第十章 信号

非会话首进程的组长进程:

->HUP

```
tongue aupe # ./process-quit
Child created!
```

```
tongue ~ # ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init|bash'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
    0     0     0 root     29424 29424 29424 29422 29424 pts/2    Ss+  bash
    0     0     0 root     30100 29449 29449 29447 29449 pts/3    Ss   bash
    0     0     0 root     30098 29976 29976 29974 29976 pts/1    Ss   bash
    0     0     0 root     30098 29976 30098 29976 30098 pts/1    S+   process-quit
    0     0     0 root     30098 29976 30098 30098 30099 pts/1    S+   process-quit
```

```
tongue ~ # kill -s HUP 30098; ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
    0     0     0 root     29976 29976 30098     1 30099 pts/1    S    process-quit
```

```
ppid: 29976, pid: 30098, signo: 1
tongue aupe #
```

->INT

信号来源分两种情况:

1、来自终端;

```
tongue aupe # ./process-quit
Child created!
^Cppid: 29976, pid: 30178, signo: 2
ppid: 30178, pid: 30179, signo: 2
```

2、来自kill;

```
tongue aupe # ./process-quit
Child created!
```

```
tongue ~ # ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init|bash'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
    0     0     0 root     29424 29424 29424 29422 29424 pts/2    Ss+  bash
    0     0     0 root     30117 29449 29449 29447 29449 pts/3    Ss   bash
    0     0     0 root     30115 29976 29976 29974 29976 pts/1    Ss   bash
    0     0     0 root     30115 29976 30115 29976 30115 pts/1    S+   process-quit
    0     0     0 root     30115 29976 30115 30115 30116 pts/1    S+   process-quit
```

```
tongue ~ # kill -s INT 30115; ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
    0     0     0 root     29976 29976 30115     1 30116 pts/1    S    process-quit
```

```
ppid: 29976, pid: 30115, signo: 2
tongue aupe #
```

->QUIT

信号来源分两种情况:

1、来自终端;

```
tongue aupe # ./process-quit
Child created!
^\ppid: 30180, pid: 30181, signo: 3
ppid: 29976, pid: 30180, signo: 3
```

2、来自kill;

```
tongue aupe # ./process-quit
Child created!
```

```
tongue ~ # ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init|bash'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
    0     0     0 root     29424 29424 29424 29422 29424 pts/2    Ss+  bash
    0     0     0 root     30127 29449 29449 29447 29449 pts/3    Ss   bash
    0     0     0 root     30125 29976 29976 29974 29976 pts/1    Ss   bash
    0     0     0 root     30125 29976 30125 29976 30125 pts/1    S+   process-quit
    0     0     0 root     30125 29976 30125 30125 30126 pts/1    S+   process-quit
tongue ~ # kill -s QUIT 30125; ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
    0     0     0 root     29976 29976 30125     1 30126 pts/1    S    process-quit
```

```
ppid: 29976, pid: 30125, signo: 3
tongue aupe #
```

->ABRT

```
tongue aupe # ./process-quit
Child created!
tongue ~ # ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init|bash'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
    0     0     0 root     29424 29424 29424 29422 29424 pts/2    Ss+  bash
    0     0     0 root     30136 29449 29449 29447 29449 pts/3    Ss   bash
    0     0     0 root     30134 29976 29976 29974 29976 pts/1    Ss   bash
    0     0     0 root     30134 29976 30134 29976 30134 pts/1    S+   process-quit
    0     0     0 root     30134 29976 30134 30134 30135 pts/1    S+   process-quit
tongue ~ # kill -s ABRT 30134; ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
    0     0     0 root     29976 29976 30134     1 30135 pts/1    S    process-quit
```

```
ppid: 29976, pid: 30134, signo: 6
tongue aupe #
```

->KILL

```
tongue aupe # ./process-quit
Child created!
tongue ~ # ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init|bash'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
    0     0     0 root     29424 29424 29424 29422 29424 pts/2    Ss+  bash
    0     0     0 root     30158 29449 29449 29447 29449 pts/3    Ss   bash
    0     0     0 root     30156 29976 29976 29974 29976 pts/1    Ss   bash
    0     0     0 root     30156 29976 30156 29976 30156 pts/1    S+   process-quit
    0     0     0 root     30156 29976 30156 30156 30157 pts/1    S+   process-quit
tongue ~ # kill -s KILL 30156; ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
    0     0     0 root     29976 29976 30156     1 30157 pts/1    S    process-quit
```

```
Killed
tongue aupe #
```

->TERM

```
tongue aupe # ./process-quit
Child created!
tongue ~ # ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init|bash'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
    0     0     0 root     29424 29424 29424 29422 29424 pts/2    Ss+  bash
    0     0     0 root     30166 29449 29449 29447 29449 pts/3    Ss   bash
    0     0     0 root     30164 29976 29976 29974 29976 pts/1    Ss   bash
    0     0     0 root     30164 29976 30164 29976 30164 pts/1    S+   process-quit
    0     0     0 root     30164 29976 30164 30164 30165 pts/1    S+   process-quit
```

```
tongue ~ # kill -s TERM 30164; ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
    0     0     0 root     29976 29976 30164     1 30165 pts/1    S    process-quit
```

```
ppid: 29976, pid: 30164, signo: 15
tongue aupe #
```

会话首进程:

->HUP

```
tongue aupe # ./process-quit > ~/out.msg
```

```
tongue ~ # ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init|bash'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
    0     0     0 root     29424 29424 29424 29422 29424 pts/2    Ss+  bash
    0     0     0 root     30254 29449 29449 29447 29449 pts/3    Ss   bash
    0     0     0 root     30252 30239 30239 30237 30239 pts/0    Ss   bash
    0     0     0 root     30252 30239 30252 30239 30252 pts/0    S+   process-quit
    0     0     0 root     30252 30239 30252 30252 30253 pts/0    S+   process-quit
```

SIGHUP　如果终端接口检测到一个连接断开，则将此信号发送给与该终端相关的控制进程（会话首进程）。见图9-11，此信号被送给session结构中的s_leader字段所指向的进程。仅当终端的CLOCAL标志没有设置时，在上述条件下才产生此信号。（如果所连接的终端是本地的，则设置该终端的CLOCAL标志。它告诉终端驱动程序忽略所有调制解调器的状态行。第18章将说明如何设置此标志。）

