```
**********************************************************
   48836 Fri Jun 13 09:21:32 2014
new/usr/src/cmd/avs/rdc/sndrd.c
3910 t_look(3NSL) should never return T_ERROR
**********************************************************
```

```
   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 2010 Sun Microsystems, Inc.  All rights reserved.
  24  * Use is subject to license terms.
  25  */
  26 /*
  27  * Copyright 2014 Nexenta Systems, Inc.  All rights reserved.
  28  * Copyright 2014 Gary Mills
  29  */

  31 /*
  32  * Network SNDR/ncall-ip server - based on nfsd
  33  */
  34 #include <sys/types.h>
  35 #include <rpc/types.h>
  36 #include <errno.h>
  37 #include <netdb.h>
  38 #include <sys/socket.h>
  39 #include <netconfig.h>
  40 #include <stropts.h>
  41 #include <fcntl.h>
  42 #include <stdio.h>
  43 #include <strings.h>
  44 #include <signal.h>
  45 #include <unistd.h>
  46 #include <stdlib.h>
  47 #include <netdir.h>
  48 #include <rpc/rpc_com.h>
  49 #include <rpc/rpc.h>
  50 #include <tiuser.h>
  51 #include <netinet/tcp.h>
  52 #include <netinet/in.h>
  53 #include <syslog.h>
  54 #include <locale.h>
  55 #include <langinfo.h>
  56 #include <libintl.h>
  57 #include <libgen.h>
  58 #include <deflt.h>
  59 #include <sys/resource.h>

  61 #include <sys/nsctl/nsctl.h>
```



```
  63 #ifdef  __NCALL__

  65 #include <sys/ncall/ncall.h>
  66 #include <sys/ncall/ncall_ip.h>
  67 #include <sys/nsctl/libncall.h>

  69 #define RDC_POOL_CREATE NC_IOC_POOL_CREATE
  70 #define RDC_POOL_RUN    NC_IOC_POOL_RUN
  71 #define RDC_POOL_WAIT   NC_IOC_POOL_WAIT
  72 #define RDC_PROGRAM     NCALL_PROGRAM
  73 #define RDC_SERVICE     "ncall"
  74 #undef RDC_SVCPOOL_ID   /* We are overloading this value */
  75 #define RDC_SVCPOOL_ID  NCALL_SVCPOOL_ID
  76 #define RDC_SVC_NAME    "NCALL"
  77 #define RDC_VERS_MIN    NCALL_VERS_MIN
  78 #define RDC_VERS_MAX    NCALL_VERS_MAX

  80 #else   /* !__NCALL__ */

  82 #include <sys/nsctl/rdc_ioctl.h>
  83 #include <sys/nsctl/rdc_io.h>
  84 #include <sys/nsctl/librdc.h>

  86 #define RDC_SERVICE     "rdc"
  87 #define RDC_SVC_NAME    "RDC"

  89 #endif  /* __NCALL__ */

  91 #define RDCADMIN        "/etc/default/sndr"

  93 #include <nsctl.h>

  95 struct conn_ind {
  96         struct conn_ind *conn_next;
  97         struct conn_ind *conn_prev;
  98         struct t_call   *conn_call;
  99 };
______unchanged_portion_omitted_

1185 static int
1186 do_poll_cots_action(int fd, int conn_index)
1187 {
1188         char buf[256];
1189         int event;
1190         int i1;
1191         int flags;
1192         struct conn_entry *connent = &conn_polled[conn_index];
1193         struct netconfig *nconf = &(connent->nc);
1194         const char *errorstr;

1196         while (event = t_look(fd)) {
1197                 switch (event) {
1198                 case T_LISTEN:
1199                         cots_listen_event(fd, conn_index);
1200                         break;

1202                 case T_DATA:
1203                         /*
1204                          * Receive a private notification from CONS rpcmod.
1205                          */
1206                         i1 = t_rcv(fd, buf, sizeof (buf), &flags);
1207                         if (i1 == -1) {
1208                                 syslog(LOG_ERR, "t_rcv failed");
1209                                 break;
1210                         }
```

```
1211                        if (i1 < sizeof (int))
1212                                break;
1213                        i1 = BE32_TO_U32(buf);
1214                        if (i1 == 1 || i1 == 2) {
1215                                /*
1216                                 * This connection has been idle for too long,
1217                                 * so release it as politely as we can.  If we
1218                                 * have already initiated an orderly release
1219                                 * and we get notified that the stream is
1220                                 * still idle, pull the plug.  This prevents
1221                                 * hung connections from continuing to consume
1222                                 * resources.
1223                                 */
1224                                if (nconf->nc_semantics == NC_TPI_COTS ||
1225                                    connent->closing != 0) {
1226                                        (void) t_snddis(fd, (struct t_call *)0);
1227                                        goto fdclose;
1228                                }
1229                                /*
1230                                 * For NC_TPI_COTS_ORD, the stream is closed
1231                                 * and removed from the poll list when the
1232                                 * T_ORDREL is received from the provider.  We
1233                                 * don't wait for it here because it may take
1234                                 * a while for the transport to shut down.
1235                                 */
1236                                if (t_sndrel(fd) == -1) {
1237                                        syslog(LOG_ERR,
1238                                        "unable to send orderly release %m");
1239                                }
1240                                connent->closing = 1;
1241                        } else
1242                                syslog(LOG_ERR,
1243                                    "unexpected event from CONS rpcmod %d", i1);
1244                        break;

1246                case T_ORDREL:
1247                        /* Perform an orderly release. */
1248                        if (t_rcvrel(fd) == 0) {
1249                                /* T_ORDREL on listen fd's should be ignored */
1250                                if (!is_listen_fd_index(fd)) {
1251                                        (void) t_sndrel(fd);
1252                                        goto fdclose;
1253                                }
1254                                break;

1256                        } else if (t_errno == TLOOK) {
1257                                break;
1258                        } else {
1259                                rdcd_log_tli_error("t_rcvrel", fd, nconf);
1260                                /*
1261                                 * check to make sure we do not close
1262                                 * listen fd
1263                                 */
1264                                if (!is_listen_fd_index(fd))
1265                                        break;
1266                                else
1267                                        goto fdclose;
1268                        }

1270                case T_DISCONNECT:
1271                        if (t_rcvdis(fd, (struct t_discon *)NULL) == -1)
1272                                rdcd_log_tli_error("t_rcvdis", fd, nconf);

1274                        /*
1275                         * T_DISCONNECT on listen fd's should be ignored.
1276                         */
```



```
1277                        if (!is_listen_fd_index(fd))
1278                                break;
1279                        else
1280                                goto fdclose;

1281                case T_ERROR:
1282                default:
1283                        if (t_errno == TSYSERR) {
1283                        if (event == T_ERROR || t_errno == TSYSERR) {
1284                                if ((errorstr = strerror(errno)) == NULL) {
1285                                        (void) snprintf(buf, sizeof (buf),
1286                                            "Unknown error num %d", errno);
1287                                        errorstr = (const char *)buf;
1288                                }
1289                        } else if (event == -1)
1290                                errorstr = t_strerror(t_errno);
1291                        else
1292                                errorstr = "";
1293 #ifdef DEBUG
1294                        syslog(LOG_ERR,
1295                            "unexpected TLI event (0x%x) on "
1296                            "connection-oriented transport(%s, %d):%s",
1297                            event, nconf->nc_proto, fd, errorstr);
1298 #endif

1300 fdclose:
1301                        num_conns--;
1302                        remove_from_poll_list(fd);
1303                        (void) t_close(fd);
1304                        return (0);
1305                }
1306        }

1308        return (0);
1309 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   10907 Fri Jun 13 09:21:32 2014
new/usr/src/cmd/bnu/interface.c
3910 t_look(3NSL) should never return T_ERROR
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2014 Gary Mills
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

/*
 *      interface(label)
 *      provide alternate definitions for the I/O functions through global
 *      interfaces.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"
/*      interface( label )
        provide alternate definitions for the I/O functions through global
        interfaces.
*/
#include        "uucp.h"

#ifdef TLI
#include        <tiuser.h>
char *t_alloc();
int t_bind(), t_close(), t_connect(), t_free(), t_look(), t_open(), t_rcvdis();
int t_getinfo(), t_getstate(), t_look(), t_rcv(), t_snd(), t_sync(), t_unbind();
#endif /*  TLI  */

#ifdef DATAKIT
#include        "dk.h"

static int      dksetup();
static int      dkteardown();
#endif  /* DATAKIT */

EXTERN void     sethup();
EXTERN int      restline();
#if defined(__STDC__)
extern int      ioctl(int, int, ...);
```



```
#else
extern int      ioctl();
#endif
static int      usetup(), uteardown();

GLOBAL ssize_t  (*Read)() = read,
        (*Write)() = write;
#if defined(__STDC__)
GLOBAL int      (*Ioctl)(int, int, ...) = ioctl,
GLOBAL int      (*Ioctl)(int,int,...) = ioctl,
#else
GLOBAL int      (*Ioctl)() = ioctl,
#endif
        (*Setup)() = usetup,
        (*Teardown)() = uteardown;

#ifdef TLI
EXTERN void tfaillog(), show_tlook();
static ssize_t  tread(), twrite();      /* TLI i/o */
#ifdef __STDC__
static int      tioctl(int, int, ...),
#else
static int      tioctl(),               /* TLI i/o control */
#endif
                tsetup(),       /* TLI setup without streams module */
                tssetup(),      /* TLI setup with streams module */
                tteardown();    /* TLI teardown, works with either setup */
#endif /*  TLI  */
/*
 *      The IN_label in Interface[] imply different caller routines:
 *      e.g. tlicall().
 *      If so, the names here and the names in callers.c must match.
                tteardown();            /* TLI teardown, works with either setup
 */
#endif /*  TLI  */
/*      The IN_label in Interface[] imply different caller routines:
        e.g. tlicall().
        If so, the names here and the names in callers.c must match.
*/

static
struct Interface {
  struct Interface {
        char    *IN_label;              /* interface name */
        ssize_t (*IN_read)();           /* read function */
        ssize_t (*IN_write)();          /* write function */
#ifdef __STDC__
        int     (*IN_ioctl)(int, int, ...);
        int     (*IN_ioctl)(int,int,...);
#else
        int     (*IN_ioctl)();          /* ioctl function */
#endif
        int     (*IN_setup)();          /* setup function, called before */
                                        /* first i/o operation */
        int     (*IN_teardown)();       /* teardown function, called after */
                                        /* last i/o operation */
        int     (*IN_setup)();          /* setup function, called before first
                                        i/o operation */
        int     (*IN_teardown)();       /* teardown function, called after last
                                        i/o operation */
} Interface[] = {
                        /* vanilla UNIX */
                { "UNIX", read, write, ioctl, usetup, uteardown },
#ifdef TCP
                        /* TCP over sockets or UNET */
                { "TCP", read, write, ioctl, usetup, uteardown },
```

```
#endif /* TCP */
#ifdef SYTEK
                        /* Sytek network */
                { "Sytek", read, write, ioctl, usetup, uteardown },
#endif /* Sytek network */
#ifdef DIAL801
                        /* 801 auto dialers */
                { "801", read, write, ioctl, usetup, uteardown },
#endif /* DIAL801 */
#ifdef DIAL801
                        /* 212 auto dialers */
                { "212", read, write, ioctl, usetup, uteardown },
#endif /* DIAL801 */
#ifdef TLI
                        /* AT&T Transport Interface Library WITHOUT streams */
                { "TLI", tread, twrite, tioctl, tsetup, tteardown },
#ifdef TLIS
                        /* AT&T Transport Interface Library WITH streams */
                { "TLIS", read, write, tioctl, tssetup, uteardown },
#endif /*  TLIS  */
#endif /*  TLI  */
#ifdef DATAKIT
                { "DK", read, write, ioctl, dksetup, dkteardown },
#endif /* DATAKIT */
#ifdef UNET
                { "Unetserver", read, write, ioctl, usetup, uteardown },
#endif
                { 0, 0, 0, 0, 0, 0 }
        };


GLOBAL int
interface(label)
char    *label;
{
        register int    i;

