进程信号

信号的概念

信号是一个软件中断，打断当前的正在运行的进程，让该进程去处理信号的事件。

信号的种类

使用**kill -l**命令查看信号

```
[run@localhost ~]$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Linux操作系统中，总共有62种信号

- 前1 - 31，不可靠信号，信号有可能会丢失，非实时信号
- 后34 - 64，可靠信号，信号不可能被丢失的，实时信号，它们没有固定的含义（可以由用户自定义）。所有的实时信号的默认动作都为终止进程。

信号的生命周期

信号的产生

- 终端产生
  - Ctrl + c ： 给前台发送一个SIGINT信号，中断当前的前台进程
  - Ctrl + z：给前台发送一个SIGTSTP信号，使当前进程暂停
  - Ctrl + \：给前台发送一个SIGQUIT信号，退出进程同时产生一个core文件
- 命令产生
  - kill -[信号标识] [进程ID]：给进程发送某个信号
- 函数产生
  - kill函数：给指定进程发送指定信号
    - ```
      #include <sys/types.h>
      #include <signal.h>
      int kill(pid_t pid, int sig);
      ```
    - pid>0：发送信号给指定的进程
    - pid=0：将信号传给和目前进程相同组的所有进程
    - pid=-1：将信号广播传送给系统内所有的进程
    - pid<0：取pid的绝对值发送给对应的进程组
      ```

        - 返回值：执行成功返回0，如果有错误则返回-1
    - abort函数：
        - ```
          #include <stdlib.h>
          void abort(void);
          ```
        - 给自己发送异常终止信号SIGABRT，终止并产生core文件
    - raise函数：
        - ```
          #include <signal.h>
          int raise(int sig);
          ```
        - 给当前进程发送指定信号

信号的注册

前提：

1、在内核当中的task_struct结构体当中，保存一个struct sigpenging 的对象pending(未决信号集合，就是信号产生了还没有决定怎么做)，struct sigpenging这个结构体当中保存了两个元素：

```
struct sigpending{
    struct sigqueue *head,*tail;
    sigset_t signal;
}
```

sigset_t类型也是一个结构体，struct sigset_t保存了一个unsigned long _val[xxx] ——>sig这个数组是按照比特位来使用的，我们称为位图。

2、内核当中还维护了一个sigqueue队列，队列当中的每一个元素对应信号的一个处理节点。

信号的注册分两种：

- 非可靠信号：
    - 当程序收到一个信号时，操作系统会给当前程序维护的sig位图当中对应的比特位为1，并且在sigqueue队列当中增加对应信号的节点；当多次收到同样的一个信号时(判断多次收到同样信号的标准就是判断sig位图中的比特位)，只添加一次节点，也就是意味着第二次(多次)收到的同样的信号就被丢弃了。
- 可靠信号：
    - 当程序收到一个信号时，操作系统会给当前程序维护的sig位图当中对应的比特位为1，并且在sigqueue队列当中增加对应信号的节点；当多次收到同样的一个信号时，多次添加对应信号的节点到sigqueue队列当中去，也就是意味着，每一个信号操作系统在处理信号的时候都会从sigqueue队列拿出对应的节点进行处理，也就是信号没有丢失。

信号的注销

在信号处理之前，会先销毁信号的信息；信号注销存在的目的就是为了抹除信号存在的痕迹，防止对同一个信号进行多次处理。

- 非可靠信号

  将sig位图当中对应信号的比特位从1置为0，并且将sifqueue队列当中对应的节点去除掉
- 可靠信号

从sigqueue队列当中获取对应信号的节点，并且**判断sigqueue队列当中是否还有对应相同信号的节点**；如果有，去除sigqueue队列当中刚才拿出来的对应信号的节点，不会将sig位图中的比特位置为0；如果没有，去除sigqueue队列当中对应信号的节点，并且将sig位图置为0

信号的处理方式

执行默认动作

- SIG_DEL——>执行一个操作系统定义好的动作（操作系统执行了一个函数）

忽略(丢弃)

- SIG_IGN——>操作系统不会干任何事情
- 僵尸进程：子进程退出的时候，给父进程发送了一个SIGCHID信号，但是操作系统对SIGCHID信号的处理方式为忽略处理，而导致父进程不去处理信号，从而子进程变成僵尸进程

捕获处理

捕获处理：程序员自己定义信号的处理函数

- signal(设置信号处理方式)
  - ```
    #inlclude <signal.h>
    sighandler_t signal(int signum,sighandler_t handler);
    ```
  - signum：需要更改自定义处理函数的信号
  - handler：接收一个函数的地址，将信号的处理函数更改为什么函数
- ```
  typedef void (*sighandler)(int);
  ```
  - int：参数，指的是哪一个信号触发操作系统调用该函数
- 我们注册的函数，被称为回调函数，并不是我们调用signal函数的时候，回调函数就会被操作系统调用执行，而是当我们收到自定义信号的时候，操作系统才会帮我们调用该函数，进行处理信号。
- 回调函数的执行是操作系统的执行流调用执行的。

信号的阻塞

信号的阻塞就是阻止一个信号的抵达，当一种信号被阻塞时，它仍然可以被发送，但是产生的待处理信号不会被接受，直到进程取消对这种信号的阻塞。

在PCB中有一个阻塞信号集合(block位图，实现方式与pending相同)，凡是添加到这个集合中的信号，都表示需要阻塞，暂时不需要处理。

sigprocmask()函数显式地阻塞和取消阻塞选择的信号

```
#include <signal.h>
int sigprocmask(int how ,const sigset_t *set,sigset_t *oldset);
```

- how：
  - SIG_BLOCK：设置某个信号为阻塞状态，用修改位图到达目的 block(new)=block(old)|set
  - SIG_UNBLOCK：设置某个信号为非阻塞状态，block(new)=block(new)|(~set)
  - SIG_SETMASK：block(new)=set
- set：要设置的新的阻塞位图

- oldset：之前程序当中阻塞的位图

在所有的信号中，有两个信号9) SIGKILL和19)SIGSTOP，不可被阻塞，不可被自定义修改处理方式，也不可被忽略。