注意，接到此信号的会话首进程可能在后台，例如，参见图9-7。这有别于由终端正常产生的几个信号（中断、退出和挂起），这些信号总是传递给前台进程组。

如果会话首进程终止，则也产生此信号。在这种情况下，此信号将被发送给前台进程组中的每一个进程。

通常用此信号通知守护进程（见第13章），以重新读取它们的配置文件。为此目的选用SIGHUP的理由是，守护进程不会有控制终端，而且通常决不会接收到这种信号。

```
tongue ~ # kill -s HUP 30239; ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
```

```
tongue ~ # tail out.msg
ppid: 30239, pid: 30252, signo: 1
Child created!  顺序值得研究哦
ppid: 1, pid: 30253, signo: 1
```

->INT

```
tongue aupe # ./process-quit > ~/out.msg
```

```
tongue ~ # ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init|bash'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
    0     0     0 root     29424 29424 29424 29422 29424 pts/2    Ss+  bash
    0     0     0 root     30268 29449 29449 29447 29449 pts/3    Ss   bash
    0     0     0 root     30266 30261 30261 30259 30261 pts/0    Ss   bash
    0     0     0 root     30266 30261 30266 30261 30266 pts/0    S+   process-quit
    0     0     0 root     30266 30261 30266 30266 30267 pts/0    S+   process-quit
```

```
tongue ~ # kill -s INT 30261; ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
    0     0     0 root     30266 30261 30266 30261 30266 pts/0    S+   process-quit
    0     0     0 root     30266 30261 30266 30266 30267 pts/0    S+   process-quit
tongue ~ # tail out.msg
tongue ~ #
```

郁闷吧！继续看下面。

bash对此信号做了特殊处理，故在接受到该信号后不执行默认操作(可参阅242页，**程序启动**部分)。

```
tongue aupe # ^C
tongue aupe #
```

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。于是，当按中断键时就不会影响到后台进程。如果没有执行这样的处理，那么当按中断键时，它不但会终止前台进程，还会终止所有后台进程。

很多捕捉这两个信号的交互式程序具有下列形式的代码：

```
void sig_int(int), sig_quit(int);

if (signal(SIGINT, SIG_IGN) != SIG_IGN)
    signal(SIGINT, sig_int);
if (signal(SIGQUIT, SIG_IGN) != SIG_IGN)
    signal(SIGQUIT, sig_quit);
```

这样处理后，仅当信号当前未被忽略时，进程才会捕捉它们。

信号产生，并未做默认操作。

试试同样的步骤，对另一个进程组是什么效果。

```
tongue ~ # ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init|bash|nginx'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
    0     0     0 root     29424 29424 29424 29422 29424 pts/2    Ss+  bash
    0     0     0 root     30276 29449 29449 29447 29449 pts/3    Ss   bash
    0     0     0 root        -1 29685 29685     1 29685 ?        Ss   nginx
 1000  1000  1000 www         -1 29685 29685 29685 29686 ?        S    nginx
 1000  1000  1000 www         -1 29685 29685 29685 29687 ?        S    nginx
 1000  1000  1000 www         -1 29685 29685 29685 29688 ?        S    nginx
 1000  1000  1000 www         -1 29685 29685 29685 29689 ?        S    nginx
    0     0     0 root     30261 30261 30261 30259 30261 pts/0    Ss+  bash
tongue ~ # kill -s INT 29685; ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init|nginx'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
tongue ~ #                    都对号入座了
```

->QUIT

```
tongue aupe # ./process-quit > ~/out.msg
```

```
tongue ~ # ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init|bash'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
    0     0     0 root     29424 29424 29424 29422 29424 pts/2    Ss+  bash
    0     0     0 root     30285 29449 29449 29447 29449 pts/3    Ss   bash
    0     0     0 root     30283 30261 30261 30259 30261 pts/0    Ss   bash
    0     0     0 root     30283 30261 30283 30261 30283 pts/0    S+   process-quit
    0     0     0 root     30283 30261 30283 30283 30284 pts/0    S+   process-quit
tongue ~ # kill -s QUIT 30261; ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
    0     0     0 root     30283 30261 30283 30261 30283 pts/0    S+   process-quit
    0     0     0 root     30283 30261 30283 30283 30284 pts/0    S+   process-quit
```

原因与INT一样。

```
tongue ~ # /etc/init.d/nginx restart
 * Checking nginx' configuration ...
 * Stopping nginx ...
 * start-stop-daemon: no pid found in `/run/nginx.pid'
 * Starting nginx ...
tongue ~ # ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init|bash|nginx'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
    0     0     0 root     29424 29424 29424 29422 29424 pts/2    Ss+  bash
    0     0     0 root     30320 29449 29449 29447 29449 pts/3    Ss   bash
    0     0     0 root     30261 30261 30261 30259 30261 pts/0    Ss+  bash
    0     0     0 root        -1 30313 30313     1 30313 ?        Ss   nginx
 1000  1000  1000 www         -1 30313 30313 30313 30314 ?        S    nginx
 1000  1000  1000 www         -1 30313 30313 30313 30315 ?        S    nginx
 1000  1000  1000 www         -1 30313 30313 30313 30316 ?        S    nginx
 1000  1000  1000 www         -1 30313 30313 30313 30317 ?        S    nginx
tongue ~ # kill -s QUIT 30313; ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init|nginx'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
```

->ABRT

```
tongue aupe # ./process-quit > ~/out.msg
```

```
tongue ~ # ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init|bash'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
    0     0     0 root     29424 29424 29424 29422 29424 pts/2    Ss+  bash
    0     0     0 root     30326 29449 29449 29447 29449 pts/3    Ss   bash
    0     0     0 root     30324 30261 30261 30259 30261 pts/0    Ss   bash
    0     0     0 root     30324 30261 30324 30261 30324 pts/0    S+   process-quit
    0     0     0 root     30324 30261 30324 30324 30325 pts/0    S+   process-quit
tongue ~ # kill -s ABRT 30261; ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
    0     0     0 root        -1 30261 30324     1 30324 ?        S    process-quit
    0     0     0 root        -1 30261 30324 30324 30325 ?        S    process-quit
tongue ~ # tail out.msg
```

