程序 14-12 linux/include/unistd.h

```
1  #ifndef _UNISTD_H
2  #define _UNISTD_H
3
4  /* ok, this may be a joke, but I'm working on it */
   /* ok, 这也许是个玩笑，但我正在着手处理 */
   // 下面符号常数指出符合 IEEE 标准 1003.1 实现的版本号，是一个整数值。
5  #define _POSIX_VERSION 198808L
6
   // chown()和 fchown()的使用受限于进程的权限。/* 只有超级用户可以执行 chown（我想..）*/
7  #define _POSIX_CHOWN_RESTRICTED /* only root can do a chown (I think..) */
   // 长于(NAME_MAX)的路径名将产生错误，而不会自动截断。/* 路径名不截断（但是请看内核代码）*/
8  #define _POSIX_NO_TRUNC         /* no pathname truncation (but see in kernel) */
   // 下面这个符号将定义成字符值，该值将禁止终端对其的处理。/* 禁止象^C 这样的字符 */
   // _POSIX_VDISABLE 用于控制终端某些特殊字符的功能。当一个终端 termios 结构中 c_cc[]
   // 数组某项字符代码值等于_POSIX_VDISABLE 的值时，表示禁止使用相应的特殊字符。
9  #define _POSIX_VDISABLE '\0'    /* character to disable things like ^C */
   // 系统实现支持作业控制。
10 #define _POSIX_JOB_CONTROL
   // 每个进程都有一保存的 set-user-ID 和一保存的 set-group-ID。 /* 已经实现。 */
11 #define _POSIX_SAVED_IDS        /* Implemented, for whatever good it is */
12
13 #define STDIN_FILENO    0           // 标准输入文件句柄（描述符）号。
14 #define STDOUT_FILENO   1           // 标准输出文件句柄号。
15 #define STDERR_FILENO   2           // 标准出错文件句柄号。
16
17 #ifndef NULL
18 #define NULL    ((void *)0)         // 定义空指针。
19 #endif
20
21 /* access */   /* 文件访问 */
   // 以下定义的符号常数用于 access()函数。
22 #define F_OK    0                   // 检测文件是否存在。
23 #define X_OK    1                   // 检测是否可执行（搜索）。
24 #define W_OK    2                   // 检测是否可写。
25 #define R_OK    4                   // 检测是否可读。
26
27 /* lseek */   /* 文件指针重定位 */
   // 以下符号常数用于 lseek()和 fcntl()函数。
28 #define SEEK_SET        0           // 将文件读写指针设置为偏移值。
29 #define SEEK_CUR        1           // 将文件读写指针设置为当前值加上偏移值。
30 #define SEEK_END        2           // 将文件读写指针设置为文件长度加上偏移值。
31
32 /* _SC stands for System Configuration. We don't use them much */
   /* _SC 表示系统配置。我们很少使用 */
   // 下面的符号常数用于 sysconf()函数。
33 #define _SC_ARG_MAX             1   // 最大变量数。
34 #define _SC_CHILD_MAX           2   // 子进程最大数。
35 #define _SC_CLOCKS_PER_SEC      3   // 每秒滴答数。
36 #define _SC_NGROUPS_MAX         4   // 最大组数。
37 #define _SC_OPEN_MAX            5   // 最大打开文件数。
38 #define _SC_JOB_CONTROL         6   // 作业控制。
39 #define _SC_SAVED_IDS           7   // 保存的标识符。
```

```
40 #define _SC_VERSION             8    // 版本。
41
42 /* more (possibly) configurable things - now pathnames */
   /* 更多的（可能的）可配置参数 - 现在用于路径名 */
   // 下面的符号常数用于 pathconf()函数。
43 #define _PC_LINK_MAX            1    // 连接最大数。
44 #define _PC_MAX_CANON           2    // 最大常规文件数。
45 #define _PC_MAX_INPUT           3    // 最大输入长度。
46 #define _PC_NAME_MAX            4    // 名称最大长度。
47 #define _PC_PATH_MAX            5    // 路径最大长度。
48 #define _PC_PIPE_BUF            6    // 管道缓冲大小。
49 #define _PC_NO_TRUNC            7    // 文件名不截断。
50 #define _PC_VDISABLE            8    //
51 #define _PC_CHOWN_RESTRICTED    9    // 改变宿主受限。
52
53 #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。
54 #include <sys/time.h>
55 #include <sys/times.h>    // 定义了进程中运行时间结构 tms 以及 times()函数原型。
56 #include <sys/utsname.h>  // 系统名称结构头文件。
57 #include <sys/resource.h>
58 #include <utime.h>        // 用户时间头文件。定义了访问和修改时间结构以及 utime()原型。
59
60 #ifdef __LIBRARY__
61
   // 以下是实现的系统调用符号常数，用作系统调用函数表中索引值(参见 include/linux/sys.h) 。
62 #define __NR_setup      0       /* used only by init, to get system going */
63 #define __NR_exit       1       /* __NR_setup 仅用于初始化，以启动系统 */
64 #define __NR_fork       2
65 #define __NR_read       3
66 #define __NR_write      4
67 #define __NR_open       5
68 #define __NR_close      6
69 #define __NR_waitpid    7
70 #define __NR_creat      8
71 #define __NR_link       9
72 #define __NR_unlink     10
73 #define __NR_execve     11
74 #define __NR_chdir      12
75 #define __NR_time       13
76 #define __NR_mknod      14
77 #define __NR_chmod      15
78 #define __NR_chown      16
79 #define __NR_break      17
80 #define __NR_stat       18
81 #define __NR_lseek      19
