程序 14-20 linux/include/linux/fs.h

```
/*
 * This file has definitions for some important file table
 * structures etc.
 */
/*
 * 本文件含有某些重要文件表结构的定义等。
 */

#ifndef _FS_H
#define _FS_H

#include <sys/types.h>    // 类型头文件。定义了基本的系统数据类型。

/* devices are as follows: (same as minix, so we can use the minix
 * file system. These are major numbers.)
 *
 * 0 - unused (nodev)
 * 1 - /dev/mem
 * 2 - /dev/fd
 * 3 - /dev/hd
 * 4 - /dev/ttyx
 * 5 - /dev/tty
 * 6 - /dev/lp
 * 7 - unnamed pipes
 */
/*
 * 系统所含的设备如下：（与 minix 系统的一样，所以我们可以使用 minix 的
 * 文件系统。以下这些是主设备号。）
 *
 * 0 - 没有用到（nodev）
 * 1 - /dev/mem         内存设备。
 * 2 - /dev/fd          软盘设备。
 * 3 - /dev/hd          硬盘设备。
 * 4 - /dev/ttyx        tty 串行终端设备。
 * 5 - /dev/tty         tty 终端设备。
 * 6 - /dev/lp          打印设备。
 * 7 - unnamed pipes    没有命名的管道。
 */

#define IS_SEEKABLE(x) ((x)>=1 && (x)<=3)      // 判断设备是否是可以寻找定位的。

#define READ 0
#define WRITE 1
#define READA 2         /* read-ahead - don't pause */
#define WRITEA 3        /* "write-ahead" - silly, but somewhat useful */

void buffer_init(long buffer_end);              // 高速缓冲区初始化函数。

#define MAJOR(a) (((unsigned)(a))>>8)           // 取高字节（主设备号）。
#define MINOR(a) ((a)&0xff)                     // 取低字节（次设备号）。

#define NAME_LEN 14                             // 名字长度值。
```

```
37 #define ROOT_INO 1                              // 根 i 节点。
38
39 #define I_MAP_SLOTS 8                          // i 节点位图槽数。
40 #define Z_MAP_SLOTS 8                          // 逻辑块（区段块）位图槽数。
41 #define SUPER_MAGIC 0x137F                     // 文件系统魔数。
42
43 #define NR_OPEN 20                             // 进程最多打开文件数。
44 #define NR_INODE 32                            // 系统同时最多使用 I 节点个数。
45 #define NR_FILE 64                             // 系统最多文件个数（文件数组项数）。
46 #define NR_SUPER 8                             // 系统所含超级块个数（超级块数组项数）。
47 #define NR_HASH 307                            // 缓冲区 Hash 表数组项数值。
48 #define NR_BUFFERS nr_buffers                  // 系统所含缓冲块个数。初始化后不再改变。
49 #define BLOCK_SIZE 1024                        // 数据块长度（字节值）。
50 #define BLOCK_SIZE_BITS 10                     // 数据块长度所占比特位数。
51 #ifndef NULL
52 #define NULL ((void *) 0)
53 #endif
54
   // 每个逻辑块可存放的 i 节点数。
55 #define INODES_PER_BLOCK ((BLOCK_SIZE)/(sizeof (struct d_inode)))
   // 每个逻辑块可存放的目录项数。
56 #define DIR_ENTRIES_PER_BLOCK ((BLOCK_SIZE)/(sizeof (struct dir_entry)))
57
   // 管道头、管道尾、管道大小、管道空？、管道满？、管道头指针递增。
58 #define PIPE_READ_WAIT(inode) ((inode).i_wait)
59 #define PIPE_WRITE_WAIT(inode) ((inode).i_wait2)
60 #define PIPE_HEAD(inode) ((inode).i_zone[0])
61 #define PIPE_TAIL(inode) ((inode).i_zone[1])
62 #define PIPE_SIZE(inode) ((PIPE_HEAD(inode)-PIPE_TAIL(inode))&(PAGE_SIZE-1))
63 #define PIPE_EMPTY(inode) (PIPE_HEAD(inode)==PIPE_TAIL(inode))
64 #define PIPE_FULL(inode) (PIPE_SIZE(inode)==(PAGE_SIZE-1))
65
66 #define NIL_FILP        ((struct file *)0)    // 空文件结构指针。
67 #define SEL_IN          1
68 #define SEL_OUT         2
69 #define SEL_EX          4
70
71 typedef char buffer_block[BLOCK_SIZE];        // 块缓冲区。
72
   // 缓冲块头数据结构。（极为重要！！！）
   // 在程序中常用 bh 来表示 buffer_head 类型的缩写。
73 struct buffer_head {
74         char * b_data;                  /* pointer to data block (1024 bytes) */ //指针。
75         unsigned long b_blocknr;        /* block number */       // 块号。
76         unsigned short b_dev;           /* device (0 = free) */  // 数据源的设备号。
77         unsigned char b_uptodate;       // 更新标志：表示数据是否已更新。
78         unsigned char b_dirt;           /* 0-clean,1-dirty */ //修改标志:0 未修改,1 已修改.