后台进程实例

```
tongue ~ # /etc/init.d/nginx restart
 * Checking nginx' configuration ...
 * Stopping nginx ...
 * start-stop-daemon: no pid found in `/run/nginx.pid'
 * Starting nginx ...
tongue ~ # ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init|bash|nginx'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
    0     0     0 root     29424 29424 29424 29422 29424 pts/2    Ss+  bash
    0     0     0 root     30431 29449 29449 29447 29449 pts/3    Ss   bash
    0     0     0 root     30348 30348 30348 30346 30348 pts/0    Ss+  bash
    0     0     0 root        -1 30424 30424     1 30424 ?        Ss   nginx
 1000  1000  1000 www         -1 30424 30424 30424 30425 ?        S    nginx
 1000  1000  1000 www         -1 30424 30424 30424 30426 ?        S    nginx
 1000  1000  1000 www         -1 30424 30424 30424 30427 ?        S    nginx
 1000  1000  1000 www         -1 30424 30424 30424 30428 ?        S    nginx
tongue ~ # kill -s ABRT 30424; ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init|nginx'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
 1000  1000  1000 www         -1 30424 30424     1 30425 ?        S    nginx
 1000  1000  1000 www         -1 30424 30424     1 30426 ?        S    nginx
 1000  1000  1000 www         -1 30424 30424     1 30427 ?        S    nginx
 1000  1000  1000 www         -1 30424 30424     1 30428 ?        S    nginx
```

->KILL

```
tongue aupe # ./process-quit > ~/out.msg
```

```
tongue ~ # ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init|bash'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
    0     0     0 root     29424 29424 29424 29422 29424 pts/2    Ss+  bash
    0     0     0 root     30341 29449 29449 29447 29449 pts/3    Ss   bash
    0     0     0 root     30339 30335 30335 30333 30335 pts/0    Ss   bash
    0     0     0 root     30339 30335 30339 30335 30339 pts/0    S+   process-quit
    0     0     0 root     30339 30335 30339 30339 30340 pts/0    S+   process-quit
```

```
tongue ~ # kill -s KILL 30335; ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
tongue ~ # tail out.msg
Child created!
ppid: 30339, pid: 30340, signo: 1
ppid: 1, pid: 30339, signo: 1
```

注意这里的PPID，还有接收到的信号ID。

SIGHUP对应的信号ID为1，可图上我们明明发送的是SIGKILL呀，SIGKILL对应的信号ID为9。

解答：

> 如果会话首进程终止，则也产生此信号。在这种情况下，此信号将被发送给前台进程组中的每一个进程。

后台进程实例

```
tongue ~ # killall nginx
tongue ~ # /etc/init.d/nginx restart
 * Checking nginx' configuration ...
 * Stopping nginx ...
 * start-stop-daemon: no pid found in `/run/nginx.pid'
 * Starting nginx ...
tongue ~ # ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init|bash|nginx'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
    0     0     0 root     29424 29424 29424 29422 29424 pts/2    Ss+  bash
    0     0     0 root     30511 29449 29449 29447 29449 pts/3    Ss   bash
    0     0     0 root     30348 30348 30348 30346 30348 pts/0    Ss+  bash
    0     0     0 root        -1 30504 30504     1 30504 ?        Ss   nginx
 1000  1000  1000 www         -1 30504 30504 30504 30505 ?        S    nginx
 1000  1000  1000 www         -1 30504 30504 30504 30506 ?        S    nginx
 1000  1000  1000 www         -1 30504 30504 30504 30507 ?        S    nginx
 1000  1000  1000 www         -1 30504 30504 30504 30508 ?        S    nginx
```

```
tongue ~ # kill -s KILL 30504; ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init|nginx'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
 1000  1000  1000 www         -1 30504 30504     1 30505 ?        S    nginx
 1000  1000  1000 www         -1 30504 30504     1 30506 ?        S    nginx
 1000  1000  1000 www         -1 30504 30504     1 30507 ?        S    nginx
 1000  1000  1000 www         -1 30504 30504     1 30508 ?        S    nginx
```

->TERM

```
tongue aupe # ./process-quit > ~/out.msg
```

```
tongue ~ # ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init|bash'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
    0     0     0 root     29424 29424 29424 29422 29424 pts/2    Ss+  bash
    0     0     0 root     30356 29449 29449 29447 29449 pts/3    Ss   bash
    0     0     0 root     30354 30348 30348 30346 30348 pts/0    Ss   bash
    0     0     0 root     30354 30348 30354 30348 30354 pts/0    S+   process-quit
    0     0     0 root     30354 30348 30354 30354 30355 pts/0    S+   process-quit
```

```
tongue ~ # kill -s TERM 30348; ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
    0     0     0 root     30354 30348 30354 30348 30354 pts/0    S+   process-quit
    0     0     0 root     30354 30348 30354 30354 30355 pts/0    S+   process-quit
tongue ~ # tail out.msg
```

bash应该是对此信号做了特殊处理，避免执行默认动作。

后台进程实例

```
tongue ~ # killall nginx
tongue ~ # /etc/init.d/nginx restart
 * Checking nginx' configuration ...
 * Stopping nginx ...
 * start-stop-daemon: no pid found in `/run/nginx.pid'
 * Starting nginx ...
tongue ~ # ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init|bash|nginx'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
    0     0     0 root     29424 29424 29424 29422 29424 pts/2    Ss+  bash
    0     0     0 root     30582 29449 29449 29447 29449 pts/3    Ss   bash
    0     0     0 root     30348 30348 30348 30346 30348 pts/0    Ss+  bash
    0     0     0 root        -1 30575 30575     1 30575 ?        Ss   nginx
 1000  1000  1000 www         -1 30575 30575 30575 30576 ?        S    nginx
 1000  1000  1000 www         -1 30575 30575 30575 30577 ?        S    nginx
 1000  1000  1000 www         -1 30575 30575 30575 30578 ?        S    nginx
 1000  1000  1000 www         -1 30575 30575 30575 30579 ?        S    nginx
tongue ~ # kill -s TERM 30575; ps ax -o uid,gid,euid,user,tpgid,sid,pgid,ppid,pid,tty,stat,comm|egrep 'USER|process-quit|init|nginx'
  UID   GID  EUID USER     TPGID   SID  PGID  PPID   PID TT       STAT COMMAND
    0     0     0 root        -1     1     1     0     1 ?        Ss   init
```

