SCSI EH

 This document describes SCSI midlayer error handling infrastructure.
Please refer to Documentation/scsi/scsi_mid_low_api.txt for more information regarding SCSI midlayer.

TABLE OF CONTENTS

[1] How SCSI commands travel through the midlayer and to EH

[1-1] struct scsi_cmnd

 Each SCSI command is represented with struct scsi_cmnd (== scmd). A scmd has two list_head's to link itself into lists. The two are scmd->list and scmd->eh_entry. The former is used for free list or per-device allocated scmd list and not of much interest to this EH discussion. The latter is used for completion and EH lists and unless otherwise stated scmds are always linked using scmd->eh_entry in this discussion.

[1-2] How do scmd's get completed?

 Once LLDD gets hold of a scmd, either the LLDD will complete the command by calling scsi_done callback passed from midlayer when invoking hostt->queuecommand() or SCSI midlayer will time it out.

[1-2-1] Completing a scmd w/ scsi_done

 For all non-EH commands, scsi_done() is the completion callback. It does the following.

 1. Delete timeout timer. If it fails, it means that timeout timer
    has expired and is going to finish the command. Just return.

 2. Link scmd to per-cpu scsi_done_q using scmd->en_entry

3. Raise SCSI_SOFTIRQ

SCSI_SOFTIRQ handler scsi_softirq calls scsi_decide_disposition() to determine what to do with the command. scsi_decide_disposition() looks at the scmd->result value and sense data to determine what to do with the command.

- SUCCESS
  scsi_finish_command() is invoked for the command. The function does some maintenance choirs and notify completion by calling scmd->done() callback, which, for fs requests, would be HLD completion callback - sd:sd_rw_intr, sr:rw_intr, st:st_intr.

- NEEDS_RETRY
- ADD_TO_MLQUEUE
  scmd is requeued to blk queue.

- otherwise
  scsi_eh_scmd_add(scmd, 0) is invoked for the command. See [1-3] for details of this function.

[1-2-2] Completing a scmd w/ timeout

The timeout handler is scsi_times_out(). When a timeout occurs, this function

1. invokes optional hostt->eh_timed_out() callback. Return value can be one of

   - EH_HANDLED
     This indicates that eh_timed_out() dealt with the timeout. The scmd is passed to __scsi_done() and thus linked into per-cpu scsi_done_q. Normal command completion described in [1-2-1] follows.

   - EH_RESET_TIMER
     This indicates that more time is required to finish the command. Timer is restarted. This action is counted as a retry and only allowed scmd->allowed + 1(!) times. Once the limit is reached, action for EH_NOT_HANDLED is taken instead.

     NOTE This action is racy as the LLDD could finish the scmd after the timeout has expired but before it's added back. In such cases, scsi_done() would think that timeout has occurred and return without doing anything. We lose completion and the command will time out again.

   - EH_NOT_HANDLED
     This is the same as when eh_timed_out() callback doesn't exist. Step #2 is taken.

2. scsi_eh_scmd_add(scmd, SCSI_EH_CANCEL_CMD) is invoked for the command. See [1-3] for more information.

[1-3] How EH takes over

scmds enter EH via scsi_eh_scmd_add(), which does the following.

1. Turns on scmd->eh_eflags as requested. It's 0 for error completions and SCSI_EH_CANCEL_CMD for timeouts.

2. Links scmd->eh_entry to shost->eh_cmd_q

3. Sets SHOST_RECOVERY bit in shost->shost_state

4. Increments shost->host_failed

5. Wakes up SCSI EH thread if shost->host_busy == shost->host_failed

As can be seen above, once any scmd is added to shost->eh_cmd_q, SHOST_RECOVERY shost_state bit is turned on. This prevents any new scmd to be issued from blk queue to the host; eventually, all scmds on the host either complete normally, fail and get added to eh_cmd_q, or time out and get added to shost->eh_cmd_q.

