第十章 信号

1. 信号的概念

- 信号的产生
  - 用户按键：DELETE、Ctrl+C
  - 硬件异常：除0错误、无效的内存访问
  - 进程用kill函数将信号发送给另一个进程或进程组
  - 用户用kill命令将信号发送给其它进程
  - 检测到某种软件条件已经发生，并通知有关进程：SIGURG、SIGPIPE、SIGALRM

1. 信号的概念

- 信号的处理
  - 忽略信息（SIGKILL和SIGSTOP除外）
  - 捕捉信号
  - 执行系统默认动作

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

1. 信号的概念

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

2. signal函数

- UNIX信号机制最简单的界面是signal函数

```
#include <signal.h>

void  (*signal ( int  signo,  void (*func) (int) ))  (int);
```

SIG_IGN 忽略信号
SIG_DFL 系统默认动作
signal handler函数

2. signal函数

- UNIX信号机制最简单的界面是signal函数

```
#include <signal.h>

typedef  void  sigfunc(int);

sigfunc  *signal(int,  sigfunc  *);

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

```
#include    <signal.h>
#include    "ourhdr.h"

static void sig_usr(int);    /* one handler for both signals */

int
main(void)
{
    if (signal(SIGUSR1, sig_usr) == SIG_ERR)
        err_sys("can't catch SIGUSR1");
    if (signal(SIGUSR2, sig_usr) == SIG_ERR)
        err_sys("can't catch SIGUSR2");

    for ( ; ; )
        pause();
}

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);
    return;
}
```

2. signal函数


```
$ a.out &                          在后台启动进程
[1]     4720                       作业控制shell打印作业号和进程ID
$ kill -USR1 4720                  向该进程发送SIGUSR1
received SIGUSR1
$ kill -USR2 4720                  向该进程发送SIGUSR2
received SIGUSR2
$ kill 4720                        向该进程发送SIGTERM
[1] + Terminated          a.out &
```

3. 中断的系统调用

函　数	系　统	信号处理程序仍被安装	阻塞信号的能力	被中断系统调用的自动再起动
signal,	V7, SVR2, SVR3, SVR4			决不
sigset,sighold,sigrelse sigignore,sigpause	SVR3, SVR4	•	•	决不
signal,sigvec,sigblock	4.2BSD	•	•	总是
sigsetmask,sigpause	4.3BSD, 4.3+BSD	•	•	默认
sigaction,sigprocmask sigpending,sigsuspend	POSIX.1	•	•	未说明
	SVR4	•	•	可选
	4.3+BSD	•	•	可选

4. 可再入函数

versity

_exit	fork	pipe	stat
abort*	fstat	read	sysconf
access	getegid	rename	tcdrain
alarm	geteuid	rmdir	tcflow
cfgetispeed	getgid	setgid	tcflush
cfgetospeed	getgroups	setpgid	tcgetattr
cfsetispeed	getpgrp	setsid	tcgetpgrp
cfsetospeed	getpid	setuid	tcsendbreak
chdir	getppid	sigaction	tcsetattr
chmod	getuid	sigaddset	tcsetpgrp
chown	kill	sigdelset	time
close	link	sigemptyset	times
creat	longjmp*	sigfillset	umask
dup	lseek	sigismember	uname
dup2	mkdir	signal*	unlink
execle	mkfifo	sigpending	utime
execve	open	sigprocmask	wait
exit*	pathconf	sigsuspend	waitpid
fcntl	pause	sleep	write

5. kill和raise函数

- kill函数将信号发送给进程或进程组

```
#include <sys/types.h>
#include <signal.h>
int kill (pid_t pid, int signo);
```

- raise函数允许进程向自身发送信号

```
#include <sys/types.h>
#include <signal.h>
int raise(int signo);
```

6. alarm和pause函数

- alarm函数设置一个闹钟，产生SIGALRM信号

```
#include <unistd.h>

unsigned int alarm(unsigned int seconds);
```

- pause函数使调用进程挂起直至捕捉到一个信号

```
#include <unistd.h>

int pause(void);
```

6. alarm和pause函数


```
#include    <signal.h>
#include    <unistd.h>

static void
sig_alrm(int signo)
{
    return; /* nothing to do, just return to wake up the pause */
}

unsigned int
sleep1(unsigned int nsecs)
{
    if (signal(SIGALRM, sig_alrm) == SIG_ERR)
        return(nsecs);
    alarm(nsecs);          /* start the timer */
    pause();               /* next caught signal wakes us up */
    return( alarm(0) ); /* turn off timer, return unslept time */
}
```

