10-信号

10.1 信号概念

- 在头文件<signal.h>中，信号都被定义为正整数常量，不存在编号为0的信号

产生信号条件

- 按某些终端键，产生信号
- 硬件异常产生信号
- 进程用kill(2)函数将信号发送给进程或进程组
  - **接收信号进程和发送信号进程的所有者必须相同**，或发送信号进程的所有者必须是超级用户。
- 进程用kill(1)命令发送给其他进程
- 软件条件产生；如除零异常；SIGURG，SIGPIPE（管道读进程终止后写该管道），SIGALRM

对信号的操作

- 忽略该信号
- 捕捉该信号
- 执行系统默认动作

表10-1 UNIX信号

名 字	说 明	ANSI C	POSIX.1	SVR4	4.3+BSD	缺省动作
SIGABRT	异常终止(abort)	•	•	•	•	终止w/core
SIGALRM	超时(alarm)		•	•	•	终止
SIGBUS	硬件故障			•	•	终止w/core
SIGCHLD	子进程状态改变		作业	•	•	忽略
SIGCONT	使暂停进程继续		作业	•	•	继续/忽略
SIGEMT	硬件故障			•	•	终止w/core
SIGFPE	算术异常	•	•	•	•	终止w/core
SIGHUP	连接断开		•	•	•	终止
SIGILL	非法硬件指令	•	•	•	•	终止w/core
SIGINFO	键盘状态请求				•	忽略
SIGINT	终端中断符	•	•	•	•	终止
SIGIO	异步I/O			•	•	终止/忽略
SIGIOT	硬件故障			•	•	终止w/core
SIGKILL	终止		•	•	•	终止
SIGPIPE	写至无读进程的管道		•	•	•	终止
SIGPOLL	可轮询事件(poll)			•		终止
SIGPROF	梗概时间超时(setitimer)			•	•	终止
SIGPWR	电源失效/再起动			•		忽略
SIGQUIT	终端退出符		•	•	•	终止w/core
SIGSEGV	无效存储访问	•	•	•	•	终止w/core
SIGSTOP	停止		作业	•	•	暂停进程

（续）

名 字	说 明	ANSI C	POSIX.1	SVR4	4.3+BSD	缺省动作
SIGSYS	无效系统调用			•	•	终止w/core
SIGTERM	终止	•	•	•	•	终止
SIGTRAP	硬件故障			•	•	终止w/core
SIGTSTP	终端挂起符		作业	•	•	停止进程
SIGTTIN	后台从控制tty读		作业	•	•	停止进程
SIGTTOU	后台向控制tty写		作业	•	•	停止进程
SIGURG	紧急情况			•	•	忽略
SIGUSR1	用户定义信号		•	•	•	终止
SIGUSR2	用户定义信号		•	•	•	终止
SIGVTALRM	虚拟时间闹钟(setitimer)			•	•	终止
SIGWINCH	终端窗口大小改变			•	•	忽略
SIGXCPU	超过CPU限制(setrlimit)			•	•	终止w/core
SIGXFSZ	超过文件长度限制(setrlimit)			•	•	终止w/core

10.2 signal函数

- 给定两个参数；一个整形变量，一个int参数的函数指针；返回一个信号处理程序的函数指针

```c
#include <signal.h>
void (*signal(int signo, void (*func)(int)) (int);

typedef void Sigfunc(int);
Sigfunc* signal(int, Sigfunc*);

#define SIG_ERR (void(*)())-1
#define SIG_DFL (void(*)())0
#define SIG_IGN (void(*)())1
```

- func的值是：
  - SIG_IGN，则向内核表示忽略此信号
  - SIG_DFL，则表示接到此信号后的动作是系统默认动作
  - 当指定函数地址时，我们称此为捕捉此信号。我们称此函数为信号处理程序（signal handler）或信号捕捉函数（signal-catching function）。

```c
// 信号处理函数
static void sig_usr(int signo) /* argument is signal number */
{
  if (signo == SIGUSR1)
    printf("received SIGUSR1\n");
  else if (signo == SIGUSR2)
    printf("received SIGUSR2\n");
  else
    err_dump("received signal %d\n", signo);
}
```

```
lzy@lzy:~/Workspace/APUE/chapter10$ ./sigusr &
[1] 1378
lzy@lzy:~/Workspace/APUE/chapter10$ kill -USR1 1378
received SIGUSR1
lzy@lzy:~/Workspace/APUE/chapter10$ kill -USR2 1378
received SIGUSR2
```

- **exec函数将设置捕捉的信号重新设置为默认**
- **shell自动将后台进程对中断和退出信号的处理方式设置为忽略**
  - ctrl-C对后台程序无效；使用kill -9 pid
- fork后，子进程继承父进程的信号处理方式