If all scmds either complete or fail, the number of in-flight scmds becomes equal to the number of failed scmds - i.e. shost->host_busy == shost->host_failed. This wakes up SCSI EH thread. So, once woken up, SCSI EH thread can expect that all in-flight commands have failed and are linked on shost->eh_cmd_q.

Note that this does not mean lower layers are quiescent. If a LLDD completed a scmd with error status, the LLDD and lower layers are assumed to forget about the scmd at that point. However, if a scmd has timed out, unless hostt->eh_timed_out() made lower layers forget about the scmd, which currently no LLDD does, the command is still active as long as lower layers are concerned and completion could occur at any time. Of course, all such completions are ignored as the timer has already expired.

We'll talk about how SCSI EH takes actions to abort - make LLDD forget about - timed out scmds later.

[2] How SCSI EH works

LLDD's can implement SCSI EH actions in one of the following two ways.

- Fine-grained EH callbacks
  LLDD can implement fine-grained EH callbacks and let SCSI midlayer drive error handling and call appropriate callbacks. This will be discussed further in [2-1].

- eh_strategy_handler() callback
  This is one big callback which should perform whole error handling. As such, it should do all choirs SCSI midlayer performs during recovery. This will be discussed in [2-2].

 Once recovery is complete, SCSI EH resumes normal operation by calling scsi_restart_operations(), which

 1. Checks if door locking is needed and locks door.

 2. Clears SHOST_RECOVERY shost_state bit

 3. Wakes up waiters on shost->host_wait. This occurs if someone calls scsi_block_when_processing_errors() on the host. (*QUESTION* why is it needed? All operations will be blocked anyway after it reaches blk queue.)

 4. Kicks queues in all devices on the host in the asses

[2-1] EH through fine-grained callbacks

[2-1-1] Overview

 If eh_strategy_handler() is not present, SCSI midlayer takes charge of driving error handling. EH's goals are two - make LLDD, host and device forget about timed out scmds and make them ready for new commands. A scmd is said to be recovered if the scmd is forgotten by lower layers and lower layers are ready to process or fail the scmd again.

 To achieve these goals, EH performs recovery actions with increasing severity. Some actions are performed by issuing SCSI commands and others are performed by invoking one of the following fine-grained hostt EH callbacks. Callbacks may be omitted and omitted ones are considered to fail always.

```
int (* eh_abort_handler)(struct scsi_cmnd *);
int (* eh_device_reset_handler)(struct scsi_cmnd *);
int (* eh_bus_reset_handler)(struct scsi_cmnd *);
int (* eh_host_reset_handler)(struct scsi_cmnd *);
```

 Higher-severity actions are taken only when lower-severity actions cannot recover some of failed scmds. Also, note that failure of the highest-severity action means EH failure and results in offlining of all unrecovered devices.

 During recovery, the following rules are followed

 - Recovery actions are performed on failed scmds on the to do list, eh_work_q. If a recovery action succeeds for a scmd, recovered scmds are removed from eh_work_q.

   Note that single recovery action on a scmd can recover multiple scmds. e.g. resetting a device recovers all failed scmds on the device.

 - Higher severity actions are taken iff eh_work_q is not empty after lower severity actions are complete.

- EH reuses failed scmds to issue commands for recovery. For timed-out scmds, SCSI EH ensures that LLDD forgets about a scmd before reusing it for EH commands.

When a scmd is recovered, the scmd is moved from eh_work_q to EH local eh_done_q using scsi_eh_finish_cmd(). After all scmds are recovered (eh_work_q is empty), scsi_eh_flush_done_q() is invoked to either retry or error-finish (notify upper layer of failure) recovered scmds.

scmds are retried iff its sdev is still online (not offlined during EH), REQ_FAILFAST is not set and ++scmd->retries is less than scmd->allowed.

