# Linux 信号机制

## 一、基础知识

### 1、信号的概念

1）本质：是在软件层次上对中断的一种模拟，也称为“软中断”。信号用来通知进程发生了异步事件，当一个进程收到一个信号时会停止当前任务而转去处理该信号。信号是由内核发送并处理的。
2）特征：简单（开销小）、不能携带大量信息。
3）分类：Linux 一共有 62 个信号（可以通过 kill –l 命令查看），前 31 个是不可靠信号，也称为非实时信号，它们不支持排队，可能会丢失（若该信号被屏蔽期间产生多次该信号则只记录一次），并且都有自己固定的含义和默认处理动作；后 31 个是可靠信号，也称为实时信号，支持排队，不会丢失，没有固定含义并且默认动作都为终止进程（本文中只关注前 31 个信号）。
4）信号四要素：名称、编号、默认处理动作、信号产生事件（可以通过 man 7 signal 查看）。

### 2、信号的产生方式

1）按键产生，如：Ctrl + c、Ctrl + z、Ctrl + \
2）系统调用产生，如：kill、raise、abort
3）软件条件产生，如：定时器（alarm）
4）硬件异常产生，如：非法访问内存（段错误）、除 0（浮点数例外）、内存对齐出错（总线错误）
5）命令产生，如：kill 命令

### 3、信号的处理机制

1）在进程 PCB 中维护着两个集合：阻塞信号集（信号屏蔽字）和未决信号集，阻塞信号集中存放被该进程屏蔽的信号，未决信号集中存放着已经产生但是还未被该进程处理的信号。
2）处理流程：当一个信号由于某种原因产生后，内核将该信号传递给目标进程，此时，目标进程首先将该信号加入未决信号集，然后判断该信号是否在阻塞信号集中，如果不在，则立即处理该信号，然后将该信号从未决信号集中移除，如果存在，则不作处理，直到将该信号从阻塞信号集中移除后才会被处理并将该信号移出未决信号集。
3）信号的处理方式：
（1）执行默认动作：
①Term：终止进程
②Ign：忽略信号，什么都不做
③Core：终止进程并生成 Core 文件（查验进程死亡原因，用于 gdb 调试）

④Stop：暂停进程

⑤Cont：如果进程暂停的话继续运行

（2）忽略（丢弃该信号）

（3）捕捉（自定义用户处理函数）

**（注：9 号信号 SIGKILL 和 19 号信号 SIGSTOP 不能被捕捉、屏蔽或者忽略，只能执行默认动作）**

# 二、信号相关操作

## 1、kill 命令和函数

功能：给指定进程发送指定信号。

kill 命令：kill –信号编号 pid

kill 函数： int kill(pid_t pid, int sig);

pid > 0：给指定进程发送信号

pid = 0：给调用 kill 命令/函数的进程的同组所有进程发送信号

pid < -1：给进程组 id 为|pid|的进程组中的所有进程发送信号

pid = -1：给调用 kill 命令/函数的进程有权限发送信号的所有进程发送信号

扩展：

raise 函数（int raise(int sig)）：给自己发送信号；

abort 函数（void abort(void)）：给自己发送异常终止信号（SIGABRT），进程终止并产生 Core 文件。

## 2、定时器函数

1）alarm 函数（unsigned int alarm(unsigned int seconds)）：

功能：设置定时器，在指定时间后，内核会给当前进程发送 SIGALRM 信号，该信号的默认动作是终止进程。

返回值：0 或上次定时剩余秒数，不会失败。

**注：每个进程有且只有一个定时器；用 alarm(0) 来取消定时器；采用自然定时法，无论进程处于什么状态（阻塞、挂起、就绪……）都会计时。**

扩展：可以在可执行程序前加上 time 命令（如：time ./a.out），用来统计程序执行时间，实际执行时间 = 在内核中消耗时间 + 在用户空间中消耗时间 + 等待时间（等待时间片、等待 IO 等）。

2）setitimer 函数：

功能：设置定时器，精度可达到微妙，并且可以实现周期定时。

函数原型：int setitimer(int which, const struct itimerval *new_value, struct itimerval *old_value);

```
struct itimerval {
        struct timeval it_interval;    /* Interval for periodic timer */
        struct timeval it_value;       /* Time until next expiration */
};
```

```
struct timeval {
        time_t      tv_sec;     /* seconds */
        suseconds_t tv_usec;    /* microseconds */
};
```

（1）which：指定定时方式，①ITIMER_REAL —— 自然计时，计算自然时间，发送 SIGALRM 信号；②ITIMER_VIRTUAL —— 虚拟空间计时（用户空间），只计算进程占用 cpu 时间，发送 SIGVTALRM 信号；③ITIMER_PROF —— 运行时计时（用户空间 + 内核空间），计算进程占用 cpu 时间以及系统调用的时间，发送 SIGPROF 信号。

