7  * modification, are permitted provided that the following conditions
8  * are met:
9  * 1. Redistributions of source code must retain the above copyright
10 * notice, this list of conditions and the following disclaimer.
11 * 2. Redistributions in binary form must reproduce the above copyright
12 * notice, this list of conditions and the following disclaimer in the
13 * documentation and/or other materials provided with the distribution.
14 *
15 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
16 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
17 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
18 * ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
19 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
20 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
21 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
22 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
23 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
24 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
25 * SUCH DAMAGE.
26 */

28 /*
29  * Copyright 2011 Nexenta Systems, Inc. All rights reserved.
30 */

32 /*
33  * Client-side RPC wrappers (nlm_..._rpc)
34  * Called from nlm_client.c
35  *
36  * Source code derived from FreeBSD nlm_advlock.c
37  */

39 #include <sys/param.h>
40 #include <sys/fcntl.h>
41 #include <sys/lock.h>
42 #include <sys/flock.h>
43 #include <sys/mount.h>
44 #include <sys/mutex.h>
45 #include <sys/proc.h>
46 #include <sys/syslog.h>
47 #include <sys/system.h>
48 #include <sys/unistd.h>
49 #include <sys/vnode.h>
50 #include <sys/queue.h>

52 #include <rpcsvc/nlm_prot.h>

54 #include <nfs/nfs.h>
55 #include <nfs/nfs_clnt.h>
56 #include <nfs/export.h>
57 #include <nfs/rnode.h>

59 #include "nlm_impl.h"

61 static void

```

```

62 nlm_convert_to_nlm_lock(struct nlm_lock *dst, struct nlm4_lock *src)
63 {
64     dst->caller_name = src->caller_name;
65     dst->fh = src->fh;
66     dst->oh = src->oh;
67     dst->svid = src->svid;
68     dst->l_offset = src->l_offset;
69     dst->l_len = src->l_len;
70 }

72 static void
73 nlm_convert_to_nlm4_holder(struct nlm4_holder *dst, struct nlm_holder *src)
74 {
75     dst->exclusive = src->exclusive;
76     dst->svid = src->svid;
77     dst->oh = src->oh;
78     dst->l_offset = src->l_offset;
79     dst->l_len = src->l_len;
80 }

82 static void
83 nlm_convert_to_nlm4_res(struct nlm4_res *dst, struct nlm_res *src)
84 {
85     dst->cookie = src->cookie;
86     dst->stat.stat = (enum nlm4_stats) src->stat.stat;
87 }

89 enum clnt_stat
90 nlm_test_rpc(nlm4_testargs *args, nlm4_testres *res,
91             CLIENT *client, rpcvers_t vers)
92 {
93     if (vers == NLM4_VERS) {
94         return (nlm4_test_4(args, res, client));
95     } else {
96         nlm_testargs args1;
97         nlm_testres res1;
98         enum clnt_stat stat;

100         args1.cookie = args->cookie;
101         args1.exclusive = args->exclusive;
102         nlm_convert_to_nlm_lock(&args1.alock, &args->alock);
103         (void) memset(&res1, 0, sizeof (res1));

105         stat = nlm_test_1(&args1, &res1, client);

107         if (stat == RPC_SUCCESS) {
108             res->cookie = res1.cookie;
109             res->stat.stat = (enum nlm4_stats) res1.stat.stat;
110             if (res1.stat.stat == nlm_denied)
111                 nlm_convert_to_nlm4_holder(
112                     &res->stat.nlm4_testreply_u.holder,
113                     &res1.stat.nlm_testreply_u.holder);
114         }

116         return (stat);
117     }
118 }

120 enum clnt_stat
121 nlm_lock_rpc(nlm4_lockargs *args, nlm4_res *res,
122             CLIENT *client, rpcvers_t vers)
123 {
124     if (vers == NLM4_VERS) {
125         return (nlm4_lock_4(args, res, client));
126     } else {
127         nlm_lockargs args1;

```

```

128     nlm_res res1;
129     enum clnt_stat stat;

131     args1.cookie = args->cookie;
132     args1.block = args->block;
133     args1.exclusive = args->exclusive;
134     nlm_convert_to_nlm_lock(&args1.alock, &args->alock);
135     args1.reclaim = args->reclaim;
136     args1.state = args->state;
137     (void) memset(&res1, 0, sizeof(res1));

139     stat = nlm_lock_1(&args1, &res1, client);

141     if (stat == RPC_SUCCESS) {
142         nlm_convert_to_nlm4_res(res, &res1);
143     }

145     return (stat);
146 }
147 }

149 enum clnt_stat
150 nlm_cancel_rpc(nlm4_cancargs *args, nlm4_res *res,
151 CLIENT *client, rpcvers_t vers)
152 {
153     if (vers == NLM4_VERS) {
154         return (nlm4_cancel_4(args, res, client));
155     } else {
156         nlm_cancargs args1;
157         nlm_res res1;
158         enum clnt_stat stat;

160         args1.cookie = args->cookie;
161         args1.block = args->block;
162         args1.exclusive = args->exclusive;
163         nlm_convert_to_nlm_lock(&args1.alock, &args->alock);
164         (void) memset(&res1, 0, sizeof(res1));

166         stat = nlm_cancel_1(&args1, &res1, client);

168         if (stat == RPC_SUCCESS) {
169             nlm_convert_to_nlm4_res(res, &res1);
170         }

172         return (stat);
173     }
174 }

176 enum clnt_stat
177 nlm_unlock_rpc(nlm4_unlockargs *args, nlm4_res *res,
178 CLIENT *client, rpcvers_t vers)
179 {
180     if (vers == NLM4_VERS) {
181         return (nlm4_unlock_4(args, res, client));
182     } else {
183         nlm_unlockargs args1;
184         nlm_res res1;
185         enum clnt_stat stat;

187         args1.cookie = args->cookie;
188         nlm_convert_to_nlm_lock(&args1.alock, &args->alock);
189         (void) memset(&res1, 0, sizeof(res1));

191         stat = nlm_unlock_1(&args1, &res1, client);
193         if (stat == RPC_SUCCESS) {

```

```

194             nlm_convert_to_nlm4_res(res, &res1);
195         }

197         return (stat);
198     }
199 }

201 enum clnt_stat
202 nlm_null_rpc(CLIENT *client, rpcvers_t vers)
203 {
204     if (vers == NLM4_VERS)
205         return (nlm4_null_4(NULL, NULL, client));

207     return (nlm_null_1(NULL, NULL, client));
208 }

210 /*
211  * Share reservations
212  */

214 static void
215 nlm_convert_to_nlm_share(struct nlm_share *dst, struct nlm4_share *src)
216 {
218     dst->caller_name = src->caller_name;
219     dst->fh = src->fh;
220     dst->oh = src->oh;
221     dst->mode = src->mode;
222     dst->access = src->access;
223 }

225 static void
226 nlm_convert_to_nlm4_shres(struct nlm4_sharereres *dst,
227 struct nlm_sharereres *src)
228 {
229     dst->cookie = src->cookie;
230     dst->stat = (enum nlm4_stats) src->stat;
231     dst->sequence = src->sequence;
232 }

235 enum clnt_stat
236 nlm_share_rpc(nlm4_shareargs *args, nlm4_sharereres *res,
237 CLIENT *client, rpcvers_t vers)
238 {
239     if (vers == NLM4_VERS) {
240         return (nlm4_share_4(args, res, client));
241     } else {
242         nlm_shareargs args3;
243         nlm_sharereres res3;
244         enum clnt_stat stat;

246         args3.cookie = args->cookie;
247         nlm_convert_to_nlm_share(&args3.share, &args->share);
248         args3.reclaim = args->reclaim;
249         (void) memset(&res3, 0, sizeof(res3));

251         stat = nlm_share_3(&args3, &res3, client);

253         if (stat == RPC_SUCCESS) {
254             nlm_convert_to_nlm4_shres(res, &res3);
255         }

257         return (stat);
258     }
259 }

```

```
261 enum clnt_stat
262 nlm_unshare_rpc(nlm4_shareargs *args, nlm4_sharereres *res,
263 CLIENT *client, rpcvers_t vers)
264 {
265     if (vers == NLM4_VERS) {
266         return (nlm4_unshare_4(args, res, client));
267     } else {
268         nlm_shareargs args3;
269         nlm_sharereres res3;
270         enum clnt_stat stat;
271
272         args3.cookie = args->cookie;
273         nlm_convert_to_nlm_share(&args3.share, &args->share);
274         args3.reclaim = args->reclaim;
275         (void) memset(&res3, 0, sizeof (res3));
276
277         stat = nlm_unshare_3(&args3, &res3, client);
278
279         if (stat == RPC_SUCCESS) {
280             nlm_convert_to_nlm4_shres(res, &res3);
281         }
282
283         return (stat);
284     }
285 }
```

```

*****
9353 Fri Aug 16 09:04:54 2013
new/usr/src/uts/common/klm/nlm_rpc_handle.c
195 Need replacement for nfs/lockd+klm
*****
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 2011 Nexenta Systems, Inc. All rights reserved.
24  * Copyright (c) 2012 by Delphix. All rights reserved.
25  */

27 #include <sys/param.h>
28 #include <sys/system.h>
29 #include <sys/socket.h>
30 #include <sys/syslog.h>
31 #include <sys/system.h>
32 #include <sys/unistd.h>
33 #include <sys/queue.h>
34 #include <sys/sdt.h>
35 #include <netinet/in.h>

37 #include <rpc/rpc.h>
38 #include <rpc/xdr.h>
39 #include <rpc/pmap_prot.h>
40 #include <rpc/pmap_clnt.h>
41 #include <rpc/rpcb_prot.h>

43 #include <rpcsvc/nlm_prot.h>
44 #include <rpcsvc/sm_inter.h>

46 #include "nlm_impl.h"

48 /*
49  * The following errors codes from nlm_null_rpc indicate that the port we have
50  * cached for the client's NLM service is stale and that we need to establish
51  * a new RPC client.
52  */
53 #define NLM_STALE_CLNT(_status) \
54     ((_status) == RPC_PROGUNAVAIL || \
55     (_status) == RPC_PROGVERSISMATCH || \
56     (_status) == RPC_PROCUNAVAIL || \
57     (_status) == RPC_CANTCONNECT || \
58     (_status) == RPC_XPRTFAILED)

60 static struct kmem_cache *nlm_rpch_cache = NULL;

```

```

62 static int nlm_rpch_ctor(void *, void *, int);
63 static void nlm_rpch_dtor(void *, void *);
64 static void destroy_rpch(nlm_rpc_t *);
65 static nlm_rpc_t *get_nlm_rpc_fromcache(struct nlm_host *, int);
66 static void update_host_rpcbinding(struct nlm_host *, int);
67 static int refresh_nlm_rpc(struct nlm_host *, nlm_rpc_t *);
68 static void nlm_host_rele_rpc_locked(struct nlm_host *, nlm_rpc_t *);

70 static nlm_rpc_t *
71 get_nlm_rpc_fromcache(struct nlm_host *hostp, int vers)
72 {
73     nlm_rpc_t *rpcp;
74     bool_t found = FALSE;

76     ASSERT(MUTEX_HELD(&hostp->nh_lock));
77     if (TAILQ_EMPTY(&hostp->nh_rpcch))
78         return (NULL);

80     TAILQ_FOREACH(rpcp, &hostp->nh_rpcch, nr_link) {
81         if (rpcp->nr_vers == vers) {
82             found = TRUE;
83             break;
84         }
85     }

87     if (!found)
88         return (NULL);

90     TAILQ_REMOVE(&hostp->nh_rpcch, rpcp, nr_link);
91     return (rpcp);
92 }

94 /*
95  * Update host's RPC binding (host->nh_addr).
96  * The function is executed by only one thread at time.
97  */
98 static void
99 update_host_rpcbinding(struct nlm_host *hostp, int vers)
100 {
101     enum clnt_stat stat;

103     ASSERT(MUTEX_HELD(&hostp->nh_lock));

105     /*
106      * Mark RPC binding state as "update in progress" in order
107      * to say other threads that they need to wait until binding
108      * is fully updated.
109      */
110     hostp->nh_rpcb_state = NRPCB_UPDATE_INPROGRESS;
111     hostp->nh_rpcb_ustat = RPC_SUCCESS;
112     mutex_exit(&hostp->nh_lock);

114     stat = rpcbind_getaddr(&hostp->nh_knc, NLM_PROG, vers, &hostp->nh_addr);
115     mutex_enter(&hostp->nh_lock);

117     hostp->nh_rpcb_state = ((stat == RPC_SUCCESS) ?
118         NRPCB_UPDATED : NRPCB_NEED_UPDATE);

120     hostp->nh_rpcb_ustat = stat;
121     cv_broadcast(&hostp->nh_rpcb_cv);
122 }

124 /*
125  * Refresh RPC handle taken from host handles cache.
126  * This function is called when an RPC handle is either
127  * uninitialized or was initialized using a binding that's

```

```

128 * no longer current.
129 */
130 static int
131 refresh_nlm_rpc(struct nlm_host *hostp, nlm_rpc_t *rpcp)
132 {
133     int ret;
134
135     if (rpcp->nr_handle == NULL) {
136         bool_t clset = TRUE;
137
138         ret = clnt_tli_kcreate(&hostp->nh_knc, &hostp->nh_addr,
139                               NLM_PROG, rpcp->nr_vers, 0, NLM_RPC_RETRIES,
140                               CRED(), &rpcp->nr_handle);
141
142         /*
143          * Set the client's CLSET_NODELAYONERR option to true. The
144          * RPC clnt_call interface creates an artificial delay for
145          * certain call errors in order to prevent RPC consumers
146          * from getting into tight retry loops. Since this function is
147          * called by the NLM service routines we would like to avoid
148          * this artificial delay when possible. We do not retry if the
149          * NULL request fails so it is safe for us to turn this option
150          * on.
151          */
152         if (clnt_control(rpcp->nr_handle, CLSET_NODELAYONERR,
153                         (char *)&clset)) {
154             NLM_ERR("Unable to set CLSET_NODELAYONERR\n");
155         }
156     } else {
157         ret = clnt_tli_kinit(rpcp->nr_handle, &hostp->nh_knc,
158                             &hostp->nh_addr, 0, NLM_RPC_RETRIES, CRED());
159         if (ret == 0) {
160             enum clnt_stat stat;
161
162             /*
163              * Check whether host's RPC binding is still
164              * fresh, i.e. if remote program is still sits
165              * on the same port we assume. Call NULL proc
166              * to do it.
167              *
168              * Note: Even though we set no delay on error on the
169              * client handle the call to nlm_null_rpc can still
170              * delay for 10 seconds before returning an error. For
171              * example the no delay on error option is not honored
172              * for RPC_XPRTFAILED errors (see clnt_cots_kcallit).
173              */
174             stat = nlm_null_rpc(rpcp->nr_handle, rpcp->nr_vers);
175             if (NLM_STALE_CLNT(stat)) {
176                 ret = ESTALE;
177             }
178         }
179     }
180
181     return (ret);
182 }
183
184 /*
185 * Get RPC handle that can be used to talk to the NLM
186 * of given version running on given host.
187 * Saves obtained RPC handle to rpcpp argument.
188 *
189 * If error occurs, return nonzero error code.
190 */
191 int
192 nlm_host_get_rpc(struct nlm_host *hostp, int vers, nlm_rpc_t **rpcpp)
193 {

```

```

194     nlm_rpc_t *rpcp = NULL;
195     int rc;
196
197     mutex_enter(&hostp->nh_lock);
198
199     /*
200     * If this handle is either uninitialized, or was
201     * initialized using binding that's now stale
202     * do the init or re-init.
203     * See comments to enum nlm_rpcb_state for more
204     * details.
205     */
206     again:
207     while (hostp->nh_rpcb_state != NRPCB_UPDATED) {
208         if (hostp->nh_rpcb_state == NRPCB_UPDATE_INPROGRESS) {
209             rc = cv_wait_sig(&hostp->nh_rpcb_cv, &hostp->nh_lock);
210             if (rc == 0) {
211                 mutex_exit(&hostp->nh_lock);
212                 return (EINTR);
213             }
214         }
215
216         /*
217         * Check if RPC binding was marked for update.
218         * If so, start RPC binding update operation.
219         * NOTE: the operation can be executed by only
220         * one thread at time.
221         */
222         if (hostp->nh_rpcb_state == NRPCB_NEED_UPDATE)
223             update_host_rpcbinding(hostp, vers);
224
225         /*
226         * Check if RPC error occurred during RPC binding
227         * update operation. If so, report a corresponding
228         * error.
229         */
230         if (hostp->nh_rpcb_ustat != RPC_SUCCESS) {
231             mutex_exit(&hostp->nh_lock);
232             return (ENOENT);
233         }
234     }
235
236     rpcp = get_nlm_rpc_fromcache(hostp, vers);
237     mutex_exit(&hostp->nh_lock);
238     if (rpcp == NULL) {
239         /*
240         * There weren't any RPC handles in a host
241         * cache. No luck, just create a new one.
242         */
243         rpcp = kmem_cache_alloc(nlm_rpcb_cache, KM_SLEEP);
244         rpcp->nr_vers = vers;
245     }
246
247     /*
248     * Refresh RPC binding
249     */
250     rc = refresh_nlm_rpc(hostp, rpcp);
251     if (rc != 0) {
252         if (rc == ESTALE) {
253             /*
254              * Host's RPC binding is stale, we have
255              * to update it. Put the RPC handle back
256              * to the cache and mark the host as
257              * "need update".
258              */
259             mutex_enter(&hostp->nh_lock);

```

```

260         hostp->nh_rpcb_state = NRPCB_NEED_UPDATE;
261         nlm_host_rele_rpc_locked(hostp, rpcp);
262         goto again;
263     }
264
265     destroy_rpch(rpcp);
266     return (rc);
267 }
268
269 DTRACE_PROBE2(end, struct nlm_host *, hostp,
270              nlm_rpc_t *, rpcp);
271
272 *rpcpp = rpcp;
273 return (0);
274 }
275
276 void
277 nlm_host_rele_rpc(struct nlm_host *hostp, nlm_rpc_t *rpcp)
278 {
279     mutex_enter(&hostp->nh_lock);
280     nlm_host_rele_rpc_locked(hostp, rpcp);
281     mutex_exit(&hostp->nh_lock);
282 }
283
284 static void
285 nlm_host_rele_rpc_locked(struct nlm_host *hostp, nlm_rpc_t *rpcp)
286 {
287     ASSERT(mutex_owned(&hostp->nh_lock));
288     TAILQ_INSERT_HEAD(&hostp->nh_rpcch, rpcp, nr_link);
289 }
290
291 /*
292  * The function invalidates host's RPC binding by marking it
293  * as not fresh. In this case another time thread tries to
294  * get RPC handle from host's handles cache, host's RPC binding
295  * will be updated.
296  *
297  * The function should be executed when RPC call invoked via
298  * handle taken from RPC cache returns RPC_PROCUAVAIL.
299  */
300 void
301 nlm_host_invalidate_binding(struct nlm_host *hostp)
302 {
303     mutex_enter(&hostp->nh_lock);
304     hostp->nh_rpcb_state = NRPCB_NEED_UPDATE;
305     mutex_exit(&hostp->nh_lock);
306 }
307
308 void
309 nlm_rpc_init(void)
310 {
311     nlm_rpc_cache = kmem_cache_create("nlm_rpc_cache",
312                                     sizeof (nlm_rpc_t), 0, nlm_rpc_ctor, nlm_rpc_dtor,
313                                     NULL, NULL, NULL, 0);
314 }
315
316 void
317 nlm_rpc_cache_destroy(struct nlm_host *hostp)
318 {
319     nlm_rpc_t *rpcp;
320
321     /*
322      * There's no need to lock host's mutex here,
323      * nlm_rpc_cache_destroy() should be called from
324      * only one place: nlm_host_destroy, when all
325      * resources host owns are already cleaned up.

```

```

326     * So there shouldn't be any raises.
327     */
328     while ((rpcp = TAILQ_FIRST(&hostp->nh_rpcch)) != NULL) {
329         TAILQ_REMOVE(&hostp->nh_rpcch, rpcp, nr_link);
330         destroy_rpch(rpcp);
331     }
332 }
333
334 /* ARGSUSED */
335 static int
336 nlm_rpc_ctor(void *datap, void *cdrarg, int kmflags)
337 {
338     nlm_rpc_t *rpcp = (nlm_rpc_t *)datap;
339
340     bzero(rpcp, sizeof (*rpcp));
341     return (0);
342 }
343
344 /* ARGSUSED */
345 static void
346 nlm_rpc_dtor(void *datap, void *cdrarg)
347 {
348     nlm_rpc_t *rpcp = (nlm_rpc_t *)datap;
349     ASSERT(rpcp->nr_handle == NULL);
350 }
351
352 static void
353 destroy_rpch(nlm_rpc_t *rpcp)
354 {
355     if (rpcp->nr_handle != NULL) {
356         AUTH_DESTROY(rpcp->nr_handle->cl_auth);
357         CLNT_DESTROY(rpcp->nr_handle);
358         rpcp->nr_handle = NULL;
359     }
360
361     kmem_cache_free(nlm_rpc_cache, rpcp);
362 }

```

```

*****
21456 Fri Aug 16 09:04:54 2013
new/usr/src/uts/common/klm/nlm_rpc_svc.c
195 Need replacement for nfs/lockd+klm
*****
1 /*
2  * Copyright (c) 2008 Isilon Inc http://www.isilon.com/
3  * Authors: Doug Rabson <df@rabson.org>
4  * Developed with Red Inc: Alfred Perlstein <alfred@freebsd.org>
5  *
6  * Redistribution and use in source and binary forms, with or without
7  * modification, are permitted provided that the following conditions
8  * are met:
9  * 1. Redistributions of source code must retain the above copyright
10 * notice, this list of conditions and the following disclaimer.
11 * 2. Redistributions in binary form must reproduce the above copyright
12 * notice, this list of conditions and the following disclaimer in the
13 * documentation and/or other materials provided with the distribution.
14 *
15 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
16 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
17 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
18 * ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
19 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
20 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
21 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
22 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
23 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
24 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
25 * SUCH DAMAGE.
26 */

28 /*
29 * Copyright 2013 Nexenta Systems, Inc. All rights reserved.
30 * Copyright (c) 2012 by Delphix. All rights reserved.
31 */

33 /*
34 * NFS Lock Manager, RPC service functions (nlm..._svc)
35 * Called via nlm_dispatch.c tables.
36 *
37 * Source code derived from FreeBSD nlm_prot_server.c
38 *
39 * The real service functions all use nlm4... args and return
40 * data types. These wrappers convert older forms to and from
41 * the new forms and call the nlm_do... service functions.
42 */

44 #include <sys/param.h>
45 #include <sys/system.h>

47 #include <rpcsvc/nlm_prot.h>
48 #include "nlm_impl.h"

50 /*
51 * Convert between various versions of the protocol structures.
52 */

54 /*
55 * Down-convert, for granted_1 call
56 *
57 * This converts a 64-bit lock to 32-bit form for our granted
58 * call-back when we're dealing with a 32-bit NLM client.
59 * Our NLM_LOCK handler ensures that any lock we grant to a
60 * 32-bit client can be represented in 32-bits. If the
61 * ASSERTs here fire, then the call to nlm_init_flock in

```

```

62 * nlm_do_lock has failed to restrict a 32-bit client to
63 * 32-bit lock ranges.
64 */
65 static void
66 nlm_convert_to_nlm_lock(struct nlm_lock *dst, struct nlm4_lock *src)
67 {
68     dst->caller_name = src->caller_name;
69     dst->fh = src->fh;
70     dst->oh = src->oh;
71     dst->svid = src->svid;
72     ASSERT(src->l_offset <= MAX_UOFF32);
73     dst->l_offset = (uint32_t)src->l_offset;
74     ASSERT(src->l_len <= MAX_UOFF32);
75     dst->l_len = (uint32_t)src->l_len;
76 }

78 /*
79 * Up-convert for v1 svc functions with a 32-bit lock range arg.
80 * Note that lock range checks (like overflow) are done later,
81 * in nlm_init_flock().
82 */
83 static void
84 nlm_convert_to_nlm4_lock(struct nlm4_lock *dst, struct nlm_lock *src)
85 {
86
87     dst->caller_name = src->caller_name;
88     dst->fh = src->fh;
89     dst->oh = src->oh;
90     dst->svid = src->svid;
91     dst->l_offset = src->l_offset;
92     dst->l_len = src->l_len;
93 }

95 static void
96 nlm_convert_to_nlm4_share(struct nlm4_share *dst, struct nlm_share *src)
97 {
98
99     dst->caller_name = src->caller_name;
100     dst->fh = src->fh;
101     dst->oh = src->oh;
102     dst->mode = src->mode;
103     dst->access = src->access;
104 }

106 /*
107 * Down-convert for v1 NLM_TEST or NLM_TEST_MSG response.
108 * Note that nlm_do_test is careful to give us lock ranges
109 * that can be represented with 32-bit values. If the
110 * ASSERTs here fire, then the code in nlm_do_test that
111 * builds an nlm4 holder for a 32-bit client has failed to
112 * restrict the reported conflicting lock range so it's a
113 * valid 32-bit lock range.
114 */
115 static void
116 nlm_convert_to_nlm_holder(struct nlm_holder *dst, struct nlm4_holder *src)
117 {
118     dst->exclusive = src->exclusive;
119     dst->svid = src->svid;
120     dst->oh = src->oh;
121     ASSERT(src->l_offset <= MAX_UOFF32);
122     dst->l_offset = (uint32_t)src->l_offset;
123     ASSERT(src->l_len <= MAX_UOFF32);
124     dst->l_len = (uint32_t)src->l_len;
125 }

