SCSI 命令（scsi_cmnd）的生命周期

Rajat Jain <rajatja@google.com> 于 2015 年 5 月 12 日

翻译于 2023年3月15日

&&& 翻译说明 &&&

本文对应的原文在之前已向上游社区提交，但在当前的代码中并没有看到此文档。

~ ~ ~

```
$ ls Documentation/scsi/
53c700.rst                    aic7xxx.rst                       qlogicfas.rst
BusLogic.rst                  arcmsr_spec.rst                   scsi-changer.rst
ChangeLog.arcmsr              bfa.rst                           scsi-generic.rst
ChangeLog.ips                 bnx2fc.rst                        scsi-parameters.rst
ChangeLog.lpfc                cxgb3i.rst                        scsi.rst
ChangeLog.megaraid            dc395x.rst                        scsi_eh.rst
ChangeLog.megaraid_sas        dpti.rst                  scsi_fc_transport.rst
ChangeLog.ncr53c8xx           g_NCR5380.rst             scsi_mid_low_api.rst
ChangeLog.sym53c8xx           hpsa.rst                      scsi_transport_srp
ChangeLog.sym53c8xx_2         hptiop.rst                    sd-parameters.rst
FlashPoint.rst                index.rst                     smartpqi.rst
LICENSE.FlashPoint            libsas.rst                    st.rst
NinjaSCSI.rst                 link_power_management_policy.rst sym53c500_cs.rst
aacraid.rst                   lpfc.rst                      sym53c8xx_2.rst
advansys.rst                  megaraid.rst                  tcm_qla2xxx.rst
aha152x.rst                   ncr53c8xx.rst                 ufs.rst
aic79xx.rst                   ppa.rst                       wd719x.rst
~ ~ ~
```

提交的信息可参考如下链接：

https://lwn.net/Articles/644318/

https://lkml.org/lkml/2015/5/12/853

文中出现的一些名词说明：

主机： SCSI host，即对应的主机适配器。可能为 RAID 卡、FC HBA 等。

（本文档大致匹配 Linux 内核 v4.0）

本文档描述了 SCSI 命令 (struct scsi_cmnd) 生命周期的各个阶段，因为它包含了 SCSI中间层驱动程序 (SCSI mid level driver) 的不同部分。 它描述了在什么条件下以及如何中止、重试或安排 scsi_cmnd 进行错误处理，它是如何恢复 (recover) 的，以及 SCSI中间层驱动程序通常如何处理块请求。 它详细介绍了调用哪些函数以及每个函数的用途等。

为了用一个例子来帮助解释，它以一个 scsi_cmnd 的例子为例，它经历了这一切——超时（timeout）、中止 (abort)、错误处理 (error handling)、重试 (retry，也会导致 CHECK_CONDITION 并获取有意义的 sense 信息）。 最后一部分以示例 scsi_cmnd 跟踪在其生命周期中所可能出现的路径（不含底层部分）。

目录

[1] scsi_cmnd 的生命周期
[2] scsi_cmnd 如何排队到 LLD（底层驱动程序）进行处理?
[3] 一个 scsi_cmnd 是如何完成的?
  [3.1] 命令通过 scsi_softirq_done() 完成
  [3.2] 命令通过 scsi_times_out()$ 完成
[4] SCSI 错误处理
  [4.1] 我们是怎么来到这里的?
  [4.2] 错误处理什么时候实际运行?
  [4.3] SCSI 错误处理程序线程
[5] SCSI 命令可以被“劫持（hijacked）”
[6] SCSI 命令中止
  [6.1]中间层何时会尝试中止命令?
  [6.2] SCSI 命令中止如何工作?
  [6.3] 中止也可能失败
[7] SCSI 命令重试
  [7.1] 中级何时重试命令?
  [7.2] 重试资格标准
[8] 示例：遵循 scsi_cmnd（导致 CHECK_CONDITION）
  [8.1] 示例 scsi_cmnd 所用路径的高级视图
  [8.2] 实际采取的路径
[9] 参考资料