（2）new_value 传入定时时间间隔；old_value 传出上次定时时间间隔，不关心可设为 NULL。其中 it_value 指定定时器第一次定时的时间间隔，it_interval 指定定时器第一次定时后周期性定时的时间间隔，如果只指定 it_value 则只完成一次定时，两个都为 0 时取消定时器。

# 3、信号集操作函数

1）内核通过读取未决信号集判断是否有信号未被处理，通过读取阻塞信号集判断信号是否被屏蔽。用户可以自定义阻塞信号集，以达到屏蔽指定信号的目的。

2）信号集处理函数：

```
sigset_t set;   //信号集类型，typedef unsigned long sigset_t
int sigemptyset(sigset_t *set);   //将某个信号集清 0
int sigfillset(sigset_t *set);        //将某个信号集置 1
int sigaddset(sigset_t *set, int signum);   //将某个信号加入信号集
int sigdelset(sigset_t *set, int signum);   //将某个信号清出信号集
int sigismember(const sigset_t *set, int signum);   //判断某个信号是否在信号集中
```

3）sigprocmask 函数：

功能：用来屏蔽信号和解除屏蔽，会读取和修改进程的阻塞信号集。

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

set：传入参数，自定义的信号集

oldset：传出参数，保存旧的阻塞信号集

how：指定如何设置进程的阻塞信号集，设阻塞信号集为 mask

①SIG_BLOCK —— set 表示需要屏蔽的信号集，相当于 mask = mask | set

②SIG_UNBLOCK —— set 表示需要解除屏蔽的信号集，相当于 mask = mask & ~set

③SIG_SETMASK —— set 来代替原有信号集，相当于 mask = set

**注：9 号信号 SIGKILL 和 19 号信号 SIGSTOP 设置为屏蔽不起作用，依然会执行默认动作**

4）sigpending 函数（int sigpending(sigset_t *set)）：

功能：读取当前进程的未决信号集。

# 4、注册信号捕捉函数

用户可以自定义信号处理函数，然后在内核中注册，当有信号到来时，内核会自动调用该处理函数。

1）signal 函数：

```
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);
```

功能：将对 signum 信号自定义的处理函数 handler 注册给内核。
返回值：成功返回先前的处理函数，失败返回 SIG_ERR。
2）sigaction 函数：

```
int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
struct sigaction {
        void        (*sa_handler)(int);
        void        (*sa_sigaction)(int, siginfo_t *, void *);
        sigset_t    sa_mask;
        int         sa_flags;
        void        (*sa_restorer)(void); //这个成员已经被废弃
};
```

sa_handler —— 指定信号处理函数，也可设置为 SIG_DFL（执行默认动作）和 SIG_IGN（忽略）。
sa_mask —— 指定信号处理函数执行期间的阻塞信号集（临时的）。
sa_flags —— 常用设置：①0，表示默认属性（在信号处理函数执行期间自动屏蔽本信号）；②SA_NODEFER，不自动屏蔽本信号。
sa_sigaction —— 当 sa_flags 为 SA_SIGINFO 时，使用该处理函数（很少使用）。

# 5、进程挂起等待信号函数

1）pause 函数：
功能：将进程挂起，等待信号唤醒。

```
int pause(void);
```

返回值：
①如果进程收到的信号的处理动作是结束进程，则 pause 没有机会返回；
②如果进程收到的信号的处理动作是忽略或者该信号被屏蔽，则进程继续挂起；
③如果进程收到的信号的处理动作是捕捉，则信号处理函数结束后，pause 函数返回-1，并将 errno 设置为 EINTR。
2）sigsuspend 函数：
功能：将进程挂起，等待信号唤醒。

```
int sigsuspend(const sigset_t *mask);
```

在进程挂起等待期间临时设置 mask 为阻塞信号集。

# 6、信号传参

1）sigqueue 函数：
功能：给指定进程发送信号和参数

```
int sigqueue(pid_t pid, int sig, const union sigval value);
union sigval {
    int     sival_int;
    void *sival_ptr; //虚拟地址
};
```

注：不同进程的虚拟地址空间各自独立，所以给另一进程发送地址无实际意义。

2）sigaction 函数：

当 sigaction 函数的 sa_flags 为 SA_SIGINFO 时，使用 sa_sigaction 作为信号处理函数。

void (*sa_sigaction)(int, siginfo_t *, void *);

siginfo_t 是一个包含很多信号信息的结构体类型。

系统调用时，内核会将参数传入信号处理函数的第三个参数（void *）中。