程序 9-1 linux/kernel/blk_drv/blk.h

```
 1 #ifndef _BLK_H
 2 #define _BLK_H
 3
 4 #define NR_BLK_DEV      7          // 块设备类型数量。
 5 /*
 6  * NR_REQUEST is the number of entries in the request-queue.
 7  * NOTE that writes may use only the low 2/3 of these: reads
 8  * take precedence.
 9  *
10  * 32 seems to be a reasonable number: enough to get some benefit
11  * from the elevator-mechanism, but not so much as to lock a lot of
12  * buffers when they are in the queue. 64 seems to be too many (easily
13  * long pauses in reading when heavy writing/syncing is going on)
14  */
   /*
    * 下面定义的 NR_REQUEST 是请求队列中所包含的项数。
    * 注意，写操作仅使用这些项中低端的 2/3 项；读操作优先处理。
    *
    * 32 项好象是一个合理的数字：该数已经足够从电梯算法中获得好处，
    * 但当缓冲区在队列中而锁住时又不显得是很大的数。64 就看上去太
    * 大了（当大量的写/同步操作运行时很容易引起长时间的暂停）。
    */
15 #define NR_REQUEST      32
16
17 /*
18  * Ok, this is an expanded form so that we can use the same
19  * request for paging requests when that is implemented. In
20  * paging, 'bh' is NULL, and 'waiting' is used to wait for
21  * read/write completion.
22  */
   /*
    * OK，下面是 request 结构的一个扩展形式，因而当实现以后，我们
    * 就可以在分页请求中使用同样的 request 结构。 在分页处理中，
    * 'bh'是 NULL，而'waiting'则用于等待读/写的完成。
    */
   // 下面是请求队列中项的结构。其中如果字段 dev = -1，则表示队列中该项没有被使用。
   // 字段 cmd 可取常量 READ（0）或 WRITE（1）（定义在 include/linux/fs.h 中）。
   // 其中，内核并没有用到 waiting 指针，起而代之地内核使用了缓冲块的等待队列。因为
   // 等待一个缓冲块与等待请求项完成是对等的。
23 struct request {
24         int dev;                    /* -1 if no request */  // 发请求的设备号。
25         int cmd;                    /* READ or WRITE */     // READ 或 WRITE 命令。
26         int errors;                  //操作时产生的错误次数。
27         unsigned long sector;        // 起始扇区。(1 块=2 扇区)
28         unsigned long nr_sectors;    // 读/写扇区数。
29         char * buffer;               // 数据缓冲区。
30         struct task_struct * waiting;   // 任务等待请求完成操作的地方（队列）。
31         struct buffer_head * bh;     // 缓冲区头指针(include/linux/fs.h,68)。
32         struct request * next;       // 指向下一请求项。
33 };
34
35 /*
```

```
 * This is used in the elevator algorithm: Note that
 * reads always go before writes. This is natural: reads
 * are much more time-critical than writes.
 */
/*
 * 下面的定义用于电梯算法：注意读操作总是在写操作之前进行。
 * 这是很自然的：读操作对时间的要求要比写操作严格得多。
 */
// 下面宏中参数 s1 和 s2 的取值是上面定义的请求结构 request 的指针。该宏定义用于根据两个参数
// 指定的请求项结构中的信息（命令 cmd（READ 或 WRITE）、设备号 dev 以及所操作的扇区号 sector）
// 来判断出两个请求项结构的前后排列顺序。这个顺序将用作访问块设备时的请求项执行顺序。
// 这个宏会在程序 blk_drv/ll_rw_blk.c 中函数 add_request()中被调用（第 96 行）。该宏部分
// 地实现了 I/O 调度功能，即实现了对请求项的排序功能（另一个是请求项合并功能）。
#define IN_ORDER(s1,s2) \
((s1)->cmd<(s2)->cmd || (s1)->cmd==(s2)->cmd && \
((s1)->dev < (s2)->dev || ((s1)->dev == (s2)->dev && \
(s1)->sector < (s2)->sector)))

// 块设备处理结构。
struct blk_dev_struct {
        void (*request_fn)(void);                       // 请求处理函数指针。
        struct request * current_request;               // 当前处理的请求结构。
};

// 块设备表（数组）。每种块设备占用一项，共 7 项。
extern struct blk_dev_struct blk_dev[NR_BLK_DEV];
// 请求队列数组，共 32 项。
extern struct request request[NR_REQUEST];
// 等待空闲请求项的进程队列头指针。
extern struct task_struct * wait_for_request;

// 设备数据块总数指针数组。每个指针项指向指定主设备号的总块数数组 hd_sizes[]。该总
// 块数数组每一项对应子设备号确定的一个子设备上所拥有的数据块总数（1 块大小 = 1KB）。
extern int * blk_size[NR_BLK_DEV];

// 在块设备驱动程序（如 hd.c）包含此头文件时，必须先定义驱动程序处理设备的主设备号。
// 这样，在下面 63 行—90 行就能为包含本文件的驱动程序给出正确的宏定义。
#ifdef MAJOR_NR                     // 主设备号。

/*
 * Add entries as needed. Currently the only block devices
 * supported are hard-disks and floppies.