        for (i = 0;  Interface[i].IN_label;  ++i) {
                if (0 == strcmp(Interface[i].IN_label, label)) {
        for ( i = 0;  Interface[i].IN_label;  ++i ) {
                if( !strcmp( Interface[i].IN_label, label ) ) {
                        Read = Interface[i].IN_read;
                        Write = Interface[i].IN_write;
                        Ioctl = Interface[i].IN_ioctl;
                        Setup = Interface[i].IN_setup;
                        Teardown = Interface[i].IN_teardown;
                        DEBUG(5, "set interface %s\n", label);
                        return (0);
                        return( 0 );
                }
        }
        return (FAIL);
        return( FAIL );
}

/*
 *      usetup - vanilla unix setup routine
 */
static int
usetup(role, fdreadp, fdwritep)
usetup( role, fdreadp, fdwritep )
int     *fdreadp, *fdwritep;
{
        if (role == SLAVE)
        if ( role == SLAVE )
        {
```



```
                *fdreadp = 0;
                *fdwritep = 1;
                /* 2 has been re-opened to RMTDEBUG in main() */
        }
        return (SUCCESS);
        return(SUCCESS);
}

/*
 *      uteardown - vanilla unix teardown routine
 */
static int
uteardown(role, fdread, fdwrite)
uteardown( role, fdread, fdwrite )
{
        int ret;
        char *ttyn;

        if (role == SLAVE) {
                ret = restline();
                DEBUG(4, "ret restline - %d\n", ret);
                if (fdread != -1)
                        sethup(fdread);
        }
        if (fdread != -1) {
                ttyn = ttyname(fdread);
                if (ttyn != NULL && Dev_mode != 0)
                        chmod(ttyn, Dev_mode);  /* can fail, but who cares? */
                (void) close(fdread);
                (void) close(fdwrite);
        }
        return (SUCCESS);
        return(SUCCESS);
}

#ifdef DATAKIT
/*
 *      dksetup - DATAKIT setup routine
 *
 * Put line in block mode.
 */

static int
dksetup(role, fdreadp, fdwritep)
dksetup (role, fdreadp, fdwritep)

int     role;
int     *fdreadp;
int     *fdwritep;
int *   fdreadp;
int *   fdwritep;

{
        static short dkrmode[3] = { DKR_BLOCK | DKR_TIME, 0, 0 };
        int     ret;

        (void) usetup(role, fdreadp, fdwritep);
        if ((ret = (*Ioctl)(*fdreadp, DIOCRMODE, dkrmode)) < 0) {
        if((ret = (*Ioctl)(*fdreadp, DIOCRMODE, dkrmode)) < 0) {
                DEBUG(4, "dksetup: failed to set block mode. ret=%d,\n", ret);
                DEBUG(4, "read fd=%d, ", *fdreadp);
                DEBUG(4, "errno=%d\n", errno);
                return (FAIL);
                return(FAIL);
        }
        return (SUCCESS);
```

```
226         return(SUCCESS);
227 }

229 /*
230  *      dkteardown  -  DATAKIT teardown routine
231  */
232 static int
233 dkteardown(role, fdread, fdwrite)
233 dkteardown( role, fdread, fdwrite )
234 int     role, fdread, fdwrite;
235 {
236         char    *ttyn;

238         if (role == MASTER) {
238         if ( role == MASTER ) {
239                 ttyn = ttyname(fdread);
240                 if (ttyn != NULL && Dev_mode != 0)
240                 if ( ttyn != NULL && Dev_mode != 0 )
241                         chmod(ttyn, Dev_mode);  /* can fail, but who cares? */
242         }

244         /*      must flush fd's for datakit     */
245         /*      else close can hang             */
246         if (ioctl(fdread, DIOCFLUSH, NULL) != 0)
247                 DEBUG(4, "dkteardown: DIOCFLUSH of input fd %d failed",
248                     fdread);
249         if (ioctl(fdwrite, DIOCFLUSH, NULL) != 0)
250                 DEBUG(4, "dkteardown: DIOCFLUSH of output fd %d failed",
251                     fdwrite);
246         if ( ioctl(fdread, DIOCFLUSH, NULL) != 0 )
247                 DEBUG(4, "dkteardown: DIOCFLUSH of input fd %d failed", fdread);
248         if ( ioctl(fdwrite, DIOCFLUSH, NULL) != 0 )
249                 DEBUG(4, "dkteardown: DIOCFLUSH of output fd %d failed", fdwrite

253         (void) close(fdread);
254         (void) close(fdwrite);
255         return (SUCCESS);
251         (void)close(fdread);
252         (void)close(fdwrite);
253         return(SUCCESS);
256 }
257 #endif /* DATAKIT */


260 #ifdef TLI
261 /*
262  *      tread - tli read routine
263  */
264 static ssize_t
265 tread(fd, buf, nbytes)
266 int             fd;
267 char            *buf;
268 unsigned        nbytes;
269 {
270         int             rcvflags;

272         return ((ssize_t)t_rcv(fd, buf, nbytes, &rcvflags));
270         return((ssize_t)t_rcv(fd, buf, nbytes, &rcvflags));
273 }

275 /*
276  *      twrite - tli write routine
277  */
278 #define N_CHECK 100
279 static ssize_t
280 twrite(fd, buf, nbytes)
```



```
281 int             fd;
282 char            *buf;
283 unsigned        nbytes;
284 {
285         register int            i, ret;
286         static int              n_writ, got_info;
287         static struct t_info    info;

289         if (got_info == 0) {
290                 if (t_getinfo(fd, &info) != 0) {
287         if ( got_info == 0 ) {
288                 if ( t_getinfo(fd, &info) != 0 ) {
291                         tfaillog(fd, "twrite: t_getinfo\n");
292                         return (FAIL);
290                         return(FAIL);
293                 }
294                 got_info = 1;
295         }

297         /* on every N_CHECKth call, check that are still in DATAXFER state */
298         if (++n_writ == N_CHECK) {
296         if ( ++n_writ == N_CHECK ) {
299                 n_writ = 0;
300                 if (t_getstate(fd) != T_DATAXFER)
301                         return (FAIL);
298                 if ( t_getstate(fd) != T_DATAXFER )
299                         return(FAIL);
302         }

304         if (info.tsdu <= 0 || nbytes <= info.tsdu)
305                 return (t_snd(fd, buf, nbytes, NULL));
302         if ( info.tsdu <= 0 || nbytes <= info.tsdu )
303                 return(t_snd(fd, buf, nbytes, NULL));

307         /* if get here, then there is a limit on transmit size  */
308         /* (info.tsdu > 0) and buf exceeds it                   */
309         i = ret = 0;
310         while (nbytes >= info.tsdu) {
311                 if ((ret = t_snd(fd,  &buf[i], info.tsdu, NULL)) != info.tsdu)
312                         return ((ret >= 0 ? (i + ret) : ret));
308         while ( nbytes >= info.tsdu ) {
309                 if ( (ret = t_snd(fd,  &buf[i], info.tsdu, NULL)) != info.tsdu )
310                         return( ( ret >= 0 ? (i + ret) : ret ) );
313                 i += info.tsdu;
314                 nbytes -= info.tsdu;
315         }
316         if (nbytes != 0) {
317                 if ((ret = t_snd(fd,  &buf[i], nbytes, NULL)) != nbytes)
318                         return ((ssize_t)(ret >= 0 ? (i + ret) : ret));
314         if ( nbytes != 0 ) {
315                 if ( (ret = t_snd(fd,  &buf[i], nbytes, NULL)) != nbytes )
316                         return( (ssize_t)( ret >= 0 ? (i + ret) : ret ) );
319                 i += nbytes;
320         }
321         return ((ssize_t)i);
319         return((ssize_t)i);
322 }


325 /*
326  *      tioctl - stub for tli ioctl routine
327  */
328 /*ARGSUSED*/
329 static int
330 #ifdef __STDC__
331 tioctl(int fd, int request, ...)
```

```
#else
tioctl(fd, request, arg)
int     fd, request;
#endif
{
        return (SUCCESS);
        return(SUCCESS);
}

/*
 *      tsetup - tli setup routine
 *      note blatant assumption that *fdreadp == *fdwritep == 0
 */
static int
tsetup(role, fdreadp, fdwritep)
tsetup( role, fdreadp, fdwritep )
int     *fdreadp, *fdwritep;
{

        if (role == SLAVE) {
        if ( role == SLAVE ) {
                *fdreadp = 0;
                *fdwritep = 1;
                /* 2 has been re-opened to RMTDEBUG in main() */
                errno = t_errno = 0;
                if (t_sync(*fdreadp) == -1 || t_sync(*fdwritep) == -1) {
                if ( t_sync(*fdreadp) == -1 || t_sync(*fdwritep) == -1 ) {
                        tfaillog(*fdreadp, "tsetup: t_sync\n");
                        return (FAIL);
                        return(FAIL);
                }
        }
        return (SUCCESS);
        return(SUCCESS);
}

/*
 *      tteardown - tli shutdown routine
 */
/*ARGSUSED*/
static int
tteardown(role, fdread, fdwrite)
tteardown( role, fdread, fdwrite )
{
        (void) t_unbind(fdread);
        (void) t_close(fdread);
        return (SUCCESS);
        (void)t_unbind(fdread);
        (void)t_close(fdread);
        return(SUCCESS);
}

#ifdef TLIS
/*
 *      tssetup - tli, with streams module, setup routine
 *      note blatant assumption that *fdreadp == *fdwritep
 */
static int
tssetup(role, fdreadp, fdwritep)
tssetup( role, fdreadp, fdwritep )
int     role;
int     *fdreadp;
int     *fdwritep;
{
        if (role == SLAVE) {
        if ( role == SLAVE ) {
```



```
                *fdreadp = 0;
                *fdwritep = 1;
                /* 2 has been re-opened to RMTDEBUG in main() */
                DEBUG(5, "tssetup: SLAVE mode: leaving ok\n%s", "");
                return (SUCCESS);
                return(SUCCESS);
        }

        DEBUG(4, "tssetup: MASTER mode: leaving ok\n%s", "");
        return (SUCCESS);
        return(SUCCESS);
}

/*
 *      Report why a TLI call failed.
 */
GLOBAL void
tfaillog(fd, s)
int     fd;
char    *s;
{
        char    fmt[ BUFSIZ ];

        if (0 < t_errno && t_errno < t_nerr) {
                sprintf(fmt, "%s: %%s\n", s);
                sprintf( fmt, "%s: %%s\n", s );
                DEBUG(5, fmt, t_errlist[t_errno]);
                logent(s, t_errlist[t_errno]);
                if (t_errno == TSYSERR) {
                if ( t_errno == TSYSERR ) {
                        strcpy(fmt, "tlicall: system error: %s\n");
                        DEBUG(5, fmt, strerror(errno));
                } else if (t_errno == TLOOK) {
                } else if ( t_errno == TLOOK ) {
                        show_tlook(fd);
                }
        } else {
                sprintf(fmt, "unknown tli error %d", t_errno);
                logent(s, fmt);
                sprintf(fmt, "%s: unknown tli error %d", s, t_errno);
                DEBUG(5, fmt, 0);
                sprintf(fmt, "%s: %%s\n", s);
                DEBUG(5, fmt, strerror(errno));
        }
        return;
}

GLOBAL void
show_tlook(fd)
int fd;
{
        register int reason;
        register char *msg;
        extern int      t_errno;
/*
 * Find out the current state of the interface.
 */
        errno = t_errno = 0;
        switch (reason = t_getstate(fd)) {
        switch( reason = t_getstate(fd) ) {
        case T_UNBND:           msg = "T_UNBIND";       break;
        case T_IDLE:            msg = "T_IDLE";         break;
        case T_OUTCON:          msg = "T_OUTCON";       break;
        case T_INCON:           msg = "T_INCON";        break;
        case T_DATAXFER:        msg = "T_DATAXFER";     break;
        case T_OUTREL:          msg = "T_OUTREL";       break;
```

```
 445         case T_INREL:           msg = "T_INREL";        break;
 446         default:                msg = NULL;             break;
 447         }
 448         if (msg == NULL)
 447         if( msg == NULL )
 449                 return;

 451         DEBUG(5, "state is %s", msg);
 452         switch (reason = t_look(fd)) {
 451         switch( reason = t_look(fd) ) {
 453         case -1:                msg = ""; break;
 454         case 0:                 msg = "NO ERROR"; break;
 455         case T_LISTEN:          msg = "T_LISTEN"; break;
 456         case T_CONNECT:         msg = "T_CONNECT"; break;
 457         case T_DATA:            msg = "T_DATA";  break;
 458         case T_EXDATA:          msg = "T_EXDATA"; break;
 459         case T_DISCONNECT:      msg = "T_DISCONNECT"; break;
 460         case T_ORDREL:          msg = "T_ORDREL"; break;
 460         case T_ERROR:           msg = "T_ERROR"; break;
 461         case T_UDERR:           msg = "T_UDERR"; break;
 462         default:                msg = "UNKNOWN ERROR"; break;
 463         }
 464         DEBUG(4, " reason is %s\n", msg);

 466         if (reason == T_DISCONNECT)
 466         if ( reason == T_DISCONNECT )
 467         {
 468                 struct t_discon *dropped;
 469                 if (((dropped =
 470                     (struct t_discon *)t_alloc(fd, T_DIS, T_ALL)) == 0) ||
 471                     (t_rcvdis(fd, dropped) == -1)) {
 469                 if ( ((dropped =
 470                         (struct t_discon *)t_alloc(fd, T_DIS, T_ALL)) == 0)
 471                 ||  (t_rcvdis(fd, dropped) == -1 )) {
 472                         if (dropped)
 473                                 t_free((char *)dropped, T_DIS);
 474                         return;
 475                 }
 476                 DEBUG(5, "disconnect reason #%d\n", dropped->reason);
 477                 t_free((char *)dropped, T_DIS);
 478         }
 479         return;
 479 }
_____unchanged_portion_omitted_
```

```
**********************************************************
   44708 Fri Jun 13 09:21:32 2014
new/usr/src/cmd/fs.d/nfs/lib/nfs_tbind.c
3910 t_look(3NSL) should never return T_ERROR
**********************************************************
```

