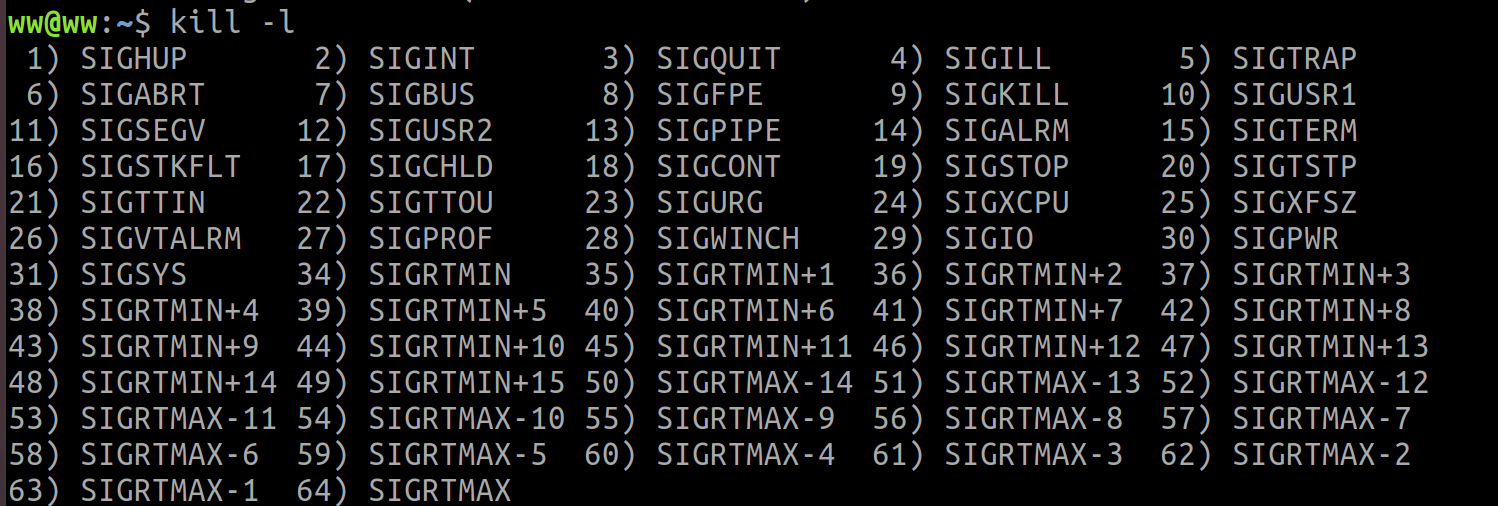
# 信号

## 信号的概念

kill -l 命令查看系统支持的信号类型和信号编号

标准信号： 1) SIGHUP～ 31) SIGSYS

响应无顺序 会丢失

实时信号： 34) SIGRTMIN ～ 64) SIGRTMAX

响应有顺序 不会丢失

信号是软件中断。信号的响应依靠硬件中断（调度）来实现。

note: SIGUSR1 和 SIGUSR2 没有绑定默认的信号处理函数，由用户来实现

## signal（）

//将信号类型和信号处理函数绑定

typedef void (\*sighandler\_t)(int);

sighandler\_t signal(int signum, sighandler\_t handler);

int signum：信号类型

sighandler\_t handler：要执行的动作(SIG\_IGN 忽略该信号)

信号会打断阻塞的系统调用（ag: sleep open）

1. 信号的不可靠

标准信号会丢失并不是信号不可靠的原因而是信号的行为不可靠（执行现场由内核决定信号处理函数的执行现场不是程序员布置的，而是内核布置的，因为程序中不会有调用信号处理函数的地方。同一个信号处理函数的执行现场会被布置在同一个地方，所以当一次信号处理函数未执行完成时再次触发了相同的信号，信号处理函数发生了第二次调用，则第一次调用的执行现场会被覆盖）

