程序 14-6　linux/include/signal.h

```
 1 #ifndef _SIGNAL_H
 2 #define _SIGNAL_H
 3
 4 #include <sys/types.h>                  // 类型头文件。定义了基本的系统数据类型。
 5
 6 typedef int sig_atomic_t;               // 定义信号原子操作类型。
 7 typedef unsigned int sigset_t;          /* 32 bits */  // 定义信号集类型。
 8
 9 #define _NSIG             32            // 定义信号种类 -- 32 种。
10 #define NSIG              _NSIG          // NSIG = _NSIG
11
   // 以下这些是 Linux 0.12 内核中定义的信号。其中包括了 POSIX.1 要求的所有 20 个信号。
12 #define SIGHUP            1             // Hang Up     -- 挂断控制终端或进程。
13 #define SIGINT            2             // Interrupt   -- 来自键盘的中断。
14 #define SIGQUIT           3             // Quit        -- 来自键盘的退出。
15 #define SIGILL            4             // Illeagle    -- 非法指令。
16 #define SIGTRAP           5             // Trap        -- 跟踪断点。
17 #define SIGABRT           6             // Abort       -- 异常结束。
18 #define SIGIOT            6             // IO Trap     -- 同上。
19 #define SIGUNUSED         7             // Unused      -- 没有使用。
20 #define SIGFPE            8             // FPE         -- 协处理器出错。
21 #define SIGKILL           9             // Kill        -- 强迫进程终止。
22 #define SIGUSR1           10            // User1       -- 用户信号 1，进程可使用。
23 #define SIGSEGV           11            // Segment Violation -- 无效内存引用。
24 #define SIGUSR2           12            // User2       -- 用户信号 2，进程可使用。
25 #define SIGPIPE           13            // Pipe        -- 管道写出错，无读者。
26 #define SIGALRM           14            // Alarm       -- 实时定时器报警。
27 #define SIGTERM           15            // Terminate   -- 进程终止。
28 #define SIGSTKFLT         16            // Stack Fault -- 栈出错（协处理器）。
29 #define SIGCHLD           17            // Child       -- 子进程停止或被终止。
30 #define SIGCONT           18            // Continue    -- 恢复进程继续执行。
31 #define SIGSTOP           19            // Stop        -- 停止进程的执行。
32 #define SIGTSTP           20            // TTY Stop    -- tty 发出停止进程，可忽略。
33 #define SIGTTIN           21            // TTY In      -- 后台进程请求输入。
34 #define SIGTTOU           22            // TTY Out     -- 后台进程请求输出。
35
36 /* Ok, I haven't implemented sigactions, but trying to keep headers POSIX */
   /* OK，我还没有实现 sigactions 的编制，但在头文件中仍希望遵守 POSIX 标准 */
   // 上面原注释已经过时，因为在 0.12 内核中已经实现了 sigaction()。下面是 sigaction 结构
   // sa_flags 标志字段可取的符号常数值。
37 #define SA_NOCLDSTOP      1             // 当子进程处于停止状态，就不对 SIGCHLD 处理。
38 #define SA_INTERRUPT      0x20000000    // 系统调用被信号中断后不重新启动系统调用。
39 #define SA_NOMASK         0x40000000    // 不阻止在指定的信号处理程序中再收到该信号。
40 #define SA_ONESHOT        0x80000000    // 信号句柄一旦被调用过就恢复到默认处理句柄。
41
   // 以下常量用于 sigprocmask(how, )-- 改变阻塞信号集(屏蔽码)。用于改变该函数的行为。
42 #define SIG_BLOCK         0   /* for blocking signals */   // 在阻塞信号集中加上给定信号。
43 #define SIG_UNBLOCK       1   /* for unblocking signals */ // 从阻塞信号集中删除指定信号。
44 #define SIG_SETMASK       2   /* for setting the signal mask */   // 设置阻塞信号集。
45
   // 以下两个常数符号都表示指向无返回值的函数指针，且都有一个 int 整型参数。这两个指针
   // 值是逻辑上讲实际上不可能出现的函数地址值。可作为下面 signal 函数的第二个参数。用
```

```
   // 于告知内核，让内核处理信号或忽略对信号的处理。使用方法参见 kernel/signal.c 程序，