上面用到的源码

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/signal.h>

void sig_fun(int signo) {
    printf( "ppid: %d, pid: %d, signo: %d\n", getppid(), getpid(), signo);
}

int main(int argc, const char *argv[]) {
    pid_t pid;
    signal(SIGHUP, sig_fun);        // 1
    signal(SIGINT, sig_fun);        // 2
    signal(SIGQUIT, sig_fun);       // 3
    signal(SIGABRT, sig_fun);       // 6
    signal(SIGKILL, sig_fun);       // 9
    signal(SIGTERM, sig_fun);       // 15
    if (0 > (pid = fork())) {
        printf( "error by fork!\n");
    } else if (0 == pid) {
        printf( "Child created!\n");
    }
    sleep(100);
    return 0;
}
```

```
tongue aupe # gcc process-quit.c -o process-quit
tongue aupe #
```

补充材料：http://en.wikipedia.org/wiki/Unix_signal

> SIGTERM:
> SIGTERM是kill或killall命令发送给进程的默认信号。它始一个进程终止，但与SIGKILL信号不同，它可被进程捕获、解释(或忽略)。因此，SIGTERM如同询问进程来很好的终止，允许清理操作和文件的关闭。为此，unix系统关闭期间，init给所有进程发送SIGTERM，并不急着关闭电源，等待数秒，然后发布SIGKILL给任何或者的进程，强制终止它们。

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/signal.h>

void sig_fun(int signo) {
    printf( "ppid: %d, pid: %d, signo: %d\n", getppid(), getpid(), signo);
}

int main(int argc, const char *argv[]) {
    pid_t pid;
    signal(SIGHUP, sig_fun);        // 1
    signal(SIGINT, sig_fun);        // 2
    signal(SIGQUIT, sig_fun);       // 3
    signal(SIGABRT, sig_fun);       // 6
    signal(SIGKILL, sig_fun);       // 9
    signal(SIGTERM, sig_fun);       // 15
    while(1) {
        sleep(100);
        printf("Oh\n");
    }
    return 0;
}
/opt/drill_ground/aupe/signal-int-cont.c [FORMAT=unix:utf-8] [TYPE=C] [COL=
```

上述代码，通常会在sleep时收到信号，这时会系统中断sleep，然后执行信号处理函数。完成后，接着执行sleep下面的函数，这里是printf。

http://www.ibm.com/developerworks/cn/linux/l-ipc/part2/index1.html
http://www.ibm.com/developerworks/cn/linux/l-ipc/part2/index2.html

习题：

10-1、没有了for(;;)，pause()将不在任何循环当中，那么程序会在接收到一次中断后便退出。原因是中断pause后，进程会执行pause后面的逻辑。显然程序清单10-1的pause之后，没其他什么逻辑了；

```c
#include <stdio.h>
#include <sys/types.h>
#include <signal.h>

static void sig_user(int signo) {
    switch (signo) {
        case SIGUSR1: {
            printf("Capture the SIGUSR1\n");
        }
        break;
        case SIGUSR2: {
            printf("Capture the SIGUSR2\n");
        }
        break;
        default:
            printf("Does not recognize signo: %d\n", signo);
    }
}

int main(int argc, const char *argv[]) {
    signal( SIGUSR1, sig_user);
    signal( SIGUSR2, sig_user);
    pause();
    return 0;
}
```

```
tongue aupe # gcc 10-1.c -o 10-1
tongue aupe # ./10-1
```

```
tongue ~ # ps aux|grep -v 'grep'|grep './10-1'
root      31898  0.0  0.0   4056   352 pts/0    S+   00:58   0:00 ./10-1
tongue ~ # kill -s USR1 31898
tongue ~ # tty
/dev/pts/1
```

```
tongue aupe # ./10-1

Capture the SIGUSR1
tongue aupe #
```

10-2、以下是sig2str实现的示例，可在switch中加入更多接收；

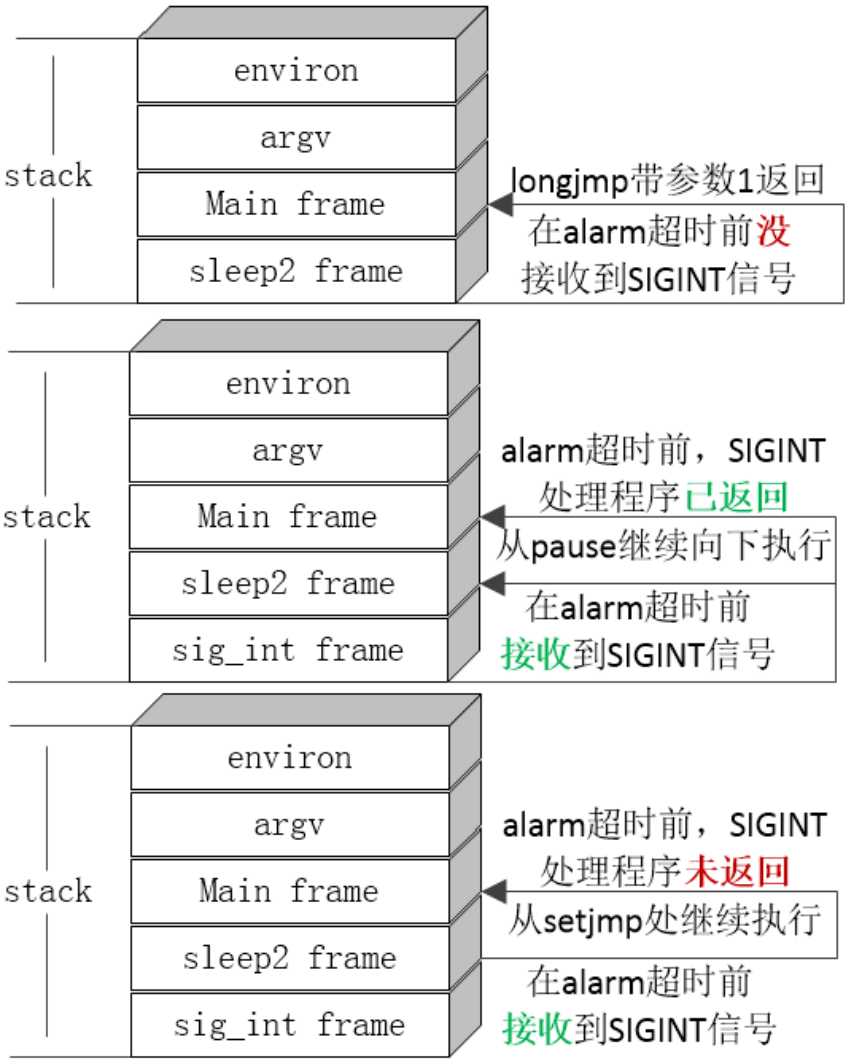
```c
#include <stdio.h>
#include <sys/types.h>
#include <string.h>
#include <signal.h>

#define SIG2STR_MAX 256

int sig2str(int signo, char *str) {
    char tmp_str[SIG2STR_MAX], *pTmp_str = tmp_str;
    bzero(tmp_str, sizeof(SIG2STR_MAX));
    switch (signo) {
        case SIGUSR1: {
            pTmp_str = "SIGUSR1";
        }
        break;
        case SIGUSR2: {
            pTmp_str = "SIGUSR2";
        }
        break;
        default:
            sprintf( tmp_str, "Does not recognize signo: %d", signo);
    }
    if (SIG2STR_MAX < strlen(tmp_str)) {
        sprintf( tmp_str, "You need more space in your str object!");
        return 1;
    }
    memcpy( str, pTmp_str, strlen(pTmp_str) + 1); // last is '\0';
    return 0;
}
```

```c
int main(int argc, const char *argv[]) {
    char sig_str[SIG2STR_MAX];
    if (2 > argc) {
        printf("Usage: ./10-2 12\n");
        return 1;
    }
    sig2str( atoi(argv[1]), sig_str);
    printf("%s\n", sig_str);
    return 0;
}
```