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

/*
 * Copyright (c) 1996, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2012 by Delphix. All rights reserved.
 * Copyright 2014 Nexenta Systems, Inc.  All rights reserved.
 * Copyright 2014 Gary Mills
 */


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

#include <tiuser.h>
#include <fcntl.h>
#include <netconfig.h>
#include <stropts.h>
#include <errno.h>
#include <syslog.h>
#include <rpc/rpc.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <signal.h>
#include <netdir.h>
#include <unistd.h>
#include <string.h>
#include <netinet/tcp.h>
#include <malloc.h>
#include <stdlib.h>
#include "nfs_tbind.h"
#include <nfs/nfs.h>
#include <nfs/nfs_acl.h>
#include <nfs/nfssys.h>
#include <nfs/nfs4.h>
#include <zone.h>
#include <sys/socket.h>
#include <tsol/label.h>

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



```
#define OK_TPI_TYPE(_nconf) \
        (_nconf->nc_semantics == NC_TPI_CLTS || \
        _nconf->nc_semantics == NC_TPI_COTS || \
        _nconf->nc_semantics == NC_TPI_COTS_ORD)

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

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

/*
 * Number of file descriptors by which the process soft limit may be
 * increased on each call to nofile_increase(0).
 */
#define NOFILE_INC_SIZE 64

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

struct conn_ind {
        struct conn_ind *conn_next;
        struct conn_ind *conn_prev;
        struct t_call   *conn_call;
};
```

*unchanged_portion_omitted_*

```
static int
do_poll_cots_action(int fd, int conn_index)
{
        char buf[256];
        int event;
        int i1;
        int flags;
        struct conn_entry *connent = &conn_polled[conn_index];
        struct netconfig *nconf = &(connent->nc);
        const char *errorstr;

        while (event = t_look(fd)) {
                switch (event) {
                case T_LISTEN:
#ifdef DEBUG
printf("do_poll_cots_action(%s,%d): T_LISTEN event\n", nconf->nc_proto, fd);
#endif
                        cots_listen_event(fd, conn_index);
                        break;

                case T_DATA:
#ifdef DEBUG
printf("do_poll_cots_action(%d,%s): T_DATA event\n", fd, nconf->nc_proto);
#endif
                        /*
                         * Receive a private notification from CONS rpcmod.
                         */
                        i1 = t_rcv(fd, buf, sizeof (buf), &flags);
                        if (i1 == -1) {
                                syslog(LOG_ERR, "t_rcv failed");
                                break;
                        }
```

```
1511                        if (i1 < sizeof (int))
1512                                break;
1513                        i1 = BE32_TO_U32(buf);
1514                        if (i1 == 1 || i1 == 2) {
1515                                /*
1516                                 * This connection has been idle for too long,
1517                                 * so release it as politely as we can.  If we
1518                                 * have already initiated an orderly release
1519                                 * and we get notified that the stream is
1520                                 * still idle, pull the plug.  This prevents
1521                                 * hung connections from continuing to consume
1522                                 * resources.
1523                                 */
1524 #ifdef DEBUG
1525 printf("do_poll_cots_action(%s,%d): ", nconf->nc_proto, fd);
1526 printf("initiating orderly release of idle connection\n");
1527 #endif
1528                                if (nconf->nc_semantics == NC_TPI_COTS ||
1529                                    connent->closing != 0) {
1530                                        (void) t_snddis(fd, (struct t_call *)0);
1531                                        goto fdclose;
1532                                }
1533                                /*
1534                                 * For NC_TPI_COTS_ORD, the stream is closed
1535                                 * and removed from the poll list when the
1536                                 * T_ORDREL is received from the provider.  We
1537                                 * don't wait for it here because it may take
1538                                 * a while for the transport to shut down.
1539                                 */
1540                                if (t_sndrel(fd) == -1) {
1541                                        syslog(LOG_ERR,
1542                                        "unable to send orderly release %m");
1543                                }
1544                                connent->closing = 1;
1545                        } else
1546                                syslog(LOG_ERR,
1547                                "unexpected event from CONS rpcmod %d", i1);
1548                        break;

1550                case T_ORDREL:
1551 #ifdef DEBUG
1552 printf("do_poll_cots_action(%s,%d): T_ORDREL event\n", nconf->nc_proto, fd);
1553 #endif
1554                        /* Perform an orderly release. */
1555                        if (t_rcvrel(fd) == 0) {
1556                                /* T_ORDREL on listen fd's should be ignored */
1557                                if (!is_listen_fd_index(conn_index)) {
1558                                        (void) t_sndrel(fd);
1559                                        goto fdclose;
1560                                }
1561                                break;

1563                        } else if (t_errno == TLOOK) {
1564                                break;
1565                        } else {
1566                                nfslib_log_tli_error("t_rcvrel", fd, nconf);

1568                                /*
1569                                 * check to make sure we do not close
1570                                 * listen fd
1571                                 */
1572                                if (is_listen_fd_index(conn_index))
1573                                        break;
1574                                else
1575                                        goto fdclose;
1576                        }
```



```
1578                case T_DISCONNECT:
1579 #ifdef DEBUG
1580 printf("do_poll_cots_action(%s,%d): T_DISCONNECT event\n", nconf->nc_proto, fd);
1581 #endif
1582                        if (t_rcvdis(fd, (struct t_discon *)NULL) == -1)
1583                                nfslib_log_tli_error("t_rcvdis", fd, nconf);

1585                        /*
1586                         * T_DISCONNECT on listen fd's should be ignored.
1587                         */
1588                        if (is_listen_fd_index(conn_index))
1589                                break;
1590                        else
1591                                goto fdclose;

1592                case T_ERROR:
1593                default:
1594                        if (t_errno == TSYSERR) {
1594                        if (event == T_ERROR || t_errno == TSYSERR) {
1595                                if ((errorstr = strerror(errno)) == NULL) {
1596                                        (void) sprintf(buf,
1597                                            "Unknown error num %d", errno);
1598                                        errorstr = (const char *) buf;
1599                                }
1600                        } else if (event == -1)
1601                                errorstr = t_strerror(t_errno);
1602                        else
1603                                errorstr = "";
1604                        syslog(LOG_ERR,
1605                            "unexpected TLI event (0x%x) on "
1606                            "connection-oriented transport(%s,%d):%s",
1607                            event, nconf->nc_proto, fd, errorstr);
1608 fdclose:
1609                        num_conns--;
1610                        remove_from_poll_list(fd);
1611                        (void) t_close(fd);
1612                        return (0);
1613                }
1614        }

1616        return (0);
1617 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    8818 Fri Jun 13 09:21:32 2014
new/usr/src/lib/libnsl/dial/interface.c
3910 t_look(3NSL) should never return T_ERROR
**********************************************************
```

```
   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  */

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

  26 /*
  27  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  28  * Use is subject to license terms.
  29  * Copyright 2014 Gary Mills
  30  */

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

  32 /*
  33  * interface( label )
  34  *      provide alternate definitions for the I/O functions through global
  35  *      interfaces.
  36  */
  37 #include "mt.h"
  38 #include "uucp.h"
  39 #include <unistd.h>

  41 #ifdef TLI
  42 #include <tiuser.h>
  43 #endif /*  TLI  */

  45 static void     sethup(int);
  46 static int      restline(void);
  47 static int      usetup(int, int *, int *);
  48 static int      uteardown(int, int, int);

  50 static ssize_t  (*Read)() = read,
  51         (*Write)() = write;
  52 static int      (*Ioctl)(int, int, ...) = ioctl,
  53         (*Setup)() = usetup;

  55 #ifdef TLI
  56 static void tfaillog(int fd, const char *s);
  57 static void show_tlook(int);
  58 static ssize_t  tread(int, char *, unsigned);
  59 static ssize_t  twrite(int, char *, unsigned);
```



```
  60 static int tioctl(int, int, ...);
  61 static int tsetup(int, int *, int *); /* TLI setup without streams module */
  62 static int tssetup(int, int *, int *); /* TLI setup with streams module */
  63 static int tteardown(int, int, int); /* TLI teardown, works with either setup */
  64 #endif /*  TLI  */

  66 /*
  67  *      The IN_label in Interface[] imply different caller routines:
  68  *      e.g. tlicall().
  69  *      If so, the names here and the names in callers.c must match.
  70  */
  71 static struct Interface {
  72         const   char    *IN_label;      /* interface name */
  73         ssize_t (*IN_read)();           /* read function */
  74         ssize_t (*IN_write)();          /* write function */
  75         int     (*IN_ioctl)(int, int, ...);
  76         int     (*IN_setup)();          /* setup function, called before */
  77                                         /* first i/o operation */
  78         int     (*IN_teardown)();       /* teardown function, called after */
  79                                         /* last i/o operation */
  80 } Interface[] = {
_____unchanged_portion_omitted_

 299 static void
 300 show_tlook(int fd)
 301 {
 302         int reason;
 303         const char *msg;

 305 /*
 306  * Find out the current state of the interface.
 307  */
 308         errno = t_errno = 0;
 309         switch (reason = t_getstate(fd)) {
 310         case T_UNBND:           msg = (const char *)"T_UNBIND"; break;
 311         case T_IDLE:            msg = (const char *)"T_IDLE";  break;
 312         case T_OUTCON:          msg = (const char *)"T_OUTCON"; break;
 313         case T_INCON:           msg = (const char *)"T_INCON";  break;
 314         case T_DATAXFER:        msg = (const char *)"T_DATAXFER"; break;
 315         case T_OUTREL:          msg = (const char *)"T_OUTREL"; break;
 316         case T_INREL:           msg = (const char *)"T_INREL";  break;
 317         default:                msg = NULL;             break;
 318         }
 319         if (msg == NULL)
 320                 return;
 321         DEBUG(5, "state is %s", msg);
 322         switch (reason = t_look(fd)) {
 323         case -1:                msg = (const char *)""; break;
 324         case 0:                 msg = (const char *)"NO ERROR"; break;
 325         case T_LISTEN:          msg = (const char *)"T_LISTEN"; break;
 326         case T_CONNECT:         msg = (const char *)"T_CONNECT"; break;
 327         case T_DATA:            msg = (const char *)"T_DATA";    break;
 328         case T_EXDATA:          msg = (const char *)"T_EXDATA"; break;
 329         case T_DISCONNECT:      msg = (const char *)"T_DISCONNECT"; break;
 330         case T_ORDREL:          msg = (const char *)"T_ORDREL"; break;
 332         case T_ERROR:           msg = (const char *)"T_ERROR"; break;
 331         case T_UDERR:           msg = (const char *)"T_UDERR"; break;
 332         default:                msg = (const char *)"UNKNOWN ERROR"; break;
 333         }
 334         DEBUG(4, " reason is %s\n", msg);

 336         if (reason == T_DISCONNECT) {
 337                 struct t_discon *dropped;
 338                 if (((dropped =
 339                     /* LINTED pointer cast */
 340                     (struct t_discon *)t_alloc(fd, T_DIS, T_ALL)) == 0) ||
```

```
341                     (t_rcvdis(fd, dropped) == -1)) {
342                         if (dropped)
343                                 (void) t_free((char *)dropped, T_DIS);
344                         return;
345                 }
346                 DEBUG(5, "disconnect reason #%d\n", dropped->reason);
347                 (void) t_free((char *)dropped, T_DIS);
348         }
349 }
```