127 static enum nlm_stats

```



```

128 nlm_convert_to_nlm_stats(enum nlm4_stats src)
129 {
130     if (src > nlm4_deadlck)
131         return (nlm_denied);
132     return ((enum nlm_stats)src);
133 }

135 static void
136 nlm_convert_to_nlm_res(struct nlm_res *dst, struct nlm4_res *src)
137 {
138     dst->cookie = src->cookie;
139     dst->stat.stat = nlm_convert_to_nlm_stats(src->stat.stat);
140 }

142 /* ***** */
144 /*
145  * Version 1 svc functions
146  */

148 bool_t
149 nlm_test_1_svc(struct nlm_testargs *argp, nlm_testres *resp,
150              struct svc_req *sr)
151 {
152     nlm4_testargs args4;
153     nlm4_testres res4;

155     bzero(&args4, sizeof (args4));
156     bzero(&res4, sizeof (res4));

158     args4.cookie = argp->cookie;
159     args4.exclusive = argp->exclusive;
160     nlm_convert_to_nlm4_lock(&args4.alock, &argp->alock);

162     nlm_do_test(&args4, &res4, sr, NULL);

164     resp->cookie = res4.cookie;
165     resp->stat.stat = nlm_convert_to_nlm_stats(res4.stat.stat);
166     if (resp->stat.stat == nlm_denied)
167         nlm_convert_to_nlm_holder(
168             &resp->stat.nlm_testreply_u.holder,
169             &res4.stat.nlm4_testreply_u.holder);

171     return (TRUE);
172 }

174 /*
175  * Callback functions for nlm_lock_1_svc
176  */
177 static bool_t nlm_lock_1_reply(SVCXPRT *, nlm4_res *);
178 static enum clnt_stat nlm_granted_1_cb(nlm4_testargs *, void *, CLIENT *);

180 bool_t
181 nlm_lock_1_svc(nlm_lockargs *argp, nlm_res *resp,
182              struct svc_req *sr)
183 {
184     nlm4_lockargs args4;
185     nlm4_res res4;

187     bzero(&res4, sizeof (res4));

189     args4.cookie = argp->cookie;
190     args4.block = argp->block;
191     args4.exclusive = argp->exclusive;
192     nlm_convert_to_nlm4_lock(&args4.alock, &argp->alock);
193     args4.reclaim = argp->reclaim;

```

```

194     args4.state = argp->state;

196     /* NLM_LOCK */
197     nlm_do_lock(&args4, &res4, sr,
198               nlm_lock_1_reply, NULL,
199               nlm_granted_1_cb);

201     /* for freeresult */
202     nlm_convert_to_nlm_res(resp, &res4);

204     /* above does its own reply */
205     return (FALSE);
206 }

208 static bool_t
209 nlm_lock_1_reply(SVCXPRT *transp, nlm4_res *resp)
210 {
211     nlm_res res1;

213     nlm_convert_to_nlm_res(&res1, resp);
214     return (svc_sendreply(transp, xdr_nlm_res, (char *)&res1));
215 }

217 static enum clnt_stat
218 nlm_granted_1_cb(nlm4_testargs *argp, void *resp, CLIENT *clnt)
219 {
220     nlm_testargs args1;
221     nlm_res res1;
222     int rv;

224     bzero(&res1, sizeof (res1));

226     args1.cookie = argp->cookie;
227     args1.exclusive = argp->exclusive;
228     nlm_convert_to_nlm_lock(&args1.alock, &argp->alock);

230     rv = nlm_granted_1(&args1, &res1, clnt);

232     /* NB: We have a result our caller will not free. */
233     xdr_free((xdrproc_t)xdr_nlm_res, (void *)&res1);
234     (void) resp;

236     return (rv);
237 }

239 bool_t
240 nlm_cancel_1_svc(struct nlm_cancargs *argp, nlm_res *resp,
241               struct svc_req *sr)
242 {
243     nlm4_cancargs args4;
244     nlm4_res res4;

246     bzero(&res4, sizeof (res4));

248     args4.cookie = argp->cookie;
249     args4.block = argp->block;
250     args4.exclusive = argp->exclusive;
251     nlm_convert_to_nlm4_lock(&args4.alock, &argp->alock);

253     nlm_do_cancel(&args4, &res4, sr, NULL);

255     nlm_convert_to_nlm_res(resp, &res4);

257     return (TRUE);
258 }

```

```

260 bool_t
261 nlm_unlock_1_svc(struct nlm_unlockargs *argp, nlm_res *resp,
262                struct svc_req *sr)
263 {
264     nlm4_unlockargs args4;
265     nlm4_res res4;
266
267     bzero(&res4, sizeof (res4));
268
269     args4.cookie = argp->cookie;
270     nlm_convert_to_nlm4_lock(&args4.alock, &argp->alock);
271
272     nlm_do_unlock(&args4, &res4, sr, NULL);
273
274     nlm_convert_to_nlm_res(resp, &res4);
275
276     return (TRUE);
277 }
278
279 bool_t
280 nlm_granted_1_svc(struct nlm_testargs *argp, nlm_res *resp,
281                 struct svc_req *sr)
282 {
283     nlm4_testargs args4;
284     nlm4_res res4;
285
286     bzero(&res4, sizeof (res4));
287
288     args4.cookie = argp->cookie;
289     args4.exclusive = argp->exclusive;
290     nlm_convert_to_nlm4_lock(&args4.alock, &argp->alock);
291
292     nlm_do_granted(&args4, &res4, sr, NULL);
293
294     nlm_convert_to_nlm_res(resp, &res4);
295
296     return (TRUE);
297 }
298
299 /*
300 * The _msg_calls get no reply. Instead, these callers
301 * expect an RPC call to the corresponding _res function.
302 * We pass this callback function to nlm_do_test so it will
303 * use it to do the RPC callback, with the correct res type.
304 *
305 * The callback functions have nearly the same arg signature
306 * as the client call functions so that many of those can be
307 * optimized to nothing by the compiler. Also, passing the
308 * null result arg for these just to reduce warnings.
309 *
310 * See similar callbacks for other _msg functions below.
311 */
312
313 static enum clnt_stat nlm_test_res_1_cb(nlm4_testres *, void *, CLIENT *);
314
315 bool_t
316 nlm_test_msg_1_svc(struct nlm_testargs *argp, void *resp,
317                  struct svc_req *sr)
318 {
319     nlm4_testargs args4;
320     nlm4_testres res4;
321
322     bzero(&res4, sizeof (res4));
323
324     args4.cookie = argp->cookie;
325     args4.exclusive = argp->exclusive;

```

```

326     nlm_convert_to_nlm4_lock(&args4.alock, &argp->alock);
327
328     nlm_do_test(&args4, &res4, sr,
329               nlm_test_res_1_cb);
330
331     /* NB: We have a result our caller will not free. */
332     xdr_free((xdrproc_t)xdr_nlm4_testres, (void *)&res4);
333     (void) resp;
334
335     /* The _msg_calls get no reply. */
336     return (FALSE);
337 }
338
339 static enum clnt_stat
340 nlm_test_res_1_cb(nlm4_testres *res4, void *null, CLIENT *clnt)
341 {
342     nlm_testres res1;
343
344     res1.cookie = res4->cookie;
345     res1.stat.stat = nlm_convert_to_nlm_stats(res4->stat.stat);
346     if (res1.stat.stat == nlm_denied)
347         nlm_convert_to_nlm_holder(
348             &res1.stat.nlm_testreply_u.holder,
349             &res4->stat.nlm4_testreply_u.holder);
350
351     return (nlm_test_res_1(&res1, null, clnt));
352 }
353
354 /*
355 * Callback functions for nlm_lock_msg_1_svc
356 */
357 static enum clnt_stat nlm_lock_res_1_cb(nlm4_res *, void *, CLIENT *);
358 static enum clnt_stat nlm_granted_msg_1_cb(nlm4_testargs *, void *, CLIENT *);
359
360 bool_t
361 nlm_lock_msg_1_svc(nlm_lockargs *argp, void *resp,
362                  struct svc_req *sr)
363 {
364     nlm4_lockargs args4;
365     nlm4_res res4;
366
367     bzero(&res4, sizeof (res4));
368
369     args4.cookie = argp->cookie;
370     args4.block = argp->block;
371     args4.exclusive = argp->exclusive;
372     nlm_convert_to_nlm4_lock(&args4.alock, &argp->alock);
373     args4.reclaim = argp->reclaim;
374     args4.state = argp->state;
375
376     /* NLM_LOCK_MSG */
377     nlm_do_lock(&args4, &res4, sr,
378               NULL, nlm_lock_res_1_cb,
379               nlm_granted_msg_1_cb);
380
381     /* NB: We have a result our caller will not free. */
382     xdr_free((xdrproc_t)xdr_nlm4_res, (void *)&res4);
383     (void) resp;
384
385     /* The _msg_calls get no reply. */
386     return (FALSE);
387 }
388
389 static enum clnt_stat
390 nlm_lock_res_1_cb(nlm4_res *res4, void *null, CLIENT *clnt)
391 {

```

```

392     nlm_res res1;

394     nlm_convert_to_nlm_res(&res1, resp);
395     return (nlm_lock_res_1(&res1, null, clnt));
396 }

398 static enum clnt_stat
399 nlm_granted_msg_1_cb(nlm4_testargs *argp, void *null, CLIENT *clnt)
400 {
401     nlm_testargs args1;

403     args1.cookie = argp->cookie;
404     args1.exclusive = argp->exclusive;
405     nlm_convert_to_nlm_lock(&args1.alock, &argp->alock);

407     return (nlm_granted_msg_1(&args1, null, clnt));

409 }

412 static enum clnt_stat nlm_cancel_res_1_cb(nlm4_res *, void *, CLIENT *);

414 bool_t
415 nlm_cancel_msg_1_svc(struct nlm_cancargs *argp, void *resp,
416                    struct svc_req *sr)
417 {
418     nlm4_cancargs args4;
419     nlm4_res res4;

421     bzero(&res4, sizeof (res4));

423     args4.cookie = argp->cookie;
424     args4.block = argp->block;
425     args4.exclusive = argp->exclusive;
426     nlm_convert_to_nlm4_lock(&args4.alock, &argp->alock);

428     nlm_do_cancel(&args4, &res4, sr,
429                  nlm_cancel_res_1_cb);

431     /* NB: We have a result our caller will not free. */
432     xdr_free((xdrproc_t)xdr_nlm4_res, (void *)&res4);
433     (void) resp;

435     /* The _msg_ calls get no reply. */
436     return (FALSE);
437 }

439 static enum clnt_stat
440 nlm_cancel_res_1_cb(nlm4_res *res4, void *null, CLIENT *clnt)
441 {
442     nlm_res res1;

444     nlm_convert_to_nlm_res(&res1, res4);
445     return (nlm_cancel_res_1(&res1, null, clnt));
446 }

449 static enum clnt_stat nlm_unlock_res_1_cb(nlm4_res *, void *, CLIENT *);

451 bool_t
452 nlm_unlock_msg_1_svc(struct nlm_unlockargs *argp, void *resp,
453                    struct svc_req *sr)
454 {
455     nlm4_unlockargs args4;
456     nlm4_res res4;

```

```

458     bzero(&res4, sizeof (res4));

460     args4.cookie = argp->cookie;
461     nlm_convert_to_nlm4_lock(&args4.alock, &argp->alock);

463     nlm_do_unlock(&args4, &res4, sr,
464                  nlm_unlock_res_1_cb);

466     /* NB: We have a result our caller will not free. */
467     xdr_free((xdrproc_t)xdr_nlm4_res, (void *)&res4);
468     (void) resp;

470     /* The _msg_ calls get no reply. */
471     return (FALSE);
472 }

474 static enum clnt_stat
475 nlm_unlock_res_1_cb(nlm4_res *res4, void *null, CLIENT *clnt)
476 {
477     nlm_res res1;

479     nlm_convert_to_nlm_res(&res1, res4);
480     return (nlm_unlock_res_1(&res1, null, clnt));
481 }

484 static enum clnt_stat nlm_granted_res_1_cb(nlm4_res *, void *, CLIENT *);

486 bool_t
487 nlm_granted_msg_1_svc(struct nlm_testargs *argp, void *resp,
488                    struct svc_req *sr)
489 {
490     nlm4_testargs args4;
491     nlm4_res res4;

493     bzero(&res4, sizeof (res4));

495     args4.cookie = argp->cookie;
496     args4.exclusive = argp->exclusive;
497     nlm_convert_to_nlm4_lock(&args4.alock, &argp->alock);

499     nlm_do_granted(&args4, &res4, sr,
500                   nlm_granted_res_1_cb);

502     /* NB: We have a result our caller will not free. */
503     xdr_free((xdrproc_t)xdr_nlm4_res, (void *)&res4);
504     (void) resp;

506     /* The _msg_ calls get no reply. */
507     return (FALSE);
508 }

510 static enum clnt_stat
511 nlm_granted_res_1_cb(nlm4_res *res4, void *null, CLIENT *clnt)
512 {
513     nlm_res res1;

515     nlm_convert_to_nlm_res(&res1, res4);
516     return (nlm_granted_res_1(&res1, null, clnt));
517 }

519 /*
520  * The _res_ calls get no reply. These RPC calls are
521  * "call backs" in response to RPC _msg_ calls.
522  * We don't care about these responses.
523  */

```

```

525 /* ARGSUSED */
526 bool_t
527 nlm_test_res_1_svc(nlm_testres *argp, void *resp, struct svc_req *sr)
528 {
529     /* The _res_ calls get no reply. */
530     return (FALSE);
531 }

533 /* ARGSUSED */
534 bool_t
535 nlm_lock_res_1_svc(nlm_res *argp, void *resp, struct svc_req *sr)
536 {
537     /* The _res_ calls get no reply. */
538     return (FALSE);
539 }

541 /* ARGSUSED */
542 bool_t
543 nlm_cancel_res_1_svc(nlm_res *argp, void *resp, struct svc_req *sr)
544 {
545     /* The _res_ calls get no reply. */
546     return (FALSE);
547 }

549 /* ARGSUSED */
550 bool_t
551 nlm_unlock_res_1_svc(nlm_res *argp, void *resp, struct svc_req *sr)
552 {
553     /* The _res_ calls get no reply. */
554     return (FALSE);
555 }

557 /* ARGSUSED */
558 bool_t
559 nlm_granted_res_1_svc(nlm_res *argp, void *resp, struct svc_req *sr)
560 {
561     /* The _res_ calls get no reply. */
562     return (FALSE);
563 }

565 /*
566  * Version 2 svc functions (used by local statd)
567  */

569 bool_t
570 nlm_sm_notify1_2_svc(struct nlm_sm_status *argp, void *resp,
571     struct svc_req *sr)
572 {
573     nlm_do_notify1(argp, resp, sr);
574     return (TRUE);
575 }

577 bool_t
578 nlm_sm_notify2_2_svc(struct nlm_sm_status *argp, void *resp,
579     struct svc_req *sr)
580 {
581     nlm_do_notify2(argp, resp, sr);
582     return (TRUE);
583 }

585 /*
586  * Version 3 svc functions
587  */

589 bool_t

```

```

590 nlm_share_3_svc(nlm_shareargs *argp, nlm_sharereres *resp,
591     struct svc_req *sr)
592 {
593     nlm4_shareargs args4;
594     nlm4_sharereres res4;

596     bzero(&res4, sizeof (res4));

598     args4.cookie = argp->cookie;
599     nlm_convert_to_nlm4_share(&args4.share, &argp->share);
600     args4.reclaim = argp->reclaim;

602     nlm_do_share(&args4, &res4, sr);

604     resp->cookie = res4.cookie;
605     resp->stat = nlm_convert_to_nlm_stats(res4.stat);
606     resp->sequence = res4.sequence;

608     return (TRUE);
609 }

611 bool_t
612 nlm_unshare_3_svc(nlm_shareargs *argp, nlm_sharereres *resp,
613     struct svc_req *sr)
614 {
615     nlm4_shareargs args4;
616     nlm4_sharereres res4;

618     bzero(&res4, sizeof (res4));

620     args4.cookie = argp->cookie;
621     nlm_convert_to_nlm4_share(&args4.share, &argp->share);
622     args4.reclaim = argp->reclaim;

624     nlm_do_unshare(&args4, &res4, sr);

626     resp->cookie = res4.cookie;
627     resp->stat = nlm_convert_to_nlm_stats(res4.stat);
628     resp->sequence = res4.sequence;

630     return (TRUE);
631 }

633 bool_t
634 nlm_nm_lock_3_svc(nlm_lockargs *argp, nlm_res *resp, struct svc_req *sr)
635 {
636     nlm4_lockargs args4;
637     nlm4_res res4;

639     bzero(&res4, sizeof (res4));

641     args4.cookie = argp->cookie;
642     args4.block = argp->block;
643     args4.exclusive = argp->exclusive;
644     nlm_convert_to_nlm4_lock(&args4.alock, &argp->alock);
645     args4.reclaim = argp->reclaim;
646     args4.state = argp->state;

648     /*
649     * Don't allow blocking for non-monitored (nm_lock) calls.
650     * These clients don't handle any callbacks, including
651     * the granted call we make after a blocking lock.
652     * Same reply callback as nlm_lock_1_svc
653     */
654     args4.block = FALSE;

```

```

656     /* NLM_NM_LOCK */
657     nlm_do_lock(&args4, &res4, sr,
658               nlm_lock_1_reply, NULL,
659               NULL); /* indicates non-monitored */

661     /* for freeresult */
662     nlm_convert_to_nlm_res(resp, &res4);

664     /* above does its own reply */
665     return (FALSE);
666 }

668 bool_t
669 nlm_free_all_3_svc(nlm_notify *argp, void *resp, struct svc_req *sr)
670 {
671     struct nlm4_notify args4;
672
673     args4.name = argp->name;
674     args4.state = argp->state;

676     nlm_do_free_all(&args4, resp, sr);

678     return (TRUE);
679 }

681 /*
682  * Version 4 svc functions
683  */

685 bool_t
686 nlm4_test_4_svc(nlm4_testargs *argp, nlm4_testres *resp, struct svc_req *sr)
687 {
688     nlm_do_test(argp, resp, sr, NULL);
689     return (TRUE);
690 }

692 /*
693  * Callback functions for nlm4_lock_4_svc
694  */
695 static bool_t nlm4_lock_4_reply(SVCXPRT *, nlm4_res *);
696 static enum clnt_stat nlm4_granted_4_cb(nlm4_testargs *, void *, CLIENT *);

698 bool_t
699 nlm4_lock_4_svc(nlm4_lockargs *argp, nlm4_res *resp,
700                struct svc_req *sr)
701 {
702
703     /* NLM4_LOCK */
704     nlm_do_lock(argp, resp, sr,
705               nlm4_lock_4_reply, NULL,
706               nlm4_granted_4_cb);

708     /* above does its own reply */
709     return (FALSE);
710 }

712 static bool_t
713 nlm4_lock_4_reply(SVCXPRT *transp, nlm4_res *resp)
714 {
715     return (svc_sendreply(transp, xdr_nlm4_res, (char *)resp));
716 }

718 static enum clnt_stat
719 nlm4_granted_4_cb(nlm4_testargs *argp, void *resp, CLIENT *clnt)
720 {
721     nlm4_res res4;

```

```

722     int rv;

724     bzero(&res4, sizeof (res4));
725     rv = nlm4_granted_4(argp, &res4, clnt);

727     /* NB: We have a result our caller will not free. */
728     xdr_free((xdrproc_t)xdr_nlm4_res, (void *)&res4);
729     (void) resp;

731     return (rv);
732 }

734 bool_t
735 nlm4_cancel_4_svc(nlm4_cancargs *argp, nlm4_res *resp, struct svc_req *sr)
736 {
737     nlm_do_cancel(argp, resp, sr, NULL);
738     return (TRUE);
739 }

741 bool_t
742 nlm4_unlock_4_svc(nlm4_unlockargs *argp, nlm4_res *resp, struct svc_req *sr)
743 {
744     nlm_do_unlock(argp, resp, sr, NULL);
745     return (TRUE);
746 }

748 bool_t
749 nlm4_granted_4_svc(nlm4_testargs *argp, nlm4_res *resp, struct svc_req *sr)
750 {
751     nlm_do_granted(argp, resp, sr, NULL);
752     return (TRUE);
753 }

755 bool_t
756 nlm4_test_msg_4_svc(nlm4_testargs *argp, void *resp, struct svc_req *sr)
757 {
758     nlm4_testres res4;

760     bzero(&res4, sizeof (res4));
761     nlm_do_test(argp, &res4, sr,
762               nlm4_test_res_4);

764     /* NB: We have a result our caller will not free. */
765     xdr_free((xdrproc_t)xdr_nlm4_testres, (void *)&res4);
766     (void) resp;

768     /* The _msg_ calls get no reply. */
769     return (FALSE);
770 }

772 /*
773  * Callback functions for nlm4_lock_msg_4_svc
774  * (using the RPC client stubs directly)
775  */

777 bool_t
778 nlm4_lock_msg_4_svc(nlm4_lockargs *argp, void *resp,
779                    struct svc_req *sr)
780 {
781     nlm4_res res4;

783     /* NLM4_LOCK_MSG */
784     bzero(&res4, sizeof (res4));
785     nlm_do_lock(argp, &res4, sr,
786               NULL, nlm4_lock_res_4,
787               nlm4_granted_msg_4);

```

```

789     /* NB: We have a result our caller will not free. */
790     xdr_free((xdrproc_t)xdr_nlm4_res, (void *)&res4);
791     (void) resp;

793     /* The _msg_ calls get no reply. */
794     return (FALSE);
795 }

797 bool_t
798 nlm4_cancel_msg_4_svc(nlm4_cancargs *argp, void *resp, struct svc_req *sr)
799 {
800     nlm4_res res4;

802     bzero(&res4, sizeof (res4));
803     nlm_do_cancel(argp, &res4, sr,
804                 nlm4_cancel_res_4);

806     /* NB: We have a result our caller will not free. */
807     xdr_free((xdrproc_t)xdr_nlm4_res, (void *)&res4);
808     (void) resp;

810     /* The _msg_ calls get no reply. */
811     return (FALSE);
812 }

814 bool_t
815 nlm4_unlock_msg_4_svc(nlm4_unlockargs *argp, void *resp, struct svc_req *sr)
816 {
817     nlm4_res res4;

819     bzero(&res4, sizeof (res4));
820     nlm_do_unlock(argp, &res4, sr,
821                 nlm4_unlock_res_4);

823     /* NB: We have a result our caller will not free. */
824     xdr_free((xdrproc_t)xdr_nlm4_res, (void *)&res4);
825     (void) resp;

827     /* The _msg_ calls get no reply. */
828     return (FALSE);
829 }

831 bool_t
832 nlm4_granted_msg_4_svc(nlm4_testargs *argp, void *resp, struct svc_req *sr)
833 {
834     nlm4_res res4;

836     bzero(&res4, sizeof (res4));
837     nlm_do_granted(argp, &res4, sr,
838                 nlm4_granted_res_4);

840     /* NB: We have a result our caller will not free. */
841     xdr_free((xdrproc_t)xdr_nlm4_res, (void *)&res4);
842     (void) resp;

844     /* The _msg_ calls get no reply. */
845     return (FALSE);
846 }

848 /* ARGSUSED */
849 bool_t
850 nlm4_test_res_4_svc(nlm4_testres *argp, void *resp, struct svc_req *sr)
851 {
852     /* The _res_ calls get no reply. */
853     return (FALSE);

```

```

854 }

856 /* ARGSUSED */
857 bool_t
858 nlm4_lock_res_4_svc(nlm4_res *argp, void *resp, struct svc_req *sr)
859 {
860     /* The _res_ calls get no reply. */
861     return (FALSE);
862 }

864 /* ARGSUSED */
865 bool_t
866 nlm4_cancel_res_4_svc(nlm4_res *argp, void *resp, struct svc_req *sr)
867 {
868     /* The _res_ calls get no reply. */
869     return (FALSE);
870 }

872 /* ARGSUSED */
873 bool_t
874 nlm4_unlock_res_4_svc(nlm4_res *argp, void *resp, struct svc_req *sr)
875 {
876     /* The _res_ calls get no reply. */
877     return (FALSE);
878 }

880 /* ARGSUSED */
881 bool_t
882 nlm4_granted_res_4_svc(nlm4_res *argp, void *resp, struct svc_req *sr)
883 {
884     /* The _res_ calls get no reply. */
885     return (FALSE);
886 }

888 /* ARGSUSED */
889 bool_t
890 nlm4_share_4_svc(nlm4_shareargs *argp, nlm4_sharerres *resp,
891                struct svc_req *sr)
892 {
893     nlm_do_share(argp, resp, sr);
894     return (TRUE);
895 }

897 /* ARGSUSED */
898 bool_t
899 nlm4_unshare_4_svc(nlm4_shareargs *argp, nlm4_sharerres *resp,
900                  struct svc_req *sr)
901 {
902     nlm_do_unshare(argp, resp, sr);
903     return (TRUE);
904 }