标准信号的响应没有严格的顺序

1. 可重入函数

所有的系统调用和一部分库函数都是可重入的

man手册中后缀为“\_r”的函数表示可重入版本。

1. 信号的响应过程

内核为每个进程维护了两个（32位）位图，MASK和pending，MASK是屏蔽信号位图（响应中断置1，不响应置0），pending是信号是否发生位图（发生了中断为1否则为0）。在每个时间片用完，发生任务调度时，内核会根据这两个位图判断要运行进程是否收到了信号。如果收到了信号，首先更改这两个位图（MASK置0），屏蔽同类型的信号，然后更改PC指针到信号响应函数开始运行。

使用信号和alarm函数完成时间控制的程序，时间误差最小可以控制在10ms。

MASK置0？？

不能随便的从信号处理函数往外跳（setjmp longjmp），因为处理函数完成后要扎内核把MASK位改回去，跳出去MASK就改不回去了。

可以使用：sigsetjmp，siglongjmp 完成有保存和恢复MASK状态的跳转。

标准信号的丢失是因为，在一个信号未得到响应前，无论重复发多少次相同的信号，都只能响应一次，因为记录信号发生的位图只有1位，多余的信号只是在重复对位图进行设置，发生一次penging置1，发生100次penging还是1。（实时信号不会丢失）

信号的响应由一定的延时，因为需要等待调度到再次运行时，才能判断是否受到信号进行响应。

1. 常用函数

kill:

//发送信号给进程 检测某个进程是否存在

int kill(pid\_t pid, int sig);

raise:

//给自己（进程或线程）发信号

int raise(int sig);

alarm:

//设置seconds秒后的定时器让内核给自己发信号 这个信号的默认功能会杀死进程， //可采用signal()函数修改信号的功能。

unsigned int alarm(unsigned int seconds);

//当多个alarm存在时只有最后一个alarm生效。也就是alarm只能设置一个计时值。

//单次单值计时

pause:

//堵塞的等待任何一个信号

int pause(void);

setitimer:

//微妙级别定时（单次/周期） 内核发送信号给自己

int setitimer(int which, const struct itimerval \*new\_value,

struct itimerval \*old\_value);

which 时钟类型 （类型不同触发的信号不同）：

ITIMER\_REAL（全部时间）

ITIMER\_VIRTUAL（用户态时间）

ITIMER\_PROF（用户态时间+内核态时间）

new\_value ：计时周期

settimer工作机制是，先对it\_value倒计时，当it\_value为零时触发信号，然后重 置为it\_interval，继续对it\_value倒计时，一直这样循环下去。就是只设置it\_value 为单次计时，it\_interval也设置为循环计时。

old\_value： 参数，通常用不上，设置为NULL，它是用来存储上一次setitimer调用时设置的new\_value值

int getitimer(int which, struct itimerval \*curr\_value);

abort:

//给自己发送SIGABRT使进程异常终止，并产生coredump文件。

void abort(void);

system:

调用shell执行一个可执行文文件。

相当于：execl("/bin/sh", "sh", "-c", command, (char \*) 0);

During execution of the command, SIGCHLD will be blocked, and SIGINT and SIGQUIT will be ignored, in the process that calls system() (these signals will be handled according to their defaults inside the child process that executes command).

sleep：

sleep可能由alarm+pause封装而来mixing calls to alarm and sleep() is a bad idea.(一个进程只能使用一个alarm 所以使用sleep不安全)

可以用：

nanosleep()

usleep()

select()

1. 信号集 一个整形数 每一位代表一种信号。

信号集类型：sigset\_t

sigemptyset, //信号集清空

sigfillset, //与清空相反

sigaddset, //信号集添加一个信号

sigdelset, //信号集删除一个信号

sigismember //检测某个集合是否存在一个信号

1. 信号屏蔽字/pending集 的处理

sigprocmask

//对某个信号的响应作修改（我们不能决定信号的到来，但可以修改响应方式）

int sigprocmask(int how, const sigset\_t \*set, sigset\_t \*oldset);

how 对信号集信号采取的操作 ：

SIG\_BLOCK (锁)

SIG\_UNBLOCK(解锁)