***unchanged_portion_omitted_***

```
**********************************************************
    6204 Fri Jun 13 09:21:33 2014
new/usr/src/lib/libnsl/nsl/t_look.c
3910 t_look(3NSL) should never return T_ERROR
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

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

/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 * Copyright 2014 Gary Mills
 */

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

#include "mt.h"
#include <errno.h>
#include <unistd.h>
#include <sys/stream.h>
#include <stropts.h>
#define _SUN_TPI_VERSION 2
#include <sys/tihdr.h>
#include <sys/timod.h>
#include <xti.h>
#include <assert.h>
#include "tx.h"

int
_tx_look(int fd, int api_semantics)
{
        int state;
        int sv_errno;
        int do_expinline_peek;   /* unusual XTI specific processing */
        struct _ti_user *tiptr;

        if ((tiptr = _t_checkfd(fd, 0, api_semantics)) == NULL)
                return (-1);
        sig_mutex_lock(&tiptr->ti_lock);

        if (_T_IS_XTI(api_semantics))
                do_expinline_peek = 1;
        else
                do_expinline_peek = 0;
```



```
        state = _t_look_locked(fd, tiptr, do_expinline_peek, api_semantics);

        sv_errno = errno;

        sig_mutex_unlock(&tiptr->ti_lock);
        errno = sv_errno;
        return (state);
}

/*
 * _t_look_locked() assumes tiptr->ti_lock lock is already held and signals
 * already blocked in MT case.
 * Intended for use by other TLI routines only.
 */
int
_t_look_locked(
        int fd,
        struct _ti_user *tiptr,
        int do_expinline_peek,
        int api_semantics
)
{
        struct strpeek strpeek;
        int retval;
        union T_primitives *pptr;
        t_scalar_t type;
        t_scalar_t ctltype;

        assert(MUTEX_HELD(&tiptr->ti_lock));

#ifdef notyet
        if (_T_IS_XTI(api_semantics)) {
                /*
                 * XTI requires the strange T_GODATA and T_GOEXDATA
                 * events which are almost brain-damaged but thankfully
                 * not tested. Anyone feeling the need for those should
                 * consider the need for using non-blocking endpoint.
                 * Probably introduced at the behest of some weird-os
                 * vendor which did not understand the non-blocking endpoint
                 * option.
                 * We choose not to implment these mis-features.
                 * Here is the plan-of-action (POA)if we are ever forced
                 * to implement these.
                 * - When returning TFLOW set state to indicate if it was
                 *   a normal or expedited data send attempt.
                 * - In routines that set TFLOW, clear the above set state
                 *   on each entry/reentry
                 * - In this routine, if that state flag is set,
                 * do a I_CANPUT on appropriate band to to see if it
                 * is writeable. If that indicates that the band is
                 * writeable, return T_GODATA or T_GOEXDATA event.
                 *
                 * Actions are also influenced by whether T_EXDATA_REQ stays
                 * band 1 or goes to band 0 if EXPINLINE is set
                 *
                 * We will also need to sort out if "write side" events
                 * (such as T_GODATA/T_GOEXDATA) take precedence over
                 * all other events (all read side) or not.
                 */
        }
#endif /* notyet */

        strpeek.ctlbuf.maxlen = (int)sizeof (ctltype);
        strpeek.ctlbuf.len = 0;
        strpeek.ctlbuf.buf = (char *)&ctltype;
        strpeek.databuf.maxlen = 0;
```

```
126         strpeek.databuf.len = 0;
127         strpeek.databuf.buf = NULL;
128         strpeek.flags = 0;

130         do {
131                 retval = ioctl(fd, I_PEEK, &strpeek);
132         } while (retval < 0 && errno == EINTR);

134         if (retval < 0) {
136                 if (_T_IS_TLI(api_semantics)) {
135                 /*
136                  * XTI semantics (also identical to documented
137                  * TLI semantics).
138                         * This return of T_ERROR event is ancient
139                         * SVR3 TLI semantics and not documented for
140                         * current SVR4 TLI interface.
141                         * Fixing this will impact some apps
142                         * (e.g. nfsd,lockd) in ON consolidation
143                         * so they need to be fixed first before TLI
144                         * can be fixed.
145                         * XXX Should we never fix this because it might
146                         * break apps in field ?
138                  */
148                         return (T_ERROR);
149                 }
150                 /*
151                  * XTI semantics (also identical to documented,
152                  * but not implemented TLI semantics).
153                  */
139                 t_errno = TSYSERR;
140                 return (-1);
141         }

143         /*
144          * if something there and cntl part also there
145          */
146         if ((tiptr->ti_lookcnt > 0) ||
147             ((retval > 0) && (strpeek.ctlbuf.len >=
148             (int)sizeof (t_scalar_t)))) {
162         ((retval > 0) && (strpeek.ctlbuf.len >= (int)sizeof (t_scalar_t)))) {
149                 /* LINTED pointer cast */
150                 pptr = (union T_primitives *)strpeek.ctlbuf.buf;
151                 if (tiptr->ti_lookcnt > 0) {
152                         /* LINTED pointer cast */
153                         type = *((t_scalar_t *)tiptr->ti_lookbufs.tl_lookcbuf);
154                         /*
155                          * If message on stream head is a T_DISCON_IND, that
156                          * has priority over a T_ORDREL_IND in the look
157                          * buffer.
158                          * (This assumes that T_ORDREL_IND can only be in the
159                          * first look buffer in the list)
160                          */
161                         if ((type == T_ORDREL_IND) && retval &&
162                             (pptr->type == T_DISCON_IND)) {
163                                 type = pptr->type;
164                                 /*
165                                  * Blow away T_ORDREL_IND
166                                  */
167                                 _t_free_looklist_head(tiptr);
168                         }
169                 } else
170                         type = pptr->type;

172                 switch (type) {

174                 case T_CONN_IND:
```



```
175                         return (T_LISTEN);

177                 case T_CONN_CON:
178                         return (T_CONNECT);

180                 case T_DISCON_IND:
181                         return (T_DISCONNECT);

183                 case T_DATA_IND: {
184                         int event = T_DATA;
185                         int retval, exp_on_q;

187                         if (do_expinline_peek &&
188                             (tiptr->ti_prov_flag & EXPINLINE)) {
189                                 assert(_T_IS_XTI(api_semantics));
190                                 retval = _t_expinline_queued(fd, &exp_on_q);
191                                 if (retval < 0) {
192                                         t_errno = TSYSERR;
193                                         return (-1);
194                                 }
195                                 if (exp_on_q)
196                                         event = T_EXDATA;
197                         }
198                         return (event);
199                 }

201                 case T_UNITDATA_IND:
202                         return (T_DATA);

204                 case T_EXDATA_IND:
205                         return (T_EXDATA);

207                 case T_UDERROR_IND:
208                         return (T_UDERR);

210                 case T_ORDREL_IND:
211                         return (T_ORDREL);

213                 default:
214                         t_errno = TSYSERR;
215                         errno = EPROTO;
216                         return (-1);
217                 }
218         }

220         /*
221          * if something there put no control part
222          * it must be data on the stream head.
223          */
224         if ((retval > 0) && (strpeek.ctlbuf.len <= 0)) {
225                 int event = T_DATA;
226                 int retval, exp_on_q;

228                 if (do_expinline_peek &&
229                     (tiptr->ti_prov_flag & EXPINLINE)) {
230                         assert(_T_IS_XTI(api_semantics));
231                         retval = _t_expinline_queued(fd, &exp_on_q);
232                         if (retval < 0)
233                                 return (-1);
234                         if (exp_on_q)
235                                 event = T_EXDATA;
236                 }
237                 return (event);
238         }

240         /*
```

```
241          * if msg there and control
242          * part not large enough to determine type?
243          * it must be illegal TLI message
244          */
245         if ((retval > 0) && (strpeek.ctlbuf.len > 0)) {
246                 t_errno = TSYSERR;
247                 errno = EPROTO;
248                 return (-1);
249         }
250         return (0);
251 }
```