906 bool_t
907 nlm4_nm_lock_4_svc(nlm4_lockargs *argp, nlm4_res *resp, struct svc_req *sr)
908 {
909     /*
910      * Don't allow blocking for non-monitored (nm_lock) calls.
911      * These clients don't handle any callbacks, including
912      * the granted call we make after a blocking lock.
913      * Same reply callback as nlm4_lock_4_svc
914      */
915     argp->block = FALSE;

918     /* NLM4_NM_LOCK */
919     nlm_do_lock(argp, resp, sr,

```

```
920         nlm4_lock_4_reply, NULL,  
921         NULL); /* indicates non-monitored */  
  
923         /* above does its own reply */  
924         return (FALSE);  
925     }  
  
927     bool_t  
928     nlm4_free_all_4_svc(nlm4_notify *argp, void *resp, struct svc_req *sr)  
929     {  
930         nlm_do_free_all(argp, resp, sr);  
931         return (TRUE);  
932     }
```

```
*****
```

```
30560 Fri Aug 16 09:04:55 2013
```

```
new/usr/src/uts/common/klm/nlm_service.c
```

```
195 Need replacement for nfs/lockd+klm
```

```
*****
```

```

1 /*
2  * Copyright (c) 2008 Isilon Inc http://www.isilon.com/
3  * Authors: Doug Rabson <df@rabson.org>
4  * Developed with Red Inc: Alfred Perlstein <alfred@freebsd.org>
5  *
6  * Redistribution and use in source and binary forms, with or without
7  * modification, are permitted provided that the following conditions
8  * are met:
9  * 1. Redistributions of source code must retain the above copyright
10 * notice, this list of conditions and the following disclaimer.
11 * 2. Redistributions in binary form must reproduce the above copyright
12 * notice, this list of conditions and the following disclaimer in the
13 * documentation and/or other materials provided with the distribution.
14 *
15 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
16 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
17 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
18 * ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
19 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
20 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
21 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
22 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
23 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
24 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
25 * SUCH DAMAGE.
26 */

28 /*
29  * Copyright 2011 Nexenta Systems, Inc. All rights reserved.
30  * Copyright (c) 2012 by Delphix. All rights reserved.
31 */

33 /*
34  * NFS Lock Manager service functions (nlm_do...)
35  * Called from nlm_rpc_svc.c wrappers.
36  *
37  * Source code derived from FreeBSD nlm_prot_impl.c
38 */

40 #include <sys/param.h>
41 #include <sys/system.h>
42 #include <sys/thread.h>
43 #include <sys/fcntl.h>
44 #include <sys/flock.h>
45 #include <sys/mount.h>
46 #include <sys/priv.h>
47 #include <sys/proc.h>
48 #include <sys/share.h>
49 #include <sys/socket.h>
50 #include <sys/syscall.h>
51 #include <sys/syslog.h>
52 #include <sys/system.h>
53 #include <sys/taskq.h>
54 #include <sys/unistd.h>
55 #include <sys/vnode.h>
56 #include <sys/vfs.h>
57 #include <sys/queue.h>
58 #include <sys/sdt.h>
59 #include <netinet/in.h>

61 #include <rpc/rpc.h>

```

```

62 #include <rpc/xdr.h>
63 #include <rpc/pmap_prot.h>
64 #include <rpc/pmap_clnt.h>
65 #include <rpc/rpcb_prot.h>

67 #include <rpcsvc/nlm_prot.h>
68 #include <rpcsvc/sm_inter.h>

70 #include <nfs/nfs.h>
71 #include <nfs/nfs_clnt.h>
72 #include <nfs/export.h>
73 #include <nfs/rnode.h>

75 #include "nlm_impl.h"

77 #define NLM_IN_GRACE(g) (ddi_get_lbolt() < (g)->grace_threshold)

79 struct nlm_block_cb_data {
80     struct nlm_host      *hostp;
81     struct nlm_vhold    *nvp;
82     struct flock64      *flp;
83 };

85 /*
86  * Invoke an asynchronous RPC callback
87  * (used when NLM server needs to reply to MSG NLM procedure).
88 */
89 #define NLM_INVOKE_CALLBACK(descr, rpcp, resp, callb) \
90     do { \
91         enum clnt_stat _stat; \
92         \
93         _stat = (*(callb))(resp, NULL, (rpcp)->nr_handle); \
94         if (_stat != RPC_SUCCESS && _stat != RPC_TIMEDOUT) { \
95             struct rpc_err _err; \
96             \
97             CLNT_GETERR((rpcp)->nr_handle, &_err); \
98             NLM_ERR("NLM: %s callback failed: " \
99                 "stat %d, err %d\n", descr, _stat, \
100                 _err.re_errno); \
101         } \
102         \
103         _NOTE(CONSTCOND) } while (0)

105 static void nlm_block(
106     nlm4_lockargs *lockargs,
107     struct nlm_host *host,
108     struct nlm_vhold *nvp,
109     nlm_rpc_t *rpcp,
110     struct flock64 *fl,
111     nlm_testargs_cb grant_cb);

113 static vnode_t *nlm_fh_to_vp(struct netobj *);
114 static struct nlm_vhold *nlm_fh_to_vhold(struct nlm_host *, struct netobj *);
115 static void nlm_init_shrlock(struct shrlock *, nlm4_share *, struct nlm_host *);
116 static callb_cpr_t *nlm_block_callback(flk_cb_when_t, void *);
117 static int nlm_vop_frlock(vnode_t *, int, flock64_t *, int, offset_t,
118     struct flk_callback *, cred_t *, caller_context_t *);

120 /*
121  * Convert a lock from network to local form, and
122  * check for valid range (no overflow).
123 */
124 static int
125 nlm_init_flock(struct flock64 *fl, struct nlm4_lock *nl,
126     struct nlm_host *host, rpcvers_t vers, short type)
127 {

```



```

128     uint64_t off, len;

130     bzero(fl, sizeof (*fl));
131     off = nl->l_offset;
132     len = nl->l_len;

134     if (vers < NLM4_VERS) {
135         if (off > MAX_UOFF32 || len > MAX_UOFF32)
136             return (EINVAL);
137         if (off + len > MAX_UOFF32 + 1)
138             return (EINVAL);
139     } else {
140         /*
141          * Check range for 64-bit client (no overflow).
142          * Again allow len == ~0 to mean lock to EOF.
143          */
144         if (len == MAX_U_OFFSET_T)
145             len = 0;
146         if (len != 0 && off + (len - 1) < off)
147             return (EINVAL);
148     }

150     fl->l_type = type;
151     fl->l_whence = SEEK_SET;
152     fl->l_start = off;
153     fl->l_len = len;
154     fl->l_sysid = host->nh_sysid;
155     fl->l_pid = nl->svid;
156     /* l_pad */

158     return (0);
159 }

161 /*
162  * Gets vnode from client's filehandle
163  * NOTE: Holds vnode, it _must_ be explicitly
164  * released by VN_RELE().
165  */
166 static vnode_t *
167 nlm_fh_to_vp(struct netobj *fh)
168 {
169     fhandle_t *fhp;

171     /*
172      * Get a vnode pointer for the given NFS file handle.
173      * Note that it could be an NFSv2 for NFSv3 handle,
174      * which means the size might vary. (don't copy)
175      */
176     if (fh->n_len < sizeof (*fhp))
177         return (NULL);

179     /* We know this is aligned (kmem_alloc) */
180     /* LINTED E_BAD_PTR_CAST_ALIGN */
181     fhp = (fhandle_t *)fh->n_bytes;
182     return (lm_fhtovp(fhp));
183 }

185 /*
186  * Get vhold from client's filehandle, but in contrast to
187  * The function tries to check some access rights as well.
188  *
189  * NOTE: vhold object _must_ be explicitly released by
190  * nlm_vhold_release().
191  */
192 static struct nlm_vhold *
193 nlm_fh_to_vhold(struct nlm_host *hostp, struct netobj *fh)

```

```

194 {
195     vnode_t *vp;
196     struct nlm_vhold *nvp;

198     vp = nlm_fh_to_vp(fh);
199     if (vp == NULL)
200         return (NULL);

203     nvp = nlm_vhold_get(hostp, vp);

205     /*
206      * Both nlm_fh_to_vp() and nlm_vhold_get()
207      * do VN_HOLD(), so we need to drop one
208      * reference on vnode.
209      */
210     VN_RELE(vp);
211     return (nvp);
212 }

214 /* ***** */

216 /*
217  * NLM implementation details, called from the RPC svc code.
218  */

220 /*
221  * Call-back from NFS statd, used to notify that one of our
222  * hosts had a status change. The host can be either an
223  * NFS client, NFS server or both.
224  * According to NSM protocol description, the state is a
225  * number that is increases monotonically each time the
226  * state of host changes. An even number indicates that
227  * the host is down, while an odd number indicates that
228  * the host is up.
229  *
230  * Here we ignore this even/odd difference of status number
231  * reported by the NSM, we launch notification handlers
232  * every time the state is changed. The reason we why do so
233  * is that client and server can talk to each other using
234  * connectionless transport and it's easy to lose packet
235  * containing NSM notification with status number update.
236  *
237  * In nlm_host_monitor(), we put the sysid in the private data
238  * that statd carries in this callback, so we can easliiy find
239  * the host this call applies to.
240  */
241 /* ARGSUSED */
242 void
243 nlm_do_notify1(nlm_sm_status *argp, void *res, struct svc_req *sr)
244 {
245     struct nlm_globals *g;
246     struct nlm_host *host;
247     uint16_t sysid;

249     g = zone_getspecific(nlm_zone_key, curzone);
250     bcopy(&argp->priv, &sysid, sizeof (sysid));

252     DTRACE_PROBE2(nsm_notify, uint16_t, sysid,
253                 int, argp->state);

255     host = nlm_host_find_by_sysid(g, (sysid_t)sysid);
256     if (host == NULL)
257         return;

259     nlm_host_notify_server(host, argp->state);

```

```

260     nlm_host_notify_client(host, argp->state);
261     nlm_host_release(g, host);
262 }

264 /*
265  * Another available call-back for NFS statd.
266  * Not currently used.
267  */
268 /* ARGSUSED */
269 void
270 nlm_do_notify2(nlm_sm_status *argp, void *res, struct svc_req *sr)
271 {
272     ASSERT(0);
273 }

276 /*
277  * NLM_TEST, NLM_TEST_MSG,
278  * NLM4_TEST, NLM4_TEST_MSG,
279  * Client inquiry about locks, non-blocking.
280  */
281 void
282 nlm_do_test(nlm4_testargs *argp, nlm4_testres *resp,
283            struct svc_req *sr, nlm_testres_cb cb)
284 {
285     struct nlm_globals *g;
286     struct nlm_host *host;
287     struct nlm4_holder *lh;
288     struct nlm_owner_handle *oh;
289     nlm_rpc_t *rpcp = NULL;
290     vnode_t *vp = NULL;
291     struct netbuf *addr;
292     char *netid;
293     char *name;
294     int error;
295     struct flock64 fl;

297     nlm_copy_netobj(&resp->cookie, &argp->cookie);

299     name = argp->alock.caller_name;
300     netid = svc_getnetid(sr->rq_xprt);
301     addr = svc_getrpccaller(sr->rq_xprt);

303     g = zone_getspecific(nlm_zone_key, curzone);
304     host = nlm_host_findcreate(g, name, netid, addr);
305     if (host == NULL) {
306         resp->stat.stat = nlm4_denied_nolocks;
307         return;
308     }
309     if (cb != NULL) {
310         error = nlm_host_get_rpc(host, sr->rq_vers, &rpcp);
311         if (error != 0) {
312             resp->stat.stat = nlm4_denied_nolocks;
313             goto out;
314         }
315     }

317     vp = nlm_fh_to_vp(&argp->alock.fh);
318     if (vp == NULL) {
319         resp->stat.stat = nlm4_stale_fh;
320         goto out;
321     }

323     if (NLM_IN_GRACE(g)) {
324         resp->stat.stat = nlm4_denied_grace_period;
325         goto out;

```

```

326     }

328     /* Convert to local form. */
329     error = nlm_init_flock(&fl, &argp->alock, host, sr->rq_vers,
330                          (argp->exclusive) ? F_WRLCK : F_RDLCK);
331     if (error) {
332         resp->stat.stat = nlm4_failed;
333         goto out;
334     }

336     /* BSD: VOP_ADVLOCK(nv->nv_vp, NULL, F_GETLK, &fl, F_REMOTE); */
337     error = nlm_vop_frlock(vp, F_GETLK, &fl,
338                          F_REMOTELOCK | F_READ | F_WRITE,
339                          (u_offset_t)0, NULL, CRED(), NULL);
340     if (error) {
341         resp->stat.stat = nlm4_failed;
342         goto out;
343     }

345     if (fl.l_type == F_UNLCK) {
346         resp->stat.stat = nlm4_granted;
347         goto out;
348     }
349     resp->stat.stat = nlm4_denied;

351     /*
352      * This lock "test" fails due to a conflicting lock.
353      *
354      * If this is a v1 client, make sure the conflicting
355      * lock range we report can be expressed with 32-bit
356      * offsets. The lock range requested was expressed
357      * as 32-bit offset and length, so at least part of
358      * the conflicting lock should lie below MAX_UOFF32.
359      * If the conflicting lock extends past that, we'll
360      * trim the range to end at MAX_UOFF32 so this lock
361      * can be represented in a 32-bit response. Check
362      * the start also (paranoid, but a low cost check).
363      */
364     if (sr->rq_vers < NLM4_VERS) {
365         uint64 maxlen;
366         if (fl.l_start > MAX_UOFF32)
367             fl.l_start = MAX_UOFF32;
368         maxlen = MAX_UOFF32 + 1 - fl.l_start;
369         if (fl.l_len > maxlen)
370             fl.l_len = maxlen;
371     }

373     /*
374      * Build the nlm4_holder result structure.
375      *
376      * Note that lh->oh is freed via xdr_free,
377      * xdr_nlm4_holder, xdr_netobj, xdr_bytes.
378      */
379     oh = kmem_zalloc(sizeof(*oh), KM_SLEEP);
380     oh->oh_sysid = (sysid_t)fl.l_sysid;
381     lh = &resp->stat.nlm4_testreply_u	holder;
382     lh->exclusive = (fl.l_type == F_WRLCK);
383     lh->svid = fl.l_pid;
384     lh->oh.n_len = sizeof(*oh);
385     lh->oh.n_bytes = (void *)oh;
386     lh->l_offset = fl.l_start;
387     lh->l_len = fl.l_len;

389 out:
390     /*
391      * If we have a callback funtion, use that to

```

```

392     * deliver the response via another RPC call.
393     */
394     if (cb != NULL && rpcp != NULL)
395         NLM_INVOKE_CALLBACK("test", rpcp, resp, cb);
397
398     if (vp != NULL)
399         VN_RELE(vp);
400     if (rpcp != NULL)
401         nlm_host_rele_rpc(host, rpcp);
402
403     nlm_host_release(g, host);
404 }
405
406 /*
407 * NLM_LOCK, NLM_LOCK_MSG, NLM_NM_LOCK
408 * NLM4_LOCK, NLM4_LOCK_MSG, NLM4_NM_LOCK
409 * Client request to set a lock, possibly blocking.
410 *
411 * If the lock needs to block, we return status blocked to
412 * this RPC call, and then later call back the client with
413 * a "granted" callback. Tricky aspects of this include:
414 * sending a reply before this function returns, and then
415 * borrowing this thread from the RPC service pool for the
416 * wait on the lock and doing the later granted callback.
417 *
418 * We also have to keep a list of locks (pending + granted)
419 * both to handle retransmitted requests, and to keep the
420 * vnodes for those locks active.
421 */
422 void
423 nlm_do_lock(nlm4_lockargs *argp, nlm4_res *resp, struct svc_req *sr,
424            nlm_reply_cb reply_cb, nlm_res_cb res_cb, nlm_testargs_cb grant_cb)
425 {
426     struct nlm_globals *g;
427     struct flock64 fl;
428     struct nlm_host *host = NULL;
429     struct netbuf *addr;
430     struct nlm_vhold *nvp = NULL;
431     nlm_rpc_t *rpcp = NULL;
432     char *netid;
433     char *name;
434     int error, flags;
435     bool_t do_blocking = FALSE;
436     bool_t do_mon_req = FALSE;
437     enum nlm4_stats status;
438
439     nlm_copy_netobj(&resp->cookie, &argp->cookie);
440
441     name = argp->alock.caller_name;
442     netid = svc_getnetid(sr->rq_xprt);
443     addr = svc_getrpccaller(sr->rq_xprt);
444
445     g = zone_getspecific(nlm_zone_key, curzone);
446     host = nlm_host_findcreate(g, name, netid, addr);
447     if (host == NULL) {
448         DTRACE_PROBE4(no_host, struct nlm_globals *, g,
449                     char *, name, char *, netid, struct netbuf *, addr);
450         status = nlm4_denied_nolocks;
451         goto doreply;
452     }
453
454     DTRACE_PROBE3(start, struct nlm_globals *, g,
455                 struct nlm_host *, host, nlm4_lockargs *, argp);
456
457     /*

```

```

458     * If we may need to do _msg_call needing an RPC
459     * callback, get the RPC client handle now,
460     * so we know if we can bind to the NLM service on
461     * this client.
462     *
463     * Note: host object carries transport type.
464     * One client using multiple transports gets
465     * separate sysids for each of its transports.
466     */
467     if (res_cb != NULL || (grant_cb != NULL && argp->block == TRUE)) {
468         error = nlm_host_get_rpc(host, sr->rq_vers, &rpcp);
469         if (error != 0) {
470             status = nlm4_denied_nolocks;
471             goto doreply;
472         }
473     }
474
475     /*
476     * During the "grace period", only allow reclaim.
477     */
478     if (argp->reclaim == 0 && NLM_IN_GRACE(g)) {
479         status = nlm4_denied_grace_period;
480         goto doreply;
481     }
482
483     /*
484     * Check whether we missed host shutdown event
485     */
486     if (nlm_host_get_state(host) != argp->state)
487         nlm_host_notify_server(host, argp->state);
488
489     /*
490     * Get a hold on the vnode for a lock operation.
491     * Only lock() and share() need vhold objects.
492     */
493     nvp = nlm_fh_to_vhold(host, &argp->alock.fh);
494     if (nvp == NULL) {
495         status = nlm4_stale_fh;
496         goto doreply;
497     }
498
499     /* Convert to local form. */
500     error = nlm_init_flock(&fl, &argp->alock, host, sr->rq_vers,
501                          (argp->exclusive) ? F_WRLCK : F_RDLCK);
502     if (error) {
503         status = nlm4_failed;
504         goto doreply;
505     }
506
507     /*
508     * Try to lock non-blocking first. If we succeed
509     * getting the lock, we can reply with the granted
510     * status directly and avoid the complications of
511     * making the "granted" RPC callback later.
512     *
513     * This also let's us find out now about some
514     * possible errors like EROFS, etc.
515     */
516     flags = F_REMOTELOCK | FREAD | FWRITE;
517     error = nlm_vop_frlock(nvp->nv_vp, F_SETLK, &fl, flags,
518                          (u_offset_t)0, NULL, CRED(), NULL);
519
520     DTRACE_PROBE3(setlk_res, struct flock64 *, &fl,
521                 int, flags, int, error);
522
523     switch (error) {

```

```

524     case 0:
525         /* Got it without waiting! */
526         status = nlm4_granted;
527         do_mon_req = TRUE;
528         break;

530     /* EINPROGRESS too? */
531     case EAGAIN:
532         /* We did not get the lock. Should we block? */
533         if (argp->block == FALSE || grant_cb == NULL) {
534             status = nlm4_denied;
535             break;
536         }
537         /*
538          * Should block. Try to reserve this thread
539          * so we can use it to wait for the lock and
540          * later send the granted message. If this
541          * reservation fails, say "no resources".
542          */
543         if (!svc_reserve_thread(sr->rq_xprt)) {
544             status = nlm4_denied_nolocks;
545             break;
546         }
547         /*
548          * OK, can detach this thread, so this call
549          * will block below (after we reply).
550          */
551         status = nlm4_blocked;
552         do_blocking = TRUE;
553         do_mon_req = TRUE;
554         break;

556     case ENOLCK:
557         /* Failed for lack of resources. */
558         status = nlm4_denied_nolocks;
559         break;

561     case EROFS:
562         /* read-only file system */
563         status = nlm4_rofs;
564         break;

566     case EFBIG:
567         /* file too big */
568         status = nlm4_fbig;
569         break;

571     case EDEADLK:
572         /* dead lock condition */
573         status = nlm4_deadlck;
574         break;

576     default:
577         status = nlm4_denied;
578         break;
579 }

581 doreply:
582     resp->stat.stat = status;

584     /*
585     * We get one of two function pointers; one for a
586     * normal RPC reply, and another for doing an RPC
587     * "callback" _res reply for a _msg function.
588     * Use either of those to send the reply now.
589     */

```

```

590     * If sending this reply fails, just leave the
591     * lock in the list for retransmitted requests.
592     * Cleanup is via unlock or host rele (statmon).
593     */
594     if (reply_cb != NULL) {
595         /* i.e. nlm_lock_1_reply */
596         if (!(*reply_cb)(sr->rq_xprt, resp))
597             svcerr_systemerr(sr->rq_xprt);
598     }
599     if (res_cb != NULL && rpcp != NULL)
600         NLM_INVOKE_CALLBACK("lock", rpcp, resp, res_cb);

602     /*
603     * The reply has been sent to the client.
604     * Start monitoring this client (maybe).
605     *
606     * Note that the non-monitored (NM) calls pass grant_cb=NULL
607     * indicating that the client doesn't support RPC callbacks.
608     * No monitoring for these (lame) clients.
609     */
610     if (do_mon_req && grant_cb != NULL)
611         nlm_host_monitor(g, host, argp->state);

613     if (do_blocking) {
614         /*
615         * We need to block on this lock, and when that
616         * completes, do the granted RPC call. Note that
617         * we "reserved" this thread above, so we can now
618         * "detach" it from the RPC SVC pool, allowing it
619         * to block indefinitely if needed.
620         */
621         ASSERT(rpcp != NULL);
622         (void) svc_detach_thread(sr->rq_xprt);
623         nlm_block(argp, host, nvp, rpcp, &fl, grant_cb);
624     }

626     DTRACE_PROBE3(lock__end, struct nlm_globals *, g,
627                 struct nlm_host *, host, nlm4_res *, resp);

629     if (rpcp != NULL)
630         nlm_host_rele_rpc(host, rpcp);

632     nlm_vhold_release(host, nvp);
633     nlm_host_release(g, host);
634 }

636 /*
637 * Helper for nlm_do_lock(), partly for observability,
638 * (we'll see a call blocked in this function) and
639 * because nlm_do_lock() was getting quite long.
640 */
641 static void
642 nlm_block(nlm4_lockargs *lockargs,
643          struct nlm_host *host,
644          struct nlm_vhold *nvp,
645          nlm_rpc_t *rpcp,
646          struct flock64 *flp,
647          nlm_testargs_cb grant_cb)
648 {
649     nlm4_testargs args;
650     int error;
651     flk_callback_t flk_cb;
652     struct nlm_block_cb_data cb_data;

654     /*
655     * Keep a list of blocked locks on nh_pending, and use it

```

```

656     * to cancel these threads in nlm_destroy_client_pending.
657     *
658     * Check to see if this lock is already in the list
659     * and if not, add an entry for it. Allocate first,
660     * then if we don't insert, free the new one.
661     * Caller already has vp held.
662     */
664     error = nlm_slreq_register(host, nvp, flp);
665     if (error != 0) {
666         /*
667          * Sleeping lock request with given fl is already
668          * registered by someone else. This means that
669          * some other thread is handling the request, let
670          * him to do its work.
671          */
672         ASSERT(error == EEXIST);
673         return;
674     }
676     cb_data.hostp = host;
677     cb_data.nvp = nvp;
678     cb_data.flp = flp;
679     flk_init_callback(&flk_cb, nlm_block_callback, &cb_data);
681     /* BSD: VOP_ADVLOCK(vp, NULL, F_SETLK, fl, F_REMOTE); */
682     error = nlm_vop_frlock(nvp->nv_vp, F_SETLKW, flp,
683         F_REMOTELOCK | F_READ | F_WRITE,
684         (u_offset_t)0, &flk_cb, CRED(), NULL);
686     if (error != 0) {
687         /*
688          * We failed getting the lock, but have no way to
689          * tell the client about that. Let 'em time out.
690          */
691         (void) nlm_slreq_unregister(host, nvp, flp);
692         return;
693     }
695     /*
696     * Do the "granted" call-back to the client.
697     */
698     args.cookie = lockargs->cookie;
699     args.exclusive = lockargs->exclusive;
700     args.alock = lockargs->alock;
702     NLM_INVOKE_CALLBACK("grant", rpcp, &args, grant_cb);
703 }
705 /*
706 * The function that is used as flk callback when NLM server
707 * sets new sleeping lock. The function unregisters NLM
708 * sleeping lock request (nlm_slreq) associated with the
709 * sleeping lock _before_ lock becomes active. It prevents
710 * potential race condition between nlm_block() and
711 * nlm_do_cancel().
712 */
713 static callb_cpr_t *
714 nlm_block_callback(flk_cb_when_t when, void *data)
715 {
716     struct nlm_block_cb_data *cb_data;
718     cb_data = (struct nlm_block_cb_data *)data;
719     if (when == FLK_AFTER_SLEEP) {
720         (void) nlm_slreq_unregister(cb_data->hostp,
721             cb_data->nvp, cb_data->flp);

```