82 #define __NR_getpid     20
83 #define __NR_mount      21
84 #define __NR_umount     22
85 #define __NR_setuid     23
86 #define __NR_getuid     24
87 #define __NR_stime      25
88 #define __NR_ptrace     26
89 #define __NR_alarm      27
```

```
 90 #define __NR_fstat      28
 91 #define __NR_pause      29
 92 #define __NR_utime      30
 93 #define __NR_stty       31
 94 #define __NR_gtty       32
 95 #define __NR_access     33
 96 #define __NR_nice       34
 97 #define __NR_ftime      35
 98 #define __NR_sync       36
 99 #define __NR_kill       37
100 #define __NR_rename     38
101 #define __NR_mkdir      39
102 #define __NR_rmdir      40
103 #define __NR_dup        41
104 #define __NR_pipe       42
105 #define __NR_times      43
106 #define __NR_prof       44
107 #define __NR_brk        45
108 #define __NR_setgid     46
109 #define __NR_getgid     47
110 #define __NR_signal     48
111 #define __NR_geteuid    49
112 #define __NR_getegid    50
113 #define __NR_acct       51
114 #define __NR_phys       52
115 #define __NR_lock       53
116 #define __NR_ioctl      54
117 #define __NR_fcntl      55
118 #define __NR_mpx        56
119 #define __NR_setpgid    57
120 #define __NR_ulimit     58
121 #define __NR_uname      59
122 #define __NR_umask      60
123 #define __NR_chroot     61
124 #define __NR_ustat      62
125 #define __NR_dup2       63
126 #define __NR_getppid    64
127 #define __NR_getpgrp    65
128 #define __NR_setsid     66
129 #define __NR_sigaction  67
130 #define __NR_sgetmask   68
131 #define __NR_ssetmask   69
132 #define __NR_setreuid   70
133 #define __NR_setregid   71
134 #define __NR_sigsuspend 72
135 #define __NR_sigpending 73
136 #define __NR_sethostname 74
137 #define __NR_setrlimit  75
138 #define __NR_getrlimit  76
139 #define __NR_getrusage  77
140 #define __NR_gettimeofday 78
141 #define __NR_settimeofday 79
142 #define __NR_getgroups  80
```

```
143 #define __NR_setgroups  81
144 #define __NR_select     82
145 #define __NR_symlink    83
146 #define __NR_lstat      84
147 #define __NR_readlink   85
148 #define __NR_uselib     86
149
    // 以下定义系统调用嵌入式汇编宏函数。
    // 不带参数的系统调用宏函数。type name(void)。
    // %0 - eax(__res), %1 - eax(__NR_##name)。其中 name 是系统调用的名称，与 __NR_ 组合形成上面
    // 的系统调用符号常数，从而用来对系统调用表中函数指针寻址。
    // 返回：如果返回值大于等于 0，则返回该值，否则置出错号 errno，并返回-1。
    // 在宏定义中，若在两个标记符号之间有两个连续的井号'##'，则表示在宏替换时会把这两个标记
    // 符号连接在一起。例如下面第 139 行上的__NR_##name，在替换了参数 name（例如是 fork）之后，
    // 最后在程序中出现的将会是符号__NR_fork。参见《The C Programming Language》附录 A.12.3。
150 #define _syscall0(type,name) \
151 type name(void) \
152 { \
153 long __res; \
154 __asm__ volatile ("int $0x80" \          // 调用系统中断 0x80。
155         : "=a" (__res) \                 // 返回值➔eax(__res)。
156         : "0" (__NR_##name)); \          // 输入为系统中断调用号__NR_name。
157 if (__res >= 0) \                        // 如果返回值>=0，则直接返回该值。
158         return (type) __res; \
159 errno = -__res; \                        // 否则置出错号，并返回-1。
160 return -1; \
161 }
162
    // 有 1 个参数的系统调用宏函数。type name(atype a)
    // %0 - eax(__res), %1 - eax(__NR_name), %2 - ebx(a)。
163 #define _syscall1(type,name,atype,a) \
164 type name(atype a) \
165 { \
166 long __res; \
167 __asm__ volatile ("int $0x80" \
168         : "=a" (__res) \
169         : "0" (__NR_##name),"b" ((long)(a))); \
170 if (__res >= 0) \
171         return (type) __res; \
172 errno = -__res; \
173 return -1; \
174 }
175
    // 有 2 个参数的系统调用宏函数。type name(atype a, btype b)
    // %0 - eax(__res), %1 - eax(__NR_name), %2 - ebx(a), %3 - ecx(b)。
176 #define _syscall2(type,name,atype,a,btype,b) \
177 type name(atype a,btype b) \
178 { \
179 long __res; \
180 __asm__ volatile ("int $0x80" \
181         : "=a" (__res) \
182         : "0" (__NR_##name),"b" ((long)(a)),"c" ((long)(b))); \
183 if (__res >= 0) \
```

```
184         return (type) __res; \
185 errno = -__res; \
186 return -1; \
187 }
188
    // 有 3 个参数的系统调用宏函数。type name(atype a, btype b, ctype c)