***unchanged_portion_omitted_***

```
**********************************************************
   12738 Fri Jun 13 09:21:33 2014
new/usr/src/lib/libnsl/nsl/tx.h
3910 t_look(3NSL) should never return T_ERROR
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2014 Gary Mills
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _TX_H
#define _TX_H

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

#include <sys/uio.h>

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * This file contains declarations local to the TLI/XTI implmentation
 */

/*
 * Look buffer list
 * Could be multiple buffers for MT case
 */
struct _ti_lookbufs {
        struct _ti_lookbufs *tl_next; /* next in list   */
        int     tl_lookclen;    /* "look" ctl part length */
        char    *tl_lookcbuf;   /* pointer to "look" ctl        */
        int     tl_lookdlen;    /* "look" data length   */
        char    *tl_lookdbuf;   /* pointer to "look" data */
};
_____unchanged_portion_omitted_

/*
 * Local flags used with ti_flags field in instance structure of
 * type 'struct _ti_user' declared above. Historical note:
 * This namespace constants were previously declared in a
 * a very messed up namespace in timod.h
```



```
 */
#define USED            0x0001  /* data structure in use                */
#define MORE            0x0008  /* more data                            */
#define EXPEDITED       0x0010  /* processing expedited TSDU            */
#define V_ACCEPTOR_ID   0x0020  /* acceptor_id field is has valid value */
#define TX_TQFULL_NOTIFIED 0x0040  /* TQFULL error has been returned once  */


/*
 * Valid flags that can be passed by user in t_sndv() or t_snd()
 */

#define TX_ALL_VALID_FLAGS (T_MORE|T_EXPEDITED|T_PUSH)

#define _T_MAX(x, y)            ((x) > (y) ? (x) : (y))

/*
 * Following are used to indicate which API entry point is calling common
 * routines
 */
#define         TX_TLI_API      1       /* The API semantics is TLI */
#define         TX_XTI_XNS4_API 2       /* The API semantics is XTI Unix95 */
#define         TX_XTI_XNS5_API 3       /* The API semantics is XTI Unix98 */
#define         TX_XTI_API      TX_XTI_XNS4_API
                                        /* The base XTI semantics is Unix95 */

/* _T_IS_XTI(x) - Is 'x' an XTI inspired api_semantics */
#define         _T_IS_XTI(x)    ((x) != TX_TLI_API)
#define         _T_IS_TLI(x)    ((x) == TX_TLI_API)

/* _T_API_VER_LT(x, y) - Is API version 'x' older than API version 'y' */
#define         _T_API_VER_LT(x, y)     ((x) < (y))

/*
 * Note: T_BADSTATE also defined in <sys/tiuser.h>
 */
#define T_BADSTATE 8

#ifdef DEBUG
#include <syslog.h>
#define _T_TX_SYSLOG2(tiptr, X, Y) if ((tiptr)->ti_state == T_BADSTATE)\
        syslog(X, Y)
#else
#define _T_TX_SYSLOG2(tiptr, X, Y)
#endif  /* DEBUG */

/*
 * Macro to change state and log invalid state error
 */

#define _T_TX_NEXTSTATE(event, tiptr, errstr)   \
        {       tiptr->ti_state = tiusr_statetbl[event][(tiptr)->ti_state]; \
                _T_TX_SYSLOG2((tiptr), LOG_ERR, errstr); \
        }

/*
 * External declarations
 */
extern mutex_t _ti_userlock;

/*
 * Useful shared local constants
 */

/*
 * TX_XTI_LEVEL_MAX_OPTBUF:
```

```
 *      Max option buffer requirement reserved for any XTI level options
 *      passed in an option buffer. This is intended as an upper bound.
 *      Regardless of what the providers states in OPT_size of T_info_ack,
 *      XTI level options can also be added to the option buffer and XTI
 *      test suite in particular stuffs XTI level options whether we support
 *      them or not.
 *
 * Here is the heuristic used to arrive at a value:
 *      2* [            // factor of 2 for "repeat options" type testing
 *              (sizeof(struct t_opthdr)+10*sizeof(t_scalar_t)) // XTI_DEBUG
 *             +(sizeof(struct t_opthdr)+ 2*sizeof(t_scalar_t)) // XTI_LINGER
 *             +(sizeof(struct t_opthdr)+ sizeof(t_scalar_t))   // XTI_RCVBUF
 *             +(sizeof(struct t_opthdr)+ sizeof(t_scalar_t))   // XTI_RCVLOWAT
 *             +(sizeof(struct t_opthdr)+ sizeof(t_scalar_t))   // XTI_SNDBUF
 *             +(sizeof(struct t_opthdr)+ sizeof(t_scalar_t))   // XTI_SNDLOWAT
 *         ]
 * => 2* [ 56+24+20+20+20+20 ]
 * =>
 */
#define TX_XTI_LEVEL_MAX_OPTBUF 320


/*
 * Historic information note:
 * The libnsl/nsl code implements TLI and XTI interfaces using common
 * code. Most data structures are similar in the exposed interfaces for
 * the two interfaces (<tiuser.h> and <xti.h>).
 * The common implementation C files include only <xti.h> which is the
 * superset in terms of the exposed interfaces. However the file <tiuser.h>
 * exposes (via <sys/tiuser.h>), in the past contained certain declarations
 * that are strictly internal to the implementation but were exposed through
 * their presence in the public header (<tiuser.h>).
 * Since the implmentation still needs these declarations, they follow
 * in this file and are removed from exposure through the TLI public header
 * (<tiuser.h>) which exposed them in the past.
 */

/*
 * The following are TLI/XTI user level events which cause
 * state changes.
 * NOTE: Historical namespace pollution warning.
 * Some of the event names share the namespace with structure tags
 * so there are defined inside comments here and exposed through
 * TLI and XTI headers (<tiuser.h> and <xti.h>
 */

#define T_OPEN          0
/* #define      T_BIND          1 */
/* #define      T_OPTMGMT       2 */
#define T_UNBIND        3
#define T_CLOSE         4
#define T_SNDUDATA      5
#define T_RCVUDATA      6
#define T_RCVUDERR      7
#define T_CONNECT1      8
#define T_CONNECT2      9
#define T_RCVCONNECT    10
#define T_LISTN         11
#define T_ACCEPT1       12
#define T_ACCEPT2       13
#define T_ACCEPT3       14
#define T_SND           15
#define T_RCV           16
#define T_SNDDIS1       17
#define T_SNDDIS2       18
#define T_RCVDIS1       19
```



```
#define T_RCVDIS2       20
#define T_RCVDIS3       21
#define T_SNDREL        22
#define T_RCVREL        23
#define T_PASSCON       24

#define T_NOEVENTS      25

#define T_NOSTATES      9       /* number of legal states */

extern char tiusr_statetbl[T_NOEVENTS][T_NOSTATES];

/*
 * Band definitions for data flow.
 */
#define TI_NORMAL       0
#define TI_EXPEDITED    1

/*
 * Bogus states from tiuser.h
 */
#define T_FAKE          8       /* fake state used when state   */
                                /* cannot be determined         */

/*
 * Flags for t_getname() from tiuser.h
 * Note: This routine's counterpart in XTI is substatnially modified
 * (i.e. t_getprotaddr() and does not use these flags)
 */
#define LOCALNAME       0
#define REMOTENAME      1

/*
 * Obsolete error event for t_look() in TLI, still needed for compatibility
 * to broken apps that are affected (e.g nfsd,lockd) if real error returned.
 */
#define T_ERROR 0x0020

/*
 * GENERAL UTILITY MACROS
 */
#define A_CNT(arr)      (sizeof (arr)/sizeof (arr[0]))
#define A_END(arr)      (&arr[A_CNT(arr)])
#define A_LAST(arr)     (&arr[A_CNT(arr)-1])

/*
 * Following macro compares a signed size obtained from TPI primitive
 * to unsigned size of buffer where it needs to go into passed using
 * the "struct netbuf" type.
 * Since many programs are buggy and forget to initialize "netbuf" or
 * (while unlikely!) allocated buffer can legally even be larger than
 * max signed integer, we use the following macro to do unsigned comparison
 * after verifying that signed quantity is positive.
 */
#define TLEN_GT_NLEN(tpilen, netbuflen) \
        (((tpilen) > 0) && ((unsigned int)(tpilen) > (netbuflen)))


/*
 *      N.B.:  this interface is deprecated.  Use t_strerror() instead.
 */
extern char *t_errlist[];
extern int t_nerr;

/*
 * UTILITY ROUTINES FUNCTION PROTOTYPES
```

```
 280  */

 282 extern void _t_adjust_iov(int, struct iovec *, int *);
 283 extern struct _ti_user *_t_checkfd(int, int, int);
 284 extern int _t_delete_tilink(int);
 285 extern int _t_rcv_conn_con(struct _ti_user *, struct t_call *, struct strbuf *,
 286                                                         int);
 287 extern int _t_snd_conn_req(struct _ti_user *, const struct t_call *,
 288                                                         struct strbuf *);
 289 extern int _t_aligned_copy(struct strbuf *, int, int, char *, t_scalar_t *);
 290 extern struct _ti_user *_t_create(int, struct t_info *, int, int *);
 291 extern int _t_do_ioctl(int, char *, int, int, int *);
 292 extern int _t_is_event(int, struct _ti_user *);
 293 extern int _t_is_ok(int, struct _ti_user *, t_scalar_t);
 294 extern int _t_look_locked(int, struct _ti_user *, int, int);
 295 extern int _t_register_lookevent(struct _ti_user *, caddr_t, int, caddr_t, int);
 296 extern void _t_free_looklist_head(struct _ti_user *);
 297 extern void _t_flush_lookevents(struct _ti_user *);
 298 extern int _t_acquire_ctlbuf(struct _ti_user *, struct strbuf *, int *);
 299 extern int _t_acquire_databuf(struct _ti_user *, struct strbuf *, int *);

 301 /*
 302  * Core function TLI/XTI routines function prototypes
 303  */
 304 extern int _tx_accept(int, int, const struct t_call *, int);
 305 extern char *_tx_alloc(int, int, int, int);
 306 extern int _tx_bind(int, const struct t_bind *, struct t_bind *, int);
 307 extern int _tx_close(int, int);
 308 extern int _tx_connect(int, const struct t_call *, struct t_call *, int);
 309 extern int _tx_error(const char *, int);
 310 extern int _tx_free(char *, int, int);
 311 extern int _tx_getinfo(int, struct t_info *, int);
 312 extern int _tx_getname(int, struct netbuf *, int, int);
 313 extern int _tx_getstate(int, int);
 314 extern int _tx_getprotaddr(int, struct t_bind *, struct t_bind *, int);
 315 extern int _tx_listen(int, struct t_call *, int);
 316 extern int _tx_look(int, int);
 317 extern int _tx_open(const char *, int, struct t_info *, int);
 318 extern int _tx_optmgmt(int, const struct t_optmgmt *, struct t_optmgmt *, int);
 319 extern int _tx_rcv(int, char *, unsigned, int *, int);
 320 extern int _tx_rcvconnect(int, struct t_call *, int);
 321 extern int _tx_rcvdis(int, struct t_discon *, int);
 322 extern int _tx_rcvrel(int, int);
 323 extern int _tx_rcvudata(int, struct t_unitdata *, int *, int);
 324 extern int _tx_rcvuderr(int, struct t_uderr *, int);
 325 extern int _tx_snd(int, char *, unsigned, int, int);
 326 extern int _tx_snddis(int, const struct t_call *, int);
 327 extern int _tx_sndrel(int, int);
 328 extern int _tx_sndudata(int, const struct t_unitdata *, int);
 329 extern char *_tx_strerror(int, int);
 330 extern int _tx_sync(int, int);
 331 extern int _tx_unbind(int, int);
 332 extern int _tx_unbind_locked(int, struct _ti_user *, struct strbuf *);
 333 extern int _t_expinline_queued(int, int *);
 334 extern int _t_do_postconn_sync(int, struct _ti_user *);

 336 /*
 337  * The following helper functions are used by scatter/gather functions,
 338  * which are defined only for XTI and not available in TLI. Moreover
 339  * the definition of struct t_iovec which is used below is not visible to
 340  * TLI. Hence tli_wrappers.c should not see the prototypes below.
 341  */
 342 #ifndef TLI_WRAPPERS
 343 unsigned int _t_bytecount_upto_intmax(const struct t_iovec *, unsigned int);
 344 void _t_scatter(struct strbuf *, struct t_iovec *, int);
 345 void _t_gather(char *, const struct t_iovec *, unsigned int);
```



```
 346 void _t_copy_tiov_to_iov(const struct t_iovec *, int, struct iovec *, int *);

 348 /*
 349  * The following scatter/gather and other misc. functions are defined only
 350  * for XTI and not available in TLI. Moreover the definition of struct t_iovec
 351  * which is used below is not visible to TLI. Hence tli_wrappers.c should not
 352  * see the prototypes below.
 353  */
 354 extern int _tx_rcvv(int, struct t_iovec *, unsigned int,  int *, int);
 355 extern int _tx_rcvreldata(int, struct t_discon *, int);
 356 extern int _tx_rcvvudata(int, struct t_unitdata *, struct t_iovec *,
 357     unsigned int, int *, int);
 358 extern int _tx_sndv(int, const struct t_iovec *, unsigned int, int, int);
 359 extern int _tx_sndreldata(int, struct t_discon *, int);
 360 extern int _tx_sndvudata(int, const struct t_unitdata *, struct t_iovec *,
 361     unsigned int, int);
 362 extern int _tx_sysconf(int, int);
 363 #endif /* TLI_WRAPPERS */

 365 #ifdef  __cplusplus
 366 }
```