```

722     }
724     return (0);
725 }
727 /*
728 * NLM_CANCEL, NLM_CANCEL_MSG,
729 * NLM4_CANCEL, NLM4_CANCEL_MSG,
730 * Client gives up waiting for a blocking lock.
731 */
732 void
733 nlm_do_cancel(nlm4_cancargs *argp, nlm4_res *resp,
734     struct svc_req *sr, nlm_res_cb cb)
735 {
736     struct nlm_globals *g;
737     struct nlm_host *host;
738     struct netbuf *addr;
739     struct nlm_vhold *nvp = NULL;
740     nlm_rpc_t *rpcp = NULL;
741     char *netid;
742     char *name;
743     int error;
744     struct flock64 fl;
746     nlm_copy_netobj(&resp->cookie, &argp->cookie);
747     netid = svc_getnetid(sr->rq_xprt);
748     addr = svc_getrpccaller(sr->rq_xprt);
749     name = argp->alock.caller_name;
751     g = zone_getspecific(nlm_zone_key, curzone);
752     host = nlm_host_findcreate(g, name, netid, addr);
753     if (host == NULL) {
754         resp->stat.stat = nlm4_denied_nolocks;
755         return;
756     }
757     if (cb != NULL) {
758         error = nlm_host_get_rpc(host, sr->rq_vers, &rpcp);
759         if (error != 0) {
760             resp->stat.stat = nlm4_denied_nolocks;
761             return;
762         }
763     }
765     DTRACE_PROBE3(start, struct nlm_globals *, g,
766         struct nlm_host *, host, nlm4_cancargs *, argp);
768     if (NLM_IN_GRACE(g)) {
769         resp->stat.stat = nlm4_denied_grace_period;
770         goto out;
771     }
773     nvp = nlm_fh_to_vhold(host, &argp->alock.fh);
774     if (nvp == NULL) {
775         resp->stat.stat = nlm4_stale_fh;
776         goto out;
777     }
779     /* Convert to local form. */
780     error = nlm_init_flock(&fl, &argp->alock, host, sr->rq_vers,
781         (argp->exclusive) ? F_WRLCK : F_RDLCK);
782     if (error) {
783         resp->stat.stat = nlm4_failed;
784         goto out;
785     }
787     error = nlm_slreq_unregister(host, nvp, &fl);

```

```

788     if (error != 0) {
789         /*
790          * There's no sleeping lock request corresponding
791          * to the lock. Then requested sleeping lock
792          * doesn't exist.
793          */
794         resp->stat.stat = nlm4_denied;
795         goto out;
796     }

798     fl.l_type = F_UNLCK;
799     error = nlm_vop_frlock(nvp->nv_vp, F_SETLK, &fl,
800         F_REMOTELOCK | FREAD | FWRITE,
801         (u_offset_t)0, NULL, CRED(), NULL);

803     resp->stat.stat = (error == 0) ?
804         nlm4_granted : nlm4_denied;

806 out:
807     /*
808     * If we have a callback funtion, use that to
809     * deliver the response via another RPC call.
810     */
811     if (cb != NULL && rpcp != NULL)
812         NLM_INVOKE_CALLBACK("cancel", rpcp, resp, cb);

814     DTRACE_PROBE3(cancel__end, struct nlm_globals *, g,
815         struct nlm_host *, host, nlm4_res *, resp);

817     if (rpcp != NULL)
818         nlm_host_rele_rpc(host, rpcp);

820     nlm_vhold_release(host, nvp);
821     nlm_host_release(g, host);
822 }

824 /*
825  * NLM_UNLOCK, NLM_UNLOCK_MSG,
826  * NLM4_UNLOCK, NLM4_UNLOCK_MSG,
827  * Client removes one of their locks.
828  */
829 void
830 nlm_do_unlock(nlm4_unlockargs *argp, nlm4_res *resp,
831     struct svc_req *sr, nlm_res_cb cb)
832 {
833     struct nlm_globals *g;
834     struct nlm_host *host;
835     struct netbuf *addr;
836     nlm_rpc_t *rpcp = NULL;
837     vnode_t *vp = NULL;
838     char *netid;
839     char *name;
840     int error;
841     struct flock64 fl;

843     nlm_copy_netobj(&resp->cookie, &argp->cookie);

845     netid = svc_getnetid(sr->rq_xprt);
846     addr = svc_getrppcaller(sr->rq_xprt);
847     name = argp->alock.caller_name;

849     /*
850     * NLM_UNLOCK operation doesn't have an error code
851     * denoting that operation failed, so we always
852     * return nlm4_granted except when the server is
853     * in a grace period.

```

```

854     /*
855     resp->stat.stat = nlm4_granted;

857     g = zone_getspecific(nlm_zone_key, curzone);
858     host = nlm_host_findcreate(g, name, netid, addr);
859     if (host == NULL)
860         return;

862     if (cb != NULL) {
863         error = nlm_host_get_rpc(host, sr->rq_vers, &rpcp);
864         if (error != 0)
865             goto out;
866     }

868     DTRACE_PROBE3(start, struct nlm_globals *, g,
869         struct nlm_host *, host, nlm4_unlockargs *, argp);

871     if (NLM_IN_GRACE(g)) {
872         resp->stat.stat = nlm4_denied_grace_period;
873         goto out;
874     }

876     vp = nlm_fh_to_vp(&argp->alock.fh);
877     if (vp == NULL)
878         goto out;

880     /* Convert to local form. */
881     error = nlm_init_flock(&fl, &argp->alock, host, sr->rq_vers, F_UNLCK);
882     if (error)
883         goto out;

885     /* BSD: VOP_ADVLOCK(nv->nv_vp, NULL, F_UNLCK, &fl, F_REMOTE); */
886     error = nlm_vop_frlock(vp, F_SETLK, &fl,
887         F_REMOTELOCK | FREAD | FWRITE,
888         (u_offset_t)0, NULL, CRED(), NULL);

890     DTRACE_PROBE1(unlock__res, int, error);
891 out:
892     /*
893     * If we have a callback funtion, use that to
894     * deliver the response via another RPC call.
895     */
896     if (cb != NULL && rpcp != NULL)
897         NLM_INVOKE_CALLBACK("unlock", rpcp, resp, cb);

899     DTRACE_PROBE3(unlock__end, struct nlm_globals *, g,
900         struct nlm_host *, host, nlm4_res *, resp);

902     if (vp != NULL)
903         VN_RELE(vp);
904     if (rpcp != NULL)
905         nlm_host_rele_rpc(host, rpcp);

907     nlm_host_release(g, host);
908 }

910 /*
911  * NLM_GRANTED, NLM_GRANTED_MSG,
912  * NLM4_GRANTED, NLM4_GRANTED_MSG,
913  *
914  * This service routine is special. It's the only one that's
915  * really part of our NLM_client_support, used by _servers_
916  * to "call back" when a blocking lock from this NLM client
917  * is granted by the server. In this case, we know there is
918  * already an nlm_host allocated and held by the client code.
919  * We want to find that nlm_host here.

```

```

920 *
921 * Over in nlm_call_lock(), the client encoded the sysid for this
922 * server in the "owner handle" netbuf sent with our lock request.
923 * We can now use that to find the nlm_host object we used there.
924 * (NB: The owner handle is opaque to the server.)
925 */
926 void
927 nlm_do_granted(nlm4_testargs *argp, nlm4_res *resp,
928              struct svc_req *sr, nlm_res_cb cb)
929 {
930     struct nlm_globals *g;
931     struct nlm_owner_handle *oh;
932     struct nlm_host *host;
933     nlm_rpc_t *rpcp = NULL;
934     int error;
935
936     nlm_copy_netobj(&resp->cookie, &argp->cookie);
937     resp->stat.stat = nlm4_denied;
938
939     g = zone_getspecific(nlm_zone_key, curzone);
940     oh = (void *) argp->alock.oh.n_bytes;
941     if (oh == NULL)
942         return;
943
944     host = nlm_host_find_by_sysid(g, oh->oh_sysid);
945     if (host == NULL)
946         return;
947
948     if (cb != NULL) {
949         error = nlm_host_get_rpc(host, sr->rq_vers, &rpcp);
950         if (error != 0)
951             goto out;
952     }
953
954     if (NLM_IN_GRACE(g)) {
955         resp->stat.stat = nlm4_denied_grace_period;
956         goto out;
957     }
958
959     error = nlm_slock_grant(g, host, &argp->alock);
960     if (error == 0)
961         resp->stat.stat = nlm4_granted;
962
963 out:
964     /*
965     * If we have a callback funtion, use that to
966     * deliver the response via another RPC call.
967     */
968     if (cb != NULL && rpcp != NULL)
969         NLM_INVOKE_CALLBACK("do_granted", rpcp, resp, cb);
970
971     if (rpcp != NULL)
972         nlm_host_rele_rpc(host, rpcp);
973
974     nlm_host_release(g, host);
975 }
976
977 /*
978 * NLM_FREE_ALL, NLM4_FREE_ALL
979 *
980 * Destroy all lock state for the calling client.
981 */
982 void
983 nlm_do_free_all(nlm4_notify *argp, void *res, struct svc_req *sr)
984 {
985     struct nlm_globals *g;

```

```

986     struct nlm_host_list host_list;
987     struct nlm_host *hostp;
988
989     TAILQ_INIT(&host_list);
990     g = zone_getspecific(nlm_zone_key, curzone);
991
992     /* Serialize calls to clean locks. */
993     mutex_enter(&g->clean_lock);
994
995     /*
996     * Find all hosts that have the given node name and put them on a
997     * local list.
998     */
999     mutex_enter(&g->lock);
1000     for (hostp = avl_first(&g->nlm_hosts_tree); hostp != NULL;
1001         hostp = AVL_NEXT(&g->nlm_hosts_tree, hostp)) {
1002         if (strcasemp(hostp->nh_name, argp->name) == 0) {
1003             /*
1004             * If needed take the host out of the idle list since
1005             * we are taking a reference.
1006             */
1007             if (hostp->nh_flags & NLM_NH_IDLE) {
1008                 TAILQ_REMOVE(&g->nlm_idle_hosts, hostp, nh_link)
1009                 hostp->nh_flags &= ~NLM_NH_IDLE;
1010             }
1011             hostp->nh_refs++;
1012
1013             TAILQ_INSERT_TAIL(&host_list, hostp, nh_link);
1014         }
1015     }
1016     mutex_exit(&g->lock);
1017
1018     /* Free locks for all hosts on the local list. */
1019     while (!TAILQ_EMPTY(&host_list)) {
1020         hostp = TAILQ_FIRST(&host_list);
1021         TAILQ_REMOVE(&host_list, hostp, nh_link);
1022
1023         /*
1024         * Note that this does not do client-side cleanup.
1025         * We want to do that ONLY if statd tells us the
1026         * server has restarted.
1027         */
1028         nlm_host_notify_server(hostp, argp->state);
1029         nlm_host_release(g, hostp);
1030     }
1031
1032     mutex_exit(&g->clean_lock);
1033
1034     (void) res;
1035     (void) sr;
1036 }
1037
1038 static void
1039 nlm_init_shrlock(struct shrlock *shr,
1040                 nlm4_share *nshare, struct nlm_host *host)
1041 {
1042
1043     switch (nshare->access) {
1044     default:
1045     case fsa_NONE:
1046         shr->s_access = 0;
1047         break;
1048     case fsa_R:
1049         shr->s_access = F_RDACC;
1050         break;
1051     case fsa_W:

```

```

1052         shr->s_access = F_WRACC;
1053         break;
1054     case fsa_RW:
1055         shr->s_access = F_RWACC;
1056         break;
1057     }

1059     switch (nshare->mode) {
1060     default:
1061     case fsm_DN:
1062         shr->s_deny = F_NODNY;
1063         break;
1064     case fsm_DR:
1065         shr->s_deny = F_RDDNY;
1066         break;
1067     case fsm_DW:
1068         shr->s_deny = F_WRDNY;
1069         break;
1070     case fsm_DRW:
1071         shr->s_deny = F_RWDNY;
1072         break;
1073     }

1075     shr->s_sysid = host->nh_sysid;
1076     shr->s_pid = 0;
1077     shr->s_own_len = nshare->oh.n_len;
1078     shr->s_owner = nshare->oh.n_bytes;
1079 }

1081 /*
1082  * NLM_SHARE, NLM4_SHARE
1083  */
1084 * Request a DOS-style share reservation
1085 */
1086 void
1087 nlm_do_share(nlm4_shareargs *argp, nlm4_sharereres *resp, struct svc_req *sr)
1088 {
1089     struct nlm_globals *g;
1090     struct nlm_host *host;
1091     struct netbuf *addr;
1092     struct nlm_vhold *nvp = NULL;
1093     char *netid;
1094     char *name;
1095     int error;
1096     struct shrlock shr;

1098     nlm_copy_netobj(&resp->cookie, &argp->cookie);

1100     name = argp->share.caller_name;
1101     netid = svc_getnetid(sr->rq_xprt);
1102     addr = svc_getrpccaller(sr->rq_xprt);

1104     g = zone_getspecific(nlm_zone_key, curzone);
1105     host = nlm_host_findcreate(g, name, netid, addr);
1106     if (host == NULL) {
1107         resp->stat = nlm4_denied_nolocks;
1108         return;
1109     }

1111     DTRACE_PROBE3(share__start, struct nlm_globals *, g,
1112                  struct nlm_host *, host, nlm4_shareargs *, argp);

1114     if (argp->reclaim == 0 && NLM_IN_GRACE(g)) {
1115         resp->stat = nlm4_denied_grace_period;
1116         goto out;
1117     }

```

```

1119     /*
1120     * Get holded vnode when on lock operation.
1121     * Only lock() and share() need vhold objects.
1122     */
1123     nvp = nlm_fh_to_vhold(host, &argp->share.fh);
1124     if (nvp == NULL) {
1125         resp->stat = nlm4_stale_fh;
1126         goto out;
1127     }

1129     /* Convert to local form. */
1130     nlm_init_shrlock(&shr, &argp->share, host);
1131     error = VOP_SHRLOCK(nvp->nv_vp, F_SHARE, &shr,
1132                       FREAD | FWRITE, CRED(), NULL);

1134     if (error == 0) {
1135         resp->stat = nlm4_granted;
1136         nlm_host_monitor(g, host, 0);
1137     } else {
1138         resp->stat = nlm4_denied;
1139     }

1141 out:
1142     DTRACE_PROBE3(share__end, struct nlm_globals *, g,
1143                  struct nlm_host *, host, nlm4_sharereres *, resp);

1145     nlm_vhold_release(host, nvp);
1146     nlm_host_release(g, host);
1147 }

1149 /*
1150  * NLM_UNSHARE, NLM4_UNSHARE
1151  */
1152 * Release a DOS-style share reservation
1153 */
1154 void
1155 nlm_do_unshare(nlm4_shareargs *argp, nlm4_sharereres *resp, struct svc_req *sr)
1156 {
1157     struct nlm_globals *g;
1158     struct nlm_host *host;
1159     struct netbuf *addr;
1160     vnode_t *vp = NULL;
1161     char *netid;
1162     int error;
1163     struct shrlock shr;

1165     nlm_copy_netobj(&resp->cookie, &argp->cookie);

1167     netid = svc_getnetid(sr->rq_xprt);
1168     addr = svc_getrpccaller(sr->rq_xprt);

1170     g = zone_getspecific(nlm_zone_key, curzone);
1171     host = nlm_host_find(g, netid, addr);
1172     if (host == NULL) {
1173         resp->stat = nlm4_denied_nolocks;
1174         return;
1175     }

1177     DTRACE_PROBE3(unshare__start, struct nlm_globals *, g,
1178                  struct nlm_host *, host, nlm4_shareargs *, argp);

1180     if (NLM_IN_GRACE(g)) {
1181         resp->stat = nlm4_denied_grace_period;
1182         goto out;
1183     }

```



```
1185     vp = nlm_fh_to_vp(&argp->share.fh);
1186     if (vp == NULL) {
1187         resp->stat = nlm4_stale_fh;
1188         goto out;
1189     }
1191     /* Convert to local form. */
1192     nlm_init_shrlock(&shr, &argp->share, host);
1193     error = VOP_SHRLOCK(vp, F_UNSHARE, &shr,
1194         FREAD | FWRITE, CRED(), NULL);
1196     (void) error;
1197     resp->stat = nlm4_granted;
1199 out:
1200     DTRACE_PROBE3(unshare__end, struct nlm_globals *, g,
1201         struct nlm_host *, host, nlm4_sharereres *, resp);
1203     if (vp != NULL)
1204         VN_RELE(vp);
1206     nlm_host_release(g, host);
1207 }
1209 /*
1210  * NLM wrapper to VOP_FRLOCK that checks the validity of the lock before
1211  * invoking the vnode operation.
1212  */
1213 static int
1214 nlm_vop_frlock(vnode_t *vp, int cmd, flock64_t *bfp, int flag, offset_t offset,
1215     struct flk_callback *flk_cbp, cred_t *cr, caller_context_t *ct)
1216 {
1217     if (bfp->l_len != 0 && bfp->l_start + (bfp->l_len - 1)
1218         < bfp->l_start) {
1219         return (EOVERFLOW);
1220     }
1222     return (VOP_FRLOCK(vp, cmd, bfp, flag, offset, flk_cbp, cr, ct));
1223 }
```

new/usr/src/uts/common/klm/nsm_addr_clnt.sed

1

687 Fri Aug 16 09:04:56 2013

new/usr/src/uts/common/klm/nsm_addr_clnt.sed

195 Need replacement for nfs/lockd+klm

```
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 is of the CDDL is also available via the Internet
9 # at http://www.illumos.org/license/CDDL.
10 #
11 #
12 # Copyright (c) 2012 by Delphix. All rights reserved.
13 #
14 #
15 #
16 # This sed script is run on the client code generated by rpcgen
17 # from nsm_addr.x before it is compiled.
18 #
19 #
20 6{
21 i\
22 #include <sys/param.h>
23 i\
24 #include <sys/system.h>
25 i\
26 #include <rpcsvc/nsm_addr.h>
27 }
28 /\.include/,/^.endif/d
```

new/usr/src/uts/common/klm/sm_inter_clnt.sed

1

693 Fri Aug 16 09:04:56 2013

new/usr/src/uts/common/klm/sm_inter_clnt.sed

195 Need replacement for nfs/lockd+klm

```
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 is of the CDDL is also available via the Internet
9 # at http://www.illumos.org/license/CDDL.
10 #
11 #
12 # Copyright 2011 Nexenta Systems, Inc. All rights reserved.
13 #
14 #
15 #
16 # This sed script is run on the client code generated by rpcgen
17 # from sm_inter.x before it is compiled.
18 #
19 #
20 6{
21 i\
22 #include <sys/param.h>
23 i\
24 #include <sys/system.h>
25 i\
26 #include <rpcsvc/sm_inter.h>
27 }
28 /\.include/,/^.endif/d
```

new/usr/src/uts/common/nfs/lm.h

1

```
*****
4867 Fri Aug 16 09:04:57 2013
new/usr/src/uts/common/nfs/lm.h
195 Need replacement for nfs/lockd+klm
*****
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, Version 1.0 only
6  * (the "License"). You may not use this file except in compliance
7  * with the License.
8  *
9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 * or http://www.opensolaris.org/os/licensing.
11 * See the License for the specific language governing permissions
12 * and limitations under the License.
13 *
14 * When distributing Covered Code, include this CDDL HEADER in each
15 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 * If applicable, add the following below this CDDL HEADER, with the
17 * fields enclosed by brackets "[]" replaced with your own identifying
18 * information: Portions Copyright [yyyy] [name of copyright owner]
19 *
20 * CDDL HEADER END
21 */
22 /*
23  * Copyright 2005 Sun Microsystems, Inc. All rights reserved.
24  * Use is subject to license terms.
25 */
27 /*
28  * Copyright 2011 Nexenta Systems, Inc. All rights reserved.
29 */
31 #ifndef _NFS_LM_H
32 #define _NFS_LM_H
30 #pragma ident "%Z%M% %I% %E% SMI"
34 /*
35  * Interface definitions for the NFSv2/v3 lock manager.
36 */
38 #ifdef __cplusplus
39 extern "C" {
40 #endif
42 #include <sys/cred.h>
43 #include <sys/fcntl.h>
44 #include <sys/types.h>
45 #include <sys/vnode.h>
46 #include <rpc/rpc.h>
45 #include <nfs/export.h>
48 #ifdef _KERNEL
50 /*
51  * Common interfaces.
52 */
54 struct exportinfo;
56 /*
57  * The numeric sysid is used to identify a host and transport.
58 */
```

new/usr/src/uts/common/nfs/lm.h

2

```
59 * The local locking code uses (pid, sysid) to uniquely identify a process.
60 * This means that the client-side code must doctor up the sysid before
61 * registering a lock, so that the local locking code doesn't confuse a
62 * remote process with a local process just because they have the same pid.
63 * We currently do this by ORing LM_SYSID_CLIENT into the sysid before
64 * registering a lock.
65 *
66 * If you change LM_SYSID and LM_SYSID_MAX, be sure to pick values so that
67 * LM_SYSID_MAX > LM_SYSID using signed arithmetic, and don't use zero.
68 * You may also need a different way to tag lock manager locks that are
69 * registered locally.
70 */
71 #define LM_SYSID ((sysid_t)0x0001)
72 #define LM_SYSID_MAX ((sysid_t)0x3FFF)
73 #define LM_SYSID_CLIENT ((sysid_t)0x4000)
74 #define LM_NOSYSID ((sysid_t)-1)
76 /*
77  * Struct used to represent a host.
78 */
79 struct lm_sysid;
81 /*
82  * Given a knetconfig and network address, returns a reference to the
83  * associated lm_sysid. The 3rd argument is the hostname to assign to the
84  * lm_sysid. The 4th argument is an output parameter. It is set non-zero
85  * if the returned lm_sysid has a different protocol
86  * (knetconfig::knc_proto) than what was requested.
87 */
88 extern struct lm_sysid *lm_get_sysid(struct knetconfig *, struct netbuf *,
89 char *, bool_t *);
90 extern void lm_rel_sysid(struct lm_sysid *);
92 /*
93  * Return the integer sysid for the given lm_sysid.
94 */
95 extern sysid_t lm_sysid(struct lm_sysid *);
97 extern void lm_free_config(struct knetconfig *);
99 extern void lm_cprrsuspend(void);
100 extern void lm_cprrresume(void);
102 /*
103  * Client-side interfaces.
104 */
106 extern int lm_frlock(struct vnode *vp, int cmd,
107 struct flock64 *flk, int flag,
108 u_offset_t offset, struct cred *cr,
109 netobj *fh, struct flk_callback *);
110 extern int lm_has_sleep(const struct vnode *);
111 extern void lm_register_lock_locally(vnode_t *,
112 struct lm_sysid *, struct flock64 *, int,
113 u_offset_t);
114 extern int lm_safelock(vnode_t *, const struct flock64 *,
115 cred_t *);
116 extern int lm_safemap(const vnode_t *);
117 extern int lm_shrlock(struct vnode *vp, int cmd,
118 struct shrlock *shr, int flag, netobj *fh);
119 extern int lm4_frlock(struct vnode *vp, int cmd,
120 struct flock64 *flk, int flag,
121 u_offset_t offset, struct cred *cr,
122 netobj *fh, struct flk_callback *);
123 extern int lm4_shrlock(struct vnode *vp, int cmd,
124 struct shrlock *shr, int flag, netobj *fh);
```

```
126 /*
127 * Server-side interfaces.
128 */

130 extern void          lm_unexport(struct exportinfo *);

132 /*
133 * Clustering: functions to encode the nlmid of the node where this NLM
134 * server is running in the l_sysid of the flock struct or the s_sysid
135 * field of the shrlock struct (respectively).
136 */
137 extern void          lm_set_nlmid_flk(int *);
138 extern void          lm_set_nlmid_shr(int32_t *);
139 /* Hook for deleting all mandatory NFSv4 file locks held by a remote client */
140 extern void (*lm_remove_file_locks)(int);

142 /*
143 * The following global variable is the node id of the node where this
144 * NLM server is running.
145 */
146 extern int lm_global_nlmid;

148 /*
149 * End of clustering hooks.
150 */

152 /*
153 * Return non-zero if the given local vnode is in use.
154 */
155 extern int lm_vp_active(const struct vnode *);

157 extern sysid_t      lm_alloc_sysidt(void);
158 extern void         lm_free_sysidt(sysid_t);

160 #endif /* _KERNEL */
157 #else /* _KERNEL */

162 #ifdef __STDC__
163 extern int lm_shutdown(void);
164 #else
165 extern int lm_shutdown();
166 #endif /* __STDC__ */

165 #endif /* _KERNEL */

168 #ifdef __cplusplus
169 }
unchanged_portion_omitted
```

```

*****
107103 Fri Aug 16 09:05:00 2013
new/usr/src/uts/common/os/flock.c
195 Need replacement for nfs/lockd+klm
*****
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 2007 Sun Microsystems, Inc. All rights reserved.
24  * Use is subject to license terms.
25  */

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

30 /*
31  * Copyright 2011 Nexenta Systems, Inc. All rights reserved.
32  */

34 #include <sys/flock_impl.h>
35 #include <sys/vfs.h>
36 #include <sys/t_lock.h>      /* for <sys/callb.h> */
37 #include <sys/callb.h>
38 #include <sys/clconf.h>
39 #include <sys/cladm.h>
40 #include <sys/nbmlock.h>
41 #include <sys/cred.h>
42 #include <sys/policy.h>

