程序 12-15 linux/fs/stat.c

```
1  /*
2   *  linux/fs/stat.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  #include <sys/stat.h>     // 文件状态头文件。含有文件状态结构 stat{}和常量。
9
10 #include <linux/fs.h>     // 文件系统头文件。定义文件表结构（file、m_inode）等。
11 #include <linux/sched.h>  // 调度程序头文件，定义了任务结构 task_struct、任务 0 数据等。
12 #include <linux/kernel.h> // 内核头文件。含有一些内核常用函数的原形定义。
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
   //// 复制文件状态信息。
   // 参数 inode 是文件 i 节点，statbuf 是用户数据空间中 stat 文件状态结构指针，用于存放取
   // 得的状态信息。
15 static void cp_stat(struct m_inode * inode, struct stat * statbuf)
16 {
17         struct stat tmp;
18         int i;
19
   // 首先验证(或分配)存放数据的内存空间。然后临时复制相应节点上的信息。
20         verify_area(statbuf,sizeof (struct stat));
21         tmp.st_dev = inode->i_dev;        // 文件所在的设备号。
22         tmp.st_ino = inode->i_num;        // 文件 i 节点号。
23         tmp.st_mode = inode->i_mode;      // 文件属性。
24         tmp.st_nlink = inode->i_nlinks;   // 文件连接数。
25         tmp.st_uid = inode->i_uid;        // 文件的用户 ID。
26         tmp.st_gid = inode->i_gid;        // 文件的组 ID。
27         tmp.st_rdev = inode->i_zone[0];   // 设备号（若是特殊字符文件或块设备文件）。
28         tmp.st_size = inode->i_size;      // 文件字节长度（如果文件是常规文件）。
29         tmp.st_atime = inode->i_atime;    // 最后访问时间。
30         tmp.st_mtime = inode->i_mtime;    // 最后修改时间。
31         tmp.st_ctime = inode->i_ctime;    // 最后 i 节点修改时间。
   // 最后将这些状态信息复制到用户缓冲区中。
32         for (i=0 ; i<sizeof (tmp) ; i++)
33                 put_fs_byte(((char *) &tmp)[i],i + (char *) statbuf);
34 }
35
   //// 文件状态系统调用。
   // 根据给定的文件名获取相关文件状态信息。
   // 参数 filename 是指定的文件名，statbuf 是存放状态信息的缓冲区指针。
   // 返回：成功返回 0，若出错则返回出错码。
36 int sys_stat(char * filename, struct stat * statbuf)
37 {
38         struct m_inode * inode;
39
   // 首先根据文件名找出对应的 i 节点。然后将 i 节点上的文件状态信息复制到用户缓冲区中，
   // 并放回该 i 节点。
40         if (!(inode=namei(filename)))
41                 return -ENOENT;
```

```
42          cp_stat(inode,statbuf);
43          iput(inode);
44          return 0;
45  }
46
    //// 文件状态系统调用。
    // 根据给定的文件名获取相关文件状态信息。文件路径名中有符号链接文件名，则取符号文件
    // 的状态。
    // 参数 filename 是指定的文件名，statbuf 是存放状态信息的缓冲区指针。
    // 返回：成功返回 0，若出错则返回出错码。
47  int sys_lstat(char * filename, struct stat * statbuf)
48  {
49          struct m_inode * inode;
50
    // 首先根据文件名找出对应的 i 节点。然后将 i 节点上的文件状态信息复制到用户缓冲区中，
    // 并放回该 i 节点。
51          if (!(inode = lnamei(filename)))     // 取指定路径名 i 节点，不跟随符号链接。
52                  return -ENOENT;
53          cp_stat(inode,statbuf);
54          iput(inode);
55          return 0;
56  }
57
    //// 文件状态系统调用。
    // 根据给定的文件句柄获取相关文件状态信息。
    // 参数 fd 是指定文件的句柄(描述符)，statbuf 是存放状态信息的缓冲区指针。
    // 返回：成功返回 0，若出错则返回出错码。
58  int sys_fstat(unsigned int fd, struct stat * statbuf)
59  {
60          struct file * f;
61          struct m_inode * inode;
62
    // 首先取文件句柄对应的文件结构，然后从中得到文件的 i 节点。然后将 i 节点上的文件状
    // 态信息复制到用户缓冲区中。如果文件句柄值大于一个程序最多打开文件数 NR_OPEN，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，则出错，返回出错
    // 码并退出。
63          if (fd >= NR_OPEN || !(f=current->filp[fd]) || !(inode=f->f_inode))
64                  return -EBADF;
65          cp_stat(inode,statbuf);
66          return 0;
67  }
68
    //// 读符号链接文件系统调用。
    // 该调用读取符号链接文件的内容（即该符号链接所指向文件的路径名字符串），并放到指定
    // 长度的用户缓冲区中。若缓冲区太小，就会截断符号链接的内容。
    // 参数：path -- 符号链接文件路径名；buf -- 用户缓冲区；bufsiz -- 缓冲区长度。
    // 返回：成功则返回放入缓冲区中的字符数；若失败则返回出错码。
69  int sys_readlink(const char * path, char * buf, int bufsiz)
70  {
71          struct m_inode * inode;
72          struct buffer_head * bh;
73          int i;
74          char c;
```

```
75
   // 首先检查和验证函数参数的有效性，并对其进行调整。用户缓冲区字节长度 bufsi 必须在
   // 1--1023 之间。然后取得符号链接文件名的 i 节点，并读取该文件的第 1 块数据内容。之
   // 后放回 i 节点。
76         if (bufsiz <= 0)
77                 return -EBADF;
78         if (bufsiz > 1023)
79                 bufsiz = 1023;
80         verify_area(buf,bufsiz);
81         if (!(inode = lnamei(path)))
82                 return -ENOENT;
83         if (inode->i_zone[0])
84                 bh = bread(inode->i_dev, inode->i_zone[0]);
85         else
86                 bh = NULL;
87         iput(inode);
   // 如果读取文件数据内容成功，则从内容中复制最多 bufsiz 个字符到用户缓冲区中。不复制
   // NULL 字符。最后释放缓冲块，并返回复制的字节数。
88         if (!bh)
89                 return 0;
90         i = 0;
91         while (i<bufsiz && (c = bh->b_data[i])) {
92                 i++;
93                 put_fs_byte(c,buf++);
94         }
95         brelse(bh);
96         return i;
97 }
98
```