```
**********************************************************
    9455 Thu Apr  4 14:14:05 2019
new/usr/src/uts/common/io/comstar/port/srpt/srpt_cm.c
10689 srpt_cm_conn_closed_hdlr() needs a smatch fix
**********************************************************
/*
 * 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) 2009, 2010, Oracle and/or its affiliates. All rights reserved.
 */

/*
 * Copyright 2019, Joyent, Inc.
 */

/*
 * IB CM handlers for s Solaris SCSI RDMA Protocol Target (SRP)
 * transport port provider module for the COMSTAR framework.
 */

#include <sys/cpuvar.h>
#include <sys/types.h>
#include <sys/conf.h>
#include <sys/stat.h>
#include <sys/file.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/modctl.h>
#include <sys/sysmacros.h>
#include <sys/sdt.h>
#include <sys/taskq.h>
#include <sys/ib/ibtl/ibti.h>

#include <sys/stmf.h>
#include <sys/stmf_ioctl.h>
#include <sys/portif.h>

#include "srp.h"
#include "srpt_impl.h"
#include "srpt_cm.h"
#include "srpt_stp.h"
#include "srpt_ch.h"

extern uint16_t srpt_send_msg_depth;
extern srpt_ctxt_t  *srpt_ctxt;

/*
```



```
 * srpt_cm_req_hdlr() - Login request
 *
 * CM has called back with a CM REQ message associated with an
 * SRP initiator login request.
 */
static ibt_cm_status_t
srpt_cm_req_hdlr(srpt_target_port_t *tgt, ibt_cm_event_t *event,
    ibt_cm_return_args_t *ret_args, void *ret_priv_data,
    ibt_priv_data_len_t ret_priv_data_len)
{
        ibt_cm_status_t         status;
        ibt_cm_req_rcv_t        *req;
        srp_login_req_t         login;
        srp_login_rej_t         login_rej;
        srp_login_rsp_t         login_rsp;
        srpt_channel_t          *ch = NULL;
        char                    remote_gid[SRPT_ALIAS_LEN];
        char                    local_gid[SRPT_ALIAS_LEN];

        ASSERT(tgt != NULL);
        req = &event->cm_event.req;

        if (event->cm_priv_data_len <  sizeof (srp_login_req_t)) {
                SRPT_DPRINTF_L2("cm_req_hdlr, IU size expected (>= %d),"
                    " received size (%d)", (uint_t)sizeof (srp_login_req_t),
                    event->cm_priv_data_len);
                return (IBT_CM_REJECT);
        }

        if (event->cm_priv_data == NULL) {
                SRPT_DPRINTF_L2("cm_req_hdlr, NULL ULP private data pointer");
                return (IBT_CM_REJECT);
        }

        if (ret_priv_data_len <  sizeof (srp_login_rej_t)) {
                SRPT_DPRINTF_L2("cm_req_hdlr, return private len too"
                    " small (%d)", ret_priv_data_len);
                return (IBT_CM_REJECT);
        }

        if (ret_priv_data == NULL) {
                SRPT_DPRINTF_L2("cm_req_hdlr, NULL ULP return private data"
                    " pointer");
                return (IBT_CM_REJECT);
        }

        /*
         * Copy to avoid potential alignment problems, process login
         * creating a new channel and possibly session.
