****
231231 Wed Jul 3 14:44:06 2013
new/usr/src/uts/common/io/comstar/stmf/stmf.c
3866 panic in idm module
3867 stmfCreateLu failed: GUID_IN_USE
3868 iscsi target not accepting any new connections
Reviewed by: Sebastien Roy <sebastien.roy@delphix.com>
Reviewed by: Jeremy Jones <jeremy@delphix.com>
Reviewed by: Eric Diven <eric.diven@delphix.com>
****
_____***unchanged_portion_omitted_***

```
6102 stmf_status_t
6103 stmf_scsilib_uniq_lu_id2(uint32_t company_id, uint32_t host_id,
6104     scsi_devid_desc_t *lu_id)
6105 {
6106         uint8_t *p;
6107         struct timeval32 timestamp32;
6108         uint32_t *t = (uint32_t *)&timestamp32;
6109         struct ether_addr mac;
6110         uint8_t *e = (uint8_t *)&mac;
6111         int hid = (int)host_id;
6112         uint16_t gen_number;

6114         if (company_id == COMPANY_ID_NONE)
6115                 company_id = COMPANY_ID_SUN;

6117         if (lu_id->ident_length != 0x10)
6118                 return (STMF_INVALID_ARG);

6120         p = (uint8_t *)lu_id;

6122         gen_number = atomic_add_16_nv(&stmf_lu_id_gen_number, 1);
6121         atomic_add_16(&stmf_lu_id_gen_number, 1);

6124         p[0] = 0xf1; p[1] = 3; p[2] = 0; p[3] = 0x10;
6125         p[4] = ((company_id >> 20) & 0xf) | 0x60;
6126         p[5] = (company_id >> 12) & 0xff;
6127         p[6] = (company_id >> 4) & 0xff;
6128         p[7] = (company_id << 4) & 0xf0;
6129         if (hid == 0 && !localetheraddr((struct ether_addr *)NULL, &mac)) {
6130                 hid = BE_32((int)zone_get_hostid(NULL));
6131         }
6132         if (hid != 0) {
6133                 e[0] = (hid >> 24) & 0xff;
6134                 e[1] = (hid >> 16) & 0xff;
6135                 e[2] = (hid >> 8) & 0xff;
6136                 e[3] = hid & 0xff;
6137                 e[4] = e[5] = 0;
6138         }
6139         bcopy(e, p+8, 6);
6140         uniqtime32(&timestamp32);
6141         *t = BE_32(*t);
6142         bcopy(t, p+14, 4);
6143         p[18] = (gen_number >> 8) & 0xff;
6144         p[19] = gen_number & 0xff;
6142         p[18] = (stmf_lu_id_gen_number >> 8) & 0xff;
6143         p[19] = stmf_lu_id_gen_number & 0xff;

6146         return (STMF_SUCCESS);
6147 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   43954 Wed Jul  3 14:44:09 2013
new/usr/src/uts/common/io/idm/idm_conn_sm.c
3866 panic in idm module
3867 stmfCreateLu failed: GUID_IN_USE
3868 iscsi target not accepting any new connections
Reviewed by: Sebastien Roy <sebastien.roy@delphix.com>
Reviewed by: Jeremy Jones <jeremy@delphix.com>
Reviewed by: Eric Diven <eric.diven@delphix.com>
**********************************************************
```

```
/*
 * 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) 2008, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */

#include <sys/cpuvar.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/modctl.h>
#include <sys/socket.h>
#include <sys/strsubr.h>
#include <sys/note.h>
#include <sys/sdt.h>

#define IDM_CONN_SM_STRINGS
#define IDM_CN_NOTIFY_STRINGS
#include <sys/idm/idm.h>

boolean_t       idm_sm_logging = B_FALSE;

extern idm_global_t     idm; /* Global state */

static void
idm_conn_event_handler(void *event_ctx_opaque);

static void