***unchanged_portion_omitted_***

```
**********************************************************
   24650 Fri Jun 13 09:21:33 2014
new/usr/src/lib/libnsl/rpc/svc_dg.c
3910 t_look(3NSL) should never return T_ERROR
**********************************************************
```

```
   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 2014 Gary Mills
  24  * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
  25  * Use is subject to license terms.
  26  */
  27 /* Copyright (c) 1983, 1984, 1985, 1986, 1987, 1988, 1989 AT&T */
  28 /* All Rights Reserved */
  29 /*
  30  * Portions of this source code were derived from Berkeley
  31  * 4.3 BSD under license from the Regents of the University of
  32  * California.
  33  */

  35 /*
  36  * svc_dg.c, Server side for connectionless RPC.
  37  *
  38  * Does some caching in the hopes of achieving execute-at-most-once semantics.
  39  */

  41 #include "mt.h"
  42 #include "rpc_mt.h"
  43 #include <stdio.h>
  44 #include <sys/types.h>
  45 #include <sys/sysmacros.h>
  46 #include <rpc/rpc.h>
  47 #include <rpcsvc/svc_dg_priv.h>
  48 #include <errno.h>
  49 #include <syslog.h>
  50 #include <stdlib.h>
  51 #include <string.h>
  52 #include <ucred.h>
  53 #include <unistd.h>
  54 #include <sys/socket.h>
  55 #include <netinet/in.h>
  56 #include <arpa/inet.h>
  57 #ifdef RPC_CACHE_DEBUG
  58 #include <netconfig.h>
  59 #include <netdir.h>
  60 #endif
```



```
  62 #ifndef MAX
  63 #define MAX(a, b)       (((a) > (b)) ? (a) : (b))
  64 #endif

  66 static struct xp_ops *svc_dg_ops();
  67 static void cache_set();
  68 static int cache_get();

  70 #define rpc_buffer(xprt) ((xprt)->xp_p1)

  72 /*
  73  * Usage:
  74  *      xprt = svc_dg_create(sock, sendsize, recvsize);
  75  * Does other connectionless specific initializations.
  76  * Once *xprt is initialized, it is registered.
  77  * see (svc.h, xprt_register). If recvsize or sendsize are 0 suitable
  78  * system defaults are chosen.
  79  * The routines returns NULL if a problem occurred.
  80  */
  81 static const char svc_dg_str[] = "svc_dg_create: %s";
  82 static const char svc_dg_err1[] = "could not get transport information";
  83 static const char svc_dg_err2[] = " transport does not support data transfer";
  84 static const char svc_dg_err3[] =
  85                 "fd > FD_SETSIZE; Use rpc_control(RPC_SVC_USE_POLLFD,...);";
  86 static const char __no_mem_str[] = "out of memory";

  88 /* Structure used to initialize SVC_XP_AUTH(xprt).svc_ah_ops. */
  89 extern struct svc_auth_ops svc_auth_any_ops;
  90 extern int __rpc_get_ltaddr(struct netbuf *, struct netbuf *);

  92 void
  93 svc_dg_xprtfree(SVCXPRT *xprt)
  94 {
  95 /* LINTED pointer alignment */
  96         SVCXPRT_EXT             *xt = xprt ? SVCEXT(xprt) : NULL;
  97 /* LINTED pointer alignment */
  98         struct svc_dg_data      *su = xprt ? get_svc_dg_data(xprt) : NULL;

 100         if (xprt == NULL)
 101                 return;
 102         if (xprt->xp_netid)
 103                 free(xprt->xp_netid);
 104         if (xprt->xp_tp)
 105                 free(xprt->xp_tp);
 106         if (xt->parent == NULL)
 107                 if (xprt->xp_ltaddr.buf)
 108                         free(xprt->xp_ltaddr.buf);
 109         if (xprt->xp_rtaddr.buf)
 110                 free(xprt->xp_rtaddr.buf);
 111         if (su != NULL) {
 112                 XDR_DESTROY(&(su->su_xdrs));
 113                 free(su);
 114         }
 115         if (rpc_buffer(xprt))
 116                 free(rpc_buffer(xprt));
 117         svc_xprt_free(xprt);
 118 }
______unchanged_portion_omitted_

 359 static bool_t
 360 svc_dg_recv(SVCXPRT *xprt, struct rpc_msg *msg)
 361 {
 362 /* LINTED pointer alignment */
 363         struct svc_dg_data *su = get_svc_dg_data(xprt);
 364         XDR *xdrs = &(su->su_xdrs);
 365         struct t_unitdata *tu_data = &(su->su_tudata);
```

```
366         int moreflag;
367         struct netbuf *nbufp;
368         struct netconfig *nconf;

370         /* XXX: tudata should have been made a part of the server handle */
371         if (tu_data->addr.maxlen == 0)
372                 tu_data->addr = xprt->xp_rtaddr;
373 again:
374         tu_data->addr.len = 0;
375         tu_data->opt.len  = 0;
376         tu_data->udata.len  = 0;

378         moreflag = 0;
379         if (t_rcvudata(xprt->xp_fd, tu_data, &moreflag) == -1) {
380 #ifdef RPC_DEBUG
381                 syslog(LOG_ERR, "svc_dg_recv: t_rcvudata t_errno=%d errno=%d\n",
382                     t_errno, errno);
383 #endif
384                 if (t_errno == TLOOK) {
385                         int lookres;

387                         lookres = t_look(xprt->xp_fd);
388                         if ((lookres == T_UDERR) &&
387                         if ((lookres & T_UDERR) &&
389                             (t_rcvuderr(xprt->xp_fd,
390                                 (struct t_uderr *)0) < 0)) {
391                                 /*EMPTY*/
392 #ifdef RPC_DEBUG
393                                 syslog(LOG_ERR,
394                                 "svc_dg_recv: t_rcvuderr t_errno = %d\n",
395                                         t_errno);
396 #endif
397                         }
398                         if (lookres == T_DATA)
397                         if (lookres & T_DATA)
399                                 goto again;
400                 } else if ((errno == EINTR) && (t_errno == TSYSERR))
401                         goto again;
402                 else {
403                         return (FALSE);
404                 }
405         }

407         if ((moreflag) ||
408             (tu_data->udata.len < 4 * (uint_t)sizeof (uint32_t))) {
409                 /*
410                  * If moreflag is set, drop that data packet. Something wrong
411                  */
412                 return (FALSE);
413         }
414         su->optbuf = tu_data->opt;
415         xprt->xp_rtaddr.len = tu_data->addr.len;
416         xdrs->x_op = XDR_DECODE;
417         XDR_SETPOS(xdrs, 0);
418         if (!xdr_callmsg(xdrs, msg))
419                 return (FALSE);
420         su->su_xid = msg->rm_xid;
421         if (su->su_cache != NULL) {
422                 char *reply;
423                 uint32_t replylen;

425                 if (cache_get(xprt, msg, &reply, &replylen)) {
426                         /* tu_data.addr is already set */
427                         tu_data->udata.buf = reply;
428                         tu_data->udata.len = (uint_t)replylen;
429                         extract_cred(&tu_data->opt, &tu_data->opt);
```



```
430                         set_src_addr(xprt, &tu_data->opt);
431                         (void) t_sndudata(xprt->xp_fd, tu_data);
432                         tu_data->udata.buf = (char *)rpc_buffer(xprt);
433                         tu_data->opt.buf = (char *)su->opts;
434                         return (FALSE);
435                 }
436         }

438         /*
439          * get local ip address
440          */

442         if ((nconf = getnetconfigent(xprt->xp_netid)) != NULL) {
443                 if (strcmp(nconf->nc_protofmly, NC_INET) == 0 ||
444                     strcmp(nconf->nc_protofmly, NC_INET6) == 0) {
445                         if (nconf->nc_semantics == NC_TPI_CLTS) {
446                                 /* LINTED pointer cast */
447                                 nbufp = (struct netbuf *)(xprt->xp_p2);
448                                 if (__rpc_get_ltaddr(nbufp,
449                                     &xprt->xp_ltaddr) < 0) {
450                                         if (strcmp(nconf->nc_protofmly,
451                                             NC_INET) == 0) {
452                                                 syslog(LOG_ERR,
453                                                     "svc_dg_recv: ip(udp), "
454                                                     "t_errno=%d, errno=%d",
455                                                     t_errno, errno);
456                                         }
457                                         if (strcmp(nconf->nc_protofmly,
458                                             NC_INET6) == 0) {
459                                                 syslog(LOG_ERR,
460                                                     "svc_dg_recv: ip (udp6), "
461                                                     "t_errno=%d, errno=%d",
462                                                     t_errno, errno);
463                                         }
464                                         freenetconfigent(nconf);
465                                         return (FALSE);
466                                 }
467                         }
468                 }
469                 freenetconfigent(nconf);
470         }
471         return (TRUE);
472 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   12800 Fri Jun 13 09:21:33 2014
new/usr/src/uts/common/fs/nfs/nfs_dump.c
3910 t_look(3NSL) should never return T_ERROR
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2014 Gary Mills
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Dump memory to NFS swap file after a panic.
 * We have no timeouts, context switches, etc.
 */

#include <rpc/types.h>
#include <sys/param.h>
#include <sys/errno.h>
#include <sys/vnode.h>
#include <sys/bootconf.h>
#include <nfs/nfs.h>
#include <rpc/auth.h>
#include <rpc/xdr.h>
#include <rpc/rpc_msg.h>
#include <rpc/clnt.h>
#include <netinet/in.h>
#include <sys/tiuser.h>
#include <nfs/nfs_clnt.h>
#include <sys/t_kuser.h>
#include <sys/file.h>
#include <sys/netconfig.h>
#include <sys/utsname.h>
#include <sys/sysmacros.h>
#include <sys/thread.h>
#include <sys/cred.h>
#include <sys/strsubr.h>
#include <nfs/rnode.h>
#include <sys/varargs.h>
#include <sys/cmn_err.h>
#include <sys/systm.h>
#include <sys/dumphdr.h>
#include <sys/debug.h>
#include <sys/sunddi.h>

#define TIMEOUT         (2 * hz)
```



```
#define RETRIES         (5)
#define HDR_SIZE        (256)

static struct knetconfig        nfsdump_cf;
static struct netbuf            nfsdump_addr;
static fhandle_t                nfsdump_fhandle2;
static nfs_fh3                  nfsdump_fhandle3;
static int                      nfsdump_maxcount;
static rpcvers_t                nfsdump_version;

/*
 * nonzero dumplog enables nd_log messages
 */
static int      dumplog = 0;

static int      nd_init(vnode_t *, TIUSER **);
static int      nd_poll(TIUSER *, int, int *);
static int      nd_send_data(TIUSER *, caddr_t, int, XDR *, uint32_t *);
static int      nd_get_reply(TIUSER *, XDR *, uint32_t, int *);
static int      nd_auth_marshall(XDR *);

static void nd_log(const char *, ...) __KPRINTFLIKE(1);

/*PRINTFLIKE1*/
static void
nd_log(const char *fmt, ...)
{
        if (dumplog) {
                va_list adx;

                va_start(adx, fmt);
                vprintf(fmt, adx);
                va_end(adx);
        }
}
```