[2-1-2] Flow of scmds through EH

```
 1. Error completion / time out
    ACTION: scsi_eh_scmd_add() is invoked for scmd
        - set scmd->eh_eflags
        - add scmd to shost->eh_cmd_q
        - set SHOST_RECOVERY
        - shost->host_failed++
    LOCKING: shost->host_lock

 2. EH starts
    ACTION: move all scmds to EH's local eh_work_q.  shost->eh_cmd_q
            is cleared.
    LOCKING: shost->host_lock (not strictly necessary, just for
             consistency)

 3. scmd recovered
    ACTION: scsi_eh_finish_cmd() is invoked to EH-finish scmd
        - shost->host_failed--
        - clear scmd->eh_eflags
        - scsi_setup_cmd_retry()
        - move from local eh_work_q to local eh_done_q
    LOCKING: none

 4. EH completes
    ACTION: scsi_eh_flush_done_q() retries scmds or notifies upper
            layer of failure.
        - scmd is removed from eh_done_q and scmd->eh_entry is cleared
        - if retry is necessary, scmd is requeued using
          scsi_queue_insert()
        - otherwise, scsi_finish_command() is invoked for scmd
    LOCKING: queue or finish function performs appropriate locking
```

[2-1-3] Flow of control

EH through fine-grained callbacks start from scsi_unjam_host().

<<scsi_unjam_host>>

```
    1. Lock shost->host_lock, splice_init shost->eh_cmd_q into local
```

```
   eh_work_q and unlock host_lock.  Note that shost->eh_cmd_q is
   cleared by this action.

2. Invoke scsi_eh_get_sense.

<<scsi_eh_get_sense>>

    This action is taken for each error-completed
    (!SCSI_EH_CANCEL_CMD) commands without valid sense data.  Most
    SCSI transports/LLDDs automatically acquire sense data on
    command failures (autosense).  Autosense is recommended for
    performance reasons and as sense information could get out of
    sync inbetween occurrence of CHECK CONDITION and this action.

    Note that if autosense is not supported, scmd->sense_buffer
    contains invalid sense data when error-completing the scmd
    with scsi_done().  scsi_decide_disposition() always returns
    FAILED in such cases thus invoking SCSI EH.  When the scmd
    reaches here, sense data is acquired and
    scsi_decide_disposition() is called again.

    1. Invoke scsi_request_sense() which issues REQUEST_SENSE
       command.  If fails, no action.  Note that taking no action
       causes higher-severity recovery to be taken for the scmd.

    2. Invoke scsi_decide_disposition() on the scmd

       - SUCCESS
           scmd->retries is set to scmd->allowed preventing
           scsi_eh_flush_done_q() from retrying the scmd and
           scsi_eh_finish_cmd() is invoked.

       - NEEDS_RETRY
           scsi_eh_finish_cmd() invoked

       - otherwise
           No action.

3. If !list_empty(&eh_work_q), invoke scsi_eh_abort_cmds().

<<scsi_eh_abort_cmds>>

    This action is taken for each timed out command.
    hostt->eh_abort_handler() is invoked for each scmd.  The
    handler returns SUCCESS if it has succeeded to make LLDD and
    all related hardware forget about the scmd.

    If a timedout scmd is successfully aborted and the sdev is
    either offline or ready, scsi_eh_finish_cmd() is invoked for
    the scmd.  Otherwise, the scmd is left in eh_work_q for
    higher-severity actions.

    Note that both offline and ready status mean that the sdev is
    ready to process new scmds, where processing also implies
    immediate failing; thus, if a sdev is in one of the two
    states, no further recovery action is needed.
```

Device readiness is tested using scsi_eh_tur() which issues TEST_UNIT_READY command. Note that the scmd must have been aborted successfully before reusing it for TEST_UNIT_READY.

4. If !list_empty(&eh_work_q), invoke scsi_eh_ready_devs()

<<scsi_eh_ready_devs>>

This function takes four increasingly more severe measures to make failed sdevs ready for new commands.