44 /*
45  * The following four variables are for statistics purposes and they are
46  * not protected by locks. They may not be accurate but will at least be
47  * close to the actual value.
48  */

50 int     flk_lock_allocs;
51 int     flk_lock_frees;
52 int     edge_allocs;
53 int     edge_frees;
54 int     flk_proc_vertex_allocs;
55 int     flk_proc_edge_allocs;
56 int     flk_proc_vertex_frees;
57 int     flk_proc_edge_frees;

59 static kmutex_t flock_lock;

61 #ifdef DEBUG

```

```

62 int check_debug = 0;
63 #define CHECK_ACTIVE_LOCKS(gp)  if (check_debug) \
64                                check_active_locks(gp);
65 #define CHECK_SLEEPING_LOCKS(gp)  if (check_debug) \
66                                    check_sleeping_locks(gp);
67 #define CHECK_OWNER_LOCKS(gp, pid, sysid, vp) \
68                                if (check_debug) \
69                                    check_owner_locks(gp, pid, sysid, vp);
70 #define CHECK_LOCK_TRANSITION(old_state, new_state) \
71     { \
72         if (check_lock_transition(old_state, new_state)) { \
73             cmn_err(CE_PANIC, "Illegal lock transition \
74                 from %d to %d", old_state, new_state); \
75         } \
76     }
77 #else

79 #define CHECK_ACTIVE_LOCKS(gp)
80 #define CHECK_SLEEPING_LOCKS(gp)
81 #define CHECK_OWNER_LOCKS(gp, pid, sysid, vp)
82 #define CHECK_LOCK_TRANSITION(old_state, new_state)

84 #endif /* DEBUG */

86 struct kmem_cache      *flk_edge_cache;

88 graph_t      *lock_graph[HASH_SIZE];
89 proc_graph_t pgraph;

91 /*
92  * Clustering.
93  *
94  * NLM REGISTRY TYPE IMPLEMENTATION
95  *
96  * Assumptions:
97  * 1. Nodes in a cluster are numbered starting at 1; always non-negative
98  *    integers; maximum node id is returned by clconf_maximum_nodeid().
99  * 2. We use this node id to identify the node an NLM server runs on.
100 */

102 /*
103  * NLM registry object keeps track of NLM servers via their
104  * nlmids (which are the node ids of the node in the cluster they run on)
105  * that have requested locks at this LLM with which this registry is
106  * associated.
107  *
108  * Representation of abstraction:
109  * rep = record[      states: array[nlm_state],
110  *                  lock: mutex]
111  *
112  * Representation invariants:
113  * 1. index i of rep.states is between 0 and n - 1 where n is number
114  *    of elements in the array, which happen to be the maximum number
115  *    of nodes in the cluster configuration + 1.
116  * 2. map nlmid to index i of rep.states
117  *    0 -> 0
118  *    1 -> 1
119  *    2 -> 2
120  *    n-1 -> clconf_maximum_nodeid()+1
121  * 3. This 1-1 mapping is quite convenient and it avoids errors resulting
122  *    from forgetting to subtract 1 from the index.
123  * 4. The reason we keep the 0th index is the following. A legitimate
124  *    cluster configuration includes making a UFS file system NFS
125  *    exportable. The code is structured so that if you're in a cluster
126  *    you do one thing; otherwise, you do something else. The problem
127  *    is what to do if you think you're in a cluster with PXFS loaded,

```

```

128 *      but you're using UFS not PXFS? The upper two bytes of the sysid
129 *      encode the node id of the node where NLM server runs; these bytes
130 *      are zero for UFS. Since the nodeid is used to index into the
131 *      registry, we can record the NLM server state information at index
132 *      0 using the same mechanism used for PXFS file locks!
133 */
134 static flk_nlm_status_t *nlm_reg_status = NULL; /* state array 0..N-1 */
135 static kmutex_t nlm_reg_lock; /* lock to protect array */
136 static uint_t nlm_status_size; /* size of state array */

138 /*
139 * Although we need a global lock dependency graph (and associated data
140 * structures), we also need a per-zone notion of whether the lock manager is
141 * running, and so whether to allow lock manager requests or not.
142 *
143 * Thus, on a per-zone basis we maintain a ``global'' variable
144 * (flk_lockmgr_status), protected by flock_lock, and set when the lock
145 * manager is determined to be changing state (starting or stopping).
146 *
147 * Each graph/zone pair also has a copy of this variable, which is protected by
148 * the graph's mutex.
149 *
150 * The per-graph copies are used to synchronize lock requests with shutdown
151 * requests. The global copy is used to initialize the per-graph field when a
152 * new graph is created.
153 */
154 struct flock_globals {
155     flk_lockmgr_status_t flk_lockmgr_status;
156     flk_lockmgr_status_t lockmgr_status[HASH_SIZE];
157 };
158 unchanged_portion_omitted

2244 /*
2245 * Determine whether there are any locks for the given vnode with a remote
2246 * sysid. Returns zero if not, non-zero if there are.
2247 *
2248 * Note that the return value from this function is potentially invalid
2249 * once it has been returned. The caller is responsible for providing its
2250 * own synchronization mechanism to ensure that the return value is useful
2251 * (e.g., see nfs_lockcompletion()).
2252 */
2253 int
2254 flk_has_remote_locks(vnode_t *vp)
2255 {
2256     lock_descriptor_t *lock;
2257     int result = 0;
2258     graph_t *gp;

2260     gp = flk_get_lock_graph(vp, FLK_USE_GRAPH);
2261     if (gp == NULL) {
2262         return (0);
2263     }

2265     mutex_enter(&gp->gp_mutex);

2267     SET_LOCK_TO_FIRST_ACTIVE_VP(gp, lock, vp);

2269     if (lock) {
2270         while (lock->l_vnode == vp) {
2271             if (IS_REMOTE(lock)) {
2272                 result = 1;
2273                 goto done;
2274             }
2275             lock = lock->l_next;
2276         }
2277     }

```

```

2279     SET_LOCK_TO_FIRST_SLEEP_VP(gp, lock, vp);

2281     if (lock) {
2282         while (lock->l_vnode == vp) {
2283             if (IS_REMOTE(lock)) {
2284                 result = 1;
2285                 goto done;
2286             }
2287             lock = lock->l_next;
2288         }
2289     }

2291 done:
2292     mutex_exit(&gp->gp_mutex);
2293     return (result);
2294 }

2296 /*
2297 * Determine whether there are any locks for the given vnode with a remote
2298 * sysid matching given sysid.
2299 * Used by the new (open source) NFS Lock Manager (NLM)
2300 */
2301 int
2302 flk_has_remote_locks_for_sysid(vnode_t *vp, int sysid)
2303 {
2304     lock_descriptor_t *lock;
2305     int result = 0;
2306     graph_t *gp;

2308     if (sysid == 0)
2309         return (0);

2311     gp = flk_get_lock_graph(vp, FLK_USE_GRAPH);
2312     if (gp == NULL) {
2313         return (0);
2314     }

2316     mutex_enter(&gp->gp_mutex);

2318     SET_LOCK_TO_FIRST_ACTIVE_VP(gp, lock, vp);

2320     if (lock) {
2321         while (lock->l_vnode == vp) {
2322             if (lock->l_flock.l_sysid == sysid) {
2323                 result = 1;
2324                 goto done;
2325             }
2326             lock = lock->l_next;
2327         }
2328     }

2330     SET_LOCK_TO_FIRST_SLEEP_VP(gp, lock, vp);

2332     if (lock) {
2333         while (lock->l_vnode == vp) {
2334             if (lock->l_flock.l_sysid == sysid) {
2335                 result = 1;
2336                 goto done;
2337             }
2338             lock = lock->l_next;
2339         }
2340     }

2342 done:
2343     mutex_exit(&gp->gp_mutex);

```

new/usr/src/uts/common/os/flock.c

5

```
2344         return (result);  
2345     }  
_____unchanged_portion_omitted_
```



```
*****
```

```
16601 Fri Aug 16 09:05:04 2013
```

```
new/usr/src/uts/common/os/share.c
```

```
195 Need replacement for nfs/lockd+klm
```

```
*****
```

```
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 2008 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */
```

```
26 /*
27  * Copyright 2011 Nexenta Systems, Inc. All rights reserved.
28 */
```

```
30 #include <sys/types.h>
31 #include <sys/sysmacros.h>
32 #include <sys/param.h>
33 #include <sys/system.h>
34 #include <sys/fcntl.h>
35 #include <sys/vfs.h>
36 #include <sys/vnode.h>
37 #include <sys/share.h>
38 #include <sys/cmn_err.h>
39 #include <sys/kmem.h>
40 #include <sys/debug.h>
41 #include <sys/t_lock.h>
42 #include <sys/errno.h>
43 #include <sys/nbmlck.h>
```

```
45 int share_debug = 0;
```

```
47 #ifdef DEBUG
48 static void print_shares(struct vnode *);
49 static void print_share(struct shrlock *);
50 #endif
```

```
52 static int isreadonly(struct vnode *);
53 static void do_cleanshares(struct vnode *, pid_t, int32_t);
```

```
56 /*
57  * Add the share reservation shr to vp.
58 */
59 int
60 add_share(struct vnode *vp, struct shrlock *shr)
61 {
```

```
62     struct shrlocklist *shrl;
63
64     /*
65      * An access of zero is not legal, however some older clients
66      * generate it anyways. Allow the request only if it is
67      * coming from a remote system. Be generous in what you
68      * accept and strict in what you send.
69      */
70     if ((shr->s_access == 0) && (GETSYSID(shr->s_sysid) == 0)) {
71         return (EINVAL);
72     }
73
74     /*
75      * Sanity check to make sure we have valid options.
76      * There is known overlap but it doesn't hurt to be careful.
77      */
78     if (shr->s_access & ~(F_RDACC|F_WRACC|F_RWACC|F_RMACC|F_MDACC)) {
79         return (EINVAL);
80     }
81     if (shr->s_deny & ~(F_NODNY|F_RDDNY|F_WRDNY|F_RWDNY|F_COMPAT|
82         F_MANDDNY|F_RMDNY)) {
83         return (EINVAL);
84     }
85
86     mutex_enter(&vp->v_lock);
87     for (shrl = vp->v_shrlocks; shrl != NULL; shrl = shrl->next) {
88         /*
89          * If the share owner matches previous request
90          * do special handling.
91          */
92         if ((shrl->shr->s_sysid == shr->s_sysid) &&
93             (shrl->shr->s_pid == shr->s_pid) &&
94             (shrl->shr->s_own_len == shr->s_own_len) &&
95             bcmp(shrl->shr->s_owner, shr->s_owner,
96                 shr->s_own_len) == 0) {
97
98             /*
99              * If the existing request is F_COMPAT and
100              * is the first share then allow any F_COMPAT
101              * from the same process. Trick: If the existing
102              * F_COMPAT is write access then it must have
103              * the same owner as the first.
104              */
105             if ((shrl->shr->s_deny & F_COMPAT) &&
106                 (shr->s_deny & F_COMPAT) &&
107                 ((shrl->next == NULL) ||
108                  (shr->shr->s_access & F_WRACC)))
109                 break;
110         }
111
112         /*
113          * If a first share has been done in compatibility mode
114          * handle the special cases.
115          */
116         if ((shrl->shr->s_deny & F_COMPAT) && (shrl->next == NULL)) {
117
118             if (!(shr->s_deny & F_COMPAT)) {
119                 /*
120                  * If not compat and want write access or
121                  * want to deny read or
122                  * write exists, fails
123                  */
124                 if ((shr->s_access & F_WRACC) ||
125                     (shr->s_deny & F_RDDNY) ||
126                     (shr->shr->s_access & F_WRACC)) {
127                     mutex_exit(&vp->v_lock);
```

```

128         return (EAGAIN);
129     }
130     /*
131     * If read only file allow, this may allow
132     * a deny write but that is meaningless on
133     * a read only file.
134     */
135     if (isreadonly(vp))
136         break;
137     mutex_exit(&vp->v_lock);
138     return (EAGAIN);
139 }
140 /*
141 * This is a compat request and read access
142 * and the first was also read access
143 * we always allow it, otherwise we reject because
144 * we have handled the only valid write case above.
145 */
146 if ((shr->s_access == F_RDACC) &&
147     (shrl->shr->s_access == F_RDACC))
148     break;
149 mutex_exit(&vp->v_lock);
150 return (EAGAIN);
151 }
152
153 /*
154 * If we are trying to share in compatibility mode
155 * and the current share is compat (and not the first)
156 * we don't know enough.
157 */
158 if ((shrl->shr->s_denied & F_COMPAT) && (shr->s_denied & F_COMPAT))
159     continue;
160
161 /*
162 * If this is a compat we check for what can't succeed.
163 */
164 if (shr->s_denied & F_COMPAT) {
165     /*
166     * If we want write access or
167     * if anyone is denying read or
168     * if anyone has write access we fail
169     */
170     if ((shr->s_access & F_WRACC) ||
171         (shrl->shr->s_denied & F_RDDNY) ||
172         (shrl->shr->s_access & F_WRACC)) {
173         mutex_exit(&vp->v_lock);
174         return (EAGAIN);
175     }
176     /*
177     * If the first was opened with only read access
178     * and is a read only file we allow.
179     */
180     if (shrl->next == NULL) {
181         if ((shr->shr->s_access == F_RDACC) &&
182             isreadonly(vp)) {
183             break;
184         }
185         mutex_exit(&vp->v_lock);
186         return (EAGAIN);
187     }
188     /*
189     * We still can't determine our fate so continue
190     */
191     continue;
192 }

```

```

194     /*
195     * Simple bitwise test, if we are trying to access what
196     * someone else is denying or we are trying to deny
197     * what someone else is accessing we fail.
198     */
199     if ((shr->s_access & shrl->shr->s_denied) ||
200         (shr->s_denied & shrl->shr->s_access)) {
201         mutex_exit(&vp->v_lock);
202         return (EAGAIN);
203     }
204 }
205
206 shrl = kmem_alloc(sizeof (struct shrlocklist), KM_SLEEP);
207 shrl->shr = kmem_alloc(sizeof (struct shrlock), KM_SLEEP);
208 shrl->shr->s_access = shr->s_access;
209 shrl->shr->s_denied = shr->s_denied;
210
211 /*
212 * Make sure no other deny modes are also set with F_COMPAT
213 */
214 if (shrl->shr->s_denied & F_COMPAT)
215     shrl->shr->s_denied = F_COMPAT;
216 shrl->shr->s_sysid = shr->s_sysid; /* XXX ref cnt? */
217 shrl->shr->s_pid = shr->s_pid;
218 shrl->shr->s_own_len = shr->s_own_len;
219 shrl->shr->s_owner = kmem_alloc(shr->s_own_len, KM_SLEEP);
220 bcopy(shr->s_owner, shrl->shr->s_owner, shr->s_own_len);
221 shrl->next = vp->v_shrlocks;
222 vp->v_shrlocks = shrl;
223 #ifdef DEBUG
224     if (share_debug)
225         print_shares(vp);
226 #endif
227
228     mutex_exit(&vp->v_lock);
229
230     return (0);
231 }
232
233 _____ unchanged_portion_omitted _____
234
235 void
236 cleanshares(struct vnode *vp, pid_t pid)
237 {
238     do_cleanshares(vp, pid, 0);
239 }
240
241 /*
242 * Cleanup all remote share reservations that
243 * were made by the given sysid on given vnode.
244 */
245 void
246 cleanshares_by_sysid(struct vnode *vp, int32_t sysid)
247 {
248     if (sysid == 0)
249         return;
250
251     do_cleanshares(vp, 0, sysid);
252 }
253
254 /*
255 * Cleanup share reservations on given vnode made
256 * by the either given pid or sysid.

```

```
363 * If sysid is 0, remove all shares made by given pid,
364 * otherwise all shares made by the given sysid will
365 * be removed.
366 */
367 static void
368 do_cleanshares(struct vnode *vp, pid_t pid, int32_t sysid)
369 {
370     struct shrlock shr;
371
372     if (vp->v_shrlocks == NULL)
373         return;
374
375     shr.s_access = 0;
376     shr.s_deny = 0;
377     shr.s_pid = pid;
378     shr.s_sysid = sysid;
379     shr.s_own_len = 0;
380     shr.s_owner = NULL;
381
382     (void) del_share(vp, &shr);
383 }
384
385 unchanged_portion_omitted
```

new/usr/src/uts/common/rpcsvc/Makefile

1

```
*****
2205 Fri Aug 16 09:05:07 2013
new/usr/src/uts/common/rpcsvc/Makefile
195 Need replacement for nfs/lockd+klm
*****
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #
21 #
22 # Copyright 2009 Sun Microsystems, Inc. All rights reserved.
23 # Use is subject to license terms.
24 #
25 # Copyright (c) 2012 by Delphix. All rights reserved.
26 #
27 # uts/common/rpcsvc/Makefile
28 # This makefile installs system header files that go into
29 # /usr/include/rpcsvc.
30 #
31 # include global definitions
32 include ../../../../Makefile.master

34 # Protocol descriptions. Alas, the NFS protocol cannot be expressed
35 # completely via rpcgen. The NLM description should go here some day.
36 # Also, the v3 headers have been hacked so that they no longer
37 # quite reflect what goes over the wire.
38 IDMAP_PROT_X= idmap_prot.x
39 RPCGEN_SRC= autofs_prot.x nlm_prot.x sm_inter.x nsm_addr.x \
40             $(IDMAP_PROT_X)
37 RPCGEN_SRC= autofs_prot.x sm_inter.x nsm_addr.x $(IDMAP_PROT_X)

42 DERIVED_HDRS= $(RPCGEN_SRC:%.x=%h)

44 ALLHDRS= $(RPCGEN_SRC) $(DERIVED_HDRS)

46 ROOTDIRS= $(ROOT)/usr/include/rpcsvc

48 ROOTHDRS= $(ALLHDRS:%=$(ROOTDIRS)/%)

50 RPCGENFLAGS = -C
51 idmap_prot.h := RPCGENFLAGS += -MN
52 nlm_prot.h := RPCGENFLAGS += -M
53 sm_inter.h := RPCGENFLAGS += -M
54 nsm_addr.h := RPCGENFLAGS += -M

56 $(ROOTDIRS)%: %
57     $(INS.file)

59 .KEEP_STATE:
```

new/usr/src/uts/common/rpcsvc/Makefile

2

```
61 # all_h permits derived headers to be built here in the uts source area
62 # for the kernel to reference, without going so far as to install them.
63 #
64 all_h: $(DERIVED_HDRS)

66 install_h: all_h $(ROOTDIRS) $(ROOTHDRS)

68 clean:
69     $(RM) $(DERIVED_HDRS)

71 clobber: clean

73 # Don't check rpcgen-derived files.
74 check:

76 $(ROOTDIRS):
77     $(INS.dir)

79 %.h: %.x
80     $(RPCGEN) $(RPCGENFLAGS) -h $< -o $@
71     $(RPCGEN) -C -h $< -o $@

73 idmap_prot.h: $(IDMAP_PROT_X)
74     $(RPCGEN) -CMNh -o $@ $(IDMAP_PROT_X)
```

new/usr/src/uts/common/rpcsvc/nlm_prot.x

1

```
*****
9615 Fri Aug 16 09:05:08 2013
new/usr/src/uts/common/rpcsvc/nlm_prot.x
195 Need replacement for nfs/lockd+klm
*****
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, Version 1.0 only
6  * (the "License"). You may not use this file except in compliance
7  * with the License.
8  *
9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 * or http://www.opensolaris.org/os/licensing.
11 * See the License for the specific language governing permissions
12 * and limitations under the License.
13 *
14 * When distributing Covered Code, include this CDDL HEADER in each
15 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 * If applicable, add the following below this CDDL HEADER, with the
17 * fields enclosed by brackets "[]" replaced with your own identifying
18 * information: Portions Copyright [yyyy] [name of copyright owner]
19 *
20 * CDDL HEADER END
21 */
22 /*
23  * Network lock manager protocol definition
24  * Copyright (C) 1986, 1992, 1993, 1997, 1999 by Sun Microsystems, Inc.
25  * All rights reserved.
26  * Copyright 2011 Nexenta Systems, Inc. All rights reserved.
27  *
28  * Protocol used between local lock manager and remote lock manager.
29  *
30  * There are currently 3 versions of the protocol in use. Versions 1
31  * and 3 are used with NFS version 2. Version 4 is used with NFS
32  * version 3.
33  *
34  * (Note: there is also a version 2, but it defines an orthogonal set of
35  * procedures that the status monitor uses to notify the lock manager of
36  * changes in monitored systems.)
37  */
38 #pragma ident "%Z%M% %I% %E% SMI"
39 #if RPC_HDR
40 %
41 #include <rpc/rpc_sztypes.h>
42 %
43 #endif
44
45 #ifdef RPC_HDR
46 #define LM_MAXSTRLEN 1024
47 #define LM_MAXNAMELEN (LM_MAXSTRLEN + 1)
48 #endif
49
50 /*
51  * Types for versions 1 and 3.
52  */
53
54 /*
55  * Status of a call to the lock manager. The lower case enums violate the
56  * current style guide, but we're stuck with 'em.
57  */
```

new/usr/src/uts/common/rpcsvc/nlm_prot.x

2

```
59 enum nlm_stats {
60     nlm_granted = 0,
61     nlm_denied = 1,
62     nlm_denied_nolocks = 2,
63     nlm_blocked = 3,
64     nlm_denied_grace_period = 4,
65     nlm_deadlock = 5
66 };
    unchanged_portion_omitted
185 /*
186  * Types for version 4.
187  *
188  * This revision is designed to work with NFS V3. The main changes from
189  * NFS V2 to V3 that affect the NLM protocol are that all file offsets
190  * and sizes are now unsigned 64-bit ints, and file handles are now
191  * variable length. In NLM V1 and V3, the fixed-length V2 file handle
192  * was encoded as a 'netobj', which is a count followed by the data
193  * bytes. For NLM 4, the file handle is already a count followed by
194  * data bytes, so the handle is copied directly into the netobj, rather
195  * than being encoded with an additional byte count.
196  */
197
198 /*
199  * Status of a call to the lock manager.
200  */
201
202 enum nlm4_stats {
203     nlm4_granted = 0, /* lock was granted */
204     nlm4_denied = 1, /* lock was not granted, usually */
205     NLM4_GRANTED = 0, /* lock was granted */
206     NLM4_DENIED = 1, /* lock was not granted, usually */
207     /* due to conflicting lock */
208     nlm4_denied_nolocks = 2, /* not granted: out of resources */
209     nlm4_blocked = 3, /* not granted: expect callback */
210     NLM4_DENIED_NOLOCKS = 2, /* not granted: out of resources */
211     NLM4_BLOCKED = 3, /* not granted: expect callback */
212     /* when granted */
213     nlm4_denied_grace_period = 4, /* not granted: server is */
214     NLM4_DENIED_GRACE_PERIOD = 4, /* not granted: server is */
215     /* reestablishing old locks */
216     nlm4_deadlock = 5, /* not granted: deadlock detected */
217     nlm4_rofs = 6, /* not granted: read-only filesystem */
218     nlm4_stale_fh = 7, /* not granted: stale file handle */
219     nlm4_fbig = 8, /* not granted: offset or length */
220     NLM4_DEADLCK = 5, /* not granted: deadlock detected */
221     NLM4_ROFS = 6, /* not granted: read-only filesystem */
222     NLM4_STALE_FH = 7, /* not granted: stale file handle */
223     NLM4_FBIG = 8, /* not granted: offset or length */
224     /* too big */
225     nlm4_failed = 9 /* not granted: some other error */
226     NLM4_FAILED = 9 /* not granted: some other error */
227 };
    unchanged_portion_omitted
231 union nlm4_testrply switch (nlm4_stats stat) {
232     case nlm4_denied:
233     case NLM4_DENIED:
234         struct nlm4_holder holder;
235         void;
236 };
    unchanged_portion_omitted
289 #ifdef RPC_HDR
290 %/
```

```

291 % * The following enums are actually bit encoded for efficient
292 % * boolean algebra.... DON'T change them....
293 % */
294 #endif

296 enum fsh4_mode {
297     FSM_DN = 0, /* deny none */
298     FSM_DR = 1, /* deny read */
299     FSM_DW = 2, /* deny write */
300     FSM_DRW = 3 /* deny read/write */
301 };

303 enum fsh4_access {
304     FSA_NONE = 0, /* for completeness */
305     FSA_R = 1, /* read only */
306     FSA_W = 2, /* write only */
307     FSA_RW = 3 /* read/write */
308 };

288 struct nlm4_share {
289     string caller_name<LM_MAXSTRLEN>;
290     netobj fh;
291     netobj oh;
292     fsh4_mode mode;
293     fsh4_access access;
314     fsh4_mode mode;
315     fsh4_access access;
294 };
    unchanged_portion_omitted_

313 /*
314 * Argument for the NLM call-back procedure called by rpc.statd
315 * when a monitored host status changes. The statd calls the
316 * NLM prog,vers,proc specified in the SM_MON call.
317 * NB: This struct must exactly match sm_inter.x:sm_status
318 * and requires LM_MAXSTRLEN == SM_MAXSTRLEN
319 */
320 struct nlm_sm_status {
321     string mon_name<LM_MAXSTRLEN>; /* name of host */
322     int32 state; /* new state */
323     opaque priv[16]; /* private data */
324 };