idm_state_s1_free(idm_conn_t *ic, idm_conn_event_ctx_t *event_ctx);

static void
idm_state_s2_xpt_wait(idm_conn_t *ic, idm_conn_event_ctx_t *event_ctx);

static void
idm_state_s3_xpt_up(idm_conn_t *ic, idm_conn_event_ctx_t *event_ctx);

static void
```



```
idm_state_s4_in_login(idm_conn_t *ic, idm_conn_event_ctx_t *event_ctx);

static void
idm_state_s5_logged_in(idm_conn_t *ic, idm_conn_event_ctx_t *event_ctx);

static void
idm_state_s6_in_logout(idm_conn_t *ic, idm_conn_event_ctx_t *event_ctx);

static void
idm_logout_req_timeout(void *arg);

static void
idm_state_s7_logout_req(idm_conn_t *ic, idm_conn_event_ctx_t *event_ctx);

static void
idm_state_s8_cleanup(idm_conn_t *ic, idm_conn_event_ctx_t *event_ctx);

static void
idm_state_s9_init_error(idm_conn_t *ic, idm_conn_event_ctx_t *event_ctx);

static void
idm_state_s9a_rejected(idm_conn_t *ic, idm_conn_event_ctx_t *event_ctx);

static void
idm_state_s9b_wait_snd_done_cb(idm_pdu_t *pdu,
    idm_status_t status);

static void
idm_state_s9b_wait_snd_done(idm_conn_t *ic,
    idm_conn_event_ctx_t *event_ctx);

static void
idm_state_s10_in_cleanup(idm_conn_t *ic, idm_conn_event_ctx_t *event_ctx);

static void
idm_state_s11_complete(idm_conn_t *ic, idm_conn_event_ctx_t *event_ctx);

static void
idm_state_s12_enable_dm(idm_conn_t *ic, idm_conn_event_ctx_t *event_ctx);

static void
idm_update_state(idm_conn_t *ic, idm_conn_state_t new_state,
    idm_conn_event_ctx_t *event_ctx);

static void
idm_conn_unref(void *ic_void);

static void
idm_conn_reject_unref(void *ic_void);

static idm_pdu_event_action_t
idm_conn_sm_validate_pdu(idm_conn_t *ic, idm_conn_event_ctx_t *event_ctx,
    idm_pdu_t *pdu);

static idm_status_t
idm_ffp_enable(idm_conn_t *ic);

static void
idm_ffp_disable(idm_conn_t *ic, idm_ffp_disable_t disable_type);

static void
idm_initial_login_actions(idm_conn_t *ic, idm_conn_event_ctx_t *event_ctx);

static void
idm_login_success_actions(idm_conn_t *ic, idm_conn_event_ctx_t *event_ctx);
```

```
123 idm_status_t
124 idm_conn_sm_init(idm_conn_t *ic)
125 {
126         char taskq_name[32];

128         /*
129          * Caller should have assigned a unique connection ID.  Use this
130          * connection ID to create a unique connection name string
131          */
132         ASSERT(ic->ic_internal_cid != 0);
133         (void) snprintf(taskq_name, sizeof (taskq_name) - 1, "conn_sm%08x",
134             ic->ic_internal_cid);

136         ic->ic_state_taskq = taskq_create(taskq_name, 1, minclsyspri, 4, 16384,
137             TASKQ_PREPOPULATE);
138         if (ic->ic_state_taskq == NULL) {
139                 return (IDM_STATUS_FAIL);
140         }

142         idm_sm_audit_init(&ic->ic_state_audit);
143         mutex_init(&ic->ic_state_mutex, NULL, MUTEX_DEFAULT, NULL);
144         cv_init(&ic->ic_state_cv, NULL, CV_DEFAULT, NULL);

146         ic->ic_state = CS_S1_FREE;
147         ic->ic_last_state = CS_S1_FREE;

149         return (IDM_STATUS_SUCCESS);
150 }
_____unchanged_portion_omitted_

542 static void
543 idm_state_s4_in_login(idm_conn_t *ic, idm_conn_event_ctx_t *event_ctx)
544 {
545         idm_pdu_t *pdu;

547         /*
548          * Login timer should no longer be active after leaving this
549          * state.