1. Invoke scsi_eh_stu()

<<scsi_eh_stu>>

For each sdev which has failed scmds with valid sense data of which scsi_check_sense()'s verdict is FAILED, START_STOP_UNIT command is issued w/ start=1. Note that as we explicitly choose error-completed scmds, it is known that lower layers have forgotten about the scmd and we can reuse it for STU.

If STU succeeds and the sdev is either offline or ready, all failed scmds on the sdev are EH-finished with scsi_eh_finish_cmd().

NOTE If hostt->eh_abort_handler() isn't implemented or failed, we may still have timed out scmds at this point and STU doesn't make lower layers forget about those scmds. Yet, this function EH-finish all scmds on the sdev if STU succeeds leaving lower layers in an inconsistent state. It seems that STU action should be taken only when a sdev has no timed out scmd.

2. If !list_empty(&eh_work_q), invoke scsi_eh_bus_device_reset().

<<scsi_eh_bus_device_reset>>

This action is very similar to scsi_eh_stu() except that, instead of issuing STU, hostt->eh_device_reset_handler() is used. Also, as we're not issuing SCSI commands and resetting clears all scmds on the sdev, there is no need to choose error-completed scmds.

3. If !list_empty(&eh_work_q), invoke scsi_eh_bus_reset()

<<scsi_eh_bus_reset>>

hostt->eh_bus_reset_handler() is invoked for each channel with failed scmds. If bus reset succeeds, all failed scmds on all ready or offline sdevs on the channel are EH-finished.

4. If !list_empty(&eh_work_q), invoke scsi_eh_host_reset()

<<scsi_eh_host_reset>>

This is the last resort. hostt->eh_host_reset_handler() is invoked. If host reset succeeds, all failed scmds on all ready or offline sdevs on the host are EH-finished.

5. If !list_empty(&eh_work_q), invoke scsi_eh_offline_sdevs()

<<scsi_eh_offline_sdevs>>

Take all sdevs which still have unrecovered scmds offline and EH-finish the scmds.

5. Invoke scsi_eh_flush_done_q().

<<scsi_eh_flush_done_q>>

At this point all scmds are recovered (or given up) and put on eh_done_q by scsi_eh_finish_cmd(). This function flushes eh_done_q by either retrying or notifying upper layer of failure of the scmds.

[2-2] EH through transportt->eh_strategy_handler()

transportt->eh_strategy_handler() is invoked in the place of scsi_unjam_host() and it is responsible for whole recovery process. On completion, the handler should have made lower layers forget about all failed scmds and either ready for new commands or offline. Also, it should perform SCSI EH maintenance choirs to maintain integrity of SCSI midlayer. IOW, of the steps described in [2-1-2], all steps except for #1 must be implemented by eh_strategy_handler().

[2-2-1] Pre transportt->eh_strategy_handler() SCSI midlayer conditions

The following conditions are true on entry to the handler.

- Each failed scmd's eh_flags field is set appropriately.

- Each failed scmd is linked on scmd->eh_cmd_q by scmd->eh_entry.

- SHOST_RECOVERY is set.

- shost->host_failed == shost->host_busy

[2-2-2] Post transportt->eh_strategy_handler() SCSI midlayer conditions

The following conditions must be true on exit from the handler.

- shost->host_failed is zero.

- Each scmd's eh_eflags field is cleared.

- Each scmd is in such a state that scsi_setup_cmd_retry() on the scmd doesn't make any difference.

- shost->eh_cmd_q is cleared.

- Each scmd->eh_entry is cleared.

- Either scsi_queue_insert() or scsi_finish_command() is called on each scmd. Note that the handler is free to use scmd->retries and ->allowed to limit the number of retries.

[2-2-3] Things to consider

- Know that timed out scmds are still active on lower layers. Make lower layers forget about them before doing anything else with those scmds.

- For consistency, when accessing/modifying shost data structure, grab shost->host_lock.

- On completion, each failed sdev must have forgotten about all active scmds.

- On completion, each failed sdev must be ready for new commands or offline.

--
Tejun Heo
htejun@gmail.com
11th September 2005