326 /*
327 * Over-the-wire protocol used between the network lock managers
328 */

330 program NLM_PROG {

332     version NLM_VERS {

334         void
335         NLM_NULL(void) = 0;

337         nlm_testres
338         NLM_TEST(nlm_testargs) = 1;

340         nlm_res
341         NLM_LOCK(nlm_lockargs) = 2;

343         nlm_res
344         NLM_CANCEL(nlm_cancargs) = 3;

346         nlm_res
347         NLM_UNLOCK(nlm_unlockargs) = 4;
348         /*

```

```

349         * remote lock manager call-back to grant lock
350         */
351         nlm_res
352         NLM_GRANTED(nlm_testargs) = 5;

354         /*
355         * message passing style of requesting lock
356         */

358         void
359         NLM_TEST_MSG(nlm_testargs) = 6;
360         void
361         NLM_LOCK_MSG(nlm_lockargs) = 7;
362         void
363         NLM_CANCEL_MSG(nlm_cancargs) = 8;
364         void
365         NLM_UNLOCK_MSG(nlm_unlockargs) = 9;
366         void
367         NLM_GRANTED_MSG(nlm_testargs) = 10;
368         void
369         NLM_TEST_RES(nlm_testres) = 11;
370         void
371         NLM_LOCK_RES(nlm_res) = 12;
372         void
373         NLM_CANCEL_RES(nlm_res) = 13;
374         void
375         NLM_UNLOCK_RES(nlm_res) = 14;
376         void
377         NLM_GRANTED_RES(nlm_res) = 15;
378     } = 1;

380     /*
381     * Private (loopback-only) call-backs from statd,
382     * used to notify that some machine has restarted.
383     * The meaning of these is up to the lock manager
384     * implementation. (See the SM_MON calls.)
385     */
386     version NLM_SM {
387         void NLM_SM_NOTIFY1(struct nlm_sm_status) = 17;
388         void NLM_SM_NOTIFY2(struct nlm_sm_status) = 18;
389     } = 2;

391     version NLM_VERSX {
392         nlm_sharereres
393         NLM_SHARE(nlm_shareargs) = 20;
394         nlm_sharereres
395         NLM_UNSHARE(nlm_shareargs) = 21;
396         nlm_res
397         NLM_NM_LOCK(nlm_lockargs) = 22;
398         void
399         NLM_FREE_ALL(nlm_notify) = 23;
400     } = 3;

402     version NLM4_VERS {
403         void
404         NLM4_NULL(void) = 0;
405         NLMPROC4_NULL(void) = 0;
406         nlm4_testres
407         NLM4_TEST(nlm4_testargs) = 1;
408         NLMPROC4_TEST(nlm4_testargs) = 1;
409         nlm4_res
410         NLM4_LOCK(nlm4_lockargs) = 2;
411         NLMPROC4_LOCK(nlm4_lockargs) = 2;
412         nlm4_res
413         NLM4_CANCEL(nlm4_cancargs) = 3;
414         NLMPROC4_CANCEL(nlm4_cancargs) = 3;

```

```

411         nlm4_res
412             NLM4_UNLOCK(nlm4_unlockargs) = 4;
406             NLMPROC4_UNLOCK(nlm4_unlockargs) = 4;
413         /*
414          * remote lock manager call-back to grant lock
415          */
416         nlm4_res
417             NLM4_GRANTED(nlm4_testargs) = 5;
411             NLMPROC4_GRANTED(nlm4_testargs) = 5;

419         /*
420          * message passing style of requesting lock
421          */

423         void
424             NLM4_TEST_MSG(nlm4_testargs) = 6;
418             NLMPROC4_TEST_MSG(nlm4_testargs) = 6;
425         void
426             NLM4_LOCK_MSG(nlm4_lockargs) = 7;
420             NLMPROC4_LOCK_MSG(nlm4_lockargs) = 7;
427         void
428             NLM4_CANCEL_MSG(nlm4_cancargs) = 8;
422             NLMPROC4_CANCEL_MSG(nlm4_cancargs) = 8;
429         void
430             NLM4_UNLOCK_MSG(nlm4_unlockargs) = 9;
424             NLMPROC4_UNLOCK_MSG(nlm4_unlockargs) = 9;
431         void
432             NLM4_GRANTED_MSG(nlm4_testargs) = 10;
426             NLMPROC4_GRANTED_MSG(nlm4_testargs) = 10;
433         void
434             NLM4_TEST_RES(nlm4_testres) = 11;
428             NLMPROC4_TEST_RES(nlm4_testres) = 11;
435         void
436             NLM4_LOCK_RES(nlm4_res) = 12;
430             NLMPROC4_LOCK_RES(nlm4_res) = 12;
437         void
438             NLM4_CANCEL_RES(nlm4_res) = 13;
432             NLMPROC4_CANCEL_RES(nlm4_res) = 13;
439         void
440             NLM4_UNLOCK_RES(nlm4_res) = 14;
434             NLMPROC4_UNLOCK_RES(nlm4_res) = 14;
441         void
442             NLM4_GRANTED_RES(nlm4_res) = 15;
436             NLMPROC4_GRANTED_RES(nlm4_res) = 15;

444         /*
445          * DOS-style file sharing
446          */

448         nlm4_sharerres
449             NLM4_SHARE(nlm4_shareargs) = 20;
443             NLMPROC4_SHARE(nlm4_shareargs) = 20;
450         nlm4_sharerres
451             NLM4_UNSHARE(nlm4_shareargs) = 21;
445             NLMPROC4_UNSHARE(nlm4_shareargs) = 21;
452         nlm4_res
453             NLM4_NM_LOCK(nlm4_lockargs) = 22;
447             NLMPROC4_NM_LOCK(nlm4_lockargs) = 22;
454         void
455             NLM4_FREE_ALL(nlm4_notify) = 23;
449             NLMPROC4_FREE_ALL(nlm4_notify) = 23;
456     } = 4;

458 } = 100021;
    unchanged_portion_omitted_

```

new/usr/src/uts/common/rpcsvc/sm_inter.x

1

```
*****
3331 Fri Aug 16 09:05:09 2013
new/usr/src/uts/common/rpcsvc/sm_inter.x
195 Need replacement for nfs/lockd+klm
*****
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, Version 1.0 only
6  * (the "License"). You may not use this file except in compliance
7  * with the License.
8  *
9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 * or http://www.opensolaris.org/os/licensing.
11 * See the License for the specific language governing permissions
12 * and limitations under the License.
13 *
14 * When distributing Covered Code, include this CDDL HEADER in each
15 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 * If applicable, add the following below this CDDL HEADER, with the
17 * fields enclosed by brackets "[]" replaced with your own identifying
18 * information: Portions Copyright [yyyy] [name of copyright owner]
19 *
20 * CDDL HEADER END
21 */
22 %/*
23 % * Copyright (c) 1986, 1994 by Sun Microsystems, Inc.
24 % * All rights reserved.
25 % */

27 %/* from sm_inter.x */

29 #ifdef RPC_HDR
30 %
31 %#pragma ident "%Z%M% %I% %E% SMI"
32 %
33 #endif

29 /*
30  * Status monitor protocol specification
31  */

33 program SM_PROG {
34     version SM_VERS {
35         /* res_stat = stat_succ if status monitor agrees to monitor */
36         /* res_stat = stat_fail if status monitor cannot monitor */
37         /* if res_stat == stat_succ, state = state number of site */
38         /* sm_name */
39         struct sm_stat_res      SM_STAT(struct sm_name) = 1;

41         /* res_stat = stat_succ if status monitor agrees to monitor */
42         /* res_stat = stat_fail if status monitor cannot monitor */
43         /* stat consists of state number of local site */
44         struct sm_stat_res      SM_MON(struct mon) = 2;

46         /* stat consists of state number of local site */
47         struct sm_stat          SM_UNMON(struct mon_id) = 3;

49         /* stat consists of state number of local site */
50         struct sm_stat          SM_UNMON_ALL(struct my_id) = 4;

52         void                    SM_SIMU_CRASH(void) = 5;

54         void                    SM_NOTIFY(struct stat_chge) = 6;
55     } = 1;
```

new/usr/src/uts/common/rpcsvc/sm_inter.x

2

```
56 } = 100024;
    unchanged_portion_omitted_

95 enum sm_res {
101 enum res {
96     stat_succ = 0,          /* status monitor agrees to monitor */
97     stat_fail = 1,         /* status monitor cannot monitor */
98 };

100 struct sm_stat_res {
101     sm_res res_stat;
107     res res_stat;
102     int state;
103 };

105 /*
106  * structure of the status message sent by the status monitor to the
107  * requesting program when a monitored site changes status.
108  */
109 struct sm_status {
115 struct status {
110     string mon_name<SM_MAXSTRLEN>;
111     int state;
112     opaque priv[16];          /* stored private information */
113 };
    unchanged_portion_omitted_
```



```

*****
8612 Fri Aug 16 09:05:10 2013
new/usr/src/uts/common/sys/flock.h
195 Need replacement for nfs/lockd+klm
*****
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, Version 1.0 only
6  * (the "License"). You may not use this file except in compliance
7  * with the License.
8  *
9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 * or http://www.opensolaris.org/os/licensing.
11 * See the License for the specific language governing permissions
12 * and limitations under the License.
13 *
14 * When distributing Covered Code, include this CDDL HEADER in each
15 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 * If applicable, add the following below this CDDL HEADER, with the
17 * fields enclosed by brackets "[]" replaced with your own identifying
18 * information: Portions Copyright [yyyy] [name of copyright owner]
19 *
20 * CDDL HEADER END
21 */
22 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
23 /*      All Rights Reserved */

26 /*
27 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
28 * Use is subject to license terms.
29 */
30 /*
31 * Copyright 2011 Nexenta Systems, Inc. All rights reserved.
32 */

34 #ifndef _SYS_FLOCK_H
35 #define _SYS_FLOCK_H

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

37 #include <sys/types.h>
38 #include <sys/fcntl.h>
39 #include <sys/vnode.h>
40 #include <sys/t_lock.h>      /* for <sys/callb.h> */
41 #include <sys/callb.h>
42 #include <sys/param.h>
43 #include <sys/zone.h>

45 #ifdef __cplusplus
46 extern "C" {
47 #endif

49 /*
50 * Private declarations and instrumentation for local locking.
51 */

53 /*
54 * The flag passed to fs_frlock() may be Ored together with either
55 * 'F_REMOTELOCK' or 'F_PXFSLOCK'. Since this flag is initialized using the
56 * 'f_flag' field in the 'file' structure, and that field is an unsigned short,
57 * we do not use the first 2 bytes.
58 */
59 #define F_REMOTELOCK      (0x01 << 16) /* Set if NLM lock */

```

```

60 #define F_PXFSLOCK      (0x02 << 16) /* Clustering: set if PXFS lock */

62 /*
63 * The command passed to relock() is made by ORing together one or more of
64 * the following values.
65 */

67 #define INOFLCK      0x01      /* Vnode is locked when relock() is called. */
68 #define SETFLCK      0x02      /* Set a file lock. */
69 #define SLPFLCK      0x04      /* Wait if blocked. */
70 #define RCMDLCK      0x08      /* F_REMOTELOCK specified */
71 #define PCMDLCK      0x10      /* Clustering: F_PXFSLOCK specified */
72 #define NBMLCK      0x20      /* non-blocking mandatory locking */

74 /*
75 * Special pid value that can be passed to cleanlocks(). It means that
76 * cleanlocks() should flush all locks for the given sysid, not just the
77 * locks owned by a specific process.
78 */

80 #define IGN_PID      (-1)

82 /* file locking structure (connected to vnode) */

84 #define l_end      l_len

86 /*
87 * The lock manager is allowed to use unsigned offsets and lengths, though
88 * regular Unix processes are still required to use signed offsets and
89 * lengths.
90 */
91 typedef ulong_t u_off_t;

93 #define MAX_U_OFF_T      ((u_off_t)~0)
94 #define MAX_U_OFFSET_T      ((u_offset_t)~0)

96 /*
97 * define MAXEND as the largest positive value the signed offset_t will hold.
98 */
99 #define MAXEND      MAXOFFSET_T

101 /*
102 * Definitions for accessing the l_pad area of struct flock. The
103 * discriminant of the pad_info_t union is the fcntl command used in
104 * conjunction with the flock struct.
105 */

107 typedef union {
108     int      pi_pad[4];      /* (original pad area) */
109     int      pi_has_rmt;      /* F_HASREMOTELOCKS */
110 } pad_info_t;
111 unchanged portion omitted

221 #define FLK_QUERY_ACTIVE      0x1
222 #define FLK_QUERY_SLEEPING      0x2

224 int      relock(struct vnode *, struct flock64 *, int, int, u_offset_t,
225              flk_callback_t *);
226 int      chklock(struct vnode *, int, u_offset_t, ssize_t, int,
227              caller_context_t *);
228 int      convoff(struct vnode *, struct flock64 *, int, offset_t);
229 void      cleanlocks(struct vnode *, pid_t, int);
230 locklist_t *flk_get_sleeping_locks(int sysid, pid_t pid);
231 locklist_t *flk_get_active_locks(int sysid, pid_t pid);
232 locklist_t *flk_active_locks_for_vp(const struct vnode *vp);
233 locklist_t *flk_active_nbmand_locks_for_vp(const struct vnode *vp);

```

```
234 locklist_t *flk_active_nbmand_locks(pid_t pid);
235 void flk_free_locklist(locklist_t *);
236 int flk_convert_lock_data(struct vnode *, struct flock64 *,
237     u_offset_t *, u_offset_t *, offset_t);
238 int flk_check_lock_data(u_offset_t, u_offset_t, offset_t);
239 int flk_has_remote_locks(struct vnode *vp);
240 void flk_set_lockmgr_status(flk_lockmgr_status_t status);
241 int flk_sysid_has_locks(int sysid, int chkck);
242 int flk_has_remote_locks_for_sysid(vnode_t *vp, int);
243 void flk_init_callback(flk_callback_t *,
244     callb_cpr_t (*)(flk_cb_when_t, void *), void *);
245 void flk_add_callback(flk_callback_t *,
246     callb_cpr_t (*)(flk_cb_when_t, void *), void *,
247     flk_callback_t *);
248 callb_cpr_t *flk_invoke_callbacks(flk_callback_t *, flk_cb_when_t);

250 /* Zones hooks */
251 extern zone_key_t flock_zone_key;

253 void *flk_zone_init(zoneid_t);
254 void flk_zone_fini(zoneid_t, void *);

256 /* Clustering hooks */
257 void cl_flk_set_nlm_status(int nlmid, flk_nlm_status_t nlm_state);
258 void cl_flk_remove_locks_by_sysid(int sysid);
259 int cl_flk_has_remote_locks_for_nlmid(struct vnode *vp, int nlmid);
260 void cl_flk_change_nlm_state_to_unknown(int nlmid);
261 void cl_flk_delete_pxflocks(struct vfs *vfsp, int pxfsid);
262 #endif /* _KERNEL */

264 #ifdef __cplusplus
265 }
_____unchanged_portion_omitted_
```



```

*****
16919 Fri Aug 16 09:05:12 2013
new/usr/src/uts/intel/Makefile.intel.shared
195 Need replacement for nfs/lockd+klm
*****
1 # CDDL HEADER START
2 #
3 # The contents of this file are subject to the terms of the
4 # Common Development and Distribution License (the "License").
5 # You may not use this file except in compliance with the License.
6 #
7 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
8 # or http://www.opensolaris.org/os/licensing.
9 # See the License for the specific language governing permissions
10 # and limitations under the License.
11 #
12 # When distributing Covered Code, include this CDDL HEADER in each
13 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
14 # If applicable, add the following below this CDDL HEADER, with the
15 # fields enclosed by brackets "[]" replaced with your own identifying
16 # information: Portions Copyright [yyyy] [name of copyright owner]
17 #
18 # CDDL HEADER END
19 #
21 # Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
22 # Copyright (c) 2012 Nexenta Systems, Inc. All rights reserved.
23 #
24 #
25 # This makefile contains the common definitions for all intel
26 # implementation architecture independent modules.
27 #
28 #
29 #
30 # Machine type (implementation architecture):
31 #
32 PLATFORM = i86pc
33 #
34 #
35 # Everybody needs to know how to build modstubs.o and to locate unix.o.
36 # Note that unix.o must currently be selected from among the possible
37 # "implementation architectures". Note further, that unix.o is only
38 # used as an optional error check for undefines so (theoretically)
39 # any "implementation architectures" could be used. We choose i86pc
40 # because it is the reference port.
41 #
42 UNIX_DIR = $(UTSBASE)/i86pc/unix
43 GENLIB_DIR = $(UTSBASE)/intel/genunix
44 IPDRV_DIR = $(UTSBASE)/intel/ip
45 MODSTUBS_DIR = $(UNIX_DIR)
46 DSF_DIR = $(UTSBASE)/$(PLATFORM)/genassym
47 LINTS_DIR = $(OBJS_DIR)
48 LINT_LIB_DIR = $(UTSBASE)/intel/lint-libs/$(OBJS_DIR)
49 #
50 UNIX_O = $(UNIX_DIR)/$(OBJS_DIR)/unix.o
51 GENLIB = $(GENLIB_DIR)/$(OBJS_DIR)/libgenunix.so
52 MODSTUBS_O = $(MODSTUBS_DIR)/$(OBJS_DIR)/modstubs.o
53 LINT_LIB = $(UTSBASE)/i86pc/lint-libs/$(OBJS_DIR)/llib-lunix.ln
54 GEN_LINT_LIB = $(UTSBASE)/intel/lint-libs/$(OBJS_DIR)/llib-lgenunix.ln
55 #
56 #
57 # Include the makefiles which define build rule templates, the
58 # collection of files per module, and a few specific flags. Note
59 # that order is significant, just as with an include path. The
60 # first build rule template which matches the files name will be
61 # used. By including these in order from most machine dependent

```

```

62 # to most machine independent, we allow a machine dependent file
63 # to be used in preference over a machine independent version
64 # (Such as a machine specific optimization, which preserves the
65 # interfaces.)
66 #
67 include $(UTSTREE)/intel/Makefile.files
68 include $(UTSTREE)/common/Makefile.files
69 #
70 #
71 # ----- TRANSITIONAL SECTION -----
72 #
73 #
74 #
75 # Not everything which *should* be a module is a module yet. The
76 # following is a list of such objects which are currently part of
77 # genunix but which might someday become kmods. This must be
78 # defined before we include Makefile.uts, or else genunix's build
79 # won't be as parallel as we might like.
80 #
81 NOT_YET_KMODS = $(OLDPTY_OBJS) $(PTY_OBJS) $(VCONS_CONF_OBJS) $(MOD_OBJS)
82 #
83 #
84 # ----- END OF TRANSITIONAL SECTION -----
85 #
86 # Include machine independent rules. Note that this does not imply
87 # that the resulting module from rules in Makefile.uts is machine
88 # independent. Only that the build rules are machine independent.
89 #
90 include $(UTSBASE)/Makefile.uts
91 #
92 #
93 # The following must be defined for all implementations:
94 #
95 MODSTUBS = $(UTSBASE)/intel/ia32/ml/modstubs.s
96 #
97 #
98 # Define supported builds
99 #
100 DEF_BUILDS = $(DEF_BUILDS64) $(DEF_BUILDS32)
101 ALL_BUILDS = $(ALL_BUILDS64) $(ALL_BUILDS32)
102 #
103 #
104 # x86 or amd64 inline templates
105 #
106 INLINES_32 = $(UTSBASE)/intel/ia32/ml/ia32.il
107 INLINES_64 = $(UTSBASE)/intel/amd64/ml/amd64.il
108 INLINES += $(INLINES_$(CLASS))
109 #
110 #
111 # kernel-specific optimizations; override default in Makefile.master
112 #
113 #
114 CFLAGS_XARCH_32 = $(i386_CFLAGS)
115 CFLAGS_XARCH_64 = $(amd64_CFLAGS)
116 CFLAGS_XARCH = $(CFLAGS_XARCH_$(CLASS))
117 #
118 COPTFLAG_32 = $(COPTFLAG)
119 COPTFLAG_64 = $(COPTFLAG64)
120 COPTIMIZE = $(COPTFLAG_$(CLASS))
121 #
122 CFLAGS = $(CFLAGS_XARCH)
123 CFLAGS += $(COPTIMIZE)
124 CFLAGS += $(INLINES) -D_ASM_INLINES
125 CFLAGS += $(CCMODE)
126 CFLAGS += $(SPACEFLAG)
127 CFLAGS += $(CCUNBOUND)

```

new/usr/src/uts/intel/Makefile.intel.shared

```

128 CFLAGS          += $(CFLAGS_uts)
129 CFLAGS          += -xstrconst

131 ASFLAGS_XARCH_32 = $(i386_ASFLAGS)
132 ASFLAGS_XARCH_64 = $(amd64_ASFLAGS)
133 ASFLAGS_XARCH    = $(ASFLAGS_XARCH_$(CLASS))

135 ASFLAGS          += $(ASFLAGS_XARCH)

137 #
138 #       Define the base directory for installation.
139 #
140 BASE_INS_DIR     = $(ROOT)

142 #
143 #       Debugging level
144 #
145 #       Special knowledge of which special debugging options affect which
146 #       file is used to optimize the build if these flags are changed.
147 #
148 DEBUG_DEFS_OBJ32 =
149 DEBUG_DEFS_DBG32 = -DDEBUG
150 DEBUG_DEFS_OBJ64 =
151 DEBUG_DEFS_DBG64 = -DDEBUG
152 DEBUG_DEFS       = $(DEBUG_DEFS_$(BUILD_TYPE))

154 DEBUG_COND_OBJ32 =:sh = echo \\043
155 DEBUG_COND_DBG32 =
156 DEBUG_COND_OBJ64 =:sh = echo \\043
157 DEBUG_COND_DBG64 =
158 IF_DEBUG_OBJ     = $(DEBUG_COND_$(BUILD_TYPE))$(OBJS_DIR)/

160 $(IF_DEBUG_OBJ)syscall.o      :=      DEBUG_DEFS      += -DSYSCALLTRACE
161 $(IF_DEBUG_OBJ)clock.o       :=      DEBUG_DEFS      += -DKSLICE=1

163 #
164 #       Collect the preprocessor definitions to be associated with *all*
165 #       files.
166 #
167 ALL_DEFS          = $(DEBUG_DEFS) $(OPTION_DEFS)

169 #
170 #       The kernels modules which are "implementation architecture"
171 #       specific for this machine are enumerated below. Note that most
172 #       of these modules must exist (in one form or another) for each
173 #       architecture.
174 #
175 #       Common Drivers (usually pseudo drivers) (/kernel/drv)
176 #       DRV_KMODS are built both 32-bit and 64-bit
177 #       DRV_KMODS_32 are built only 32-bit
178 #       DRV_KMODS_64 are built only 64-bit
179 #
180 DRV_KMODS        += aac
181 DRV_KMODS        += aggr
182 DRV_KMODS        += ahci
183 DRV_KMODS        += amd64_gart
184 DRV_KMODS        += amr
185 DRV_KMODS        += apgart
186 DRV_KMODS        += srn
187 DRV_KMODS        += agptarget
188 DRV_KMODS        += arn
189 DRV_KMODS        += arp
190 DRV_KMODS        += asy
191 DRV_KMODS        += ata
192 DRV_KMODS        += ath
193 DRV_KMODS        += atu

```

3

new/usr/src/uts/intel/Makefile.intel.shared

```

194 DRV_KMODS        += audio
195 DRV_KMODS        += audio1575
196 DRV_KMODS        += audio810
197 DRV_KMODS        += audiocmi
198 DRV_KMODS        += audiocmihd
199 DRV_KMODS        += audioemul0k
200 DRV_KMODS        += audioens
201 DRV_KMODS        += audiohd
202 DRV_KMODS        += audioixp
203 DRV_KMODS        += audiols
204 DRV_KMODS        += audiopl6x
205 DRV_KMODS        += audiopci
206 DRV_KMODS        += audiosolo
207 DRV_KMODS        += audiots
208 DRV_KMODS        += audiovial823x
209 DRV_KMODS_32    += audiovial97
210 DRV_KMODS        += bl
211 DRV_KMODS        += blkdev
212 DRV_KMODS        += bge
213 DRV_KMODS        += bofi
214 DRV_KMODS        += bpf
215 DRV_KMODS        += bridge
216 DRV_KMODS        += bscbus
217 DRV_KMODS        += bscv
218 DRV_KMODS        += chxge
219 DRV_KMODS        += cxgbe
220 DRV_KMODS        += ntxn
221 DRV_KMODS        += myril0ge
222 DRV_KMODS        += clone
223 DRV_KMODS        += cmdk
224 DRV_KMODS        += cn
225 DRV_KMODS        += conskbd
226 DRV_KMODS        += consms
227 DRV_KMODS        += cpuid
228 DRV_KMODS        += cpunex
229 DRV_KMODS        += crypto
230 DRV_KMODS        += cryptoadm
231 DRV_KMODS        += dca
232 DRV_KMODS        += devinfo
233 DRV_KMODS        += dld
234 DRV_KMODS        += dlpiustub
235 DRV_KMODS_32    += dnet
236 DRV_KMODS        += dump
237 DRV_KMODS        += ecpp
238 DRV_KMODS        += emlxs
239 DRV_KMODS        += fd
240 DRV_KMODS        += fdc
241 DRV_KMODS        += fm
242 DRV_KMODS        += fssnap
243 DRV_KMODS        += hxge
244 DRV_KMODS        += i8042
245 DRV_KMODS        += i915
246 DRV_KMODS        += icmp
247 DRV_KMODS        += icmp6
248 DRV_KMODS        += intel_nb5000
249 DRV_KMODS        += intel_nhm
250 DRV_KMODS        += ip
251 DRV_KMODS        += ip6
252 DRV_KMODS        += ipf
253 DRV_KMODS        += ipnet
254 DRV_KMODS        += ippctl
255 DRV_KMODS        += ipsecah
256 DRV_KMODS        += ipsecesp
257 DRV_KMODS        += ipw
258 DRV_KMODS        += iwh
259 DRV_KMODS        += iwi

```

4