SIG\_SETMASK (锁 保存 解锁 恢复一个oldset)

set 信号集

oldset 执行操作之前的状态

sigpending

//获取pending的状态

int sigpending(sigset\_t \*set);

1. sigsuspend sigaction setitimer

sigsuspend ？？

sigaction

//响应一个信号时 同时锁住其他需要锁住的信号,因为一个信号处理函数执行时可能被//另一个信号打断

int sigaction(int signum, const struct sigaction \*act,

struct sigaction \*oldact);

signum 信号类型

struct sigaction \*act 信号回调函数

struct sigaction {

void (\*sa\_handler)(int); //回调函数

void (\*sa\_sigaction)(int, siginfo\_t \*, void \*);

sigset\_t sa\_mask; //要block的信号集

int sa\_flags; //特殊要求

void (\*sa\_restorer)(void); //不用关心

};

//回调函数

void (\*sa\_handler)(int); //int 是信号类型 多个信号共用一个回调函数通过int来判断是那个信号触发的

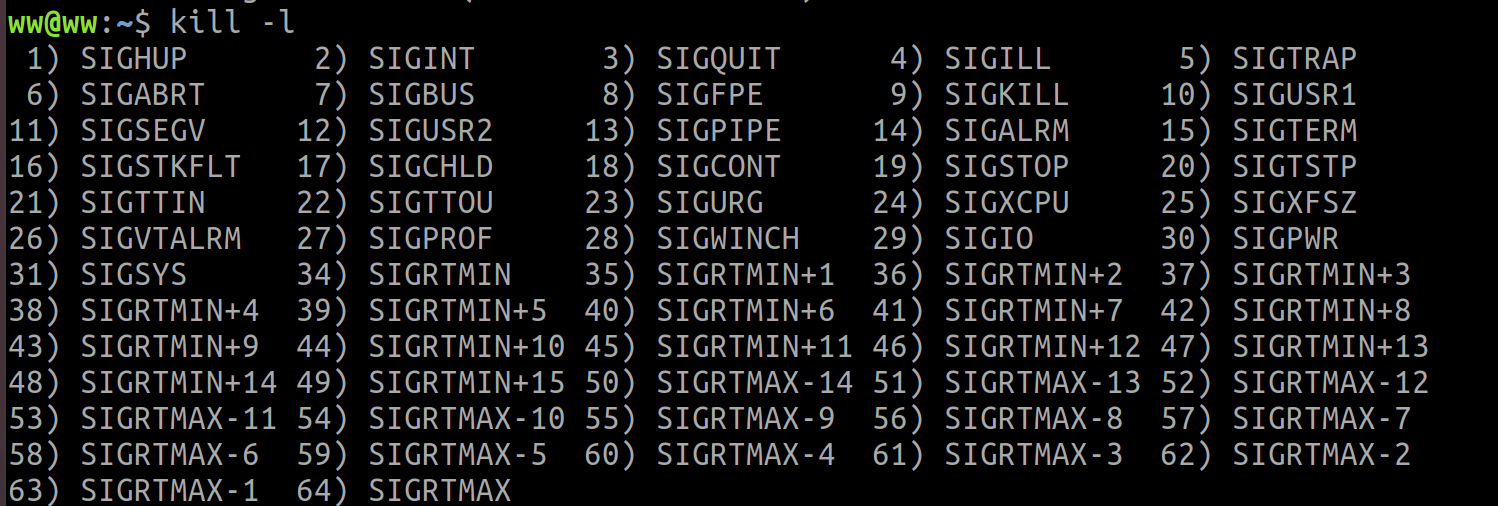
void (\*sa\_sigaction)(int, siginfo\_t \*, void \*);

int, 信号类型

siginfo\_t \*, 信号的信息（其中的si\_code成员比较重要）

void \* 弃用(其实是跳转过来前的现场)

1. 实时信号 响应有顺序 不会丢失



编号 SIGRTMIN ~ SIGRTMAX 为实时信号

信号响应是否有顺序与只与使用的信号类型有关，与使用的函数无关。

ulimit -a 中的pending signals(-i) 代表了实时信号可顺序响应的最大数量。

# 多线程