1. scsi_cmnd 的生命周期

SCSI Mid level 与块层的接口就像任何其他块驱动程序一样。 对于 SCSI中间层添加到系统中的每个块设备，它指示了一堆函数来服务于相应的请求队列。

以下函数在其生命周期内与 scsi_cmnd 相关。请注意，根据实际情况，它可能不会经历其中一些阶段，或者可能必须多次经历某些阶段。

scsi_prep_fn()
  由块层调用以准备请求。 这个函数实际上为请求分配一个新的 scsi_cmnd（来自 scsi_host->cmd_pool）并设置它。 这是 scsi_smnd “诞生”的地方。
  请注意，只有当 blk req 还没有与之关联时（req->special != NULL），才会分配一个新的 scsi_cmnd。 如果 SCSI 较早尝试过该请求，则该请求可能已经具有 scsi_cmnd，并且它导致决定稍后重试（因此请求被放回队列中）。

scsi_request_fn()
  是为请求队列提供服务的实际功能函数。 它简单地检查主机是否准备好接受新命令，如果是，它会将其提交给 LLD:

```
scsi_request_fn()
  ->scsi_dispatch_cmd()
      ->hostt->queue_command()
```

  如果 scsi_cmnd 由于某种原因无法排队到 LLD，请求将放回原始请求队列（等待稍后重试）。

scsi_softirq_done()
  是在 LLD 反馈命令完成后调用的处理程序。

```
scsi_done()
  ->blk_complete_request()
      ->导致软中断
          ->blk_done_softirq()
              ->scsi_softirq_done()
```

  此函数最重要的目标是确定此请求的进一步操作过程（基于 scsi_cmnd->result 和感知数据（如果存在）），并执行该过程。 相应的选项可以是完成对块层的请求、将其重新排队到块层、或安排它进行错误处理（如果认为有必要）。 稍后将对此进行更详细的讨论。

scsi_times_out()
  是在 LLD 长时间未响应 scsi_cmnd 的结果并且发生超时时调用的函数。 它会尝试查看是否可以通过 LLD 超时处理程序（如果可用）或中止命令来修复这种情况。 如果不是，它会为 EH 安排命令（稍后详细讨论）。

scsi_unprep_fn()
是被调用以取消准备请求的函数。 它应该撤消 scsi_prep_fn() 所做的任何事情。

2. scsi_cmnd 如何排队到 LLD 进行处理?

==========
提交部分非常简单。 一旦为块请求调用 scsi_request_fn() 并通过 blk_peek_request() 获取新的块请求，scsi_cmnd 已经设置并准备好发送到 LLD:

```
scsi_request_fn()
    ->scsi_dispatch_cmd()
        ->hostt->queue_command()
```

3. scsi_cmnd 是如何完成的?
==========
将 scsi_cmnd 提交给 LLD 后，只有两种方法可以完成它:

A。 要么 LLD 及时响应。
(即 结果为通过 scsi_softirq_done() 继续处理命令)

b. 或者，LLD没有及时响应，发生超时
(即 结果为通过 scsi_times_out() 继续处理命令)

我们在下面讨论这两种情况。

注 1:
可能有重试的 scsi_cmnd(s)。 但是重试的 scsi_cmnd 的完成与新 scsi_cmnd 的完成没有任何不同。 因此，无论重试的结果如何，scsi_cmnds 将始终使用上述两种情况之一结束。

注 2:
在 scsi_send_eh_cmnd() 的错误处理期间，scsi_cmnd 可能被“劫持”，以发送 EH 命令之一(TUR / STU / REQUEST_SENSE)。 但是，这些EH命令的完成并没有在以上两种场景中落地。 这是唯一的例外。 一旦 scsi_cmnd 被“解除劫持”，这个原始 scsi_cmnd 的结果仍将经历相同的 2 个场景之一。

