程序 12-12 linux/fs/read_write.c

```
1  /*
2   *  linux/fs/read_write.c
3   *
4   *  (C) 1991  Linus Torvalds
5   */
6
7  #include <sys/stat.h>     // 文件状态头文件。含有文件或文件系统状态结构 stat{}和常量。
8  #include <errno.h>        // 错误号头文件。包含系统中各种出错号。
9  #include <sys/types.h>    // 类型头文件。定义了基本的系统数据类型。
10
11 #include <linux/kernel.h> // 内核头文件。含有一些内核常用函数的原形定义。
12 #include <linux/sched.h>  // 调度程序头文件，定义任务结构 task_struct、任务 0 数据等。
13 #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。
14
   // 字符设备读写函数。fs/char_dev.c，第 95 行。
15 extern int rw_char(int rw,int dev, char * buf, int count, off_t * pos);
   // 读管道操作函数。fs/pipe.c，第 13 行。
16 extern int read_pipe(struct m_inode * inode, char * buf, int count);
   // 写管道操作函数。fs/pipe.c，第 41 行。
17 extern int write_pipe(struct m_inode * inode, char * buf, int count);
   // 块设备读操作函数。fs/block_dev.c，第 47 行。
18 extern int block_read(int dev, off_t * pos, char * buf, int count);
   // 块设备写操作函数。fs/block_dev.c，第 14 行。
19 extern int block_write(int dev, off_t * pos, char * buf, int count);
   // 读文件操作函数。fs/file_dev.c，第 17 行。
20 extern int file_read(struct m_inode * inode, struct file * filp,
21                 char * buf, int count);
   // 写文件操作函数。fs/file_dev.c，第 48 行。
22 extern int file_write(struct m_inode * inode, struct file * filp,
23                 char * buf, int count);
24
   //// 重定位文件读写指针系统调用。
   // 参数 fd 是文件句柄，offset 是新的文件读写指针偏移值，origin 是偏移的起始位置，可有
   // 三种选择：SEEK_SET（0，从文件开始处）、SEEK_CUR（1，从当前读写位置）、SEEK_END（
   // 2，从文件尾处）。
25 int sys_lseek(unsigned int fd,off_t offset, int origin)
26 {
27         struct file * file;
28         int tmp;
29
   // 首先判断函数提供的参数有效性。如果文件句柄值大于程序最多打开文件数 NR_OPEN(20)，
   // 或者该句柄的文件结构指针为空，或者对应文件结构的 i 节点字段为空，或者指定设备文件
   // 指针是不可定位的，则返回出错码并退出。如果文件对应的 i 节点是管道节点，则返回出错
   // 码退出。因为管道头尾指针不可随意移动!
30         if (fd >= NR_OPEN || !(file=current->filp[fd]) || !(file->f_inode)
31             || !IS_SEEKABLE(MAJOR(file->f_inode->i_dev)))
32                 return -EBADF;
33         if (file->f_inode->i_pipe)
34                 return -ESPIPE;
   // 然后根据设置的定位标志，分别重新定位文件读写指针。
35         switch (origin) {
   // origin = SEEK_SET，要求以文件起始处作为原点设置文件读写指针。若偏移值小于零，则出
```

```
// 错返回错误码。否则设置文件读写指针等于 offset。
36                  case 0:
37                          if (offset<0) return -EINVAL;
38                          file->f_pos=offset;
39                          break;
// origin = SEEK_CUR，要求以文件当前读写指针处作为原点重定位读写指针。如果文件当前指
// 针加上偏移值小于 0，则返回出错码退出。否则在当前读写指针上加上偏移值。
40                  case 1:
41                          if (file->f_pos+offset<0) return -EINVAL;
42                          file->f_pos += offset;
43                          break;
// origin = SEEK_END，要求以文件末尾作为原点重定位读写指针。此时若文件大小加上偏移值
// 小于零则返回出错码退出。否则重定位读写指针为文件长度加上偏移值。
44                  case 2:
45                          if ((tmp=file->f_inode->i_size+offset) < 0)
46                                  return -EINVAL;
47                          file->f_pos = tmp;
48                          break;
// 若 origin 设置无效，返回出错码退出。
49                  default:
50                          return -EINVAL;
51          }
52          return file->f_pos;                 // 最后返回重定位后的文件读写指针值。
