1. 塊設備驅動(Block Device):

* 塊設備指能以塊(block)為單位進行讀寫訪問，塊是Linux虛擬文件系統（VFS）基本的數據傳輸單位，需要緩衝區。字元設備是以字節(byte)為單位進行數據傳輸的，不需要緩衝。
* 塊設備在結構上是可以進行隨機訪問的，對於這些設備的讀寫都是按塊進行的，塊設備使用緩衝區來暫時存放數據，等到條件成熟以後再一次性將緩衝區中的數據寫入塊設備中。這麼做的目的是為了提高塊設備的壽命。
* Linux內核使用block\_device表示塊設備，block\_device為一個結構體，定義在include/linux/fs.h文件中

1. 塊設備常見API函式:

* 和字元設備驅動一樣，我們需要向Kernel註冊新的塊設備、申請設備號，塊設備註冊函數為register\_blkdev。
* 和字元設備驅動一樣，如果不使用某個塊設備了，那麼就需要註銷掉，函數為unregister\_blkdev。
* Linux內核使用gendisk來描述一個磁盤設備，這是一個結構體，定義在include/linux/genhd.h中，此處列出比較重要的成員變數(如下表一)。

(表一: gendisk結構體成員變數)

|  |  |
| --- | --- |
| **成員變數** | **描述** |
| major | 為磁盤設備的主設備號。 |
| first\_minor | 為磁盤的第一個次設備號。 |
| minors | 為磁盤的次設備號數量，也就是磁盤的分區數量，這些分區的主設備號一樣，次設備號不同。 |
| part\_tbl | 為磁盤對應的分區表，為結構體 disk\_part\_tbl 類型，disk\_part\_tbl 的核心是一個 hd\_struct 結構體指針數組，此數組每一項都對應一個分區信息。 |
| fops | 為塊設備操作集，為 block\_device\_operations 結構體類型。和字元設備操作集 file\_operations 一樣，是塊設備驅動中的重點！ |
| queue | 為磁盤對應的請求隊列，所以針對該磁盤設備的請求都放到此隊列中，驅動程序需要處理此隊列中的所有請求。 |

1. gendisk常見API函式

* 使用gendisk之前要先申請alloc\_disk函數用於申請一個gendisk(如下表二)。
* 如果要刪除gendisk的話，可以使用函數del\_gendisk(如下表二)。
* 使用alloc\_disk申請到gendisk以後系統還不能使用，必須使用add\_disk函數將申請到的gendisk添加到內核中(如下表二)。
* 每一個磁盤都有容量，所以在初始化gendisk的時候也需要設置其容量，使用函數set\_capacity(如下表二)。

(表二: gendisk 常用API函式)

|  |  |
| --- | --- |
| **gendisk API函數** | **功能描述** |
| alloc\_disk | 用於申請一個gendisk。 |
| del\_gendisk | 用於刪除一個gendisk。 |
| add\_disk | 將申請到的gendisk添加到內核中，使系統可以使用。 |
| set\_capacity | 設置gendisk的容量。 |

1. block\_device\_operations結構體:

* 和字元設備的 file\_operations 一樣，塊設備也有操作集，為結構體 block\_device\_operations。此結構體定義在 include/linux/blkdev.h中。結構體中常見函式及功能如下表三。

(表三: block\_device\_operations結構體中常見函式及功能)

|  |  |
| --- | --- |
| **函数** | **描述** |
| open | 用於打開指定的塊設備 |
| release | 用於關閉 (釋放) 指定的塊設備 |
| rw\_page | 用於讀寫指定的頁 |
| ioctl | 用於塊設備的 I/O 控制 |
| compat\_ioctl | 和 ioctl 函數一樣，用於塊設備的 I/O 控制 |
| getgeo | 用於獲取磁盤信息，包括磁頭、柱面和扇區等信息 |
| owner | 表示此結構體屬於哪個模塊，一般直接設置為 THIS\_MODULE |

(\*註:塊設備沒有write()/read()函式，只要是透過request\_queue、request和bio來進行數據讀寫。)

1. 塊設備 I/O 請求過程:

塊設備沒有write()/read()函式，只要是透過request\_queue、request和bio來進行數據讀寫。

1. 請求隊列 request\_queue:

* Kernel將對塊設備的讀寫請求發送到請求隊列request\_queue中。request\_queue中包含大量的請求（request結構體），每個request又包含了bio。
* bio保存了讀寫相關數據，例如讀取的起始地址、數據長度、目的地，如果是寫操作還包括要寫入的數據。
* request\_queue是一個結構體，定義在文件include/linux/blkdev.h中。在gendisk結構體中有一個request\_queue結構體指針成員變量 queue。編寫塊設備驅動時，每個磁盤（gendisk）都需要分配一個 request\_queue。

1. 請求 request:

* 請求隊列(request\_queue)包含一系列的請求(request)。request 是一個結構體，定義在include/linux/blkdev.h裡面，內含一個名為bio 的成員變量，類型為bio結構體指針。真正的數據保存在bio 裡面。
* 需要從request\_queue中取出request，再從每個request中取出bio，最後根據bio的描述將數據寫入到塊設備，或從塊設備中讀取數據。

1. bio:

* 每個request包含多個bio，bio保存著最終要讀寫的數據、地址等信息。
* 上層應用程序對塊設備的讀寫會被構造成一個或多個bio結構，描述要讀寫的起始扇區、扇區數量、讀取或寫入、頁偏移、數據長度等。上層將bio提交給I/O調度器，I/O調度器將這些bio構造成request結構，順序存放在request\_queue中。新
* 產生的bio可能被合併到現有的request中，也可能生成新的request，並插入到合適的位置，這一切由I/O調度器完成。