3.1 命令通过 scsi_softirq_done() 完成
==========
当 LLD 及时响应时就是这种情况，例如命令正常完成。
请注意，这里的“完成”并不意味着命令已成功完成。事实上，SCSI 主机硬件甚至可能在没有接受命令的情况下发生故障。然而，scsi_softirq_done() 被调用的事实表明有及时可用的“结果”。我们必须检查这个结果才能决定下一步行动。

```
scsi_softirq_done()
|
+---> scsi_decide_disposition()
| 查看 scsi_cmnd->result 和感知 (sense) 数据以确定采取的最佳行动方案。
| 在阅读此函数代码时，不应将 SUCCESS 混淆为表示命令成功，或将 FAILED
| 混淆为表示命令失败等。此函数的返回值仅指示要采取的操作过程
|
+---> 结果为 SUCCESS 时:
| (完成到块层的命令。例如，设备可能处于离线状态，因此完成命令 - 块层
|  可能稍后自行重试，但这与 SCSI ML 无关)
| |
| +---> scsi_finish_command()
| |
| +---> scsi_io_completion() (*见下面的注释)
| |
| +---> blk_finish_request()
|
+---> 结果为 RETRY/ADD_TO_MLQUEUE 时:
| (重新排队请求队列的命令。例如，设备硬件是忙的，因此 SCSI ML 知道重试可能会
|  有帮助)
|
```

```
|  |
|  +---> scsi_queue_insert()
|  |
|  +---> blk_requeue_request()
|
+---> 结果为 FAILED/default 时：
       |   ( 安排 scsi_cmnd 进入 EH 。例如，存在可能需要总线重置的总线错误。
       |     或者我们得到 CHECK_CONDITION，我们需要发出 REQ_SENSE 以获取有
       |     关故障的更多信息等)
       |
       +---> scsi_eh_scmd_add()
             将 scsi_cmnd 添加到主机 EH 队列
                 scsi_eh_wakeup()
```

注3：
scsi_io_completion() 有一个类似于 scsi_decide_disposition() 的辅助逻辑，因为它也查看结果和感知数据并确定如何处理请求。它在要采取的行动过程中做出类似的选择。 此函数中有一个特殊情况涉及在重新排队之前“取消准备 (unprepping)”scsi_cmnd，我们将在下面的部分中讨论它。

3.2 命令通过 scsi_times_out() 完成

当 LLD 没有及时响应、块层超时，并因此调用相关 SCSI 设备的请求队列的超时函数时，就会发生这种情况。

```
scsi_times_out()
|
+---> scsi_transport_template->eh_timed_out() - 成功了吗？ 如果不...
|  (给transportt一个处理的机会)
|
+---> scsi_host_template->eh_timed_out() - 成功了吗？ 如果不...
|  (给hostt一个处理的机会)
|
+---> scsi_abort_command() - 成功了吗？ 如果不...
|  (安排 scsi_cmnd 的 ABORT。如果需要，中止处理程序还将重新排队)
|
+---> scsi_eh_scmd_add()
        (为 EH 安排 scsi_cmnd。这肯定有效。因为如果它不起作用，EH 处理程序会将设备标记为离线，这算是一个很好的修复:-))
```

4. SCSI 错误处理

SCSI 错误处理应该被认为是 ML 在知道仅仅重试请求可能无济于事并且需要做其他事情（可能是破坏性的）来解决问题时决定采取的行动。 例如 停滞的 (stalled) 主机可能需要主机重置，并且只有在此之后才能完成请求的重试。

注4：
（随机想法）：将“错误处理 (Error Handling)”与“重试 (Retries)”进行对比。重试是一件正常的事情，当中层认为它看到了一个本质上是暂时的错误，并且会自行消失而无需明确地做任何事情。因此在这种情况下再次重试请求是有意义的。（另一方面，一个 cmnd 被安排用于 EH，当它知道它需要在重试 cmnd 可以给出好的结果之前做“某事”时）。

注5：
SCSI 中间层使用 scsi_host->eh_cmd_q 维护一个（每个主机）列表，其中包含已在该主机上为 EH 安排的所有 scsi_cmnd。
这是 EH 线程在运行时处理的列表。

4.1 我们是如何到达这里的？

在以下情况下，可以将 scsi_cmnd 标记为 EH：

* 命令“错误完成 (error completed)”，即 scsi_decide_disposition() 返回 FAILED 或表示需要某种错误恢复的失败的东西。 例如 设备硬件出现故障，或者我们有 CHECK_CONDITION。
    scsi_softirq_done()
      ->scsi_decide_disposition = FAILED
          ->scsi_eh_scmd_add()

* scsi_cmnd 超时，尝试中止失败。
    scsi_times_out()
      ->scsi_abort_command() != SUCCESS
          ->scsi_eh_scmd_add()

4.2 错误处理什么时候真正运行?