53  }
54
//// 读文件系统调用。
// 参数 fd 是文件句柄，buf 是缓冲区，count 是欲读字节数。
55  int sys_read(unsigned int fd,char * buf,int count)
56  {
57          struct file * file;
58          struct m_inode * inode;
59
// 函数首先对参数有效性进行判断。如果文件句柄值大于程序最多打开文件数 NR_OPEN，或者
// 需要读取的字节计数值小于 0，或者该句柄的文件结构指针为空，则返回出错码并退出。若
// 需读取的字节数 count 等于 0，则返回 0 退出
60          if (fd>=NR_OPEN || count<0 || !(file=current->filp[fd]))
61                  return -EINVAL;
62          if (!count)
63                  return 0;
// 然后验证存放数据的缓冲区内存限制。并取文件的 i 节点。用于根据该 i 节点的属性，分别
// 调用相应的读操作函数。若是管道文件，并且是读管道文件模式，则进行读管道操作，若成
// 功则返回读取的字节数，否则返回出错码，退出。如果是字符型文件，则进行读字符设备操
// 作，并返回读取的字符数。如果是块设备文件，则执行块设备读操作，并返回读取的字节数。
64          verify_area(buf,count);
65          inode = file->f_inode;
66          if (inode->i_pipe)
67                  return (file->f_mode&1)?read_pipe(inode,buf,count):-EIO;
68          if (S_ISCHR(inode->i_mode))
69                  return rw_char(READ,inode->i_zone[0],buf,count,&file->f_pos);
70          if (S_ISBLK(inode->i_mode))
71                  return block_read(inode->i_zone[0],&file->f_pos,buf,count);
// 如果是目录文件或者是常规文件，则首先验证读取字节数 count 的有效性并进行调整（若读
// 取字节数加上文件当前读写指针值大于文件长度，则重新设置读取字节数为 文件长度-当前
```

```
    // 读写指针值，若读取数等于 0，则返回 0 退出），然后执行文件读操作，返回读取的字节数
    // 并退出。
72          if (S_ISDIR(inode->i_mode) || S_ISREG(inode->i_mode)) {
73                  if (count+file->f_pos > inode->i_size)
74                          count = inode->i_size - file->f_pos;
75                  if (count<=0)
76                          return 0;
77                  return file_read(inode,file,buf,count);
78          }
    // 执行到这里，说明我们无法判断文件的属性。则打印节点文件属性，并返回出错码退出。
79          printk("(Read)inode->i_mode=%06o\n\r",inode->i_mode);
80          return -EINVAL;
81  }
82
    //// 写文件系统调用。
    // 参数 fd 是文件句柄，buf 是用户缓冲区，count 是欲写字节数。
83  int sys_write(unsigned int fd,char * buf,int count)
84  {
85          struct file * file;
86          struct m_inode * inode;
87
    // 同样地，我们首先判断函数参数的有效性。如果进程文件句柄值大于程序最多打开文件数
    // NR_OPEN，或者需要写入的字节计数小于 0，或者该句柄的文件结构指针为空，则返回出错
    // 码并退出。如果需读取的字节数 count 等于 0，则返回 0 退出
88          if (fd>=NR_OPEN || count <0 || !(file=current->filp[fd]))
89                  return -EINVAL;
90          if (!count)
91                  return 0;
    // 然后验证存放数据的缓冲区内存限制。并取文件的 i 节点。用于根据该 i 节点的属性，分别
    // 调用相应的读操作函数。若是管道文件，并且是写管道文件模式，则进行写管道操作，若成
    // 功则返回写入的字节数，否则返回出错码退出。如果是字符设备文件，则进行写字符设备操
    // 作，返回写入的字符数退出。如果是块设备文件，则进行块设备写操作，并返回写入的字节
    // 数退出。若是常规文件，则执行文件写操作，并返回写入的字节数，退出。
92          inode=file->f_inode;
93          if (inode->i_pipe)
94                  return (file->f_mode&2)?write_pipe(inode,buf,count):-EIO;
95          if (S_ISCHR(inode->i_mode))
96                  return rw_char(WRITE,inode->i_zone[0],buf,count,&file->f_pos);
97          if (S_ISBLK(inode->i_mode))
98                  return block_write(inode->i_zone[0],&file->f_pos,buf,count);
99          if (S_ISREG(inode->i_mode))
100                 return file_write(inode,file,buf,count);
    // 执行到这里，说明我们无法判断文件的属性。则打印节点文件属性，并返回出错码退出。
101         printk("(Write)inode->i_mode=%06o\n\r",inode->i_mode);
102         return -EINVAL;
103 }
104
```