```
#include    <signal.h>
#include    "ourhdr.h"

int
main(void)
{
    int     n;
    char    line[MAXLINE];

    if (signal(SIGALRM, sig_alrm) == SIG_ERR)
        err_sys("signal(SIGALRM) error");
    alarm(10);
    if ( (n = read(STDIN_FILENO, line, MAXLINE)) < 0)
        err_sys("read error");
    alarm(0);

    write(STDOUT_FILENO, line, n);

    exit(0);
}

static void
sig_alrm(int signo)
{
    return; /* nothing to do, just return to interrupt the read */
}
```

7. 信号集

- 表示信号集的数据类型

```
#include <signal.h>
sigset_t

int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set, int signo);
int sigdelset(sigset_t *set, int signo);

int sigismember(const sigset_t *set, int signo);
```

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

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

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)
{
    if (SIGBAD(signo)) { errno = EINVAL; return(-1); }

    *set &= ~(1 << (signo - 1));     /* turn bit off */
    return(0);
}

int
sigismember(const sigset_t *set, int signo)
{
    if (SIGBAD(signo)) { errno = EINVAL; return(-1); }

    return( (*set & (1 << (signo - 1))) != 0 );
}
```

8. sigprocmask函数

- 检测或更改进程的信号屏蔽字

```
#include <signal.h>

int sigprocmask(int how, const sigset_t *set, sigset_t *oset);
```

how	说　明
SIG_BLOCK	该进程新的信号屏蔽字是其当前信号屏蔽字和 *set*指向信号集的并集。*set*包含了我们希望阻塞的附加信号
SIG_UNBLOCK	该进程新的信号屏蔽字是其当前信号屏蔽字和 *set*所指向信号集的交集。*set*包含了我们希望解除阻塞的信号
SIG_SETMASK	该进程新的信号屏蔽是*set*指向的值

9. sigpending函数

- 返回对于调用进程被阻塞不能递送和当前未决的信号集

```
#include <signal.h>

int  sigpending(sigset_t  *set);
```

10. sigaction函数

- 检查或修改与指定信号相关联的处理动作

```
#include <signal.h>

int sigaction(int signo, const struct sigaction *act, struct
    sigacton *oact);

struct sigaction {
    void (*sa_handler)(); /* addr of signal handler, or SIG_IGN,
                             or SIG_DFL */
    sigset_t sa_mask;     /* additional signals to block */
    int sa_flags;         /* signal options */
}
```

可 选 项	POSIX.1	SVR4	4.3+BSD	说 明
SA_NOCLDSTOP	•	•	•	若*signo*是SIGCHLD，当一子进程停止时（作业控制），不产生此信号。当一子进程终止时，仍旧产生此信号（但请参阅下面说明的SVR4 SA_NOCLDWAIT可选项）
SA_RESTART		•	•	由此信号中断的系统调用自动再起动（参见 10.5节）
SA_ONSTACK		•	•	若用sigaltstack(2)已说明了一替换栈，则此信号递送给替换栈上的进程
SA_NOCLDWAIT		•		若*signo*是SIGCHLD，则当调用进程的子进程终止时，不创建僵死进程。若调用进程在后面调用 wait，则阻塞到它所有子进程都终止，此时返回 -1，errno设置为ECHILD（见10.7节）
SA_NODEFER		•		当捕捉到此信号时，在执行其信号捕捉函数时，系统不自动阻塞此信号。注意，此种类型的操作对应于早期的不可靠信号
SA_RESETHAND		•		对此信号的处理方式在此信号捕捉函数的入口处复置为SIG_DFL。注意，此种类型的信号对应于早期的不可靠信号
SA_SIGINFO		•		此选项对信号处理程序提供了附加信息。详细情况见10.21节

```
/* Reliable version of signal(), using POSIX sigaction().  */

#include    <signal.h>
#include    "ourhdr.h"

Sigfunc *
signal(int signo, Sigfunc *func)
{
    struct sigaction    act, oact;

    act.sa_handler = func;
    sigemptyset(&act.sa_mask);
    act.sa_flags = 0;
    if (signo == SIGALRM) {
#ifdef  SA_INTERRUPT
        act.sa_flags |= SA_INTERRUPT;    /* SunOS */
#endif
    } else {
#ifdef  SA_RESTART
        act.sa_flags |= SA_RESTART;      /* SVR4, 4.3+BSD */
#endif
    }
    if (sigaction(signo, &act, &oact) < 0)
        return(SIG_ERR);
    return(oact.sa_handler);
}
```

