程序 12-8 linux/fs/block_dev.c

```
1  /*
2   *  linux/fs/block_dev.c
3   *
4   *  (C) 1991  Linus Torvalds
5   */
6
7  #include <errno.h>        // 错误号头文件。包含系统中各种出错号。
8
9  #include <linux/sched.h>  // 调度程序头文件，定义了任务结构 task_struct、任务 0 的数据，
                             // 还有一些有关描述符参数设置和获取的嵌入式汇编函数宏语句。
10 #include <linux/kernel.h> // 内核头文件。含有一些内核常用函数的原形定义。
11 #include <asm/segment.h>  // 段操作头文件。定义了有关段寄存器操作的嵌入式汇编函数。
12 #include <asm/system.h>   // 系统头文件。定义了设置或修改描述符/中断门等的嵌入式汇编宏。
13
   // 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）。
14 extern int *blk_size[];              // blk_drv/ll_rw_blk.c，49 行。
15
   //// 数据块写函数 - 向指定设备从给定偏移处写入指定长度数据。
   // 参数：dev - 设备号；pos - 设备文件中偏移量指针；buf - 用户空间中缓冲区地址；
   // count - 要传送的字节数。
   // 返回已写入字节数。若没有写入任何字节或出错，则返回出错号。
   // 对于内核来说，写操作是向高速缓冲区中写入数据。什么时候数据最终写入设备是由高速缓
   // 冲管理程序决定并处理的。另外，因为块设备是以块为单位进行读写，因此对于写开始位置
   // 不处于块起始处时，需要先将开始字节所在的整个块读出，然后将需要写的数据从写开始处
   // 填写满该块，再将完整的一块数据写盘（即交由高速缓冲程序去处理）。
16 int block_write(int dev, long * pos, char * buf, int count)
17 {
   // 首先由文件中位置 pos 换算成开始读写盘块的块序号 block，并求出需写第 1 字节在该块中
   // 的偏移位置 offset。
18         int block = *pos >> BLOCK_SIZE_BITS;          // pos 所在文件数据块号。
19         int offset = *pos & (BLOCK_SIZE-1);           // pos 在数据块中偏移值。
20         int chars;
21         int written = 0;
22         int size;
23         struct buffer_head * bh;
24         register char * p;                    // 局部寄存器变量，被存放在寄存器中。
25
   // 在写一个块设备文件时，要求写的总数据块数当然不能超过指定设备上容许的最大数据块总
   // 数。因此这里首先取出指定设备的块总数 size 来比较和限制函数参数给定的写入数据长度。
   // 如果系统中没有对设备指定长度，就使用默认长度 0x7fffffff（2GB 个块）。
26         if (blk_size[MAJOR(dev)])
27                 size = blk_size[MAJOR(dev)][MINOR(dev)];
28         else
29                 size = 0x7fffffff;
   // 然后针对要写入的字节数 count，循环执行以下操作，直到数据全部写入。在循环执行过程
   // 中，若当前写入数据的块号已经大于或等于指定设备的总块数，则返回已写字节数并退出。
   // 然后再计算在当前处理的数据块中可写入的字节数。如果需要写入的字节数填不满一块，那
   // 么就只需写 count 字节。如果正好要写 1 块数据内容，则直接申请 1 块高速缓冲块，并把用
   // 户数据放入即可。否则就需要读入将被写入部分数据的数据块，并预读下两块数据。然后将
   // 块号递增 1，为下次操作做好准备。如果缓冲块操作失败，则返回已写字节数，如果没有写
   // 入任何字节，则返回出错号（负数）。
```

```
30          while (count>0) {
31                  if (block >= size)
32                          return written?written:-EIO;
33                  chars = BLOCK_SIZE - offset;            // 本块可写入的字节数。
34                  if (chars > count)
35                          chars=count;
36                  if (chars == BLOCK_SIZE)
37                          bh = getblk(dev,block);         // buffer.c 第 206、322 行。
38                  else
39                          bh = breada(dev,block,block+1,block+2,-1);
40                  block++;
41                  if (!bh)
42                          return written?written:-EIO;
    // 接着先把指针 p 指向读出数据的缓冲块中开始写入数据的位置处。若最后一次循环写入的数
    // 据不足一块，则需从块开始处填写（修改）所需的字节，因此这里需预先设置 offset 为零。
    // 此后将文件中偏移指针 pos 前移此次将要写的字节数 chars，并累加这些要写的字节数到统
    // 计值 written 中。再把还需要写的计数值 count 减去此次要写的字节数 chars。然后我们从