         */
        bcopy(event->cm_priv_data, &login,  sizeof (login));

        ALIAS_STR(local_gid,
            req->req_prim_addr.av_sgid.gid_prefix,
            req->req_prim_addr.av_sgid.gid_guid);
        ALIAS_STR(remote_gid,
            req->req_prim_addr.av_dgid.gid_prefix,
            req->req_prim_addr.av_dgid.gid_guid);

        ch = srpt_stp_login(tgt, &login, &login_rsp,
            &login_rej, req->req_prim_hca_port, local_gid, remote_gid);
        if (ch != NULL) {
                bcopy(&login_rsp, ret_priv_data,  SRP_LOGIN_RSP_SIZE);
                ret_args->cm_ret_len = SRP_LOGIN_RSP_SIZE;

                SRPT_DPRINTF_L3("cm_req_hdlr, rsp priv len(%d)"
```

```
128                     " ch created on port(%d)"
129                     ", cm_req_hdlr, req ra_out(%d), ra_in(%d)"
130                     ", retry(%d)",
131                     ret_args->cm_ret_len, req->req_prim_hca_port,
132                     req->req_rdma_ra_out, req->req_rdma_ra_in,
133                     req->req_retry_cnt);

135                 ret_args->cm_ret.rep.cm_channel = ch->ch_chan_hdl;
136                 ret_args->cm_ret.rep.cm_rdma_ra_out =
137                     min(tgt->tp_ioc->ioc_attr.hca_max_rdma_out_chan,
138                     req->req_rdma_ra_in);
139                 ret_args->cm_ret.rep.cm_rdma_ra_in =
140                     min(tgt->tp_ioc->ioc_attr.hca_max_rdma_in_chan,
141                     req->req_rdma_ra_out);
142                 ret_args->cm_ret.rep.cm_rnr_retry_cnt = req->req_retry_cnt;

144                 SRPT_DPRINTF_L3("cm_req_hdlr, hca_max_rdma_in_chan (%d)"
145                     ", hca_max_rdma_out_chan (%d)"
146                     ", updated ra_out(%d), ra_in(%d), retry(%d)",
147                     tgt->tp_ioc->ioc_attr.hca_max_rdma_in_chan,
148                     tgt->tp_ioc->ioc_attr.hca_max_rdma_out_chan,
149                     ret_args->cm_ret.rep.cm_rdma_ra_out,
150                     ret_args->cm_ret.rep.cm_rdma_ra_in,
151                     ret_args->cm_ret.rep.cm_rnr_retry_cnt);
152                 status = IBT_CM_ACCEPT;

154         } else {
155                 bcopy(&login_rej, ret_priv_data,  sizeof (login_rej));
156                 ret_args->cm_ret_len =  sizeof (login_rej);
157                 status = IBT_CM_REJECT;
158         }

160         return (status);
161 }
```

*unchanged_portion_omitted_*

```
199 /*
200  * srpt_cm_conn_closed_hdlr() - Channel closed
201  *
202  * CM callback indicating a channel has been completely closed.
203  */
204 /* ARGSUSED */
205 static ibt_cm_status_t
206 srpt_cm_conn_closed_hdlr(srpt_target_port_t *tgt, ibt_cm_event_t *event)
207 {
208         ibt_cm_status_t         status = IBT_CM_ACCEPT;
209         srpt_channel_t          *ch;

211         ASSERT(tgt != NULL);
212         ASSERT(event != NULL);

214         ch = (srpt_channel_t *)ibt_get_chan_private(event->cm_channel);
215         ASSERT(ch != NULL);

217         SRPT_DPRINTF_L3("cm_conn_closed_hdlr, invoked for chan_hdl(%p),"
218             " event(%d)", (void *)ch->ch_chan_hdl,
219             event->cm_event.closed);

221         switch (event->cm_event.closed) {

223         case IBT_CM_CLOSED_DREP_RCVD:
224         case IBT_CM_CLOSED_DREQ_TIMEOUT:
225         case IBT_CM_CLOSED_DUP:
226         case IBT_CM_CLOSED_ABORT:
227         case IBT_CM_CLOSED_ALREADY:
228                 /*
```



```
229                  * These cases indicate the SRP target initiated
230                  * the closing of the channel and it is now closed.
231                  * Cleanup the channel (which will remove the targets
232                  * reference) and then release CM's reference.
233                  */
234                 SRPT_DPRINTF_L3("cm_conn_closed_hdlr, local close call-back");
235                 srpt_ch_cleanup(ch);
236                 srpt_ch_release_ref(ch, 1);
237                 break;

239         case IBT_CM_CLOSED_DREQ_RCVD:
240         case IBT_CM_CLOSED_REJ_RCVD:
241         case IBT_CM_CLOSED_STALE:
242                 /*
243                  * These cases indicate that the SRP initiator is closing
244                  * the channel.  CM will have already closed the RC channel,
245                  * so simply initiate cleanup which will remove the target
246                  * ports reference to the channel and then release the
247                  * reference held by the CM.
248                  */
249                 SRPT_DPRINTF_L3("cm_conn_closed_hdlr, remote close,"
250                     " free channel");
247                 if (ch != NULL) {
251                 srpt_ch_cleanup(ch);
252                 srpt_ch_release_ref(ch, 1);
250                 } else {
251                         SRPT_DPRINTF_L2("cm_conn_closed_hdlr, NULL channel");
252                 }
253                 break;

255         default:
256                 SRPT_DPRINTF_L2("cm_conn_closed_hdlr, unknown close type (%d)",
257                     event->cm_event.closed);
258                 status = IBT_CM_DEFAULT;
259                 break;
260         }
261         return (status);
262 }
```

*unchanged_portion_omitted_*