/opt/drill_ground/aupe/10-2.c [FORMAT=unix:utf-8]

```
tongue aupe # gcc 10-2.c -o 10-2
tongue aupe # ./10-2 10
SIGUSR1
tongue aupe # ./10-2 11
Does not recognize signo: 11
tongue aupe # ./10-2 12
SIGUSR2
```

10-3、分两种情况，一种是alarm超时，一种是在alarm超时前收到SIGINT信号；alarm超时的话，直接跳过pause的返回，到达setjmp处；alarm超时前SIGINT信号到达，则pause等待该信号处理程序的返回，并向下继续执行；如果在pause等待SIGINT信号处理程序返回期间，alarm超时，这种情况结果和alarm超时是一样的；



```c
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <setjmp.h>

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

unsigned int sleep2(unsigned int nsecs) {
    if (signal(SIGALRM, sig_alrm) == SIG_ERR) {
        return (nsecs);
    }
    // 申明跳入点:
    // longjmp携带参数1过来时, 判断将为false, 跳过里面逻辑:
    if (setjmp(env_alrm) == 0) {
        alarm(nsecs);
        // 看alarm和用户输入的SIGINT哪个先打断pause(),
        // 过早打断的话, alarm(0)返回剩余秒数, 不然, 返回0:
        pause();
    }
    return (alarm(0));
}
```

```c
static void sig_int(int signo) {
    int i,j;
    volatile int k;
    printf("\nsig_int starting\n");
    for (i=0; i<300000; i++) {
        for (j=0; j<4000; j++) {
            k += i*j;
        }
    }
    printf("sig_int finished\n");
}

int main(int argc, const char *argv[]) {
    unsigned int unslept;
    if (signal(SIGINT, sig_int) == SIG_ERR) {
        printf("signal(SIGINT) error!\n");
    }
    unslept = sleep2(5);
    printf("sleep2 returned: %u\n", unslept);
    return 0;
}
/opt/drill_ground/aupe/10-3.c [FORMAT=unix:utf-8]
```

pause函数使调用进程挂起直至捕捉到一个信号。

```c
#include <unistd.h>

int pause(void);
```

返回值：-1，并将errno设置为EINTR

只有执行了一个信号处理程序并从其返回时，pause才返回。在这种情况下，pause返回-1，并将errno设置为EINTR。

有关setjmp、longjmp可参考APUE的7-10节；

```
tongue aupe # gcc 10-3.c -o 10-3
tongue aupe # ./10-3
sleep2 returned: 0
tongue aupe # alarm超时结果打印
```

```
tongue aupe # ./10-3
^C终端发送SIGINT信号
sig_int starting
sig_int finished
sleep2 returned: 1
tongue aupe #
```

```
tongue aupe # ./10-3
^C先于SIGINT返回超时
sig_int starting
sleep2 returned: 0
```

10-4、代码如此精短，超时也设置的极长，非要找错，也许是在某种极端情况下了。

这里可能在假设一种极端情况，比如系统负载极高，刚好alarm(60)后，进程被调度器切到sleep状态，等60秒后才切回来，这时alarm就超时了，接着将执行超时处理程序，如果里面是一个longjmp的话，这时整个程序应该会段错误退出，因为setjmp还没准备好；

10-5、第一问题，并发怎么解决(比如两个计时器设置相同时间，不同的处理程序)?

若先不考虑上面这个问题，暂时想到的思路是：

1、建立超时及与超时处理程序所对应的alarm_function结构体；

2、在程序全局变量中开辟一个存放alarm_function结构体的数组alarm_function_array；

3、在一个while循环体中放入alarm(n)和遍历alarm_function_array中的每一个结构对象(n的大小为轮询的时间粒度)；

4、每遍历一次，alarm_function中的超时值减n，当等于或小于0时，执行对应的函数。

找到原文，做下参考吧http://www.kohala.com/start/libes.timers.txt

好吧，看了半天居然找到了翻译http://www.csdn123.com/html/blogs/20130501/8233.htm

貌似这个文章中的计时器，也有一些未解决的问题！比如s1与s2两个计时器时差为5秒，而s1触发的程序缺执行了8秒，s1的处理程序执行完毕后虽然可以马上执行s2的处理程序，但其已经比预定时间延长3秒了(若整个流程不计入其它时间损耗)。