只要有标记为 EH 的 scsi_smnd (插入到 Scsi_Host->eh_cmd_q 中)，就会调度 SCSI 错误处理程序线程。一旦 scsi_cmnd 被标记为 EH，ML 就不再接受该特定 Scsi_Host 的任何 scsi_cmnd。然而，EH 线程实际上并没有运行，直到该特定 Scsi_Host 的所有未决 IO 到 LLD 已完成或失败。换句话说，在该主机的 LLD 上挂起的唯一命令是需要 EH (host_busy == host_failed) 的命令。

这个想法是停止总线 (quiesce the bus)，以便 EH 线程可以恢复设备，因为它可能需要重置不同的组件才能完成其工作。

补注 1:
    这里的组件即 HBTL 四个层级。Host:Bus:Target:LUN 对应的 主机：总线：目标端：盘 四者倒序分别执行。
    在实际执行重置时，实际执行的操作除与实际问题相关外，还与相应的底层设备驱动 (LLD) 设计相关。

4.3 SCSI 错误处理器线程

```
scsi_error_handler()
|
+---> transportt->eh_strategy_handler() 如果存在，否则...
|  (如果可用，使用 transportt 自己的错误恢复处理程序)
|
+---> scsi_unjam_host()
|  (下面描述的 SCSI ML 错误处理程序。也在 Documentation/scsi/scsi_eh.txt 中描述。
|   基本目标是执行任何需要从当前错误条件中恢复的操作。并在恢复后重新排队符合条件
|   的命令)
|
+---> scsi_restart_operations()
       (重启SCSI请求队列的操作)
       |
       +---> scsi_run_host_queues()
             |
             +---> scsi_run_queue()
                   |
                   +---> blk_run_queue()
```

```
scsi_unjam_host()
```

这个想法是创建 2 个列表：work_q、done_q。
最初，work_q = <All EH scsi cmds>，done_q = NULL。然后依次递增采取可能恢复 cmnd 或设备的更高严重性的操作项，对 work_q 中的所有请求进行错误处理。不断将请求从 work_q 移动到 done_q，最后一次完成所有请求，而不是单独完成它们。

```
scsi_unjam_host()
|
+--> 创建 2 个列表：work_q、done_q
| work_q = <所有 EH scsi 命令>，done_q = NULL
|
+--> scsi_eh_get_sense() - 我们完成了吗? 如果不...
```

```
    |  (对于有CHECK_CONDITION的命令，获取 sense_info)
    |  |
    |  +--> scsi_request_sense()
    |  |  (使用 scsi_send_eh_cmnd() 发送“被劫持的 (hijacked)”REQ_SENSE 命令)
    |  |  |
    |  +--> scsi_decide_disposition()
    |  |  |
    |  +--> 如果成功 (SUCCESS) 则安排完成 scsi_cmnd (通过设置 retries=allowed)
    |  |
    |  +--> scsi_eh_abort_cmds() - 我们完成了吗？ 如果不...
    |  |  (中止超时的命令)
    |  |  |
    |  +--> scsi_try_to_abort_cmd()
    |  |  (导致调用 hostt->eh_abort_handler()，它负责使 LLD 和 HW 忘记 scsi_cmnd)
    |  |  |
    |  +--> scsi_eh_test_devices()
    |  |  (通过发送适当的 EH 命令 (STU / TEST_UNIT_READY) 测试设备现在是否响应。
    |  |   同样，发送这些 EH 命令涉及劫持原始 scsi_cmnd，然后恢复上下文)
    |  |
    |  +--> scsi_eh_ready_devs() - 我们完成了吗？ 如果不...
    |  |  (采取递增的顺序进行更高严重性的操作以恢复)
    |  |  |
    |  +--> scsi_eh_bus_device_reset()
    |  |  |  (重置 scsi_device。导致调用 hostt->eh_device_reset_handler())
    |  |  |  |
    |  +--> scsi_eh_target_reset()
    |  |  |  (重置 scsi_target。导致调用 hostt->eh_target_reset_handler())
    |  |  |  |
    |  +--> scsi_eh_bus_reset()
    |  |  |  (重置 scsi_bus。导致调用 hostt->eh_bus_reset_handler())
    |  |  |  |
    |  +--> scsi_eh_host_reset()
    |  |  |  (重置 Scsi_Host。导致调用 hostt->eh_host_reset_handler())
    |  |  |  |
    |  +--> 如果没有任何效果 - scsi_eh_offline_sdevs()
    |  |  (设备不可恢复，请下线)
    |  |
    |  +--> scsi_eh_flush_done_q()
    |       (对于 done_q 上的所有 EH 命令，如果符合条件，要么将它们重新排队
    |        (通过 scsi_queue_insert())，要么将它们完成到块层 (通过 scsi_finish_command())
```