79         unsigned char b_count;          /* users using this block */   // 使用的用户数。
80         unsigned char b_lock;           /* 0 - ok, 1 -locked */  // 缓冲区是否被锁定。
81         struct task_struct * b_wait;    // 指向等待该缓冲区解锁的任务。
82         struct buffer_head * b_prev;    // hash 队列上前一块（这四个指针用于缓冲区的管理）。
83         struct buffer_head * b_next;    // hash 队列上下一块。
84         struct buffer_head * b_prev_free;   // 空闲表上前一块。
```

```
85          struct buffer_head * b_next_free;   // 空闲表上下一块。
86  };
87
    // 磁盘上的索引节点(i 节点)数据结构。
88  struct d_inode {
89          unsigned short i_mode;          // 文件类型和属性(rwx 位)。
90          unsigned short i_uid;           // 用户 id（文件拥有者标识符）。
91          unsigned long i_size;           // 文件大小（字节数）。
92          unsigned long i_time;           // 修改时间（自 1970.1.1:0 算起，秒）。
93          unsigned char i_gid;            // 组 id(文件拥有者所在的组)。
94          unsigned char i_nlinks;         // 链接数（多少个文件目录项指向该 i 节点）。
95          unsigned short i_zone[9];       // 直接(0-6)、间接(7)或双重间接(8)逻辑块号。
                                            // zone 是区的意思，可译成区段，或逻辑块。
96  };
97
    // 这是在内存中的 i 节点结构。前 7 项与 d_inode 完全一样。
98  struct m_inode {
99          unsigned short i_mode;          // 文件类型和属性(rwx 位)。
100         unsigned short i_uid;           // 用户 id（文件拥有者标识符）。
101         unsigned long i_size;           // 文件大小（字节数）。
102         unsigned long i_mtime;          // 修改时间（自 1970.1.1:0 算起，秒）。
103         unsigned char i_gid;            // 组 id(文件拥有者所在的组)。
104         unsigned char i_nlinks;         // 文件目录项链接数。
105         unsigned short i_zone[9];       // 直接(0-6)、间接(7)或双重间接(8)逻辑块号。
106 /* these are in memory also */
107         struct task_struct * i_wait;    // 等待该 i 节点的进程。
108         struct task_struct * i_wait2;    /* for pipes */
109         unsigned long i_atime;          // 最后访问时间。
110         unsigned long i_ctime;          // i 节点自身修改时间。
111         unsigned short i_dev;           // i 节点所在的设备号。
112         unsigned short i_num;           // i 节点号。
113         unsigned short i_count;         // i 节点被使用的次数，0 表示该 i 节点空闲。
114         unsigned char i_lock;           // 锁定标志。
115         unsigned char i_dirt;           // 已修改(脏)标志。
116         unsigned char i_pipe;           // 管道标志。
117         unsigned char i_mount;          // 安装标志。
118         unsigned char i_seek;           // 搜寻标志(lseek 时)。
119         unsigned char i_update;         // 更新标志。
120 };
121
    // 文件结构（用于在文件句柄与 i 节点之间建立关系）
122 struct file {
123         unsigned short f_mode;          // 文件操作模式（RW 位）
124         unsigned short f_flags;         // 文件打开和控制的标志。
125         unsigned short f_count;         // 对应文件引用计数值。
126         struct m_inode * f_inode;       // 指向对应 i 节点。
127         off_t f_pos;                    // 文件位置（读写偏移值）。
128 };
129
    // 内存中磁盘超级块结构。
130 struct super_block {
131         unsigned short s_ninodes;       // 节点数。
132         unsigned short s_nzones;        // 逻辑块数。
```

```
133         unsigned short s_imap_blocks;   // i 节点位图所占用的数据块数。
134         unsigned short s_zmap_blocks;   // 逻辑块位图所占用的数据块数。
135         unsigned short s_firstdatazone; // 第一个数据逻辑块号。
136         unsigned short s_log_zone_size; // log(数据块数/逻辑块)。（以 2 为底）。
137         unsigned long s_max_size;       // 文件最大长度。
138         unsigned short s_magic;         // 文件系统魔数。
139 /* These are only in memory */
140         struct buffer_head * s_imap[8]; // i 节点位图缓冲块指针数组(占用 8 块，可表示 64M)。
141         struct buffer_head * s_zmap[8]; // 逻辑块位图缓冲块指针数组（占用 8 块）。
142         unsigned short s_dev;           // 超级块所在的设备号。
143         struct m_inode * s_isup;        // 被安装的文件系统根目录的 i 节点。(isup-super i)
144         struct m_inode * s_imount;      // 被安装到的 i 节点。
145         unsigned long s_time;           // 修改时间。
146         struct task_struct * s_wait;    // 等待该超级块的进程。
147         unsigned char s_lock;           // 被锁定标志。
148         unsigned char s_rd_only;        // 只读标志。