 */
/*
 * 需要时加入条目。目前块设备仅支持硬盘和软盘（还有虚拟盘）。
 */

// 如果定义了 MAJOR_NR = 1（RAM 盘主设备号），就是用以下符号常数和宏。
#if (MAJOR_NR == 1)
/* ram disk */
#define DEVICE_NAME "ramdisk"               // 设备名称（“内存虚拟盘”）。
#define DEVICE_REQUEST do_rd_request        // 设备请求项处理函数。
#define DEVICE_NR(device) ((device) & 7)    // 设备号（0 - 7）。
```

```
68 #define DEVICE_ON(device)                       // 开启设备（虚拟盘无须开启和关闭）。
69 #define DEVICE_OFF(device)                      // 关闭设备。
70
   // 否则，如果定义了 MAJOR_NR = 2（软驱主设备号），就是用以下符号常数和宏。
71 #elif (MAJOR_NR == 2)
72 /* floppy */
73 #define DEVICE_NAME "floppy"                    // 设备名称（“软盘驱动器”）。
74 #define DEVICE_INTR do_floppy                   // 设备中断处理函数。
75 #define DEVICE_REQUEST do_fd_request            // 设备请求项处理函数。
76 #define DEVICE_NR(device) ((device) & 3)        // 设备号（0 - 3）。
77 #define DEVICE_ON(device) floppy_on(DEVICE_NR(device))    // 开启设备宏。
78 #define DEVICE_OFF(device) floppy_off(DEVICE_NR(device))  // 关闭设备宏。
79
   // 否则，如果定义了 MAJOR_NR = 3（硬盘主设备号），就是用以下符号常数和宏。
80 #elif (MAJOR_NR == 3)
81 /* harddisk */
82 #define DEVICE_NAME "harddisk"                  // 设备名称（“硬盘”）。
83 #define DEVICE_INTR do_hd                       // 设备中断处理函数。
84 #define DEVICE_TIMEOUT hd_timeout               // 设备超时值。
85 #define DEVICE_REQUEST do_hd_request            // 设备请求项处理函数。
86 #define DEVICE_NR(device) (MINOR(device)/5)     // 设备号。
87 #define DEVICE_ON(device)                       // 开启设备。
88 #define DEVICE_OFF(device)                      // 关闭设备。
89
   // 否则在编译预处理阶段显示出错信息：“未知块设备”。
90 #elif
91 /* unknown blk device */
92 #error "unknown blk device"
93
94 #endif
95
   // 为了便于编程表示，这里定义了两个宏：CURRENT 是指定住设备号的当前请求结构项指针，
   // CURRENT_DEV 是当前请求项 CURRENT 中设备号。
96 #define CURRENT (blk_dev[MAJOR_NR].current_request)
97 #define CURRENT_DEV DEVICE_NR(CURRENT->dev)
98
   // 如果定义了设备中断处理符号常数，则把它声明为一个函数指针，并默认为 NULL。
99 #ifdef DEVICE_INTR
100 void (*DEVICE_INTR)(void) = NULL;
101 #endif
   // 如果定义了设备超时符号常数，则令其值等于 0，并定义 SET_INTR()宏。否则只定义宏。
102 #ifdef DEVICE_TIMEOUT
103 int DEVICE_TIMEOUT = 0;
104 #define SET_INTR(x) (DEVICE_INTR = (x),DEVICE_TIMEOUT = 200)
105 #else
106 #define SET_INTR(x) (DEVICE_INTR = (x))
107 #endif
   // 声明设备请求符号常数 DEVICE_REGUEST 是一个不带参数并无反回的静态函数指针。
108 static void (DEVICE_REQUEST)(void);
109
   // 解锁指定的缓冲块。
   // 如果指定缓冲块 bh 并没有被上锁，则显示警告信息。否则将该缓冲块解锁，并唤醒等待
   // 该缓冲块的进程。此为内嵌函数。参数是缓冲块头指针。
```

```
110 extern inline void unlock_buffer(struct buffer_head * bh)
111 {
112         if (!bh->b_lock)
113                 printk(DEVICE_NAME ": free buffer being unlocked\n");
114         bh->b_lock=0;
115         wake_up(&bh->b_wait);
116 }