注 6:
在每个恢复阶段，我们均需要测试是否已完成本阶段的错误恢复 (使用 scsi_eh_test_devices())，并仅在需要时才采取下一个更高严重性操作。

注 7:
错误处理程序会检查是否可以通过重置某一相同组件 (例如，相同的 scsi_device) 来恢复的多个 scsi_cmnd，如果可以则此组件只会重置一次。

5. SCSI命令可以被“劫持 (hijacked)”

如上所示，EH 线程可能需要发送一些 EH 命令才能检查 SCSI 设备的健康状况和响应能力：
* TUR - Test Unit Ready，测试单元就绪
* STU - Start / Stop Unit，启动/停止单元
* REQUEST_SENSE - 获取响应 CHECK_CONDITION 的 Sense 数据

然而，EH 线程并没有为这种临时目的分配和设置新的 scsi_cmnd，而是劫持它试图恢复的当前 scsi_cmnd，以便发送 EH 命令。整个过程在 scsi_send_eh_cmnd() 中完成。

scsi_send_eh_cmnd 在劫持它之前保存当前命令的上下文，在将它分发到 LLD 之前用它自己的替换 scsi_done ptr，并在完成后恢复上下文。以这种方式发送的 EH 命令会遇到相同的超时/中止失败/完成 - 但它们不会采用普通命令所采用的路由 (即不采用 scsi_softirq_done() 或 scsi_times_out() 路由)。每件事都在 scsi_send_eh_cmnd() 中处理。 这将在以下各节中讨论。

6. SCSI 命令中止

它指的是 SCSI中间层想要让 底层驱动程序和它下面的硬件忘记之前提供给 LLD 的 scsi_cmnd 的所有内容的场景。最常见的原因是 LLD 未能及时响应。

6.1 中间层何时会尝试中止命令?

在以下情况下，SCSI ML 可能会尝试中止 scsi_cmnd:

1. SCSI 中间层命令超时，并试图中止它。
   scsi_times_out()
     -> scsi_abort_command()
   如果中止失败会怎样? 为 EH 安排命令。

2. EH 线程在尝试解除主机阻塞时尝试中止所有挂起的命令。
   scsi_unjam_host()
     -> scsi_eh_abort_cmds()

   如果中止失败会怎样? 我们转向更高严重性的恢复步骤（开始重置 HW 组件等），因为这可能会导致 LLD 和 HW 忘记这些命令。

3. 这是一个讨厌的事。在错误恢复期间，EH 线程可能会“劫持”scsi_cmnd 以使用 scsi_send_eh_cmnd() 向 LLD 发送 EH 命令（TUR/STU/REQ_SENSE）。 如果此类“劫持”EH 命令超时，SCSI EH 线程将尝试中止它。
   scsi_send_eh_cmnd()
     -> scsi_abort_eh_cmnd()
          -> scsi_try_to_abort_cmd()

   如果中止失败会怎样? 与前一种情况类似，scsi_abort_eh_cmnd() 将尝试采取更高严重性的操作（重置总线等），但不会再次发送 EH 命令（例如 TUR 等）以验证设备是否开始响应。