    // %0 - eax(__res), %1 - eax(__NR_name), %2 - ebx(a), %3 - ecx(b), %4 - edx(c)。
189 #define _syscall3(type,name,atype,a,btype,b,ctype,c) \
190 type name(atype a,btype b,ctype c) \
191 { \
192 long __res; \
193 __asm__ volatile ("int $0x80" \
194         : "=a" (__res) \
195         : "0" (__NR_##name),"b" ((long)(a)),"c" ((long)(b)),"d" ((long)(c))); \
196 if (__res>=0) \
197         return (type) __res; \
198 errno=-__res; \
199 return -1; \
200 }
201
202 #endif /* __LIBRARY__ */
203
204 extern int errno;                       // 出错号，全局变量。
205
    // 对应各系统调用的函数原型定义。(详细说明参见 include/linux/sys.h )
206 int access(const char * filename, mode_t mode);
207 int acct(const char * filename);
208 int alarm(int sec);
209 int brk(void * end_data_segment);
210 void * sbrk(ptrdiff_t increment);
211 int chdir(const char * filename);
212 int chmod(const char * filename, mode_t mode);
213 int chown(const char * filename, uid_t owner, gid_t group);
214 int chroot(const char * filename);
215 int close(int fildes);
216 int creat(const char * filename, mode_t mode);
217 int dup(int fildes);
218 int execve(const char * filename, char ** argv, char ** envp);
219 int execv(const char * pathname, char ** argv);
220 int execvp(const char * file, char ** argv);
221 int execl(const char * pathname, char * arg0, ...);
222 int execlp(const char * file, char * arg0, ...);
223 int execle(const char * pathname, char * arg0, ...);
    // 函数名前的关键字 volatile 用于告诉编译器 gcc 该函数不会返回。这样可让 gcc 产生更好一
    // 些的代码，更重要的是使用这个关键字可以避免产生某些（未初始化变量的）假警告信息。
    // 等同于 gcc 的函数属性说明：void do_exit(int error_code) __attribute__ ((noreturn));
224 volatile void exit(int status);
225 volatile void _exit(int status);
226 int fcntl(int fildes, int cmd, ...);
227 int fork(void);
228 int getpid(void);
229 int getuid(void);
230 int geteuid(void);
```

```
231 int getgid(void);
232 int getegid(void);
233 int ioctl(int fildes, int cmd, ...);
234 int kill(pid_t pid, int signal);
235 int link(const char * filename1, const char * filename2);
236 int lseek(int fildes, off_t offset, int origin);
237 int mknod(const char * filename, mode_t mode, dev_t dev);
238 int mount(const char * specialfile, const char * dir, int rwflag);
239 int nice(int val);
240 int open(const char * filename, int flag, ...);
241 int pause(void);
242 int pipe(int * fildes);
243 int read(int fildes, char * buf, off_t count);
244 int setpgrp(void);
245 int setpgid(pid_t pid,pid_t pgid);
246 int setuid(uid_t uid);
247 int setgid(gid_t gid);
248 void (*signal(int sig, void (*fn)(int)))(int);
249 int stat(const char * filename, struct stat * stat_buf);
250 int fstat(int fildes, struct stat * stat_buf);
251 int stime(time_t * tptr);
252 int sync(void);
253 time_t time(time_t * tloc);
254 time_t times(struct tms * tbuf);
255 int ulimit(int cmd, long limit);
256 mode_t umask(mode_t mask);
257 int umount(const char * specialfile);
258 int uname(struct utsname * name);
259 int unlink(const char * filename);
260 int ustat(dev_t dev, struct ustat * ubuf);
261 int utime(const char * filename, struct utimbuf * times);
262 pid_t waitpid(pid_t pid,int * wait_stat,int options);
263 pid_t wait(int * wait_stat);
264 int write(int fildes, const char * buf, off_t count);
265 int dup2(int oldfd, int newfd);
266 int getppid(void);
267 pid_t getpgrp(void);
268 pid_t setsid(void);
269 int sethostname(char *name, int len);
270 int setrlimit(int resource, struct rlimit *rlp);
271 int getrlimit(int resource, struct rlimit *rlp);
272 int getrusage(int who, struct rusage *rusage);
273 int gettimeofday(struct timeval *tv, struct timezone *tz);
274 int settimeofday(struct timeval *tv, struct timezone *tz);
275 int getgroups(int gidsetlen, gid_t *gidset);
276 int setgroups(int gidsetlen, gid_t *gidset);
277 int select(int width, fd_set * readfds, fd_set * writefds,
278         fd_set * exceptfds, struct timeval * timeout);
279
280 #endif
281
```