*unchanged_portion_omitted_*

```
static int
nd_get_reply(TIUSER *tiptr, XDR *xdrp, uint32_t call_xid, int *badmsg)
{
        static struct rpc_msg           reply_msg;
        static struct rpc_err           rpc_err;
        static struct nfsattrstat       na;
        static struct WRITE3res         wres;
        static struct t_kunitdata       rudata;
        int                             uderr;
        int                             type;
        int                             error;

        *badmsg = 0;

        rudata.addr.maxlen = 0;
        rudata.opt.maxlen = 0;
        rudata.udata.udata_mp = (mblk_t *)NULL;

        nd_log("nfs_dump: calling t_krcvudata\n");

        if (error = t_krcvudata(tiptr, &rudata, &type, &uderr)) {
                if (error == EBADMSG) {
                        cmn_err(CE_WARN, "\tnfs_dump:  received EBADMSG");
                        *badmsg = 1;
                        return (0);
                }
                nfs_perror(error, "\nnfs_dump: t_krcvudata failed: %m\n");
                return (EIO);
        }
```

```
381         if (type != T_DATA) {
382                 cmn_err(CE_WARN, "\tnfs_dump:  received type %d", type);
383                 *badmsg = 1;
384                 return (0);
385         }
386         if (!rudata.udata.udata_mp) {
387                 cmn_err(CE_WARN, "\tnfs_dump: null receive");
388                 *badmsg = 1;
389                 return (0);
390         }

392         /*
393          * Decode results.
394          */
395         xdrmblk_init(xdrp, rudata.udata.udata_mp, XDR_DECODE, 0);

397         reply_msg.acpted_rply.ar_verf = _null_auth;
398         switch (nfsdump_version) {
399         case NFS_VERSION:
400                 reply_msg.acpted_rply.ar_results.where = (caddr_t)&na;
401                 reply_msg.acpted_rply.ar_results.proc = xdr_attrstat;
402                 break;
403         case NFS_V3:
404                 reply_msg.acpted_rply.ar_results.where = (caddr_t)&wres;
405                 reply_msg.acpted_rply.ar_results.proc = xdr_WRITE3res;
406                 break;
407         default:
408                 return (EIO);
409         }

411         if (!xdr_replymsg(xdrp, &reply_msg)) {
412                 cmn_err(CE_WARN, "\tnfs_dump: xdr_replymsg failed");
413                 return (EIO);
414         }

416         if (reply_msg.rm_xid != call_xid) {
417                 *badmsg = 1;
418                 return (0);
419         }

421         _seterr_reply(&reply_msg, &rpc_err);

423         if (rpc_err.re_status != RPC_SUCCESS) {
424                 cmn_err(CE_WARN, "\tnfs_dump: RPC error %d (%s)",
425                     rpc_err.re_status, clnt_sperrno(rpc_err.re_status));
426                 return (EIO);
427         }

429         switch (nfsdump_version) {
430         case NFS_VERSION:
431                 if (na.ns_status) {
432                         cmn_err(CE_WARN, "\tnfs_dump: status %d", na.ns_status);
433                         return (EIO);
434                 }
435                 break;
436         case NFS_V3:
437                 if (wres.status != NFS3_OK) {
438                         cmn_err(CE_WARN, "\tnfs_dump: status %d", wres.status);
439                         return (EIO);
440                 }
441                 break;
442         default:
443                 return (EIO);
444         }

446         if (reply_msg.acpted_rply.ar_verf.oa_base != NULL) {
```



```
447                 /* free auth handle */
448                 xdrp->x_op = XDR_FREE;
449                 (void) xdr_opaque_auth(xdrp, &(reply_msg.acpted_rply.ar_verf));
450         }

452         freemsg(rudata.udata.udata_mp);

454         return (0);
455 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    7293 Fri Jun 13 09:21:33 2014
new/usr/src/uts/common/ktli/t_krcvudat.c
3910 t_look(3NSL) should never return T_ERROR
**********************************************************
```

```
  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 2014 Gary Mills
 23  * Copyright 2008 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  */

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

 40 /*
 41  * Kernel TLI-like function to read a datagram off of a
 42  * transport endpoints stream head.
 43  *
 44  * Returns:
 45  *      0       On success or positive error code.
 46  *              On sucess, type is set to:
 47  *      T_DATA          If normal data has been received
 48  *      T_UDERR         If an error indication has been received,
 49  *                      in which case uderr contains the unitdata
 50  *                      error number.
 52  *      T_ERROR
 51  */

 53 #include <sys/param.h>
 54 #include <sys/types.h>
 55 #include <sys/user.h>
 56 #include <sys/file.h>
 57 #include <sys/errno.h>
 58 #include <sys/stream.h>
```



```
 59 #include <sys/strsubr.h>
 60 #include <sys/vnode.h>
 61 #include <sys/ioctl.h>
 62 #include <sys/stropts.h>
 63 #include <sys/tihdr.h>
 64 #include <sys/timod.h>
 65 #include <sys/tiuser.h>
 66 #include <sys/t_kuser.h>
 67 #include <sys/sysmacros.h>
 68 #include <sys/strsun.h>


 71 int
 72 t_krcvudata(TIUSER *tiptr, struct t_kunitdata *unitdata, int *type, int *uderr)
 73 {
 74         int                     len;
 75         size_t                  hdrsz;
 76         union T_primitives      *pptr;
 77         struct file             *fp;
 78         mblk_t                  *bp;
 79         mblk_t                  *nbp;
 80         mblk_t                  *mp;
 81         mblk_t                  *tmp;
 82         int                     error;
 83         int                     flag;

 85         fp = tiptr->fp;

 87         if (type == NULL || uderr == NULL)
 88                 return (EINVAL);

 90         error = 0;
 91         unitdata->udata.buf = NULL;

 93         if (unitdata->udata.udata_mp) {
 94                 KTLILOG(2, "t_krcvudata: freeing existing message block\n", 0);
 95                 freemsg(unitdata->udata.udata_mp);
 96                 unitdata->udata.udata_mp = NULL;
 97         }

 99         /*
100          * XXX  Grabbing a mutex to do an atomic operation seems pointless
101          */
102         mutex_enter(&fp->f_tlock);
103         flag = fp->f_flag;
104         mutex_exit(&fp->f_tlock);

106         if ((error = tli_recv(tiptr, &bp, flag)) != 0)
107                 return (error);

109         /*
110          * Got something
111          */
112         switch (bp->b_datap->db_type) {
113         case M_PROTO:
114                 /* LINTED pointer alignment */
115                 pptr = (union T_primitives *)bp->b_rptr;
116                 switch (pptr->type) {
117                 case T_UNITDATA_IND:
118                         KTLILOG(2, "t_krcvudata: Got T_UNITDATA_IND\n", 0);
119                         hdrsz = MBLKL(bp);

121                         /*
122                          * check everything for consistency
123                          */
124                         if (hdrsz < TUNITDATAINDSZ ||
```

```
125                             hdrsz < (pptr->unitdata_ind.OPT_length +
126                             pptr->unitdata_ind.OPT_offset) ||
127                             hdrsz < (pptr->unitdata_ind.SRC_length +
128                             pptr->unitdata_ind.SRC_offset)) {
129                                 error = EPROTO;
130                                 freemsg(bp);
131                                 break;
132                         }

134                         /*
135                          * okay, so now we copy them
136                          */
137                         len = MIN(pptr->unitdata_ind.SRC_length,
138                             unitdata->addr.maxlen);
139                         bcopy(bp->b_rptr + pptr->unitdata_ind.SRC_offset,
140                             unitdata->addr.buf, len);
141                         unitdata->addr.len = len;

143                         len = MIN(pptr->unitdata_ind.OPT_length,
144                             unitdata->opt.maxlen);
145                         bcopy(bp->b_rptr + pptr->unitdata_ind.OPT_offset,
146                             unitdata->opt.buf, len);
147                         unitdata->opt.len = len;

149                         bp->b_rptr += hdrsz;

151                         /*
152                          * we assume that the client knows how to deal
153                          * with a set of linked mblks, so all we do is
154                          * make a pass and remove any that are zero
155                          * length.
156                          */
157                         nbp = NULL;
158                         mp = bp;
159                         while (mp) {
160                                 if (bp->b_wptr == bp->b_rptr) {
161                                         KTLILOG(2,
162                                             "t_krcvudata: zero length block\n",
163                                             0);
164                                         tmp = mp->b_cont;
165                                         if (nbp)
166                                                 nbp->b_cont = tmp;
167                                         else
168                                                 bp = tmp;

170                                         freeb(mp);
171                                         mp = tmp;
172                                 } else {
173                                         nbp = mp;
174                                         mp = mp->b_cont;
175                                 }
176                         }
177 #ifdef KTLIDEBUG
178 {
179         mblk_t *tp;

181         tp = bp;
182         while (tp) {
183                 struct datab *dbp = tp->b_datap;
184                 frtn_t *frp = dbp->db_frtnp;

186                 KTLILOG(2, "t_krcvudata: bp %x, ", tp);
187                 KTLILOG(2, "db_size %x, ", dbp->db_lim - dbp->db_base);
188                 KTLILOG(2, "db_ref %x", dbp->db_ref);

190                 if (frp != NULL)
```



```
191                         KTLILOG(2, ", func: %x", frp->free_func);
192                         KTLILOG(2, ", arg %x\n", frp->free_arg);
193                 } else
194                         KTLILOG(2, "\n", 0);
195                 tp = tp->b_cont;
196         }
197 }
198 #endif /* KTLIDEBUG */
199                         /*
200                          * now just point the users mblk
201                          * pointer to what we received.
202                          */
203                         if (bp == NULL) {
204                                 KTLILOG(2, "t_krcvudata: No data\n", 0);
205                                 error = EPROTO;
206                                 break;
207                         }
208                         if (bp->b_wptr != bp->b_rptr) {
209                             if (!IS_P2ALIGNED(bp->b_rptr, sizeof (uint32_t)))
210                                 if (!pullupmsg(bp, MBLKL(bp))) {
211                                         KTLILOG(1,
212                                         "t_krcvudata:  pullupmsg failed\n", 0);
213                                         error = EIO;
214                                         freemsg(bp);
215                                         break;
216                                 }
217                                 unitdata->udata.buf = (char *)bp->b_rptr;
218                                 unitdata->udata.len = (uint_t)MBLKL(bp);

220                                 KTLILOG(2, "t_krcvudata: got %d bytes\n",
221                                     unitdata->udata.len);
222                                 unitdata->udata.udata_mp = bp;
223                         } else {
224                                 KTLILOG(2,
225                                     "t_krcvudata: 0 length data message\n", 0);
226                                 freemsg(bp);
227                                 unitdata->udata.len = 0;
228                         }
229                         *type = T_DATA;
230                         break;

232                 case T_UDERROR_IND:
233                         KTLILOG(2, "t_krcvudata: Got T_UDERROR_IND\n", 0);
234                         hdrsz = MBLKL(bp);

236                         /*
237                          * check everything for consistency
238                          */
239                         if (hdrsz < TUDERRORINDSZ ||
240                             hdrsz < (pptr->uderror_ind.OPT_length +
241                             pptr->uderror_ind.OPT_offset) ||
242                             hdrsz < (pptr->uderror_ind.DEST_length +
243                             pptr->uderror_ind.DEST_offset)) {
244                                 error = EPROTO;
245                                 freemsg(bp);
246                                 break;
247                         }

249                         if (pptr->uderror_ind.DEST_length >
250                             (int)unitdata->addr.maxlen ||
251                             pptr->uderror_ind.OPT_length >
252                             (int)unitdata->opt.maxlen) {
253                                 error = EMSGSIZE;
254                                 freemsg(bp);
255                                 break;
256                         }
```

```
                        /*
                         * okay, so now we copy them
                         */
                        bcopy(bp->b_rptr + pptr->uderror_ind.DEST_offset,
                            unitdata->addr.buf,
                            (size_t)pptr->uderror_ind.DEST_length);
                        unitdata->addr.len = pptr->uderror_ind.DEST_length;

                        bcopy(bp->b_rptr + pptr->uderror_ind.OPT_offset,
                            unitdata->opt.buf,
                            (size_t)pptr->uderror_ind.OPT_length);
                        unitdata->opt.len = pptr->uderror_ind.OPT_length;

                        *uderr = pptr->uderror_ind.ERROR_type;

                        unitdata->udata.buf = NULL;
                        unitdata->udata.udata_mp = NULL;
                        unitdata->udata.len = 0;

                        freemsg(bp);

                        *type = T_UDERR;
                        break;

                default:
                        KTLILOG(1,
                            "t_krcvudata: Unknown transport primitive %d\n",
                            pptr->type);
                        error = EPROTO;
                        freemsg(bp);
                        break;
                }
                break;

        case M_FLUSH:
                KTLILOG(1, "t_krcvudata: tli_recv returned M_FLUSH\n", 0);
                freemsg(bp);
                error = EBADMSG;
                *type = T_ERROR;
                break;

        default:
                KTLILOG(1, "t_krcvudata: unknown message type %x\n",
                    bp->b_datap->db_type);
                freemsg(bp);
                error = EBADMSG;
                *type = T_ERROR;
                break;
        }

        return (error);
}
```