6.2 SCSI 命令中止如何工作?

与像 TUR 这样的 EH 命令不同，ABORT 不是中间层驱动程序发送给 LLD 的 SCSI 命令。LLD 提供了一个 eh_abort_handler() 函数指针，用于中止命令。由 LLD 来做任何需要的事情来中止命令。 它可能需要向硬件发送一些专有命令，或者修改一些位，或者做任何必要的魔术更改。

6.3 中止也可能失败

与其他事情一样，中止尝试也可能失败。 SCSI 中间层在上一节中描述的情况下做正确的事情。

注8:
   一旦块层将命令交给 SCSI 子系统，块层目前无法取消/中止请求。这需要一些工作。

7. SCSI 命令重试

SCSI 中间层不为其正在处理的 SCSI 命令维护任何队列（EH 命令队列除外）。 因此，每当 SCSI ML 认为它需要重试命令时，它会将请求重新排队返回到相应的请求队列，以便在请求函数选择下一个请求进行处理时“自然地”进行重试。

当将此类请求要请求回到请求队列时，它们被放在请求队列的头部，以便它们在该队列中的其他（现有）请求之前。

7.1 中间层何时会重试命令?

以下是导致重试 SCSI 命令的条件（通过将 blk 请求放回请求队列）:

1. 中间层在 scsi_cmnd 上超时，成功中止并重新排队。
   scsi_times_out()
     -> scsi_abort_command()
          -> 调度 scmd_eh_abort_handler()

-> scsi_queue_insert()
-> blk_requeue_request()

2. EH 线程在恢复主机后，重新排队所有符合重试条件的 scsi_cmnd：
scsi_error_handler()
-> scsi_unjam_host()
-> scsi_eh_flush_done_q()
-> scsi_queue_insert()
-> blk_requeue_request()

3. LLD 完成 scsi_cmnd，scsi_decide_disposition() 查看 scsi_cmnd->result 并认为需要重试（例如，因为总线繁忙）。
scsi_softirq_done()
-> scsi_decide_disposition() 返回 NEEDS_RETRY
-> scsi_queue_insert()
-> blk_requeue_request()

4. 在 scsi_request_fn() 中，SCSI ML 发现主机正忙，scsi_cmnd 无法发送到 LLD，因此它将请求重新排回队列。
scsi_request_fn()
->案例note_ready:
-> blk_requeue_request()

5. scsi_finish_command() 可以从多个地方被调用以完成对块级别的请求。但是，它也可能会调用 scsi_io_completion() 来查看请求并决定重试（如果符合条件）。
scsi_finish_command()
-> scsi_io_completion()
-> __scsi_queue_insert()
-> blk_requeue_request()

注 9:
上面的情况 5 有一个非常特殊的情况。在某些情况下，scsi_io_completion() 决定必须重试 blk 请求，但是应该释放此请求的 scsi_cmnd，而是应该分配一个新的 scsi_cmnd 并在下一次重试时用于此请求。例如，情况可能就是这样：如果它看到 ILLEGAL REQUEST 作为对 READ10 命令的响应，并认为这可能是因为设备仅支持 READ6。因此，在下次重试时切换到 READ6（因此是一个新的 scsi_cmnd）可能是有意义的。

7.2 重试资格标准

请注意，SCSI 中间层总是在继续之前检查重试资格并重新排队命令以进行重试。 scsi_cmnd 的资格标准包括（其中一些可能不适用于上述所有情况）：

* retries < allowed (Num of retries should be less than allowed retries，即已重试的次数应少于允许的最大重试次数)
* 花费在 EH 的时间不超过 host->eh_deadline jiffies
* scsi_noretry_cmd() 应该为命令返回 0。
* scsi_device 必须在线
* req->timeout 不能过期
* 等等。。。

8. 示例：以一个 scsi_cmnd 过程为例

8.1 示例 scsi_cmnd 所用路径的高级视图

我们以块请求为例，例如想要从 scsi 磁盘读取块，但是 LBA 地址超出当前设备的范围（假设）。 ML 将其提交给 LLD，但 HW 接受命令并在其上阻塞（再次假设通过中止序列进行跟踪）。所以超时发生，ML 中止命令，并重新排队。在下一次运行中，LLD 使用 CHECK_CONDITION 完成命令。我们假设 SCSI 主机不会自动获取感知信息。ML 为 EH 调度 cmnd。EH线程发送REQUEST_SENSE 获取 sense info ILLEGAL_REQUEST，并据此完成对block层的请求。