149         unsigned char s_dirt;           // 已修改(脏)标志。
150 };
151
    // 磁盘上超级块结构。上面 125-132 行完全一样。
152 struct d_super_block {
153         unsigned short s_ninodes;       // 节点数。
154         unsigned short s_nzones;        // 逻辑块数。
155         unsigned short s_imap_blocks;   // i 节点位图所占用的数据块数。
156         unsigned short s_zmap_blocks;   // 逻辑块位图所占用的数据块数。
157         unsigned short s_firstdatazone; // 第一个数据逻辑块。
158         unsigned short s_log_zone_size; // log(数据块数/逻辑块)。（以 2 为底）。
159         unsigned long s_max_size;       // 文件最大长度。
160         unsigned short s_magic;         // 文件系统魔数。
161 };
162
    // 文件目录项结构。
163 struct dir_entry {
164         unsigned short inode;           // i 节点号。
165         char name[NAME_LEN];            // 文件名，长度 NAME_LEN=14。
166 };
167
168 extern struct m_inode inode_table[NR_INODE];     // 定义 i 节点表数组（32 项）。
169 extern struct file file_table[NR_FILE];          // 文件表数组（64 项）。
170 extern struct super_block super_block[NR_SUPER]; // 超级块数组（8 项）。
171 extern struct buffer_head * start_buffer;        // 缓冲区起始内存位置。
172 extern int nr_buffers;                           // 缓冲块数。
173
    //// 磁盘操作函数原型。
    // 检测驱动器中软盘是否改变。
174 extern void check_disk_change(int dev);
    // 检测指定软驱中软盘更换情况。如果软盘更换了则返回 1，否则返回 0。
175 extern int floppy_change(unsigned int nr);
    // 设置启动指定驱动器所需等待的时间（设置等待定时器）。
176 extern int ticks_to_floppy_on(unsigned int dev);
    // 启动指定驱动器。
177 extern void floppy_on(unsigned int dev);
    // 关闭指定的软盘驱动器。
```

```
178 extern void floppy_off(unsigned int dev);

    //// 以下是文件系统操作管理用的函数原型。
    // 将 i 节点指定的文件截为 0。
179 extern void truncate(struct m_inode * inode);
    // 刷新 i 节点信息。
180 extern void sync_inodes(void);
    // 等待指定的 i 节点。
181 extern void wait_on(struct m_inode * inode);
    // 逻辑块(区段，磁盘块)位图操作。取数据块 block 在设备上对应的逻辑块号。
182 extern int bmap(struct m_inode * inode,int block);
    // 创建数据块 block 在设备上对应的逻辑块，并返回在设备上的逻辑块号。
183 extern int create_block(struct m_inode * inode,int block);
    // 获取指定路径名的 i 节点号。
184 extern struct m_inode * namei(const char * pathname);
    // 取指定路径名的 i 节点，不跟随符号链接。
185 extern struct m_inode * lnamei(const char * pathname);
    // 根据路径名为打开文件操作作准备。
186 extern int open_namei(const char * pathname, int flag, int mode,
187         struct m_inode ** res_inode);
    // 释放一个 i 节点(回写入设备)。
188 extern void iput(struct m_inode * inode);
    // 从设备读取指定节点号的一个 i 节点。
189 extern struct m_inode * iget(int dev,int nr);
    // 从 i 节点表(inode_table)中获取一个空闲 i 节点项。
190 extern struct m_inode * get_empty_inode(void);
    // 获取（申请一）管道节点。返回为 i 节点指针（如果是 NULL 则失败）。
191 extern struct m_inode * get_pipe_inode(void);
    // 在哈希表中查找指定的数据块。返回找到块的缓冲头指针。
192 extern struct buffer_head * get_hash_table(int dev, int block);
    // 从设备读取指定块（首先会在 hash 表中查找）。
193 extern struct buffer_head * getblk(int dev, int block);
    // 读/写数据块。
194 extern void ll_rw_block(int rw, struct buffer_head * bh);
    // 读/写数据页面，即每次 4 块数据块。
195 extern void ll_rw_page(int rw, int dev, int nr, char * buffer);
    // 释放指定缓冲块。
196 extern void brelse(struct buffer_head * buf);
    // 读取指定的数据块。
197 extern struct buffer_head * bread(int dev,int block);
    // 读 4 块缓冲区到指定地址的内存中。
198 extern void bread_page(unsigned long addr,int dev,int b[4]);
    // 读取头一个指定的数据块，并标记后续将要读的块。
199 extern struct buffer_head * breada(int dev,int block,...);
    // 向设备 dev 申请一个磁盘块（区段，逻辑块）。返回逻辑块号
200 extern int new_block(int dev);
    // 释放设备数据区中的逻辑块(区段，磁盘块)block。复位指定逻辑块 block 的逻辑块位图比特位。
201 extern void free_block(int dev, int block);
    // 为设备 dev 建立一个新 i 节点，返回 i 节点号。
202 extern struct m_inode * new_inode(int dev);
    // 释放一个 i 节点（删除文件时）。
203 extern void free_inode(struct m_inode * inode);
    // 刷新指定设备缓冲区。
```

```
204 extern int sync_dev(int dev);
    // 读取指定设备的超级块。
205 extern struct super_block * get_super(int dev);
206 extern int ROOT_DEV;
207
    // 安装根文件系统。
208 extern void mount_root(void);
209
210 #endif
211
```