```

260 DRV_KMODS += iwk
261 DRV_KMODS += iwp
262 DRV_KMODS += iwscn
263 DRV_KMODS += kb8042
264 DRV_KMODS += keysock
265 DRV_KMODS += kssl
266 DRV_KMODS += kstat
267 DRV_KMODS += ksyms
268 DRV_KMODS += kmdb
269 DRV_KMODS += llcl
270 DRV_KMODS += lofi
271 DRV_KMODS += log
272 DRV_KMODS += logindmux
273 DRV_KMODS += mega_sas
274 DRV_KMODS += mc-amd
275 DRV_KMODS += mm
276 DRV_KMODS += mouse8042
277 DRV_KMODS += mpt_sas
278 DRV_KMODS += mr_sas
279 DRV_KMODS += mwl
280 DRV_KMODS += nca
281 DRV_KMODS += nsmb
282 DRV_KMODS += nulldriver
283 DRV_KMODS += nv_sata
284 DRV_KMODS += nxge
285 DRV_KMODS += oce
286 DRV_KMODS += openeep
287 DRV_KMODS += pci_pci
288 DRV_KMODS += pcic
289 DRV_KMODS += pcieb
290 DRV_KMODS += physmem
291 DRV_KMODS += pcan
292 DRV_KMODS += pcwl
293 DRV_KMODS += pit_beeper
294 DRV_KMODS += pm
295 DRV_KMODS += poll
296 DRV_KMODS += pool
297 DRV_KMODS += power
298 DRV_KMODS += pseudo
299 DRV_KMODS += ptc
300 DRV_KMODS += ptm
301 DRV_KMODS += pts
302 DRV_KMODS += ptsl
303 DRV_KMODS += qlge
304 DRV_KMODS += radeon
305 DRV_KMODS += ral
306 DRV_KMODS += ramdisk
307 DRV_KMODS += random
308 DRV_KMODS += rds
309 DRV_KMODS += rds3
310 DRV_KMODS += rpcib
311 DRV_KMODS += rsm
312 DRV_KMODS += rts
313 DRV_KMODS += rtw
314 DRV_KMODS += rum
315 DRV_KMODS += rwd
316 DRV_KMODS += rwn
317 DRV_KMODS += sad
318 DRV_KMODS += sd
319 DRV_KMODS += sdhost
320 DRV_KMODS += sgen
321 DRV_KMODS += si3124
322 DRV_KMODS += smbios
323 DRV_KMODS += softmac
324 DRV_KMODS += spdsock
325 DRV_KMODS += smbsrv

```

```

326 DRV_KMODS += smp
327 DRV_KMODS += spps
328 DRV_KMODS += sppptun
329 DRV_KMODS += srpt
330 DRV_KMODS += st
331 DRV_KMODS += sy
332 DRV_KMODS += sysevent
333 DRV_KMODS += sysmsg
334 DRV_KMODS += tcp
335 DRV_KMODS += tcp6
336 DRV_KMODS += tl
337 DRV_KMODS += tnf
338 DRV_KMODS += tpm
339 DRV_KMODS += trill
340 DRV_KMODS += udp
341 DRV_KMODS += udp6
342 DRV_KMODS += ucode
343 DRV_KMODS += ural
344 DRV_KMODS += uath
345 DRV_KMODS += urtw
346 DRV_KMODS += vgatext
347 DRV_KMODS += heci
348 DRV_KMODS += vnic
349 DRV_KMODS += vscan
350 DRV_KMODS += wc
351 DRV_KMODS += winlock
352 DRV_KMODS += wpi
353 DRV_KMODS += xge
354 DRV_KMODS += yge
355 DRV_KMODS += zcons
356 DRV_KMODS += zyd
357 DRV_KMODS += simnet
358 DRV_KMODS += stmf
359 DRV_KMODS += stmf_sbd
360 DRV_KMODS += fct
361 DRV_KMODS += fcoe
362 DRV_KMODS += fcoet
363 DRV_KMODS += fcoei
364 DRV_KMODS += qlt
365 DRV_KMODS += iscsit
366 DRV_KMODS += pppt
367 DRV_KMODS += ncall nsctl sdbc nskern sv
368 DRV_KMODS += ii rdc rdcsrv rdcstub
369 DRV_KMODS += iptun

371 $(CLOSED_BUILD)CLOSED_DRV_KMODS += bmc
372 $(CLOSED_BUILD)CLOSED_DRV_KMODS += glm
373 $(CLOSED_BUILD)CLOSED_DRV_KMODS += intel_nhmex
374 $(CLOSED_BUILD)CLOSED_DRV_KMODS += cpqary3
375 $(CLOSED_BUILD)CLOSED_DRV_KMODS += marvell88sx
376 $(CLOSED_BUILD)CLOSED_DRV_KMODS += bcm_sata
377 $(CLOSED_BUILD)CLOSED_DRV_KMODS += memtest
378 $(CLOSED_BUILD)CLOSED_DRV_KMODS += mpt
379 $(CLOSED_BUILD)CLOSED_DRV_KMODS += atiatom
380 $(CLOSED_BUILD)CLOSED_DRV_KMODS += acpi_toshiba

382 #
383 # Common code drivers
384 #

386 DRV_KMODS += afe
387 DRV_KMODS += atge
388 DRV_KMODS += bfe
389 DRV_KMODS += dmfe
390 DRV_KMODS += e1000g
391 DRV_KMODS += efe

```

```

392 DRV_KMODS      += elxl
393 DRV_KMODS      += hme
394 DRV_KMODS      += mxfe
395 DRV_KMODS      += nge
396 DRV_KMODS      += pcn
397 DRV_KMODS      += rge
398 DRV_KMODS      += rtls
399 DRV_KMODS      += sfe
400 DRV_KMODS      += amd8111s
401 DRV_KMODS      += igb
402 DRV_KMODS      += ipmi
403 DRV_KMODS      += iprb
404 DRV_KMODS      += ixgbe
405 DRV_KMODS      += vr
406 $(CLOSED_BUILD)CLOSED_DRV_KMODS += ixgb

408 #
409 # Virtio drivers
410 #

412 # Virtio core
413 DRV_KMODS      += virtio

415 # Virtio block driver
416 DRV_KMODS      += vioblk

418 #
419 #      DTrace and DTrace Providers
420 #
421 DRV_KMODS      += dtrace
422 DRV_KMODS      += fbt
423 DRV_KMODS      += lockstat
424 DRV_KMODS      += profile
425 DRV_KMODS      += sdt
426 DRV_KMODS      += systrace
427 DRV_KMODS      += fasttrap
428 DRV_KMODS      += dcpc

430 #
431 #      I/O framework test drivers
432 #
433 DRV_KMODS      += pshot
434 DRV_KMODS      += gen_drv
435 DRV_KMODS      += tvhci tphci tclient
436 DRV_KMODS      += emul64

438 #
439 #      Machine Specific Driver Modules (/kernel/drv):
440 #
441 DRV_KMODS      += options
442 DRV_KMODS      += scsi_vhci
443 DRV_KMODS      += pmcs
444 DRV_KMODS      += pmcs8001fw
445 DRV_KMODS      += arcmsr
446 DRV_KMODS      += fcp
447 DRV_KMODS      += fcip
448 DRV_KMODS      += fcsn
449 DRV_KMODS      += fp
450 DRV_KMODS      += qlc
451 DRV_KMODS      += iscsi

453 #
454 #      PCMCIA specific module(s)
455 #
456 DRV_KMODS      += pcs
457 DRV_KMODS      += pcata

```

```

458 MISC_KMODS     += cardbus
459 $(CLOSED_BUILD)CLOSED_DRV_KMODS += pcser

461 #
462 #      SCSI Enclosure Services driver
463 #
464 DRV_KMODS      += ses

466 #
467 #      USB specific modules
468 #
469 DRV_KMODS      += hid
470 DRV_KMODS      += hwarc hwahc
471 DRV_KMODS      += hubd
472 DRV_KMODS      += uhci
473 DRV_KMODS      += ehci
474 DRV_KMODS      += ohci
475 DRV_KMODS      += usb_mid
476 DRV_KMODS      += usb_ia
477 DRV_KMODS      += scsa2usb
478 DRV_KMODS      += usbprn
479 DRV_KMODS      += ugen
480 DRV_KMODS      += usbser
481 DRV_KMODS      += usbsacm
482 DRV_KMODS      += usbsksp
483 DRV_KMODS      += usbsprl
484 DRV_KMODS      += usb_ac
485 DRV_KMODS      += usb_as
486 DRV_KMODS      += usbskel
487 DRV_KMODS      += usbvc
488 DRV_KMODS      += usbftdi
489 DRV_KMODS      += wusb_df
490 DRV_KMODS      += wusb_ca
491 DRV_KMODS      += usbecm

493 $(CLOSED_BUILD)CLOSED_DRV_KMODS += usbser_edge

495 #
496 #      1394 modules
497 #
498 MISC_KMODS     += s1394 sbp2
499 DRV_KMODS      += hcil394 scsa1394
500 DRV_KMODS      += av1394
501 DRV_KMODS      += dcam1394

503 #
504 #      InfiniBand pseudo drivers
505 #
506 DRV_KMODS      += ib ibp eibnx eoib rdsib sdp iser daplt hermon tavor sol_ucma
507 DRV_KMODS      += sol_umad

509 #
510 #      LVM modules
511 #
512 DRV_KMODS      += md
513 MISC_KMODS     += md_stripe md_hotspares md_mirror md_raid md_trans md_notify
514 MISC_KMODS     += md_sp

516 #
517 #      Brand modules
518 #
519 BRAND_KMODS    += snl_brand s10_brand

521 #
522 #      Exec Class Modules (/kernel/exec):
523 #

```

```

524 EXEC_KMODS      += elfexec intpexec shbinexec javaexec

526 #
527 #       Scheduling Class Modules (/kernel/sched):
528 #
529 SCHED_KMODS      += IA RT TS RT_DPTBL TS_DPTBL FSS FX FX_DPTBL SDC

531 #
532 #       File System Modules (/kernel/fs):
533 #
534 FS_KMODS         += autofs cachefs ctfs dcfs dev devfs fdfs fifofs hsfs lofs
535 FS_KMODS         += mntfs namefs nfs objfs zfs zut
536 FS_KMODS         += pcfs procfs sockfs specfs tmpfs udfs ufs sharefs
537 FS_KMODS         += smbfs

539 #
540 #       Streams Modules (/kernel/strmod):
541 #
542 STRMOD_KMODS     += bufmod connld dedump ldterm pckt pfmod pipemod
543 STRMOD_KMODS     += ptem redirmod rpcmod rlmod telmod timod
544 STRMOD_KMODS     += sppsasyn sppscomp
545 STRMOD_KMODS     += tirdwr ttcompat
546 STRMOD_KMODS     += usbkbm
547 STRMOD_KMODS     += usbms
548 STRMOD_KMODS     += usbwcm
549 STRMOD_KMODS     += usb_ah
550 STRMOD_KMODS     += drcompat
551 STRMOD_KMODS     += cryptmod
552 STRMOD_KMODS     += vuid2ps2
553 STRMOD_KMODS     += vuid3ps2
554 STRMOD_KMODS     += vuidm3p
555 STRMOD_KMODS     += vuidm4p
556 STRMOD_KMODS     += vuidm5p

558 #
559 #       'System' Modules (/kernel/sys):
560 #
561 SYS_KMODS        += c2audit
562 SYS_KMODS        += doorfs
563 SYS_KMODS        += exacctsys
564 SYS_KMODS        += inst_sync
565 SYS_KMODS        += kaio
566 SYS_KMODS        += msgsys
567 SYS_KMODS        += pipe
568 SYS_KMODS        += portfs
569 SYS_KMODS        += pset
570 SYS_KMODS        += semsys
571 SYS_KMODS        += shmsys
572 SYS_KMODS        += sysacct
573 SYS_KMODS        += acctctl

575 #
576 #       'Misc' Modules (/kernel/misc)
577 #       MISC_KMODS are built both 32-bit and 64-bit
578 #       MISC_KMODS_32 are built only 32-bit
579 #       MISC_KMODS_64 are built only 64-bit
580 #
581 MISC_KMODS       += ac97
582 MISC_KMODS       += acpica
583 MISC_KMODS       += agpmaster
584 MISC_KMODS       += bignum
585 MISC_KMODS       += bootdev
586 MISC_KMODS       += busra
587 MISC_KMODS       += cmlb
588 MISC_KMODS       += consconfig
589 MISC_KMODS       += ctf

```

```

590 MISC_KMODS      += dadk
591 MISC_KMODS      += dcopy
592 MISC_KMODS      += dls
593 MISC_KMODS      += drm
594 MISC_KMODS      += fssnap_if
595 MISC_KMODS      += gda
596 MISC_KMODS      += gld
597 MISC_KMODS      += hidparser
598 MISC_KMODS      += hook
599 MISC_KMODS      += hpcsvc
600 MISC_KMODS      += ibcm
601 MISC_KMODS      += ibdm
602 MISC_KMODS      += ibdma
603 MISC_KMODS      += ibmf
604 MISC_KMODS      += ibtl
605 MISC_KMODS      += idm
606 MISC_KMODS      += idmap
607 MISC_KMODS      += iomulib
608 MISC_KMODS      += ipc
609 MISC_KMODS      += kbtrans
610 MISC_KMODS      += kcf
611 MISC_KMODS      += kgssapi
612 MISC_KMODS      += kmecch_dummy
613 MISC_KMODS      += kmecch_krb5
614 MISC_KMODS      += ksocket
615 MISC_KMODS      += mac
616 MISC_KMODS      += mii
617 MISC_KMODS      += mwlfw
618 MISC_KMODS      += net80211
619 MISC_KMODS      += nfs_dlboot
620 MISC_KMODS      += nfssrv
621 MISC_KMODS      += neti
622 MISC_KMODS      += pci_autoconfig
623 MISC_KMODS      += pcicfg
624 MISC_KMODS      += pcihp
625 MISC_KMODS      += pcmcia
626 MISC_KMODS      += rpcsec
627 MISC_KMODS      += rpcsec_gss
628 MISC_KMODS      += rsmops
629 MISC_KMODS      += sata
630 MISC_KMODS      += scsi
631 MISC_KMODS      += sda
632 MISC_KMODS      += sol_ofs
633 MISC_KMODS      += spuni
634 MISC_KMODS      += strategy
635 MISC_KMODS      += strplumb
636 MISC_KMODS      += tem
637 MISC_KMODS      += tlimod
638 MISC_KMODS      += usba usba10 usbs49_fw
639 MISC_KMODS      += scsi_vhci_f_sym_hds
640 MISC_KMODS      += scsi_vhci_f_sym
641 MISC_KMODS      += scsi_vhci_f_tpgs
642 MISC_KMODS      += scsi_vhci_f_asym_sun
643 MISC_KMODS      += scsi_vhci_f_tape
644 MISC_KMODS      += scsi_vhci_f_tpgs_tape
645 MISC_KMODS      += fctl
646 MISC_KMODS      += emlxs_fw
647 MISC_KMODS      += qlc_fw_2200
648 MISC_KMODS      += qlc_fw_2300
649 MISC_KMODS      += qlc_fw_2400
650 MISC_KMODS      += qlc_fw_2500
651 MISC_KMODS      += qlc_fw_6322
652 MISC_KMODS      += qlc_fw_8100
653 MISC_KMODS      += hwa1480_fw
654 MISC_KMODS      += uathfw
655 MISC_KMODS      += uwba

```



```

657 MISC_KMODS      += klmmod klmops

657 $(CLOSED_BUILD)CLOSED_MISC_KMODS      += klmmod klmops
659 $(CLOSED_BUILD)CLOSED_MISC_KMODS      += scsi_vhci_f_asym_lsi
660 $(CLOSED_BUILD)CLOSED_MISC_KMODS      += scsi_vhci_f_asym_emc
661 $(CLOSED_BUILD)CLOSED_MISC_KMODS      += scsi_vhci_f_sym_emc

663 #
664 #       Software Cryptographic Providers (/kernel/crypto):
665 #
666 CRYPTO_KMODS    += aes
667 CRYPTO_KMODS    += arcfour
668 CRYPTO_KMODS    += blowfish
669 CRYPTO_KMODS    += des
670 CRYPTO_KMODS    += ecc
671 CRYPTO_KMODS    += md4
672 CRYPTO_KMODS    += md5
673 CRYPTO_KMODS    += rsa
674 CRYPTO_KMODS    += sha1
675 CRYPTO_KMODS    += sha2
676 CRYPTO_KMODS    += swrand

678 #
679 #       IP Policy Modules (/kernel/ipp)
680 #
681 IPP_KMODS        += dlcosmk
682 IPP_KMODS        += flowacct
683 IPP_KMODS        += ipgpc
684 IPP_KMODS        += dscompk
685 IPP_KMODS        += tokenmt
686 IPP_KMODS        += tswtclmt

688 #
689 #       generic-unix module (/kernel/genunix):
690 #
691 GENUNIX_KMODS    += genunix

693 #
694 #       SVVS Testing Modules (/kernel/strmod):
695 #
696 #       These are streams and driver modules which are not to be
697 #       delivered with a released system. However, during development
698 #       it is convenient to build and install the SVVS kernel modules.
699 #
700 SVVS_KMODS       += lmodb lmode lmodr lmodt svvslo tidg tivc tmutex

702 $(CLOSED_BUILD)SVVS      += svvs

704 #
705 #       Modules eXcluded from the product:
706 #
707 $(CLOSED_BUILD)CLOSED_XMODS = \
708     adpu320      \
709     bnx           \
710     bnxe         \
711     lsimega      \
712     sdpib

715 #
716 #       'Dacf' Modules (/kernel/dacf):
717 #

719 #
720 #       Performance Counter BackEnd modules (/usr/kernel/pcbe)

```

```

721 #
722 PCBE_KMODS       += p123_pcbe p4_pcbe opteron_pcbe core_pcbe

724 #
725 #       MAC-Type Plugin Modules (/kernel/mac)
726 #
727 MAC_KMODS        += mac_6to4
728 MAC_KMODS        += mac_ether
729 MAC_KMODS        += mac_ipv4
730 MAC_KMODS        += mac_ipv6
731 MAC_KMODS        += mac_wifi
732 MAC_KMODS        += mac_ib

734 #
735 #       socketmod (kernel/socketmod)
736 #
737 SOCKET_KMODS     += sockpfp
738 SOCKET_KMODS     += socksctp
739 SOCKET_KMODS     += socksdp
740 SOCKET_KMODS     += sockrds
741 SOCKET_KMODS     += ksslf

743 #
744 #       kiconv modules (/kernel/kiconv):
745 #
746 KICONV_KMODS     += kiconv_emea kiconv_ja kiconv_ko kiconv_sc kiconv_tc

748 #
749 #       'Dacf' Modules (/kernel/dacf):
750 #
751 DACF_KMODS       += net_dacf

```

```

*****
2101 Fri Aug 16 09:05:14 2013
new/usr/src/uts/intel/klmmod/Makefile
195 Need replacement for nfs/lockd+klm
*****
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #
21 #
22 # Copyright 2006 Sun Microsystems, Inc. All rights reserved.
23 # Use is subject to license terms.
24 #
25 # Copyright (c) 2012 by Delphix. All rights reserved.
26 #
27 # This makefile drives the production of the network lock manager server
28 # specific kernel module.
29 #
30 # intel implementation architecture dependent
31 #
32 #
33 #
34 # Path to the base of the uts directory tree (usually /usr/src/uts).
35 #
36 UTSBASE = ../../

38 #
39 # Define the module and object file sets.
40 #
41 MODULE = klmmod
42 OBJECTS = $(KLMMOD_OBJS:%=$(OBJS_DIR)/%)
43 LINTS = $(KLMMOD_OBJS:%.o=$(LINTS_DIR)/%.ln)
44 ROOTMODULE = $(ROOT_MISC_DIR)/$(MODULE)

46 #
47 # Include common rules.
48 #
49 include $(UTSBASE)/intel/Makefile.intel

51 #
52 # Define targets
53 #
54 ALL_TARGET = $(BINARY)
55 LINT_TARGET = $(MODULE).lint
56 INSTALL_TARGET = $(BINARY) $(ROOTMODULE)

58 #
59 # Overrides.
60 #
61 LDFLAGS += -dy -Nstrmod/rpcmod -Nfs/nfs

```

```

62 LDFLAGS += -M $(UTSBASE)/common/klm/mapfile-mod
63 CTFMRGFLAGS += -f

65 #
66 # Code generated by rpcgen triggers the -Wswitch warning.
67 #
68 CERRWARN += -_gcc=-Wno-switch

70 #
71 # Default build targets.
72 #
73 .KEEP_STATE:

75 def: $(DEF_DEPS)

77 all: $(ALL_DEPS)

79 clean: $(CLEAN_DEPS)

81 clobber: $(CLOBBER_DEPS)

83 lint: $(LINT_DEPS)

85 modlintlib: $(MODLINTLIB_DEPS)

87 clean.lint: $(CLEAN_LINT_DEPS)

89 install: $(INSTALL_DEPS)

91 #
92 # Include common targets.
93 #
94 include $(UTSBASE)/intel/Makefile.targ

```

```

*****
2011 Fri Aug 16 09:05:14 2013
new/usr/src/uts/intel/klmops/Makefile
195 Need replacement for nfs/lockd+klm
*****
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #
21 #
22 # Copyright 2006 Sun Microsystems, Inc. All rights reserved.
23 # Use is subject to license terms.
24 #
25 # Copyright (c) 2012 by Delphix. All rights reserved.
26 #
27 # This makefile drives the production of the network lock manager client
28 # side module.
29 #
30 # intel implementation architecture dependent
31 #
32 #
33 #
34 # Path to the base of the uts directory tree (usually /usr/src/uts).
35 #
36 UTSBASE = ../../

38 #
39 # Define the module and object file sets.
40 #
41 MODULE = klmops
42 OBJECTS = $(KLMOPS_OBJS:%=$(OBJS_DIR)/%)
43 LINTS = $(KLMOPS_OBJS:%.o=$(LINTS_DIR)/%.ln)
44 ROOTMODULE = $(ROOT_MISC_DIR)/$(MODULE)

46 #
47 # Include common rules.
48 #
49 include $(UTSBASE)/intel/Makefile.intel

51 #
52 # Define targets
53 #
54 ALL_TARGET = $(BINARY)
55 LINT_TARGET = $(MODULE).lint
56 INSTALL_TARGET = $(BINARY) $(ROOTMODULE)

58 #
59 # Overrides.
60 #
61 LDFLAGS += -dy -Nstrmod/rpcmod -Nfs/nfs -Nmisc/klmmod

```

```

62 LDFLAGS += -M $(UTSBASE)/common/klm/mapfile-ops
63 CTFMRGFLAGS += -f

65 #
66 # Default build targets.
67 #
68 .KEEP_STATE:

70 def: $(DEF_DEPS)

72 all: $(ALL_DEPS)

74 clean: $(CLEAN_DEPS)

76 clobber: $(CLOBBER_DEPS)

78 lint: $(LINT_DEPS)

80 modlintlib: $(MODLINTLIB_DEPS)

82 clean.lint: $(CLEAN_LINT_DEPS)

84 install: $(INSTALL_DEPS)

86 #
87 # Include common targets.
88 #
89 include $(UTSBASE)/intel/Makefile.targ

```

```

*****
13709 Fri Aug 16 09:05:15 2013
new/usr/src/uts/sparc/Makefile.sparc.shared
195 Need replacement for nfs/lockd+klm
*****
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 # Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.