8.2 实际采取的路径

Dispatched 派发：

```
scsi_request_fn()
|
+---> blk_peek_request()
| |
| +---> scsi_prep_fn()
|  (分配和设置 scsi_cmnd)
|
+---> scsi_dispatch_cmd()
       |
       +---> hostt->queue_command()
```

超时：

```
scsi_times_out()
|
+---> scsi_abort_command() - 返回成功 (SUCCESS)
       |
       +---> queue_delayed_work(abort_work)，队列延迟工作（中止工作）
```

中止处理程序：

```
scmd_eh_abort_handler()
|
+---> scsi_try_to_abort_cmd() - 返回成功 (SUCCESS)
| |
| +---> hostt->eh_abort_handler()
|
+---> scsi_queue_insert()
       |
       +---> __scsi_queue_insert()
             |
             +---> blk_requeue_request()
                    (req 被重新排队，req->special 指向 scsi_cmnd)
```

再次收到请求：

```
scsi_request_fn()
|
+---> blk_peek_request()
|  (req->cmd_flags 设置了 REQ_DONTPREP，因此不会再次调用 scsi_prep_fn())
|
+---> scsi_dispatch_cmd()
       |
       +---> hostt->queue_command()
```

命令以 CHECK_CONDITION 完成：

```
scsi_softirq_done()
|
+---> scsi_decide_disposition()
|  (查看 CHECK_CONDITION)
| |
| +---> scsi_check_sense() - 返回失败
| |
| +---> scsi_command_normalize_sense()
|  (未能找到有效的感知数据)
|
+---> 返回结果为 失败 (FAILED)：
       |
       +---> scsi_eh_scmd_add()
             将 scsi_cmnd 添加到主机 EH 队列
             |
             +---> scsi_eh_wakeup()
```

SCSI 错误处理程序线程运行以获取感知信息，并在其执行完成后完成请求。

```
scsi_error_handler()
|
+---> scsi_unjam_host()
       |
       +---> scsi_eh_get_sense()
       | |
       | +---> scsi_request_sense()
       | | |
       | | +---> scsi_send_eh_cmnd()
       | |  (劫持 smnd 发送 EH 命令)
       | | |
       | | +--> scsi_eh_prep_cmnd()
       | | |  (保存现有 scsi_cmndi 的上下文，分配 sence 感知数据缓冲区，
       | | |   并为 REQUEST_SENSE 设置 scsi_cmnd)
       | | |
       | | +--> hostt->queuecommand(), 然后等待...
       | | |  (获取 cmnd 的感知数据)
       | | |
       | | +--> scsi_eh_completed_normally() - 返回成功
       | | |
       | | +--> scsi_eh_restore_cmnd()
       | |  (恢复原始 scsi_cmnd 的上下文)
       | |
       | +---> scsi_decide_disposition() - 返回成功
       | |  (此时可以看到sense info)
       | |
       | +---> 设置 scmd->retries = scmd->allowed (避免重试)
       | |
       | +---> scsi_eh_finish_cmd()
       |  (将 scsi_cmnd 放在 done_q 上)
       |
       +---> scsi_eh_flush_done_q()
              (看到 scsi_cmnd 不符合重试条件)
             |
             +---> scsi_finish_command()
                    |
                    +---> scsi_io_completion()
                           |
                           +---> scsi_end_request()
                                  |
                                  +---> scsi_put_command()
                                         (释放 scsi_cmnd)
```

9. 参考文献
   ==========
   以下是极好的参考资料来源：
   Documentation/scsi/scsi_eh.txt
   http://events.linuxfoundation.org/sites/events/files/slides/SCSI-EH.pdf   <===
--

补注 2:
上述 SCSI-EH.pdf 原链接失效，根据内容推断，应为如下链接：
http://events17.linuxfoundation.org/sites/events/files/slides/SCSI-EH.pdf