关于第一个问题，我想，应该是在同一时间的，全部放在一个处理程序中。

10-6、

```c
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <fcntl.h>

static volatile sig_atomic_t sigflag;
static sigset_t newmask, oldmask, zeromask;

static void sig_usr(int signo) {
    sigflag = 1;
}

void TELL_WAIT() {
    if (SIG_ERR == signal(SIGUSR1, sig_usr)) {
        printf("Error by signal(SIGUSR1, sig_usr)"); _exit(-1);
    }
    if (SIG_ERR == signal(SIGUSR2, sig_usr)) {
        printf("Error by signal(SIGUSR2, sig_usr)"); _exit(-1);
    }
    sigemptyset(&zeromask);
    sigemptyset(&newmask);
    sigaddset(&newmask, SIGUSR1);
    sigaddset(&newmask, SIGUSR2);
    if (0 > sigprocmask(SIG_BLOCK, &newmask, &oldmask)) {
        printf("SIG_BLOCK error"); _exit(-1);
    }
}
```

```c
void TELL_PARENT(pid_t pid) {
    kill( pid, SIGUSR1);
}

void WAIT_PARENT() {
    while (0 == sigflag) {
        sigsuspend(&zeromask);
    }
    sigflag = 0;
    if (0 > sigprocmask(SIG_SETMASK, &oldmask, NULL)) {
        printf("SIG_SETMASK error"); _exit(-1);
    }
}

void TELL_CHILD(pid_t pid) {
    kill( pid, SIGUSR2);
}

void WAIT_CHILD() {
    while (0 == sigflag) {
        sigsuspend(&zeromask);
    }
    sigflag = 0;
    if (0 > sigprocmask(SIG_SETMASK, &oldmask, NULL)) {
        printf("SIG_SETMASK error"); _exit(-1);
    }
}
```

```c
int main(int argc, const char *argv[]) {
    pid_t pid;
    int fd, counter = 0, caps = 10;
    char path[BUFSIZ], flag = 'p';
    if (2 <= argc) {
        caps = atoi(argv[1]);
    }
    sprintf( path, "./pid_%d.counter", getpid());
    fd = open( path, O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
    lseek(fd, 0, SEEK_SET);
    write( fd, &counter, sizeof(counter));
    TELL_WAIT();
    if (0 > (pid = fork())) {
        printf("Error by fork!\n");
        return -1;
    } else if (0 == pid) {
        if ('c' == flag) {
            sigflag = 1;
        }
        while (caps > counter) {
            WAIT_PARENT();
            lseek(fd, 0, SEEK_SET);
            read( fd, &counter, sizeof(counter));
            printf("%c[7;33mChild  pid %d current counter: %d%c[0m\n", 27, getpid(), counter, 27);
            counter++;
            lseek(fd, 0, SEEK_SET);
            write( fd, &counter, sizeof(counter));
            TELL_PARENT(getppid());
        }
    } else {
        if ('p' == flag) {
            sigflag = 1;
        }
        while (caps > counter) {
            WAIT_CHILD();
            lseek(fd, 0, SEEK_SET);
            read( fd, &counter, sizeof(counter));
            printf("%c[7;32mParent pid %d current counter: %d%c[0m\n", 27, getpid(), counter, 27);
            counter++;
            lseek(fd, 0, SEEK_SET);
            write( fd, &counter, sizeof(counter));
            TELL_CHILD(pid);
        }
    }
    return 0;
}
```

/opt/drill_ground/aupe/10-6.c [FORMAT=unix:utf-8] [TYPE=C] [COL=001] [ROW=102/102(100%)]

```
tongue aupe # gcc -g 10-6.c -o 10-6
tongue aupe # ./10-6
Parent pid 589 current counter: 0
Child  pid 590 current counter: 1
Parent pid 589 current counter: 2
Child  pid 590 current counter: 3
Parent pid 589 current counter: 4
Child  pid 590 current counter: 5
Parent pid 589 current counter: 6
Child  pid 590 current counter: 7
Parent pid 589 current counter: 8
Child  pid 590 current counter: 9
Parent pid 589 current counter: 10
```

10-7、如果直接退出，系统不会产生core文件(进程映像)。SIGABRT处理方式设为默认，然后产生该信号，能使进程终止的更自然一些，让用户看起来和规定上描述的一样；

表10-1　UNIX系统信号

名　字	说　明	ISO C	SUS	FreeBSD 5.2.1	Linux 2.4.22	Mac OS X 10.3	Solaris 9	默认动作
SIGABRT	异常终止 (abort)	•	•	•	•	•	•	终止+core

如果*set*是空指针，则不改变该进程的信号屏蔽字，*how*的值也无意义。

```
/*
 * 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 to default */
```

若之前处理方式为忽略，则在这里置为默认处理方式(终止+core)

若为默认处理方式(终止+core)，则先冲洗I/O流

屏蔽除SIGABRT之外的所有信号

发送SIGABRT到该进程

如果能到达该处，表明用户捕捉了改信号，并且信号处理程序中没有调用_exit、_Exit之类函数

把SIGABRT的处理方式置为默认(终止+core)

发送SIGABRT到该进程，进程收到后，会终止，并产生core文件。这样做使用户看起来，调用该abort()函数和该进程直接收到SIGABRT信号没什么两样

```
sigprocmask(SIG_SETMASK, &mask, NULL);   /* just in case ... */
kill(getpid(), SIGABRT);下面的exit应该永不被执行 /* and one more time */
exit(1);     /* this should never be executed ... */
```

10-8、euid做文件系统权限判断用，ruid代表了进程属主；
10-9、如下；

```
tongue ~ # grep -r SIG_BLOCK /usr/include/bits/*
/usr/include/bits/sigaction.h:#define   SIG_BLOCK     0          /* Block signals.  */
```

也可将这三个函数在<signal.h>中实现为单行宏，但是POSIX.1要求检查信号编号参数的有效性，如果无效则设置errno。在宏中实现这一点比在函数中要困难。

sigaddset、sigdelset、sigismember

```
#include <stdio.h>
#include <sys/types.h>
#include <string.h>
#include <signal.h>
#include <errno.h>

#define SIG2STR_MAX 256

int sig2str(int signo, char *str) {
    char tmp_str[SIG2STR_MAX], *pTmp_str = tmp_str;
    bzero(tmp_str, sizeof(SIG2STR_MAX));
    switch (signo) {
        // Reference '/usr/include/bits/signum.h';
        case SIGHUP:
            pTmp_str = "SIGHUP";
        break;
        case SIGINT:
            pTmp_str = "SIGINT";
        break;
        case SIGQUIT:
            pTmp_str = "SIGQUIT";
        break;
        case SIGILL:
            pTmp_str = "SIGILL";
        break;
```