**_____unchanged_portion_omitted_**

```
**********************************************************
    9512 Fri Jun 13 09:21:33 2014
new/usr/src/uts/common/rpc/sec/authdesubr.c
3910 t_look(3NSL) should never return T_ERROR
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2014 Gary Mills
 * Copyright 2001 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

/*
 * Portions of this source code were derived from Berkeley 4.3 BSD
 * under license from the Regents of the University of California.
 */

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

/*
 * Miscellaneous support routines for kernel implentation of AUTH_DES
 */

/*
 *  rtime - get time from remote machine
 *
 *  sets time, obtaining value from host
 *  on the udp/time socket.  Since timeserver returns
 *  with time of day in seconds since Jan 1, 1900,  must
 *  subtract 86400(365*70 + 17) to get time
 *  since Jan 1, 1970, which is what get/settimeofday
 *  uses.
 */
#include <sys/param.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/systm.h>
#include <sys/errno.h>
#include <sys/proc.h>
#include <sys/user.h>
#include <sys/socket.h>
#include <sys/sysmacros.h>
#include <netinet/in.h>
```



```
#include <rpc/rpc.h>
#include <sys/stream.h>
#include <sys/strsubr.h>
#include <sys/cred.h>
#include <sys/utsname.h>
#include <sys/vnode.h>
#include <sys/file.h>
#include <sys/uio.h>
#include <sys/systeminfo.h>
#include <rpc/rpcb_prot.h>
#include <sys/cmn_err.h>

#define TOFFSET ((uint32_t)86400 * (365 * 70 + (70 / 4)))
#define WRITTEN ((uint32_t)86400 * (365 * 86 + (86 / 4)))

#define NC_INET "inet"          /* XXX */

int
rtime(struct knetconfig *synconfig, struct netbuf *addrp, int calltype,
        struct timeval *timep, struct timeval *wait)
{
        int                     error;
        int                     timo;
        time_t                  thetime;
        int32_t                 srvtime;
        uint32_t                dummy;
        struct t_kunitdata      *unitdata;
        struct t_call           *server;
        TIUSER                  *tiptr;
        int                     type;
        int                     uderr;
        int                     i;
        int                     retries;
        mblk_t                  *mp;
        mblk_t                  *mp2;

        retries = 5;
        if (calltype == 0) {
again:
                RPCLOG0(8, "rtime: using old method\n");
                if ((error = t_kopen(NULL, synconfig->knc_rdev,
                    FREAD|FWRITE, &tiptr, CRED())) != 0) {
                        RPCLOG(1, "rtime: t_kopen %d\n", error);
                        return (-1);
                }

                if ((error = t_kbind(tiptr, NULL, NULL)) != 0) {
                        (void) t_kclose(tiptr, 1);
                        RPCLOG(1, "rtime: t_kbind %d\n", error);
                        return (-1);
                }

                if (synconfig->knc_semantics == NC_TPI_CLTS) {
                        if ((error = t_kalloc(tiptr, T_UNITDATA,
                            T_UDATA|T_ADDR, (char **)&unitdata)) != 0) {
                    if ((error = t_kalloc(tiptr, T_UNITDATA, T_UDATA|T_ADDR,
                        (char **)&unitdata)) != 0) {
                                RPCLOG(1, "rtime: t_kalloc %d\n", error);
                                (void) t_kclose(tiptr, 1);
                                return (-1);
                        }

                        unitdata->addr.len = addrp->len;
                        bcopy(addrp->buf, unitdata->addr.buf,
                            unitdata->addr.len);
                    bcopy(addrp->buf, unitdata->addr.buf, unitdata->addr.len);
```

```
124                         dummy = 0;
125                         unitdata->udata.buf = (caddr_t)&dummy;
126                         unitdata->udata.len = sizeof (dummy);

128                         if ((error = t_ksndudata(tiptr, unitdata, NULL)) !=
129                             0) {
128                     if ((error = t_ksndudata(tiptr, unitdata, NULL)) != 0) {
130                                 RPCLOG(1, "rtime: t_ksndudata %d\n", error);
131                                 (void) t_kfree(tiptr, (char *)unitdata,
132                                     T_UNITDATA);
130                             (void) t_kfree(tiptr, (char *)unitdata, T_UNITDATA);
133                                 (void) t_kclose(tiptr, 1);
134                                 return (-1);
135                         }

137                         timo = TIMEVAL_TO_TICK(wait);

139                         RPCLOG(8, "rtime: timo %x\n", timo);
140                         if ((error = t_kspoll(tiptr, timo, READWAIT,
141                             &type)) != 0) {
138                     if ((error = t_kspoll(tiptr, timo, READWAIT, &type)) != 0) {
142                                 RPCLOG(1, "rtime: t_kspoll %d\n", error);
143                                 (void) t_kfree(tiptr, (char *)unitdata,
144                                     T_UNITDATA);
140                             (void) t_kfree(tiptr, (char *)unitdata, T_UNITDATA);
145                                 (void) t_kclose(tiptr, 1);
146                                 return (-1);
147                         }

149                         if (type == 0) {
150                                 RPCLOG0(1, "rtime: t_kspoll timed out\n");
151                                 (void) t_kfree(tiptr, (char *)unitdata,
152                                     T_UNITDATA);
147                             (void) t_kfree(tiptr, (char *)unitdata, T_UNITDATA);
153                                 (void) t_kclose(tiptr, 1);
154                                 return (-1);
155                         }

157                         error = t_krcvudata(tiptr, unitdata, &type, &uderr);
158                         if (error != 0) {
159                                 RPCLOG(1, "rtime: t_krcvudata %d\n", error);
160                                 (void) t_kfree(tiptr, (char *)unitdata,
161                                     T_UNITDATA);
155                             (void) t_kfree(tiptr, (char *)unitdata, T_UNITDATA);
162                                 (void) t_kclose(tiptr, 1);
163                                 if (error == EBADMSG && retries-- > 0)
164                                         goto again;
165                                 return (-1);
166                         }

168                         if (type == T_UDERR) {
169                                 if (bcmp(addrp->buf, unitdata->addr.buf,
170                                     unitdata->addr.len) != 0) {
171                                 /*
172                                  * Response comes from some other
173                                  * destination:
174                                  * ignore it since it's not related to the
175                                  * request we just sent out.
176                                  */
177                                         (void) t_kfree(tiptr, (char *)unitdata,
178                                             T_UNITDATA);
179                                         (void) t_kclose(tiptr, 1);
180                                         goto again;
181                                 }
182                         }
```



```
184                         if (type != T_DATA) {
185                                 RPCLOG(1,
186                                     "rtime: t_krcvudata returned type %d\n",
177                                 RPCLOG(1, "rtime: t_krcvudata returned type %d\n",
187                                     type);
188                                 (void) t_kfree(tiptr, (char *)unitdata,
189                                     T_UNITDATA);
179                                 (void) t_kfree(tiptr, (char *)unitdata, T_UNITDATA);
190                                 (void) t_kclose(tiptr, 1);
191                                 if (retries-- == 0)
192                                         return (-1);
193                                 goto again;
194                         }

196                         if (unitdata->udata.len < sizeof (uint32_t)) {
197                                 RPCLOG(1, "rtime: bad rcvd length %d\n",
198                                     unitdata->udata.len);
199                                 (void) t_kfree(tiptr, (char *)unitdata,
200                                     T_UNITDATA);
189                                 (void) t_kfree(tiptr, (char *)unitdata, T_UNITDATA);
201                                 (void) t_kclose(tiptr, 1);
202                                 if (retries-- == 0)
203                                         return (-1);
204                                 goto again;
205                         }

207                         thetime = (time_t)ntohl(
208                             /* LINTED pointer alignment */
209                             *(uint32_t *)unitdata->udata.buf);
197                     thetime = (time_t)ntohl(*(uint32_t *)unitdata->udata.buf);
210                         (void) t_kfree(tiptr, (char *)unitdata, T_UNITDATA);

212                 } else {

214                         if ((error = t_kalloc(tiptr, T_CALL, T_ADDR,
215                             (char **)&server)) != 0) {
216                                 RPCLOG(1, "rtime: t_kalloc %d\n", error);
217                                 (void) t_kclose(tiptr, 1);
218                                 return (-1);
219                         }

221                         server->addr.len = addrp->len;
222                         bcopy(addrp->buf, server->addr.buf, server->addr.len);

224                         if ((error = t_kconnect(tiptr, server, NULL)) != 0) {
225                                 RPCLOG(1, "rtime: t_kconnect %d\n", error);
226                                 (void) t_kfree(tiptr, (char *)server, T_CALL);
227                                 (void) t_kclose(tiptr, 1);
228                                 return (-1);
229                         }
230                         (void) t_kfree(tiptr, (char *)server, T_CALL);

232                         timo = TIMEVAL_TO_TICK(wait);

234                         RPCLOG(8, "rtime: timo %x\n", timo);

236                         i = 0;
237                         dummy = 0;

239                         /* now read up to 4 bytes from the TIME server */
240                         while (i < sizeof (dummy)) {

242                                 error = t_kspoll(tiptr, timo, READWAIT, &type);
243                                 if (error != 0) {
244                                         RPCLOG(1, "rtime: t_kspoll %d\n",
```

```
245                                     error);
232                             RPCLOG(1, "rtime: t_kspoll %d\n", error);
246                                     (void) t_kclose(tiptr, 1);
247                                     return (-1);
248                             }

250                             if (type == 0) {
251                                     RPCLOG0(1,
252                                         "rtime: t_kspoll timed out\n");
238                             RPCLOG0(1, "rtime: t_kspoll timed out\n");
253                                     (void) t_kclose(tiptr, 1);
254                                     return (-1);
255                             }

257                             error = tli_recv(tiptr, &mp,
258                                 tiptr->fp->f_flag);
243                     error = tli_recv(tiptr, &mp, tiptr->fp->f_flag);
259                             if (error != 0) {
260                                     RPCLOG(1, "rtime: tli_recv %d\n",
261                                         error);
245                             RPCLOG(1, "rtime: tli_recv %d\n", error);
262                                     (void) t_kclose(tiptr, 1);
263                                     return (-1);
264                             }

266                             if (mp->b_datap->db_type != M_DATA) {
267                                     RPCLOG(1, "rtime: wrong msg type %d\n",
268                                         mp->b_datap->db_type);
269                                     RPCLOG(1,
270                                         "rtime: wrong msg type: read %d"
253                             RPCLOG(1, "rtime: wrong msg type: read %d"
271                                         " bytes\n", i);
272                                     (void) t_kclose(tiptr, 1);
273                                     freemsg(mp);
274                                     return (-1);
275                             }

277                             mp2 = mp;

279                             /*
280                              * The outer loop iterates until we reach the
281                              * end of the mblk chain.
263                              * The outer loop iterates until we reach the end of
264                              * the mblk chain.
282                              */
283                             while (mp2 != NULL) {

285                                     /*
286                                      * The inner loop iterates until
287                                      * we've gotten 4 bytes or until
288                                      * the mblk is exhausted.
269                                      * The inner loop iterates until we've gotten
270                                      * 4 bytes or until the mblk is exhausted.
289                                      */
290                                     while (i < sizeof (dummy) &&
291                                         mp2->b_rptr < mp2->b_wptr) {

293                                             i++;

295                                     /*
296                                      * We avoid big-endian/little-endian
297                                      * issues by serializing the result
298                                      * one byte at a time.
299                                      */
300                                             dummy <<= 8;
301                                             dummy += ((*mp2->b_rptr) &
```



```
302                                                     0xFF);
283                                             dummy += ((*mp2->b_rptr) & 0xFF);

304                                             mp2->b_rptr++;
305                                     }

307                                     mp2 = mp2->b_cont;
308                                 }

310                             freemsg(mp);
311                             }

313                             thetime = (time_t)dummy;
314                     }

316                     (void) t_kclose(tiptr, 1);

318             } else {
319                     CLIENT                  *client;
320                     struct timeval          timout;

322                     RPCLOG0(8, "rtime: using new method\n");

324     new_again:
325                     /*
326                      *      We talk to rpcbind.
327                      */
328                     error = clnt_tli_kcreate(synconfig, addrp, (rpcprog_t)RPCBPROG,
329                         (rpcvers_t)RPCBVERS, 0, retries, CRED(), &client);

331                     if (error != 0) {
332                             RPCLOG(1,
333                                 "rtime: clnt_tli_kcreate returned %d\n", error);
334                             return (-1);
335                     }
336                     timout.tv_sec = 60;
337                     timout.tv_usec = 0;
338                     error = clnt_call(client, RPCBPROC_GETTIME, (xdrproc_t)xdr_void,
339                         NULL, (xdrproc_t)xdr_u_int,
340                         (caddr_t)&srvtime, timout);
341                     thetime = srvtime;
342                     auth_destroy(client->cl_auth);
343                     clnt_destroy(client);
344                     if (error == RPC_UDERROR) {
345                             if (retries-- > 0)
346                                     goto new_again;
347                     }
348                     if (error != RPC_SUCCESS) {
349                             RPCLOG(1, "rtime: time sync clnt_call returned %d\n",
350                                 error);
351                             error = EIO;
352                             return (-1);
353                     }
354             }

356             if (calltype != 0)
357                     thetime += TOFFSET;

359             RPCLOG(8, "rtime: thetime = %lx\n", thetime);

361             if (thetime < WRITTEN) {
362                     RPCLOG(1, "rtime: time returned is too far in past %lx",
363                         thetime);
364                     RPCLOG(1, "rtime: WRITTEN %x", WRITTEN);
365                     return (-1);
366             }
```

```
 367         thetime -= TOFFSET;

 369         timep->tv_sec = thetime;
 370         RPCLOG(8, "rtime: timep->tv_sec = %lx\n", timep->tv_sec);
 371         RPCLOG(8, "rtime: machine time  = %lx\n", gethrestime_sec());
 372         timep->tv_usec = 0;
 373         RPCLOG0(8, "rtime: returning success\n");
 374         return (0);
 375 }
```

_____***unchanged_portion_omitted_***

**********
***6467 Fri Jun 13 09:21:33 2014***
***new/usr/src/uts/common/sys/tiuser.h***
***3910 t_look(3NSL) should never return T_ERROR***
**********

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T
 * All Rights Reserved
 *
 */

/*
 * Copyright 2002 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 * Copyright 2014 Gary Mills
 */

#ifndef _SYS_TIUSER_H
#define _SYS_TIUSER_H

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

#include <sys/types.h>
/*
 * The following include file has declarations needed by both the kernel
 * level transport providers and the user level library.
 */
#include <sys/tpicommon.h>

#ifdef  __cplusplus
extern "C" {
#endif


/*
 * The following are the events returned by t_look
 */
#define T_LISTEN        0x0001  /* connection indication received       */
#define T_CONNECT       0x0002  /* connect confirmation received        */
#define T_DATA          0x0004  /* normal data received                 */
#define T_EXDATA        0x0008  /* expedited data received              */
#define T_DISCONNECT    0x0010  /* disconnect received                  */
#define T_ERROR         0x0020  /* fatal error occurred                 */
#define T_UDERR         0x0040  /* data gram error indication           */
#define T_ORDREL        0x0080  /* orderly release indication           */
```



```
#define T_EVENTS        0x00ff  /* event mask                           */

/*
 * Flags for data primitives.
 */
#define T_MORE          0x001   /* more data            */
#define T_EXPEDITED     0x002   /* expedited data       */


/*
 * protocol specific service limits
 */

struct t_info {
        t_scalar_t addr;        /* size of protocol address             */
        t_scalar_t options;     /* size of protocol options             */
        t_scalar_t tsdu;        /* size of max transport service data unit */
        t_scalar_t etsdu;       /* size of max expedited tsdu           */
        t_scalar_t connect;     /* max data for connection primitives   */
        t_scalar_t discon;      /* max data for disconnect primitives   */
        t_scalar_t servtype;    /* provider service type                */
};
______unchanged_portion_omitted_
```