信号

基本概念

信号是事件发生时对进程的一种通知机制，有时候又被称之为软中断，当进程运行时，我们无法预料到什么时候会收到何种类型的信号。

在绝大部分情况下，信号都是源于内核

① 硬件发生异常

硬件（如内存、CPU）检测到了一个错误并通知到内核，而后内核再发送相应的信号给对应的进程

比如最常见的除 0 操作（CPU），引用了无法访问的内存区域（内存），后者我们可能会经常看到，信号类型为 SIGSEGV，即段错误

```
int main {
    int *ptr = NULL;
    *ptr = 1024;
}
```

Process finished with exit code 139 (interrupted by signal 11: SIGSEGV)

以前的内存是分段的，比如数据段、代码段等，当进程访问错误内存地址时，就会抛出段错误的信息

② 外部设备中断

用户通过键盘或者其他设备键入了能够产生信号的特殊字符，例如最为常用的 Ctrl-C，中断当前进程的运行

③ 软件事件发生

比如进程设置的定时器到期，进程的某个子进程退出等等，都会有信号的产生和发生

定时器到期会发送一个 SIGALRM 的信号，也就是 Singal Alarm；
进程的某个子进程退出会发送一个 SIGCHLD 信号，也就是 Singal Child

从信号的名字上我们能大致的猜到这个信号是干啥的

处理信号的方式

忽略 — 这很好理解， 当一个信号来的时候进程啥都不干

编写信号处理程序 — 常见方式，通常我们会使用一些简短的函数作为信号处理函数，来处理不同的信号

阻塞 — 当某一个信号达到时，我们可以阻塞该信号向进程的传递，或者说屏蔽某些信号，当信号被解除屏蔽后才会发送至进程

以最常用的 Ctrl-C 为例

当用户对当前进程键入 Ctrl-C 以后，内核会将其转换成 SIGINT 信号，也就是 Singal Interrupt，发送对对应的进程

当该进程收到 SIGINT 信号以后，如果该进程有针对于该信号的信号处理程序，那么执行信号处理程序。否则，终止进程的执行

常用信号及其默认行为

定时器 — SIGALRM — 定时器到期后由内核发出，在使用了 timer 的进程中非常重要

终止进程相关

SIGINT — Singal Interrupt，中断进程的执行。如果我们想写一个 Ctrl-C 结束不掉的进程，编写一个 SIGINT 的信号处理程序即可

SIGKILL — 命令行中的 kill -9 就是发出的这个信号，-9 表示信号编码

该信号为"必杀"信号，进程无法阻塞或者捕获该信号，所以总能够杀死进程，一击必杀

SIGTERM — 这个信号是用来终止进程的标准信号，kill 命令所使用的默认信号

暂停/恢复

SIGSTOP — 这是一个必停信号，进程无法阻塞、捕获或者是忽略该信号，所以总是能够停止程序的运行，有点儿像打断点一样

SIGCONT — 使停止的进程继续执行，也就是恢复进程的调度属性

段错误 — SIGSEGV — 段错误，C++ 中数组访问越界，指针访问不存在的内存区域等等，内核都会发送该信号给进程

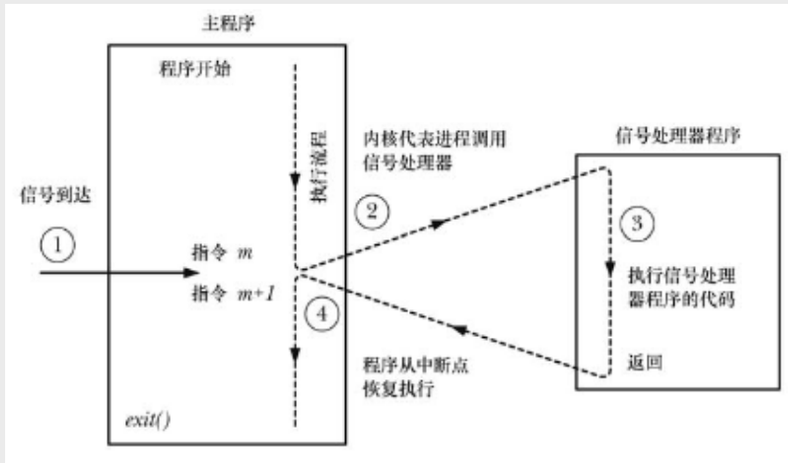
终端 — SIGHUP — 当终端断开时，将发送该信号给终端控制进程

父/子进程间通信 — SIGCHLD — 当父进程的某一个子进程终止时，内核就会向父进程发送该信号

信号处理函数

通常来说，我们会为一些信号添加信号处理函数，由进程自行处理这些送达的信号。信号处理程序应该尽可能地简短，并且函数内必须使用可重入的函数或者系统调用

处理流程



也就是说，内核调用信号处理函数可能会发生在任意时刻，并且完全有可能打断系统调用的执行

demo级函数: signal

定义在 signal.h 中的 signal 函数通常作为 demo 使用，它能够为某一个信号注册信号处理函数

示例

```
void handler(int signum) {
    printf("Got a SIGINT");      // 只是简单的打印
}

int main() {
    signal(SIGINT, handler);    // 注册信号处理函数 handler

    while (1) {
        printf("休息 1s \n");
        sleep(1);
    }
}
```

此时将无法使用 Ctrl-C 将该进程终止，因为我们改变了 SIGINT 该信号的接收动作

问题

如果当前的 SIGINT 处理函数还在执行，此时又来了一个或多个 SIGINT 信号会发生什么?

POSIX 标准将保证当前同一个信号将会被阻塞，也就是说，不会中断信号处理函数，而是等在哪儿。如果此时有多个相同信号到达，那么多个信号将会并合并成一个向进程发送

什么是可重入函数？信号处理函数为什么需要是可重入的?

可重入函数可以认为是线程安全的函数，也就是即使多个线程乱序调用某一个函数，依然能够得到预期的结果

使用不可重入的函数可能会导致进程执行混乱，甚至是陷入到休眠状态，失去进程的控制

诸如 printf()、malloc() 等函数都是不可重入的