省略一部分……

```
        default:
            sprintf( tmp_str, "Does not recognize signo: %d", signo);
    }
    if (SIG2STR_MAX < strlen(tmp_str)) {
        sprintf( tmp_str, "You need more space in your str object!");
        return 1;
    }
    memcpy( str, pTmp_str, strlen(pTmp_str) + 1); // last is '\0';
    return 0;
}
```

```
int main(int argc, const char *argv[]) {
    sigset_t newsigset, oldsigset, cursigset;
    char sig_str[SIG2STR_MAX];
    int i, errno_save;
    sigemptyset(&newsigset);
    sigaddset(&newsigset, SIGUSR1);
    sigaddset(&newsigset, SIGUSR2);
    sigaddset(&newsigset, SIGKILL);  值得注意哦
    sigaddset(&newsigset, SIGHUP);
    sigprocmask(SIG_BLOCK, &newsigset, &oldsigset);
    sigprocmask(SIG_BLOCK, NULL, &cursigset);
    for (i=1; i<=31; i++) {
        bzero(sig_str, sizeof(SIG2STR_MAX));
        errno_save = errno;
        if (sigismember(&cursigset, i)) {
            sig2str( i, sig_str);
        }
        errno = errno_save;
        if (0 < strlen(sig_str)) {
            printf("%s\n", sig_str);
        }
    }
    return 0;
}
/opt/drill_ground/aupe/10-9.c [FORMAT=unix:utf-8] [T
```

```
tongue aupe # gcc 10-9.c -o 10-9
tongue aupe # ./10-9
SIGHUP
SIGUSR1        没有看到SIGKILL
SIGUSR2        表示其不能被屏蔽
```

10-10、执行下面程序的系统中，sleep使用的是nanosleep函数；

```c
#include <stdio.h>
#include <time.h>

char str_date[BUFSIZ];
struct tm *pTm;

void now_time() {
    time_t ts = time(NULL);
    pTm = localtime(&ts);
    strftime( str_date, BUFSIZ, "%F %X", pTm);
    printf("%s\n", str_date);
}

int main(int argc, const char *argv[])
{
    setvbuf(stdout, NULL, _IONBF, 0);
    int counter = 0, cycle = 1, interval = 1;
    printf("Usage: ./10-10 interval(5) cycle(2)\n");
    if (3 <= argc) {
        interval = atoi(argv[1]);
        cycle = atoi(argv[2]);
    }
    while(1) {
        if (0 == counter % cycle) {
            now_time();
        }
        counter++;
        sleep(interval);
    }
    return 0;
}
~
/opt/drill_ground/aupe/10-10.c [FORMAT=unix:utf-8] [
```

```
tongue aupe # jobs
[1]+  Running                 nohup ./10-10 60 5 > ./time.log &
tongue aupe # logout

Connection closed.
```

经过n个小时候……

```
tongue aupe # head -5 time.log ; tail -5 time.log
Usage: ./10-10 interval(5) cycle(2)
2014-03-07 17:38:31
2014-03-07 17:43:31
2014-03-07 17:48:31
2014-03-07 17:53:31
2014-03-08 02:23:39
2014-03-08 02:28:39
2014-03-08 02:33:39
2014-03-08 02:38:39
2014-03-08 02:43:39
```

```
tongue aupe # cat /opt/drill_ground/aupe/time.log | cut -d ':' -f 3|uniq -c
      1
      6 31
     14 32      14*5=70分钟
     14 33
     14 34
     13 35      13*5=65分钟
     14 36
     14 37
     13 38
      8 39
```

精度应该和系统负载有关；

该题的目的可能是问如下

的是，经过了指定的秒数后，信号由内核产生，由于进程调度的延迟，所以进程得到控制从而能够处理该信号还需一些时间。

早期的UNIX系统实现曾提出警告，这种信号可能比预定值提前1秒发送。POSIX.1则不允许这样做。

vixie-cron程序每分钟扫描一次是否有任务需要执行；它每分钟的间隔计算是通过如下方式来实现：

```
static void
set_time(int initialize) {
    struct tm tm;
    static int isdst;

    StartTime = time(NULL);    时间戳

    /* We adjust the time to GMT so we can catch DST changes. */
    tm = *localtime(&StartTime);
    if (initialize || tm.tm_isdst != isdst) {
        isdst = tm.tm_isdst;
        GMToff = get_gmtoff(&StartTime, &tm); 时区导致的偏移量
        Debug(DSCH, ("[%ld] GMToff=%ld\n",
            (long)getpid(), (long)GMToff))
    }
    clockTime = (StartTime + GMToff) / (time_t)SECONDS_PER_MINUTE;
}   把时间戳换算成分钟
~/vixie-cron-4.1/cron.c [FORMAT=unix:utf-8] [TYPE=C] [COL=001] [ROW=
```

```
static void
cron_sleep(int target) {     相关函数
    time_t t1, t2;
    int seconds_to_wait;

    t1 = time(NULL) + GMToff;
    seconds_to_wait = (int)(target * SECONDS_PER_MINUTE - t1) + 1;
    Debug(DSCH, ("[%ld] Target time=%ld, sec-to-wait=%d\n",
        (long)getpid(), (long)target*SECONDS_PER_MINUTE, seconds_to_wait))

    while (seconds_to_wait > 0 && seconds_to_wait < 65) {
        sleep((unsigned int) seconds_to_wait);     使用sleep实现
```

```
    set_time(TRUE);   时间初始化
    run_reboot_jobs(&database);
    timeRunning = virtualTime = clockTime;
```

```
    while (TRUE) {
        int timeDiff;
        enum timejump wakeupKind;

        /* ... wait for the time (in minutes) to change ... */
        do {
            cron_sleep(timeRunning + 1);休息到当前时间的下一分钟
            set_time(FALSE);  再次获取当前分钟的时间戳
        } while (clockTime == timeRunning); 如果没有休息到下一
        timeRunning = clockTime;            分钟则继续休息
        休息完后，重置timeRunning
        /*
         * Calculate how the current time differs from our virtual
         * clock.  Classify the change into one of 4 cases.
         */
        timeDiff = timeRunning - virtualTime;
        时间差，一般情况为1
        /* shortcut for the most common case */
        if (timeDiff == 1) {
            virtualTime = timeRunning; 查看是否有任务需运行
            find_jobs(virtualTime, &database, TRUE, TRUE);
        } else {
```