25 #
26 # This makefile contains the common definitions for all sparc
27 # implementation architecture independent modules.
28 #

30 #
31 # Define supported builds
32 #
33 DEF_BUILDS = $(DEF_BUILDS64)
34 ALL_BUILDS = $(ALL_BUILDS64)

36 #
37 # Everybody needs to know how to build modstubs.o and to locate unix.o.
38 # Note that unix.o must currently be selected from among the possible
39 # "implementation architectures". Note further, that unix.o is only
40 # used as an optional error check for undefines so (theoretically)
41 # any "implementation architectures" could be used. We choose sun4u
42 # because it is the reference port.
43 #
44 UNIX_DIR = $(UTSBASE)/sun4u/unix
45 GENLIB_DIR = $(UTSBASE)/sun4u/genunix
46 IPDRV_DIR = $(UTSBASE)/sparc/ip
47 MODSTUBS_DIR = $(UNIX_DIR)
48 DSF_DIR = $(UNIX_DIR)
49 LINTS_DIR = $(OBJS_DIR)
50 LINT_LIB_DIR = $(UTSBASE)/sparc/lint-libs/$(OBJS_DIR)

52 UNIX_O = $(UNIX_DIR)/$(OBJS_DIR)/unix.o
53 MODSTUBS_O = $(MODSTUBS_DIR)/$(OBJS_DIR)/modstubs.o
54 GENLIB = $(UTSBASE)/sun4u/lint-libs/$(OBJS_DIR)/libgenunix.so

56 LINT_LIB_32 = $(UTSBASE)/sun4u/lint-libs/$(OBJS_DIR)/llib-lunix.ln
57 GEN_LINT_LIB_32 = $(UTSBASE)/sun4u/lint-libs/$(OBJS_DIR)/llib-lgenunix.ln

59 LINT_LIB_64 = $(UTSBASE)/sun4u/lint-libs/$(OBJS_DIR)/llib-lunix.ln
60 GEN_LINT_LIB_64 = $(UTSBASE)/sun4u/lint-libs/$(OBJS_DIR)/llib-lgenunix.ln

```

```

62 LINT_LIB = $(LINT_LIB_$(CLASS))
63 GEN_LINT_LIB = $(GEN_LINT_LIB_$(CLASS))

65 LINT32_DIRS = $(LINT32_BUILDS:%=$(UTSBASE)/sparc/lint-libs/%)
66 LINT32_FILES = $(LINT32_DIRS:%=/llib-l$(MODULE).ln)

68 LINT64_DIRS = $(LINT64_BUILDS:%=$(UTSBASE)/sparc/lint-libs/%)
69 LINT64_FILES = $(LINT64_DIRS:%=/llib-l$(MODULE).ln)

71 #
72 # Include the makefiles which define build rule templates, the
73 # collection of files per module, and a few specific flags. Note
74 # that order is significant, just as with an include path. The
75 # first build rule template which matches the files name will be
76 # used. By including these in order from most machine dependent
77 # to most machine independent, we allow a machine dependent file
78 # to be used in preference over a machine independent version
79 # (Such as a machine specific optimization, which preserves the
80 # interfaces.)
81 #
82 include $(UTSBASE)/sparc/Makefile.files
83 include $(UTSBASE)/sparc/v9/Makefile.files
84 include $(UTSTREE)/sun/Makefile.files
85 include $(UTSTREE)/common/Makefile.files

87 #
88 # ----- TRANSITIONAL SECTION -----
89 #

91 #
92 # Not everything which *should* be a module is a module yet. The
93 # following is a list of such objects which are currently part of
94 # genunix but which might someday become kmods. This must be
95 # defined before we include Makefile.uts, or else genunix's build
96 # won't be as parallel as we might like.
97 #
98 NOT_YET_KMODS = $(OLDPTY_OBJS) $(PTY_OBJS) $(VCONS_CONF_OBJS) $(MOD_OBJS)

100 #
101 # ----- END OF TRANSITIONAL SECTION -----
102 #
103 # Include machine independent rules. Note that this does not imply
104 # that the resulting module from rules in Makefile.uts is machine
105 # independent. Only that the build rules are machine independent.
106 #
107 include $(UTSBASE)/Makefile.uts

109 #
110 # machine specific optimization, override default in Makefile.master
111 #
112 XARCH_32 = -xarch=v8
113 XARCH_64 = -m64
114 XARCH = $(XARCH_$(CLASS))

116 COPTIMIZE_32 = -xO3
117 COPTIMIZE_64 = -xO3
118 COPTIMIZE = $(COPTIMIZE_$(CLASS))

120 CCMODE = -Xa

122 CFLAGS_32 = -xcg92
123 CFLAGS_64 = -xchip=ultra $(CCABS32) $(CCREGSYM)
124 CFLAGS = $(CFLAGS_$(CLASS))

126 CFLAGS += $(XARCH)
127 CFLAGS += $(COPTIMIZE)

```

```

128 CFLAGS      += $(EXTRA_CFLAGS)
129 CFLAGS      += $(XAOPT)
130 CFLAGS      += $(INLINES) -D_ASM_INLINES
131 CFLAGS      += $(CCMODE)
132 CFLAGS      += $(SPACEFLAG)
133 CFLAGS      += $(CERRWARN)
134 CFLAGS      += $(CTF_FLAGS_$(CLASS))
135 CFLAGS      += $(C99MODE)
136 CFLAGS      += $(CCUNBOUND)
137 CFLAGS      += $(CCSTATICSYM)
138 CFLAGS      += $(CC32BITCALLERS)
139 CFLAGS      += $(CCNOAUTOINLINE)
140 CFLAGS      += $(IROPTFLAG)
141 CFLAGS      += $(CGLOBALSTATIC)
142 CFLAGS      += -xregs=no%float
143 CFLAGS      += -xstrconst
144 CFLAGS      += $(CSOURCEDEBUGFLAGS)
145 CFLAGS      += $(USERFLAGS)

147 ASFLAGS     += $(XARCH)

149 LINT_DEFS_32 =
150 LINT_DEFS_64 = -m64
151 LINT_DEFS     += $(LINT_DEFS_$(CLASS))

153 #
154 #   The following must be defined for all implementations:
155 #
156 #   MODSTUBS:      Module stubs source file.
157 #
158 MODSTUBS      = $(UTSBASE)/sparc/ml/modstubs.s

160 #
161 #   Define the actual specific platforms - obviously none.
162 #
163 MACHINE_DEFS   =

165 #
166 #   Debugging level
167 #
168 #   Special knowledge of which special debugging options effect which
169 #   file is used to optimize the build if these flags are changed.
170 #
171 #   XXX: The above could possibly be done for more flags and files, but
172 #   is left as an experiment to the interested reader. Be forewarned,
173 #   that excessive use could lead to maintenance difficulties.
174 #
175 DEBUG_DEFS_OBJ32 =
176 DEBUG_DEFS_DBG32 = -DDEBUG
177 DEBUG_DEFS_OBJ64 =
178 DEBUG_DEFS_DBG64 = -DDEBUG
179 DEBUG_DEFS        = $(DEBUG_DEFS_$(BUILD_TYPE))

181 DEBUG_COND_OBJ32 :sh = echo \\043
182 DEBUG_COND_DBG32 =
183 DEBUG_COND_OBJ64 :sh = echo \\043
184 DEBUG_COND_DBG64 =
185 IF_DEBUG_OBJ     = $(DEBUG_COND_$(BUILD_TYPE))$(OBJDIR)/

187 $(IF_DEBUG_OBJ)syscall.o      :=      DEBUG_DEFS      += -DSYSCALLTRACE
188 $(IF_DEBUG_OBJ)clock.o       :=      DEBUG_DEFS      += -DKSLICE=1

190 # Comment these out if you don't want dispatcher lock statistics.

192 # $(IF_DEBUG_OBJ)disp_lock.o  := DEBUG_DEFS      += -DDISP_LOCK_STATS

```

```

194 #
195 #   Collect the preprocessor definitions to be associated with *all*
196 #   files.
197 #
198 ALL_DEFS      = $(MACHINE_DEFS) $(DEBUG_DEFS) $(OPTION_DEFS)
199 #
200 #
201 #   The kernels modules which are "implementation architecture"
202 #   specific for this machine are enumerated below. Note that most
203 #   of these modules must exist (in one form or another) for each
204 #   architecture.
205 #
206 #   Common Drivers (usually pseudo drivers) (/kernel/drv):
207 #
208 DRV_KMODS     += aggr arp audio bl blkdev bofi clone cn conskbd consms cpuid
209 DRV_KMODS     += crypto cryptoadm devinfo dump
210 DRV_KMODS     += dtrace fasttrap fbt lockstat profile sdt systrace dcpd
211 DRV_KMODS     += fssnap icmp icmp6 ip ip6 ipnet ipsecah
212 DRV_KMODS     += ipsecesp iptun iwscn keysock kmdb kstat ksyms llcl
213 DRV_KMODS     += lofi
214 DRV_KMODS     += log logindmux kssl mm nca physmem pm poll pool
215 DRV_KMODS     += pseudo ptc ptm pts ptsl ramdisk random rsm rts sad
216 DRV_KMODS     += simnet softmac sPPP spptun sy sysevent sysmsg
217 DRV_KMODS     += spdsock
218 DRV_KMODS     += tcp tcp6 tl tnf ttymux udp udp6 wc winlock zcons
219 DRV_KMODS     += ippctl
220 DRV_KMODS     += dld
221 DRV_KMODS     += ipf
222 DRV_KMODS     += rpcib
223 DRV_KMODS     += dlpistub
224 DRV_KMODS     += vnic
225 DRV_KMODS     += xge
226 DRV_KMODS     += rds
227 DRV_KMODS     += rdsv3
228 DRV_KMODS     += chxge
229 DRV_KMODS     += smbsrv
230 DRV_KMODS     += vscan
231 DRV_KMODS     += nsmb
232 DRV_KMODS     += fm
233 DRV_KMODS     += nulldriver
234 DRV_KMODS     += bridge trill
235 DRV_KMODS     += bpf
236 DRV_KMODS     += dca

238 $(CLOSED_BUILD)CLOSED_DRV_KMODS += glm
239 $(CLOSED_BUILD)CLOSED_DRV_KMODS += isp
240 $(CLOSED_BUILD)CLOSED_DRV_KMODS += mpt
241 $(CLOSED_BUILD)CLOSED_DRV_KMODS += qus
242 $(CLOSED_BUILD)CLOSED_DRV_KMODS += se

244 #
245 #   Hardware Drivers in common space
246 #
248 DRV_KMODS     += afe
249 DRV_KMODS     += audio1575
250 DRV_KMODS     += audioens
251 DRV_KMODS     += audiols
252 DRV_KMODS     += audiopl6x
253 DRV_KMODS     += audiopci
254 DRV_KMODS     += audiodts
255 DRV_KMODS     += e1000g
256 DRV_KMODS     += efe
257 DRV_KMODS     += hxge
258 DRV_KMODS     += mxfe
259 DRV_KMODS     += pcan

```

```

260 DRV_KMODS      += pcwl
261 DRV_KMODS      += rge
262 DRV_KMODS      += rtls
263 DRV_KMODS      += sfe
264 DRV_KMODS      += aac
265 DRV_KMODS      += igb
266 DRV_KMODS      += ixgbe
267 DRV_KMODS      += vr
268 DRV_KMODS      += mr_sas
269 $(CLOSED_BUILD)CLOSED_DRV_KMODS += ixgb
270 DRV_KMODS      += yge

272 #
273 #      Machine Specific Driver Modules (/kernel/drv):
274 #
275 DRV_KMODS      += audiocs
276 DRV_KMODS      += bge dmfe eri fas hme qfe
277 DRV_KMODS      += openeepr options sd ses st
278 DRV_KMODS      += ssd
279 DRV_KMODS      += ecpp
280 DRV_KMODS      += hid hubd ehci ohci uhci usb_mid usb_ia scsa2usb usbprn ugen
281 DRV_KMODS      += usbser usbsacm usbsksp usbsprl
282 DRV_KMODS      += usb_as usb_ac
283 DRV_KMODS      += usbskel
284 DRV_KMODS      += usbvc
285 DRV_KMODS      += usbftdi
286 DRV_KMODS      += wusb_df hwahc hwarc wusb_ca
287 DRV_KMODS      += usbecm
288 DRV_KMODS      += hcil1394 avl1394 scsa1394 dcaml1394
289 DRV_KMODS      += sbp2
290 DRV_KMODS      += ib ibp eibnx eoib rdsib sdp iser daplt hermon tavor sol_ucma
291 DRV_KMODS      += sol_umad
292 DRV_KMODS      += pci_pci pcieb pcieb_bcm
293 DRV_KMODS      += i8042 kb8042 mouse8042
294 DRV_KMODS      += fcode
295 DRV_KMODS      += mpt_sas
296 DRV_KMODS      += socal
297 DRV_KMODS      += sgen
298 DRV_KMODS      += myril0ge
299 DRV_KMODS      += smp
300 DRV_KMODS      += dad
301 DRV_KMODS      += scsi_vhci
302 DRV_KMODS      += fcp
303 DRV_KMODS      += fcip
304 DRV_KMODS      += fcsn
305 DRV_KMODS      += fp
306 DRV_KMODS      += qlc
307 DRV_KMODS      += qlge
308 DRV_KMODS      += stmf
309 DRV_KMODS      += stmf_sbd
310 DRV_KMODS      += fct
311 DRV_KMODS      += fcoe
312 DRV_KMODS      += fcoet
313 DRV_KMODS      += fcoei
314 DRV_KMODS      += qlt
315 DRV_KMODS      += iscsit
316 DRV_KMODS      += pppt
317 DRV_KMODS      += ncall nsctl sdbc nskern sv
318 DRV_KMODS      += ii rdc rdcsrv rdcstub
319 DRV_KMODS      += iscsi
320 DRV_KMODS      += emlxs
321 DRV_KMODS      += oce
322 DRV_KMODS      += srpt
323 DRV_KMODS      += pmcs
324 DRV_KMODS      += pmcs8001fw

```

```

326 $(CLOSED_BUILD)CLOSED_DRV_KMODS += ifp
327 $(CLOSED_BUILD)CLOSED_DRV_KMODS += uata
328 $(CLOSED_BUILD)CLOSED_DRV_KMODS += usbser_edge

330 #
331 #      I/O framework test drivers
332 #
333 DRV_KMODS      += pshot
334 DRV_KMODS      += gen_drv
335 DRV_KMODS      += tvhci tphci tclient
336 DRV_KMODS      += emul64

338 #
339 #      PCMCIA specific module(s)
340 #
341 DRV_KMODS      += pcs
342 MISC_KMODS     += busra cardbus dada pcmcia
343 DRV_KMODS      += pcata
344 DRV_KMODS      += pcic

346 $(CLOSED_BUILD)CLOSED_DRV_KMODS += pcser

348 # Add lvm
349 #
350 DRV_KMODS      += md
351 MISC_KMODS     += md_mirror md_stripe md_hotspares md_raid md_trans md_notify
352 MISC_KMODS     += md_sp

354 #
355 #      Exec Class Modules (/kernel/exec):
356 #
357 EXEC_KMODS     += aoutexec elfexec intpexec shbinexec javaexec

359 #
360 #      Scheduling Class Modules (/kernel/sched):
361 #
362 SCHED_KMODS    += RT TS RT_DPTBL TS_DPTBL IA FSS FX FX_DPTBL SDC

364 #
365 #      File System Modules (/kernel/fs):
366 #
367 FS_KMODS       += dev devfs fdfs fifofs hsfhs lofs namefs nfs pcfs tmpfs zfs
368 FS_KMODS       += zut specfs udfs ufs autofs cachefs procfs sockfs mntfs
369 FS_KMODS       += ctfs objfs sharefs dcfs smbfs

371 #
372 #      Streams Modules (/kernel/strmod):
373 #
374 STRMOD_KMODS  += bufmod connld dedump ldterm ms pkt pfmod
375 STRMOD_KMODS  += pipemod ptem redirmod rpcmod rlmmod telmod timod
376 STRMOD_KMODS  += spppasyn spppcomp
377 STRMOD_KMODS  += tirdwr ttcompat
378 STRMOD_KMODS  += usbkbm usbms usbwcm usb_ah
379 STRMOD_KMODS  += drcompat
380 STRMOD_KMODS  += cryptmod
381 STRMOD_KMODS  += vuid3ps2

383 #
384 #      'System' Modules (/kernel/sys):
385 #
386 SYS_KMODS     += c2audit
387 SYS_KMODS     += exacctsys
388 SYS_KMODS     += inst_sync kaio msgsys semsys shmsys sysacct pipe
389 SYS_KMODS     += doorfs pset acctctl portfs

391 #

```

```

392 #      'User' Modules (/kernel/misc):
393 #
394 MISC_KMODS      += ac97
395 MISC_KMODS      += bignum
396 MISC_KMODS      += consconfig gld ipc nfs_dlboot nfssrv scsi
397 MISC_KMODS      += strplumb swapgeneric tlimod
398 MISC_KMODS      += rpcsec rpcsec_gss kgssapi kmech_dummy
399 MISC_KMODS      += kmech_krb5
400 MISC_KMODS      += fssnap_if
401 MISC_KMODS      += hidparser kbtrans usba usbal0 usbs49_fw
402 MISC_KMODS      += sl394
403 MISC_KMODS      += hpcsvc pcihp
404 MISC_KMODS      += rsmops
405 MISC_KMODS      += kcf
406 MISC_KMODS      += ksocket
407 MISC_KMODS      += ibcm
408 MISC_KMODS      += ibdm
409 MISC_KMODS      += ibdma
410 MISC_KMODS      += ibmf
411 MISC_KMODS      += ibt1
412 MISC_KMODS      += sol_ofs
413 MISC_KMODS      += idm
414 MISC_KMODS      += idmap
415 MISC_KMODS      += hook
416 MISC_KMODS      += neti
417 MISC_KMODS      += ctf
418 MISC_KMODS      += mac_dls
419 MISC_KMODS      += cmlb
420 MISC_KMODS      += tem
421 MISC_KMODS      += pcicfg fcodem fcpci
422 MISC_KMODS      += scsi_vhci_f_sym scsi_vhci_f_tpgs scsi_vhci_f_asym_sun
423 MISC_KMODS      += scsi_vhci_f_sym_hds
424 MISC_KMODS      += scsi_vhci_f_tape scsi_vhci_f_tpgs_tape
425 MISC_KMODS      += fctl
426 MISC_KMODS      += emlxs_fw
427 MISC_KMODS      += qlc_fw_2200
428 MISC_KMODS      += qlc_fw_2300
429 MISC_KMODS      += qlc_fw_2400
430 MISC_KMODS      += qlc_fw_2500
431 MISC_KMODS      += qlc_fw_6322
432 MISC_KMODS      += qlc_fw_8100
433 MISC_KMODS      += spuni
434 MISC_KMODS      += hwa1480_fw uwba
435 MISC_KMODS      += mii

437 MISC_KMODS      += klmmod klmops

437 $(CLOSED_BUILD)CLOSED_MISC_KMODS      += klmmod klmops
439 $(CLOSED_BUILD)CLOSED_MISC_KMODS      += scsi_vhci_f_asym_lsi
440 $(CLOSED_BUILD)CLOSED_MISC_KMODS      += scsi_vhci_f_asym_emc
441 $(CLOSED_BUILD)CLOSED_MISC_KMODS      += scsi_vhci_f_sym_emc

443 #
444 #      Software Cryptographic Providers (/kernel/crypto):
445 #
446 CRYPTO_KMODS    += aes
447 CRYPTO_KMODS    += arcfour
448 CRYPTO_KMODS    += blowfish
449 CRYPTO_KMODS    += des
450 CRYPTO_KMODS    += md4
451 CRYPTO_KMODS    += md5
452 CRYPTO_KMODS    += ecc
453 CRYPTO_KMODS    += rsa
454 CRYPTO_KMODS    += sha1
455 CRYPTO_KMODS    += sha2
456 CRYPTO_KMODS    += swrand

```

```

458 #
459 # IP Policy Modules (/kernel/ipp):
460 #
461 IPP_KMODS        += dlcosmk
462 IPP_KMODS        += flowacct
463 IPP_KMODS        += ipgpc
464 IPP_KMODS        += dscppmk
465 IPP_KMODS        += tokenmt
466 IPP_KMODS        += tswtclmt

468 #
469 # 'Dacf' modules (/kernel/dacf)
470 DACF_KMODS       += consconfig_dacf

472 #
473 #      SVVS Testing Modules (/kernel/strmod):
474 #
475 #      These are streams and driver modules which are not to be
476 #      delivered with a released system. However, during development
477 #      it is convenient to build and install the SVVS kernel modules.
478 #
479 SVVS_KMODS       += lmodb lmode lmodr lmodt svvslo tidg tivc tmux

481 $(CLOSED_BUILD)SVVS      += svvs

483 #
484 #      Modules eXcluded from the product:
485 #
486 XMODS            +=
487 $(CLOSED_BUILD)CLOSED_XMODS = \
488     sdpib        \
489     wsdrv

491 #
492 #      'Dacf' Modules (/kernel/dacf):
493 #
494 DACF_KMODS       += net_dacf

496 #
497 #      MAC-Type Plugin Modules (/kernel/mac)
498 #
499 MAC_KMODS        += mac_6to4
500 MAC_KMODS        += mac_ether
501 MAC_KMODS        += mac_ipv4
502 MAC_KMODS        += mac_ipv6
503 MAC_KMODS        += mac_wifi
504 MAC_KMODS        += mac_ib

506 #
507 # socketmod (kernel/socketmod)
508 #
509 SOCKET_KMODS     += sockpfp
510 SOCKET_KMODS     += socksctp
511 SOCKET_KMODS     += socksdp
512 SOCKET_KMODS     += sockrds
513 SOCKET_KMODS     += ksslf

515 #
516 #      kiconv modules (/kernel/kiconv):
517 #
518 KICONV_KMODS     += kiconv_emea kiconv_ja kiconv_ko kiconv_sc kiconv_tc

```

```

*****
2106 Fri Aug 16 09:05:16 2013
new/usr/src/uts/sparc/klmmod/Makefile
195 Need replacement for nfs/lockd+klm
*****
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #
21 #
22 # Copyright 2006 Sun Microsystems, Inc. All rights reserved.
23 # Use is subject to license terms.
24 #
25 # Copyright (c) 2012 by Delphix. All rights reserved.
26 #
27 # This makefile drives the production of the server-side network lock
28 # manager kernel module.
29 #
30 # sparc architecture dependent
31 #
32 #
33 #
34 # Path to the base of the uts directory tree (usually /usr/src/uts).
35 #
36 UTSBASE = ../../

38 #
39 # Define the module and object file sets.
40 #
41 MODULE = klmmod
42 OBJECTS = $(KLMMOD_OBJS:%=$(OBJS_DIR)/%)
43 LINTS = $(KLMMOD_OBJS:%.o=$(LINTS_DIR)/%.ln)
44 ROOTMODULE = $(ROOT_MISC_DIR)/$(MODULE)

46 #
47 # Include common rules.
48 #
49 include $(UTSBASE)/sparc/Makefile.sparc

51 #
52 # Define targets
53 #
54 ALL_TARGET = $(BINARY)
55 LINT_TARGET = $(MODULE).lint
56 INSTALL_TARGET = $(BINARY) $(ROOTMODULE)

58 #
59 # Overrides.
60 #
61 CFLAGS += $(CCVERBOSE)

```

```

62 LDFFLAGS += -dy -Nstrmod/rpcmod -Nfs/nfs
63 LDFFLAGS += -M $(UTSBASE)/common/klm/mapfile-mod
64 CTFMRGFLAGS += -f

66 #
67 # Code generated by rpcgen triggers the -Wswitch warning.
68 #
69 CERRWARN += -_gcc=-Wno-switch

71 #
72 # Default build targets.
73 #
74 .KEEP_STATE:

76 def: $(DEF_DEPS)

78 all: $(ALL_DEPS)

80 clean: $(CLEAN_DEPS)

82 clobber: $(CLOBBER_DEPS)

84 lint: $(LINT_DEPS)

86 modlintlib: $(MODLINTLIB_DEPS)

88 clean.lint: $(CLEAN_LINT_DEPS)

90 install: $(INSTALL_DEPS)

92 #
93 # Include common targets.
94 #
95 include $(UTSBASE)/sparc/Makefile.targ

```



```

*****
2013 Fri Aug 16 09:05:16 2013
new/usr/src/uts/sparc/klmops/Makefile
195 Need replacement for nfs/lockd+klm
*****
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #
21 #
22 # Copyright 2006 Sun Microsystems, Inc. All rights reserved.
23 # Use is subject to license terms.
24 #
25 # Copyright (c) 2012 by Delphix. All rights reserved.
26 #
27 # This makefile drives the production of the client-side network lock
28 # manager kernel module.
29 #
30 # sparc architecture dependent
31 #
32 #
33 #
34 # Path to the base of the uts directory tree (usually /usr/src/uts).
35 #
36 UTSBASE = ../../

38 #
39 # Define the module and object file sets.
40 #
41 MODULE = klmops
42 OBJECTS = $(KLMOPS_OBJS:%=$(OBJS_DIR)/%)
43 LINTS = $(KLMOPS_OBJS:%.o=$(LINTS_DIR)/%.ln)
44 ROOTMODULE = $(ROOT_MISC_DIR)/$(MODULE)

46 #
47 # Include common rules.
48 #
49 include $(UTSBASE)/sparc/Makefile.sparc

51 #
52 # Define targets
53 #
54 ALL_TARGET = $(BINARY)
55 LINT_TARGET = $(MODULE).lint
56 INSTALL_TARGET = $(BINARY) $(ROOTMODULE)

58 #
59 # Overrides.
60 #
61 CFLAGS += $(CCVERBOSE)

```

```

62 LDFLAGS += -dy -Nstrmod/rpcmod -Nfs/nfs
63 LDFLAGS += -M $(UTSBASE)/common/klm/mapfile-ops
64 CTFMRGFLAGS += -f

66 #
67 # Default build targets.
68 #
69 .KEEP_STATE:

71 def: $(DEF_DEPS)

73 all: $(ALL_DEPS)

75 clean: $(CLEAN_DEPS)

77 clobber: $(CLOBBER_DEPS)

79 lint: $(LINT_DEPS)

81 modlintlib: $(MODLINTLIB_DEPS)

83 clean.lint: $(CLEAN_LINT_DEPS)

85 install: $(INSTALL_DEPS)

87 #
88 # Include common targets.
89 #
90 include $(UTSBASE)/sparc/Makefile.targ

```