10.3 不可靠信号

- 不可靠信号：信号可能会丢失；信号不能被阻塞

- 在测试sig_int_flag后，调用pause前可能发生信号中断；此时进程可能会一直调用pause

```
int sig_int();
int sig_int_flag;
int main() {
    signal(SIG_INT, sig_int);
    while(!sig_int_flag)
        pause();
}

int sig_int() {
    signal(SIG_INT, sig_int);
    sig_int_flag = 1;
}
```

10.4 中断的系统调用

低速系统调用：可能使进程阻塞的一类系统调用

- 读操作
- 写操作
- 打开操作
- Pause
- ioctl操作
- 某些进程间通信函数

自动重启动的系统调用

- Ioctl, read, readv, write, writev, wait, waitpid

10.5 SIGCLD语义

- 子进程状态改变后发送该信号
- 设置SIG_IGN，调用进程的子进程不产生僵死进程
- 设置捕捉函数，检查是否有子进程准备好被等待，如果有，调用SIGCLD处理程序

10.6 kill和raise

- kill将信号发送给进程或进程组
- raise将信号发送给进程自身

```c
#include <signal.h>
int kill(pid_t pid, int signo);
int raise(int signo);
```

10.7 alarm和pause

- alarm设置超时时间；默认信号处理是终止该进程
- pause调用进程挂起直到捕捉一个信号
  - 执行信号处理程序并返回，pause才返回；pause返回-1，errno设置为EINTR

```c
#include <unistd.h>
unsigned int alarm(unsigned int seconds);
int pause(void);
```

使用longjmp来处理信号竞争

- **解决alarm在pause前超时调用处理函数，后续如果没有其他信号导致pause永远阻塞**
- 如果此时有其他信号捕捉程序运行；可能会直接打断其他信号捕捉

```c
static void
sig_alrm(int signo)
{
  longjmp(env_alrm, 1);
}

unsigned int
sleep2(unsigned int seconds)
{
  if (signal(SIGALRM, sig_alrm) == SIG_ERR)
    return(seconds);
  if (setjmp(env_alrm) == 0) {
    alarm(seconds);   /* start the timer */
    pause();      /* next caught signal wakes us up */
  }
  return(alarm(0));   /* turn off timer, return unslept time */
}
```

```
lzy@lzy:~/Workspace/APUE/chapter10$ ./testsleep
^C
sig_int starting
sig_int finished
sleep2 returned: 4
lzy@lzy:~/Workspace/APUE/chapter10$ gcc sleep2.c testsleep.c -o testsl
eep -lapue
lzy@lzy:~/Workspace/APUE/chapter10$ ./testsleep
^C
sig_int starting
sleep2 returned: 0
```

10.8 信号集

- 信号集表示多个信号

```c
#include <signal.h>
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set);
int sigdelset(sigset_t *set);
int sigismember(const sigset_t *set, int signo);

#define sigemptyset(ptr) (*(ptr) = 0)
#define sigfillset(ptr) (*(ptr) = ~(sigset_t)0, 0)
```

```c
#include  <signal.h>
#include  <errno.h>

/*
 * <signal.h> usually defines NSIG to include signal number 0.
 */
#define SIGBAD(signo) ((signo) <= 0 || (signo) >= NSIG)

int
sigaddset(sigset_t *set, int signo)
{
  if (SIGBAD(signo)) {
    errno = EINVAL;
    return(-1);
  }
  *set |= 1 << (signo - 1);   /* turn bit on */
  return(0);
}

int
sigdelset(sigset_t *set, int signo)
```

```
22  {
23    if (SIGBAD(signo)) {
24      errno = EINVAL;
25      return(-1);
26    }
27    *set &= ~(1 << (signo - 1));  /* turn bit off */
28    return(0);
29  }
30
31  int
32  sigismember(const sigset_t *set, int signo)
33  {
34    if (SIGBAD(signo)) {
35      errno = EINVAL;
36      return(-1);
37    }
38    return((*set & (1 << (signo - 1))) != 0);
39  }
```

10.9 sigprocmask

- 对进程的信号屏蔽字进行检测或更改或同时进行

```
1  #include <signal.h>
2  int sigprocmask(int how, const sigset_t *restrict set, sigset_t *restrict
   oset);
```

- oset非空，当前信号屏蔽字由oset返回
- set非空，按照how来修改当前的屏蔽字

how	description
SIG_BLOCK	新的屏蔽字时当前信号屏蔽字和set的并集；**set是附加信号**
SIG_UNBLOCK	新的屏蔽字时当前信号屏蔽字和set的补集的交集；**set是希望解除的信号**
SIG_SETMASK	新的信号屏蔽是set