550          */
551         switch (event_ctx->iec_event) {
552         case CE_LOGIN_SUCCESS_RCV:
553         case CE_LOGIN_SUCCESS_SND:
554                 ASSERT(ic->ic_client_callback == NULL);

556                 (void) untimeout(ic->ic_state_timeout);
557                 idm_login_success_actions(ic, event_ctx);
558                 if (ic->ic_rdma_extensions) {
559                         /* T19 */
560                         idm_update_state(ic, CS_S12_ENABLE_DM, event_ctx);
561                 } else {
562                         /* T5 */
563                         idm_update_state(ic, CS_S5_LOGGED_IN, event_ctx);
564                 }
565                 break;
566         case CE_LOGIN_TIMEOUT:
567                 /* T7 */
568                 (void) idm_notify_client(ic, CN_LOGIN_FAIL, NULL);
569                 idm_update_state(ic, CS_S9_INIT_ERROR, event_ctx);
570                 break;
571         case CE_LOGIN_FAIL_SND:
572                 /*
573                  * Allow the logout response pdu to be sent and defer
574                  * the state machine cleanup until the completion callback.
575                  * Only 1 level or callback interposition is allowed.
576                  */
577                 (void) untimeout(ic->ic_state_timeout);
```



```
578                 pdu = (idm_pdu_t *)event_ctx->iec_info;
579                 ASSERT(ic->ic_client_callback == NULL);
580                 ic->ic_client_callback = pdu->isp_callback;
581                 pdu->isp_callback =
582                     idm_state_s9b_wait_snd_done_cb;
583                 idm_update_state(ic, CS_S9B_WAIT_SND_DONE,
584                     event_ctx);
585                 break;
586         case CE_LOGIN_FAIL_RCV:
587                 ASSERT(ic->ic_client_callback == NULL);
588                 /*
589                  * Need to deliver this PDU to the initiator now because after
590                  * we update the state to CS_S9_INIT_ERROR the initiator will
591                  * no longer be in an appropriate state.
592                  */
593                 event_ctx->iec_pdu_forwarded = B_TRUE;
594                 pdu = (idm_pdu_t *)event_ctx->iec_info;
595                 idm_pdu_rx_forward(ic, pdu);
596                 /* FALLTHROUGH */
597         case CE_TRANSPORT_FAIL:
598         case CE_LOGOUT_OTHER_CONN_SND:
599         case CE_LOGOUT_OTHER_CONN_RCV:
600                 /* T7 */
601                 (void) untimeout(ic->ic_state_timeout);
602                 (void) idm_notify_client(ic, CN_LOGIN_FAIL, NULL);
603                 idm_update_state(ic, CS_S9_INIT_ERROR, event_ctx);
604                 break;
605         case CE_LOGOUT_SESSION_SUCCESS:
606                 /*
607                  * T8
608                  * A session reinstatement request can be received while a
609                  * session is active and a login is in process. The iSCSI
610                  * connections are shut down by a CE_LOGOUT_SESSION_SUCCESS
611                  * event sent from the session to the IDM layer.
612                  */
613                 (void) untimeout(ic->ic_state_timeout);
614                 if (IDM_CONN_ISTGT(ic)) {
615                         ic->ic_transport_ops->it_tgt_conn_disconnect(ic);
616                 } else {
617                         ic->ic_transport_ops->it_ini_conn_disconnect(ic);
618                 }
619                 idm_update_state(ic, CS_S11_COMPLETE, event_ctx);
620                 break;

622         case CE_LOGIN_SND:
623                 ASSERT(ic->ic_client_callback == NULL);
624                 /*
625                  * Initiator connections will see initial login PDU
626                  * in this state.  Target connections see initial
627                  * login PDU in "xpt up" state.
628                  */
629                 mutex_enter(&ic->ic_state_mutex);
630                 if (!(ic->ic_state_flags & CF_INITIAL_LOGIN)) {
631                         idm_initial_login_actions(ic, event_ctx);
632                 }
633                 mutex_exit(&ic->ic_state_mutex);
634                 break;
635         case CE_MISC_TX:
636         case CE_MISC_RX:
637         case CE_LOGIN_RCV:
638         case CE_TX_PROTOCOL_ERROR:
639         case CE_RX_PROTOCOL_ERROR:
640                 /* Don't care */
641                 break;
642         default:
643                 ASSERT(0);
```

```
644                 /*NOTREACHED*/
645         }
646 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   84624 Wed Jul  3 14:44:10 2013
new/usr/src/uts/common/io/idm/idm_so.c
3866 panic in idm module
3867 stmfCreateLu failed: GUID_IN_USE
3868 iscsi target not accepting any new connections
Reviewed by: Sebastien Roy <sebastien.roy@delphix.com>
Reviewed by: Jeremy Jones <jeremy@delphix.com>
Reviewed by: Eric Diven <eric.diven@delphix.com>
**********************************************************
______unchanged_portion_omitted_

1216 /*
1217  * Watch thread for target service connection establishment.