ersity

11. abort函数

- 使程序异常终止

```
#include<stdlib.h>

void  abort(void);
```

```
void
abort(void)          /* POSIX-style abort() function */
{
    sigset_t            mask;
    struct sigaction    action;

        /* caller can't ignore SIGABRT, if so reset to default */
    sigaction(SIGABRT, NULL, &action);
    if (action.sa_handler == SIG_IGN) {
        action.sa_handler = SIG_DFL;
        sigaction(SIGABRT, &action, NULL);
    }

    if (action.sa_handler == SIG_DFL)
        fflush(NULL);           /* flush all open stdio streams */

        /* caller can't block SIGABRT; make sure it's unblocked */
    sigfillset(&mask);
    sigdelset(&mask, SIGABRT);  /* mask has only SIGABRT turned off */
    sigprocmask(SIG_SETMASK, &mask, NULL);

    kill(getpid(), SIGABRT);    /* send the signal */

        /* if we're here, process caught SIGABRT and returned */
    fflush(NULL);               /* flush all open stdio streams */

    action.sa_handler = SIG_DFL;
    sigaction(SIGABRT, &action, NULL);  /* reset disposition to default */
    sigprocmask(SIG_SETMASK, &mask, NULL);  /* just in case ... */

    kill(getpid(), SIGABRT);                /* and one more time */

    exit(1);    /* this should never be executed ... */
}
```

12. system函数

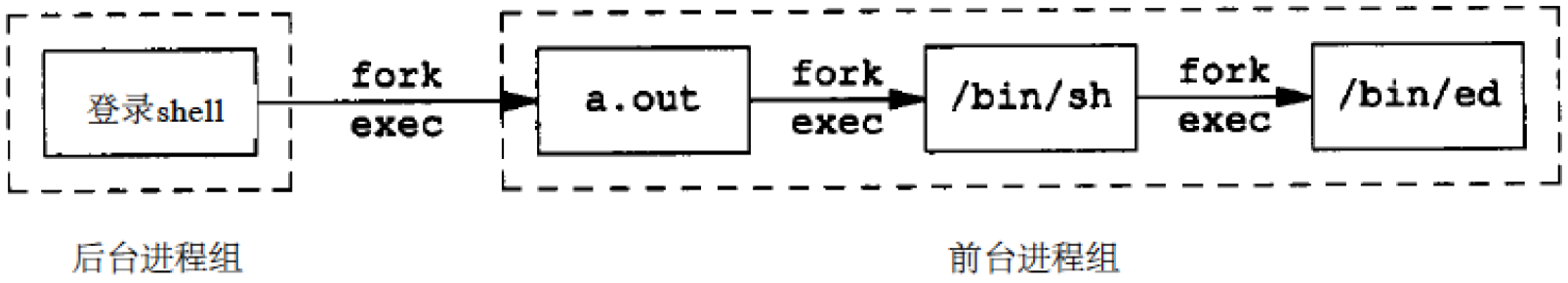
- 调用其它程序

```
#include <sys/types.h>

int system(const char *cmdstring);
```

12. system函数

- 调用其它程序



13. sleep函数

- 进程挂起

```
#include <unistd.h>

unsigned int sleep(unsigned int seconds);
```

14. 作业控制信号

- 作业相关的六种控制信号

SIGCHLD	子进程已停止或终止
SIGCONT	如果进程已停止，则使其继续运行
SIGSTOP	停止信号（不能被捕捉或忽略）
SIGTSTP	交互停止信号
SIGTTIN	后台进程组的成员读控制终端
SIGTTOU	后台进程组的成员写控制终端