10-11、这题结果很奇怪；测试的结果是，第一次写满文件，不触发该信号，第二次再写时，会触发该信号处理程序；如果把rlim_cur和BUFSIZE的值设为相同时，看起来效果会达到书上描述的结果；

	总和。
RLIMIT_FSIZE	可以创建的文件的最大字节长度。当超过此软限制时，则向该进程发送SIGXFSZ信号。

```c
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <sys/resource.h>
#define BUFSIZE 10

static void sig_xfsz() {
    fprintf(stderr, "captured the signal SIGXFSZ\n");
}

void * signal_intr(int signo, void *func) {
    struct sigaction act, oact;
    act.sa_handler = func;
    sigemptyset(&act.sa_mask);
    act.sa_flags = 0;
#ifdef SA_INTERRUPT
    act.sa_flags |= SA_INTERRUPT;
    fprintf(stderr, "SA_INTERRUPT defined\n");
#endif
    if (0 > sigaction(signo, &act, &oact)) {
        return(SIG_ERR);
    }
    return(oact.sa_handler);
}
```

```c
int main(int argc, const char *argv[]) {
    int rn, wn;
    char buf[BUFSIZE];
    struct rlimit rl;

    if (SIG_ERR == signal_intr(SIGXFSZ, sig_xfsz)) {
        fprintf(stderr, "Error by signal(SIGXFSZ, sig_usr)"); _exit(-1);
    }
    getrlimit(RLIMIT_FSIZE, &rl);
    fprintf(stderr, "rlimit_cur: %10ld\n", rl.rlim_cur);
    fprintf(stderr, "rlimit_max: %10ld\n", rl.rlim_max);
    rl.rlim_cur = BUFSIZE + 5;
    setrlimit(RLIMIT_FSIZE, &rl);
    getrlimit(RLIMIT_FSIZE, &rl);
    fprintf(stderr, "rlimit_cur: %10ld\n", rl.rlim_cur);
    fprintf(stderr, "rlimit_max: %10ld\n", rl.rlim_max);
    while (0 < (rn = read(STDIN_FILENO, buf, BUFSIZE))) {
        wn = write(STDOUT_FILENO, buf, rn);
        if (rn != wn) {
            fprintf(stderr, "write error\n"); // return (wn);
        }
    }
    if (0 > rn) {
        fprintf(stderr, "read error\n"); _exit(rn);
    }
    return 0;
}
/opt/drill_ground/aupe/10-11.c [FORMAT=unix:utf-8] [TYPE=C] [COL=001] [RO
```

```
tongue aupe # gcc -g 10-11.c -o 10-11
tongue aupe # ./10-11 > a.file
SA_INTERRUPT defined
rlimit_cur:         -1
rlimit_max:         -1
rlimit_cur:         15
rlimit_max:         -1
01234567890012345678 90
write error
captured the signal SIGXFSZ
write error
^C
```

10-12、由下面结果可以得知，SIGALRM信号处理程序要在fwrite(write也是，fwrite最终调用的还是write)数据写完之后(并不返回)才被调用；当不捕获SIGALRM时，程序会立刻退出，写操作也不会等待完成；

猜测，write可能捕获并先处理了该信号，它处理完后再调用用户对该信号的处理程序；

```c
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include <sys/resource.h>
#define BFSZ 1024 * 1024 * 1200

char tmp_buf[BFSZ];
char io_buf[BFSZ / 3];
static int n, fn;
static void sig_alarm(int signo) {
    printf("captured the signal SIGALRM at %d\n", time(NULL));
//    _exit(-1);
}
```

```c
int main(int argc, const char *argv[]) {
    FILE *fp = NULL;
    char last_str[] = "File End!";
    struct sigaction act;
    act.sa_handler = sig_alarm;
    sigemptyset(&act.sa_mask);
    act.sa_flags = 0;
    act.sa_flags |= SA_INTERRUPT;
    sigaction(SIGALRM, &act, NULL);
    fp = fopen("/dev/urandom", "r");
//      fp = fopen("./tmp2.file", "r");
    printf("Begin read\n");
    n = fread(&tmp_buf, sizeof(char), BFSZ, fp);
    memcpy( &tmp_buf[BFSZ - sizeof(last_str)], last_str, sizeof(last_str));
    printf("Finished, Read %d bytes\n", n);
    fp = fopen("./tmp.file", "w+");
    setvbuf(fp, io_buf, _IOFBF, BFSZ / 3);
    alarm(1);
    printf("Begin write, set alarm at %d\n", time(NULL));
    n = fwrite(&tmp_buf, sizeof(char), BFSZ, fp);
//    n = write(fp->_fileno, &tmp_buf, BFSZ);
    printf("Finnished, write %d bytes\n", n);
    return 0;
}
/opt/drill_ground/aupe/10-12.c [FORMAT=unix:utf-8] [TYPE=C] [COL=001] [ROW=0
```

代码中io_buf除以3是为了节省系统同内存。系统总共2G内存，tmp_buf占用了1.2G，io_buf和他一样大时，显然不够，程序会出错；

```
tongue aupe # ./10-12
Begin read
Finished, Read 1258291200 bytes
Begin write, set alarm at 1394314129
captured the signal SIGALRM at 1394314132
Finnished, write 1258291200 bytes
```

```
tongue aupe # tail -c32 tmp.file | hexdump -C
00000000  29 12 6e 43 40 12 15 9c  b9 e3 11 66 9e fb 5f dd  |).nC@......f..._.|
00000010  fe be c8 d9 21 4b 46 69  6c 65 20 45 6e 64 21 00  |....!KFile End!.|
00000020
```

做了内容结束标记，不然全是随机值，确认起来不太直观。