1218  */
1219 void
1220 idm_so_svc_port_watcher(void *arg)
1221 {
1222         idm_svc_t               *svc = arg;
1223         ksocket_t               new_so;
1224         idm_conn_t              *ic;
1225         idm_status_t            idmrc;
1226         idm_so_svc_t            *so_svc;
1227         int                     rc;
1228         const uint32_t          off = 0;
1229         struct sockaddr_in6     t_addr;
1230         socklen_t               t_addrlen;

1232         bzero(&t_addr, sizeof (struct sockaddr_in6));
1233         t_addrlen = sizeof (struct sockaddr_in6);
1234         mutex_enter(&svc->is_mutex);

1236         so_svc = svc->is_so_svc;
1237         so_svc->is_thread_running = B_TRUE;
1238         so_svc->is_thread_did = so_svc->is_thread->t_did;

1240         cv_signal(&svc->is_cv);

1242         IDM_SVC_LOG(CE_NOTE, "iSCSI service (%p/%d) online", (void *)svc,
1243             svc->is_svc_req.sr_port);

1245         while (so_svc->is_thread_running) {
1246                 mutex_exit(&svc->is_mutex);

1248                 if ((rc = ksocket_accept(so_svc->is_so,
1249                     (struct sockaddr *)&t_addr, &t_addrlen,
1250                     &new_so, CRED())) != 0) {
1251                         mutex_enter(&svc->is_mutex);
1252                         if (rc != ECONNABORTED && rc != EINTR) {
1253                                 IDM_SVC_LOG(CE_NOTE, "idm_so_svc_port_watcher:"
1254                                     " ksocket_accept failed %d", rc);
1255                         }
1256                         /*
1257                          * Unclean shutdown of this thread is not handled
1258                          * wait for !is_thread_running.
1259                          */
1252                         if (rc == ECONNABORTED)
1260                         continue;
1254                         /* Connection problem */
1255                         break;
1261                 }
1262                 /*
1263                  * Turn off SO_MAC_EXEMPT so future sobinds succeed
1264                  */
1265                 (void) ksocket_setsockopt(new_so, SOL_SOCKET, SO_MAC_EXEMPT,
1266                     (char *)&off, sizeof (off), CRED());
```



```
1268                 idmrc = idm_svc_conn_create(svc, IDM_TRANSPORT_TYPE_SOCKETS,
1269                     &ic);
1270                 if (idmrc != IDM_STATUS_SUCCESS) {
1271                         /* Drop connection */
1272                         idm_soshutdown(new_so);
1273                         idm_sodestroy(new_so);
1274                         mutex_enter(&svc->is_mutex);
1275                         continue;
1276                 }

1278                 idmrc = idm_so_tgt_conn_create(ic, new_so);
1279                 if (idmrc != IDM_STATUS_SUCCESS) {
1280                         idm_svc_conn_destroy(ic);
1281                         idm_soshutdown(new_so);
1282                         idm_sodestroy(new_so);
1283                         mutex_enter(&svc->is_mutex);
1284                         continue;
1285                 }

1287                 /*
1288                  * Kick the state machine.  At CS_S3_XPT_UP the state machine
1289                  * will notify the client (target) about the new connection.
1290                  */
1291                 idm_conn_event(ic, CE_CONNECT_ACCEPT, NULL);

1293                 mutex_enter(&svc->is_mutex);
1294         }
1295         ksocket_rele(so_svc->is_so);
1296         so_svc->is_thread_running = B_FALSE;
1297         mutex_exit(&svc->is_mutex);

1299         IDM_SVC_LOG(CE_NOTE, "iSCSI service (%p/%d) offline", (void *)svc,
1300             svc->is_svc_req.sr_port);

1302         thread_exit();
1303 }
______unchanged_portion_omitted_
```