117
    // 结束请求处理。
    // 参数 uptodate 是更新标志。
    // 首先关闭指定块设备，然后检查此次读写缓冲区是否有效。如果有效则根据参数值设置缓冲
    // 区数据更新标志，并解锁该缓冲区。 如果更新标志参数值是 0，表示此次请求项的操作已失
    // 败，因此显示相关块设备 IO 错误信息。 最后，唤醒等待该请求项的进程以及等待空闲请求
    // 项出现的进程，释放并从请求链表中删除本请求项，并把当前请求项指针指向下一请求项。
118 extern inline void end_request(int uptodate)
119 {
120         DEVICE_OFF(CURRENT->dev);                       // 关闭设备。
121         if (CURRENT->bh) {                              // CURRENT 为当前请求结构项指针。
122                 CURRENT->bh->b_uptodate = uptodate;     // 置更新标志。
123                 unlock_buffer(CURRENT->bh);             // 解锁缓冲区。
124         }
125         if (!uptodate) {                                // 若更新标志为 0 则显示出错信息。
126                 printk(DEVICE_NAME " I/O error\n\r");
127                 printk("dev %04x, block %d\n\r",CURRENT->dev,
128                         CURRENT->bh->b_blocknr);
129         }
130         wake_up(&CURRENT->waiting);                     // 唤醒等待该请求项的进程。
131         wake_up(&wait_for_request);                     // 唤醒等待空闲请求项的进程。
132         CURRENT->dev = -1;                              // 释放该请求项。
133         CURRENT = CURRENT->next;                        // 指向下一请求项。
134 }
135
    // 如果定义了设备超时符号常量 DEVICE_TIMEOUT，则定义 CLEAR_DEVICE_TIMEOUT 符号常量
    // 为“DEVICE_TIMEOUT = 0”。否则定义 CLEAR_DEVICE_TIMEOUT 为空。
136 #ifdef DEVICE_TIMEOUT
137 #define CLEAR_DEVICE_TIMEOUT DEVICE_TIMEOUT = 0;
138 #else
139 #define CLEAR_DEVICE_TIMEOUT
140 #endif
141
    // 如果定义了设备中断符号常量 DEVICE_INTR，则定义 CLEAR_DEVICE_INTR 符号常量为
    // “DEVICE_INTR = 0”，否则定义其为空。
142 #ifdef DEVICE_INTR
143 #define CLEAR_DEVICE_INTR DEVICE_INTR = 0;
144 #else
145 #define CLEAR_DEVICE_INTR
146 #endif
147
    // 定义初始化请求项宏。
    // 由于几个块设备驱动程序开始处对请求项的初始化操作相似，因此这里为它们定义了一个
    // 统一的初始化宏。该宏用于对当前请求项进行一些有效性判断。所做工作如下：
    // 如果设备当前请求项为空（NULL），表示本设备目前已无需要处理的请求项。于是略作扫尾
    // 工作就退出相应函数。否则，如果当前请求项中设备的主设备号不等于驱动程序定义的主设
```

```
    // 备号，说明请求项队列乱掉了，于是内核显示出错信息并停机。否则若请求项中用的缓冲块
    // 没有被锁定，也说明内核程序出了问题，于是显示出错信息并停机。
148 #define INIT_REQUEST \
149 repeat: \
150         if (!CURRENT) {\                        // 如果当前请求项指针为 NULL 则返回。
151                 CLEAR_DEVICE_INTR \
152                 CLEAR_DEVICE_TIMEOUT \
153                 return; \
154         } \
155         if (MAJOR(CURRENT->dev) != MAJOR_NR) \   // 如果当前设备主设备号不对则停机。
156                 panic(DEVICE_NAME ": request list destroyed"); \
157         if (CURRENT->bh) { \
158                 if (!CURRENT->bh->b_lock) \      // 如果请求项的缓冲区没锁定则停机。
159                         panic(DEVICE_NAME ": block not locked"); \
160         }
161
162 #endif
163
164 #endif
165
```