1. 請求列隊request\_queue(常見API):
2. 初始化請求列隊(分為兩部分):

* 第一部分是創建blk\_mq\_tag\_set結構體，然後使用blk\_mq\_alloc\_tag\_set函數初始化blk\_mq\_tag\_set對象(如下表四)。
* 第二部分使用blk\_mq\_init\_queue函數獲取request\_queue。

(表四:常見blk\_mq\_tag\_set結構體成員變數)

|  |  |
| --- | --- |
| **結構體成員變數** | **描述** |
| nr\_maps | 映射表數量 |
| ops | 驅動實現的操作集合，會被 request\_queue 繼承 |
| nr\_hw\_queues | 硬體隊列個數 |
| queue\_depth | 隊列深度 |
| numa\_node | 所在 numa 節點 |

* 編寫塊設備的請求隊列驅動的時候需要分配並初始化一個 blk\_mq\_tag\_set。Linux Kernel提供了一組blk\_mq\_tag\_set操作相關的函數。(如下表五)
* 給 blk\_mq\_tag\_set對象賦值後，要使用blk\_mq\_alloc\_tag\_set函數為一個或多個請求隊列分配tag和request集合。(如下表五)
* 如果要釋放請求隊列中的tag集合，可以使用 blk\_mq\_free\_tag\_set。(如下表五)
* Linux Kernel也提供了一步創建request\_queue隊列的函數blk\_mq\_init\_sq\_queue，使用此函數可以一步創建請求隊列。(如下表五)
* 當卸載塊設備驅動的時候，我們還需要刪除掉前面申請到的 request\_queue，刪除請求隊列使用函數blk\_cleanup\_queue。(如下表五)
* blk\_mq\_init\_queue函數完成了請求隊列的申請以及請求處理函數的綁定，這個一般用於像機械硬盤這樣的存儲設備，需要 I/O 調度器來優化數據讀寫過程。(如下表五)
* 但是對於 EMMC、SD 卡這樣的非機械設備，可以進行完全隨機訪問，所以就不需要複雜的I/O調度器了。對於非機械設備，我們可以先申請request\_queue，然後將申請到的 request\_queue 與“製造請求”函數綁定在一起。透過request\_queue申請函數 blk\_alloc\_queue。(如下表五)
* 我們需要為申請到的請求隊列綁定一個製造請求函數。這裡我們需要用到函數blk\_queue\_make\_request。(如下表五)

(表五:常用request\_queue API 函數)

|  |  |
| --- | --- |
| **request\_queue(API 函數)** | **功能** |
| blk\_mq\_tag\_set | 分配並初始化塊設備的請求隊列驅動時需要的 blk\_mq\_tag\_set 結構體 |
| blk\_mq\_alloc\_tag\_set | 為一個或多個請求隊列分配 tag 和 request 集合 |
| blk\_mq\_free\_tag\_set | 釋放請求隊列中的 tag 集合 |
| blk\_mq\_init\_sq\_queue | 一步創建 request\_queue 隊列 |
| blk\_cleanup\_queue | 刪除前面申請到的 request\_queue |
| blk\_mq\_init\_queue | 完成請求隊列的申請以及請求處理函數的綁定，一般用於需要 I/O 調度器優化數據讀寫過程的存儲設備 |
| blk\_alloc\_queue | 申請request\_queue，特別是對於不需要複雜I/O調度器的非機械設備(如EMMC，SD卡…etc) |
| blk\_queue\_make\_request | 為申請到的請求隊列綁定一個製造請求函數 |

1. 請求request(常見API):

* 前面說過請求隊列(request\_queue)包含一系列的請求(request)。request 是一個結構體，內含一個名為bio 的成員變量，類型為bio結構體指針。而真正的數據保存在bio 裡面。
  + 當有請求處理的時候，我們要用 blk\_mq\_start\_request 函數開啟請求處理。(如下表六)
  + 我們不用處理請求的時候，要使用 blk\_mq\_end\_request 函數結束請求處理。(如下表六)

(表六: request常用API)

|  |  |
| --- | --- |
| **request(常用API)** | **功能** |
| blk\_mq\_start\_request | 開啟請求處理 |
| blk\_mq\_end\_request | 結束請求處理 |

1. bio:

* 上層應用程序對於塊設備的讀寫會被構造成一個或多個bio結構，bio結構描述了要讀寫的起始扇區、要讀寫的扇區數量、是讀取還是寫入、頁偏移、數據長度等等信息。
* 上層會將bio提交給I/O調度器，I/O調度器會將這些bio建構成request 結構，request\_queue裡面順序存放著一系列的request。
* 新產生的bio可能被合併到request\_queue裡現有的request中，也可能產生新的request，然後插入到request\_queue中合適的位置，這一切都是由 I/O 調度器來完成的。
* bio是個結構體，定義在 include/linux/blk\_types.h中，我們的重點放在bvec\_iter結構體類型的成員變量，bio\_vec結構體指針類型的成員變量。主要功能參考如下表七。

(表七: bvec\_iter和bio\_vec結構體功能)

|  |  |
| --- | --- |
| **結構體** | **主要功能** |
| bvec\_iter | 描述和追踪在處理 bio 時的進度，包括bio\_vec的索引、偏移量和剩餘字節數。 |
| bio\_vec | 描述單個 I/O 向量的內存佈局，包括指向內存頁面的指針、頁面內的偏移量和長度。 |

※總結:

以上就是對塊設備基本介紹，接下來會針對請求列隊，以及不使用請求列隊來熟悉相關API應用。