10.10 sigpending

- 返回阻塞的信号集

```c
#include <signal.h>
int sigpending(sigset_t *set);
```

- 在休眠期间产生信号；**信号未决但不受阻塞；在sigprocmask返回前，执行sig_quit函数**

10.11 sigaction

- sa_handler包含一个信号捕捉函数的地址
- sa_mask在**信号捕捉函数调用前加入信号屏蔽集；在调用后恢复原来的信号屏蔽集**
- sa_flags指定对信号处理的选项

```c
#include <signal.h>
int sigaction(int signo, const struct sigaction *restrict act, struct
sigaction *restrict oact);

struct sigaction {
    void (*sa_handler) (int);
    sigset_t sa_mask;
    int sa_flags;
    /* alternate handler */
    void (*sa_sigaction) (int, siginfo_t*, void*) ;
};
```

10.12 sigsetjmp和siglongjmp

```c
#include <setjmp.h>
int sigsetjmp(sigjmp_buf env, int save_mask); // 在env中保存屏蔽字
int siglongjmp(sigjmp_buf env, int val);      // 从env中恢复屏蔽字
```

10.13 sigsuspend

- 使用一个原子操作先恢复信号屏蔽字，使进程休眠
- 和pause一样，sigsuspend没有成功返回值，只有执行了一个信号处理函数之后sigsuspend才返回，返回值为-1，errno设置为EINTR。
- 调用sigsuspend时，进程的信号屏蔽字由sigmask参数指定，可以通过**指定sigmask来临时解除对某个信号的屏蔽，然后挂起等待**，当sigsuspend返回时，进程的信号屏蔽字恢复为原来的值，如果原来对该信号是屏蔽的，从sigsuspend返回后仍然是屏蔽的。

```c
#include <signal.h>
int sigsuspend(const sigset_t *sigmask);
```

10.14 abort

- 使程序异常终止
- 此函数将SIGABRT信号发送给调用进程
  - 冲洗所有打开的文件流

```c
#include <stdlib.h>
void abort(void);
```

10.15 system

- 不应该使得父子进程都捕捉终端产生的信号：SIGINT和SIGQUIT
- 父进程忽略这两个信号；只有子进程接受这两个信号

```c
#include  <sys/wait.h>
#include  <errno.h>
#include  <signal.h>
#include  <unistd.h>

int
system(const char *cmdstring) /* with appropriate signal handling */
{
  pid_t       pid;
  int         status;
  struct sigaction  ignore, saveintr, savequit;
  sigset_t      chldmask, savemask;

  if (cmdstring == NULL)
    return(1);    /* always a command processor with UNIX */

  ignore.sa_handler = SIG_IGN;  /* ignore SIGINT and SIGQUIT */
  sigemptyset(&ignore.sa_mask);
  ignore.sa_flags = 0;
  if (sigaction(SIGINT, &ignore, &saveintr) < 0)
    return(-1);
  if (sigaction(SIGQUIT, &ignore, &savequit) < 0)
    return(-1);
  sigemptyset(&chldmask);     /* now block SIGCHLD */
  sigaddset(&chldmask, SIGCHLD);
```

```
26    if (sigprocmask(SIG_BLOCK, &chldmask, &savemask) < 0)
27      return(-1);
28
29    if ((pid = fork()) < 0) {
30      status = -1;  /* probably out of processes */
31    } else if (pid == 0) {      /* child */
32      /* restore previous signal actions & reset signal mask */
33      sigaction(SIGINT, &saveintr, NULL);
34      sigaction(SIGQUIT, &savequit, NULL);
35      sigprocmask(SIG_SETMASK, &savemask, NULL);
36
37      execl("/bin/sh", "sh", "-c", cmdstring, (char *)0);
38      _exit(127);   /* exec error */
39    } else {            /* parent */
40      while (waitpid(pid, &status, 0) < 0)
41        if (errno != EINTR) {
42          status = -1; /* error other than EINTR from waitpid() */
43          break;
44        }
45    }
46
47    /* restore previous signal actions & reset signal mask */
48    if (sigaction(SIGINT, &saveintr, NULL) < 0)
49      return(-1);
50    if (sigaction(SIGQUIT, &savequit, NULL) < 0)
51      return(-1);
52    if (sigprocmask(SIG_SETMASK, &savemask, NULL) < 0)
53      return(-1);
54
55    return(status);
56  }
```

10.16 sleep相关函数

- nanosleep提供了纳秒级别的精度
- clock_nanosleep相对于特定时钟的延迟时间来挂起调用的线程
  - flags为0表示时间是相对的；TIMER_ABSTIME是绝对时间

```
1  #include <time.h>
2  int nanosleep(const struct timespec *reqtp, struct timespec *remtp);
3  int clock_nanosleep(clockid_t clock_id, int flags, const struct timespec
   *reqtp, struct timespec *remtp);
```