    // 用户缓冲区复制 chars 个字节到 p 指向的高速缓冲块中开始写入的位置处。复制完后就设置
    // 该缓冲区块已修改标志，并释放该缓冲区（也即该缓冲区引用计数递减 1）。
43                  p = offset + bh->b_data;
44                  offset = 0;
45                  *pos += chars;
46                  written += chars;                       // 累计写入字节数。
47                  count -= chars;
48                  while (chars-->0)
49                          *(p++) = get_fs_byte(buf++);
50                  bh->b_dirt = 1;
51                  brelse(bh);
52          }
53          return written;                                 // 返回已写入的字节数，正常退出。
54  }
55
    //// 数据块读函数 - 从指定设备和位置处读入指定长度数据到用户缓冲区中。
    // 参数：dev - 设备号；pos - 设备文件中偏移量指针；buf - 用户空间中缓冲区地址；
    // count - 要传送的字节数。
    // 返回已读入字节数。若没有读入任何字节或出错，则返回出错号。
56  int block_read(int dev, unsigned long * pos, char * buf, int count)
57  {
58          int block = *pos >> BLOCK_SIZE_BITS;
59          int offset = *pos & (BLOCK_SIZE-1);
60          int chars;
61          int size;
62          int read = 0;
63          struct buffer_head * bh;
64          register char * p;                      // 局部寄存器变量，被存放在寄存器中。
65
    // 在读一个块设备文件时，要求读的总数据块数当然不能超过指定设备上容许的最大数据块总
    // 数。因此这里首先取出指定设备的块总数 size 来比较和限制函数参数给定的读入数据长度。
    // 如果系统中没有对设备指定长度，就使用默认长度 0x7fffffff（2GB 个块）。
66          if (blk_size[MAJOR(dev)])
67                  size = blk_size[MAJOR(dev)][MINOR(dev)];
68          else
69                  size = 0x7fffffff;
```

```
    // 然后针对要读入的字节数 count，循环执行以下操作，直到数据全部读入。在循环执行过程
    // 中，若当前读入数据的块号已经大于或等于指定设备的总块数，则返回已读字节数并退出。
    // 然后再计算在当前处理的数据块中需读入的字节数。如果需要读入的字节数还不满一块，那
    // 么就只需读 count 字节。然后调用读块函数 breada()读入需要的数据块，并预读下两块数据，
    // 如果读操作出错，则返回已读字节数，如果没有读入任何字节，则返回出错号。然后将块号
    // 递增 1。为下次操作做好准备。如果缓冲块操作失败，则返回已写字节数，如果没有读入任
    // 何字节，则返回出错号（负数）。
70          while (count>0) {
71                  if (block >= size)
72                          return read?read:-EIO;
73                  chars = BLOCK_SIZE-offset;
74                  if (chars > count)
75                          chars = count;
76                  if (!(bh = breada(dev,block,block+1,block+2,-1)))
77                          return read?read:-EIO;
78                  block++;
    // 接着先把指针 p 指向读出盘块的缓冲块中开始读入数据的位置处。若最后一次循环读操作的
    // 数据不足一块，则需从块起始处读取所需字节，因此这里需预先设置 offset 为零。 此后将
    // 文件中偏移指针 pos 前移此次将要读的字节数 chars，并且累加这些要读的字节数到统计值
    // read 中。再把还需要读的计数值 count 减去此次要读的字节数 chars。然后我们从高速缓冲
    // 块中 p 指向的开始读的位置处复制 chars 个字节到用户缓冲区中，同时把用户缓冲区指针前
    // 移。本次复制完后就释放该缓冲块。
79                  p = offset + bh->b_data;
80                  offset = 0;
81                  *pos += chars;
82                  read += chars;                  // 累计读入字节数。
83                  count -= chars;
84                  while (chars-->0)
85                          put_fs_byte(*(p++),buf++);
86                  brelse(bh);
87          }
88          return read;                            // 返回已读取的字节数，正常退出。
89  }
90
```