   // 第 94--96 行。
46 #define SIG_DFL         ((void (*)(int))0)      /* default signal handling */
                                                   // 默认信号处理程序（信号句柄）。
47 #define SIG_IGN         ((void (*)(int))1)      /* ignore signal */
                                                   // 忽略信号的处理程序。
48 #define SIG_ERR         ((void (*)(int))-1)     /* error return from signal */
                                                   // 信号处理返回错误。
49
   // 下面定义初始操作设置 sigaction 结构信号屏蔽码的宏。
50 #ifdef notdef
51 #define sigemptyset(mask) ((*(mask) = 0), 1)    // 将 mask 清零。
52 #define sigfillset(mask) ((*(mask) = ~0), 1)    // 将 mask 所有比特位置位。
53 #endif
54
   // 下面是 sigaction 的数据结构。
   // sa_handler 是对应某信号指定要采取的行动。可以用上面的 SIG_DFL，或 SIG_IGN 来忽略该
   // 信号，也可以是指向处理该信号函数的一个指针。
   // sa_mask 给出了对信号的屏蔽码，在信号程序执行时将阻塞对这些信号的处理。
   // sa_flags 指定改变信号处理过程的信号集。它是由 37-40 行的位标志定义的。
   // sa_restorer 是恢复函数指针，由函数库 Libc 提供，用于清理用户态堆栈。参见 signal.c。
   // 另外，引起触发信号处理的信号也将被阻塞，除非使用了 SA_NOMASK 标志。
55 struct sigaction {
56         void (*sa_handler)(int);
57         sigset_t sa_mask;
58         int sa_flags;
59         void (*sa_restorer)(void);
60 };
61
   // 下面 signal 函数用于是为信号_sig 安装一新的信号处理程序（信号句柄），与 sigaction()
   // 类似。该函数含有两个参数：指定需要捕获的信号_sig；具有一个参数且无返回值的函数指针
   // _func。该函数返回值也是具有一个 int 参数（最后一个(int)）且无返回值的函数指针，它是
   // 处理该信号的原处理句柄。
62 void (*signal(int _sig, void (*_func)(int)))(int);
   // 下面两函数用于发送信号。kill() 用于向任何进程或进程组发送信号。raise()用于向当前进
   // 程自身发送信号。其作用等价于 kill(getpid(),sig)。参见 kernel/exit.c，60 行。
63 int raise(int sig);
64 int kill(pid_t pid, int sig);
   // 在进程的任务结构中，除有一个以比特位表示当前进程待处理的 32 位信号字段 signal 以外，
   // 还有一个同样以比特位表示的用于屏蔽进程当前阻塞信号集（屏蔽信号集）的字段 blocked，
   // 也是 32 位，每个比特代表一个对应的阻塞信号。修改进程的屏蔽信号集可以阻塞或解除阻塞
   // 所指定的信号。 以下五个函数就是用于操作进程屏蔽信号集，虽然简单实现起来很简单，但
   // 本版本内核中还未实现。
   // 函数 sigaddset() 和 sigdelset() 用于对信号集中的信号进行增、删修改。 sigaddset()用
   // 于向 mask 指向的信号集中增加指定的信号 signo。sigdelset 则反之。函数 sigemptyset()和
   // sigfillset() 用于初始化进程屏蔽信号集。 每个程序在使用信号集前，都需要使用这两个函
   // 数之一对屏蔽信号集进行初始化。 sigemptyset()用于清空屏蔽的所有信号，也即响应所有的
   // 信号。sigfillset()向信号集中置入所有信号，也即屏蔽所有信号。当然 SIGINT 和 SIGSTOP
   // 是不能被屏蔽的。
   // sigismember()用于测试一个指定信号是否在信号集中（1 - 是，0 - 不是，-1 - 出错）。
65 int sigaddset(sigset_t *mask, int signo);
66 int sigdelset(sigset_t *mask, int signo);
67 int sigemptyset(sigset_t *mask);
```

```
68 int sigfillset(sigset_t *mask);
69 int sigismember(sigset_t *mask, int signo); /* 1 - is, 0 - not, -1 error */
   // 对 set 中的信号进行检测，看是否有挂起的信号。在 set 中返回进程中当前被阻塞的信号集。
70 int sigpending(sigset_t *set);
   // 下面函数用于改变进程目前被阻塞的信号集（信号屏蔽码）。若 oldset 不是 NULL，则通过其
   // 返回进程当前屏蔽信号集。若 set 指针不是 NULL，则根据 how（41-43 行）指示修改进程屏蔽
   // 信号集。
71 int sigprocmask(int how, sigset_t *set, sigset_t *oldset);
   // 下面函数用 sigmask 临时替换进程的信号屏蔽码,然后暂停该进程直到收到一个信号。若捕捉
   // 到某一信号并从该信号处理程序中返回，则该函数也返回，并且信号屏蔽码会恢复到调用调用
   // 前的值。
72 int sigsuspend(sigset_t *sigmask);
   // sigaction() 函数用于改变进程在收到指定信号时所采取的行动，即改变信号的处理句柄能。
   // 参见对 kernel/signal.c 程序的说明。
73 int sigaction(int sig, struct sigaction *act, struct sigaction *oldact);
74
75 #endif /* _SIGNAL_H */
76
```