学习目标

* 了解信号中的基本概念
* 熟练使用信号相关的函数
* 参考文档使用信号集操作相关函数
* 熟练使用信号捕捉函数signal
* 熟练使用信号捕捉函数sigaction
* 熟练掌握使用信号完成子进程的回收

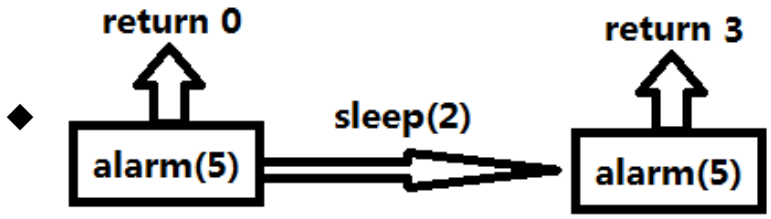
1 - 信号初步认识

1. 特点:
    - 简单
    - 携带的信息量少
    - 使用在某个特定的场景中
2. 信号的状态
    - 产生
    - 未决状态 - 没有被处理
    - 递达 - 信号被处理了
3. 处理方式
4. 信号的四要素
5. 通过man文档查看信号
    a. man 7 siganl
    b. The signals SIGKILL and SIGSTOP cannot be caught, blocked, or ignored.
6. 概念: 阻塞信号集, 未决信号集
    - pcb
    - 不能直接操作
    - 阻塞信号集:
        - 要屏蔽的信号
    - 未决信号集:
        - 没有被处理的信号的集合

2 - 信号相关函数

1. kill -- 发送信号给指定进程
   - 函数原型: int kill(pid_t pid, int sig);
2. raise -- 自己给自己发信号

   a. kill(getpid(), sigkill);
   - 函数原型: int raise(int sig);
     - 返回值:
3. abort -- 给自己发送异常终止信号
   - 函数原型: void abort(void);
     - 没有参数没有返回值, 永远不会调用失败
4. 闹钟(定时器)
   - alarm -- 设置定时器(每个进程只有一个定时器)
     - 使用的是自然定时法
       - 不受进程状态的影响
     - 函数原型: unsigned int alarm(unsigned int seconds);
       - 参数: 秒
       - 当时间到达之后, 函数发出一个信号:SIGALRM
       - 返回值:



     - 练习: 测试你的电脑1s能数多少个数字?
       - 测试程序运行的时间: time 可执行程序

         real = 用户+内核+损耗
         - 损耗来自文件IO操作
   - setitimer -- 定时器, 并实现周期性定时
     - 函数原型:

       int setitimer(int which,
       const struct itimerval *new_value,
       struct itimerval *old_value // NULL

```
);

struct itimerval {
   struct timeval it_interval;  // 定时器循环周期
    struct timeval it_value;    // 第一次触发定时器的时间
};

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

3 - 信号集

信号集操作相关函数

1. 概念：
   - 未决信号集：
     - 没有被当前进程处理的信号
   - 阻塞信号集：
     - 将某个信号放到阻塞信号集，这个信号就不会被进程处理
     - 阻塞解除之后，信号被处理
2. 自定义信号集
   - int sigemptyset(sigset_t *set); 将set集合置空
   - int sigfillset(sigset_t *set)； 将所有信号加入set集合
   - int sigaddset(sigset_t *set,int signo);
     - 将signo信号加入到set集合
   - int sigdelset(sigset_t *set,int signo);
     - 从set集合中移除signo信号
   - int sigismember(const sigset_t *set,int signo);
     - 判断信号是否存在
3. sigprocmask函数
   - 屏蔽and接触信号屏蔽，将自定义信号集设置给阻塞信号集
   - 函数原型：

     int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
4. sigpending -- 读取当前进程的未决信号集
   - 函数原型：int sigpending(sigset_t *set);
   - 参数：set -- 内核将未决信号集写入set

练习：

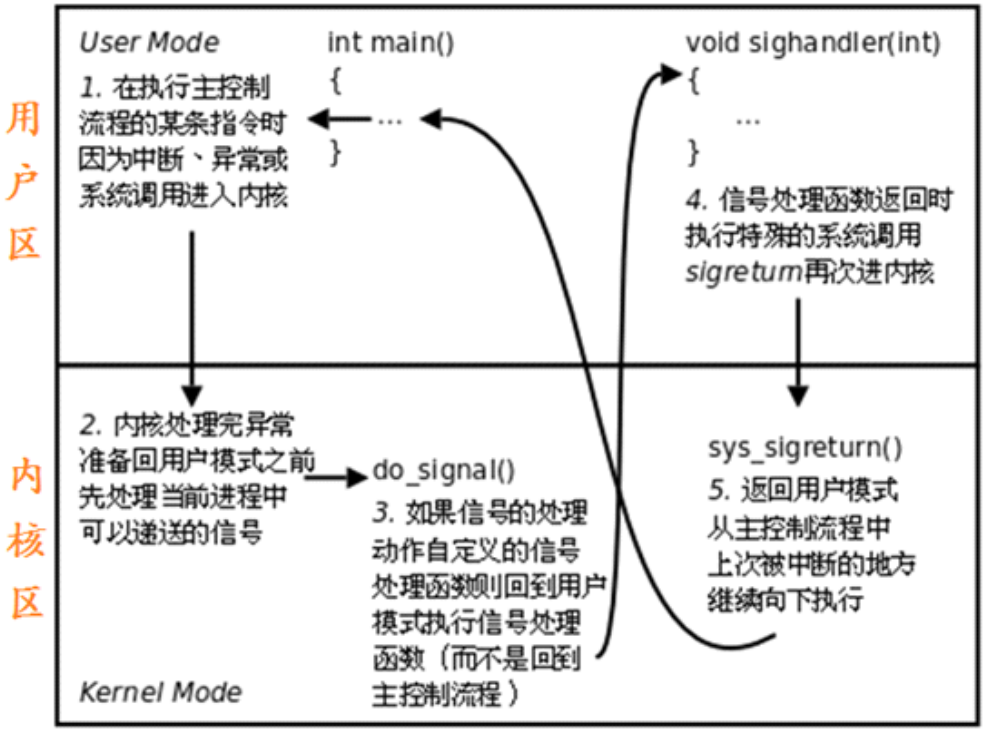
- 编写程序，设置阻塞信号集并把所有常规信号的未决状态打印至屏幕。

4 - 信号捕捉

1. siganl函数
   - typedef void (*sighandler_t)(int);
   - sighandler_t signal(int signum, sighandler_t handler);
2. sigaction函数
   - 函数原型：

```
int sigaction(int signum,  // 捕捉的信号
    const struct sigaction *act,
    struct sigaction *oldact
);
 struct sigaction {
        void     (*sa_handler)(int);
        void     (*sa_sigaction)(int, siginfo_t *, void *);
        sigset_t   sa_mask;
          ◆ 在信号处理函数执行过程中，临时屏蔽指定的信号
        int        sa_flags;
          ◆ 0 - sa_handler
        void     (*sa_restorer)(void);
 };
```

3. 内核实现信号捕捉的过程


用户区
内核区
User Mode
1. 在执行主控制流程的某条指令时因为中断、异常或系统调用进入内核
int main()
{
...
}
void sighandler(int)
{
...
}
4. 信号处理函数返回时执行特殊的系统调用 sigreturn再次进内核
2. 内核处理完异常准备回用户模式之前先处理当前进程中可以递送的信号
do_signal()
3. 如果信号的处理动作自定义的信号处理函数则回到用户模式执行信号处理函数（而不是回到主控制流程）
sys_sigreturn()
5. 返回用户模式从主控制流程中上次被中断的地方继续向下执行
Kernel Mode

5 - SIGCHLD信号

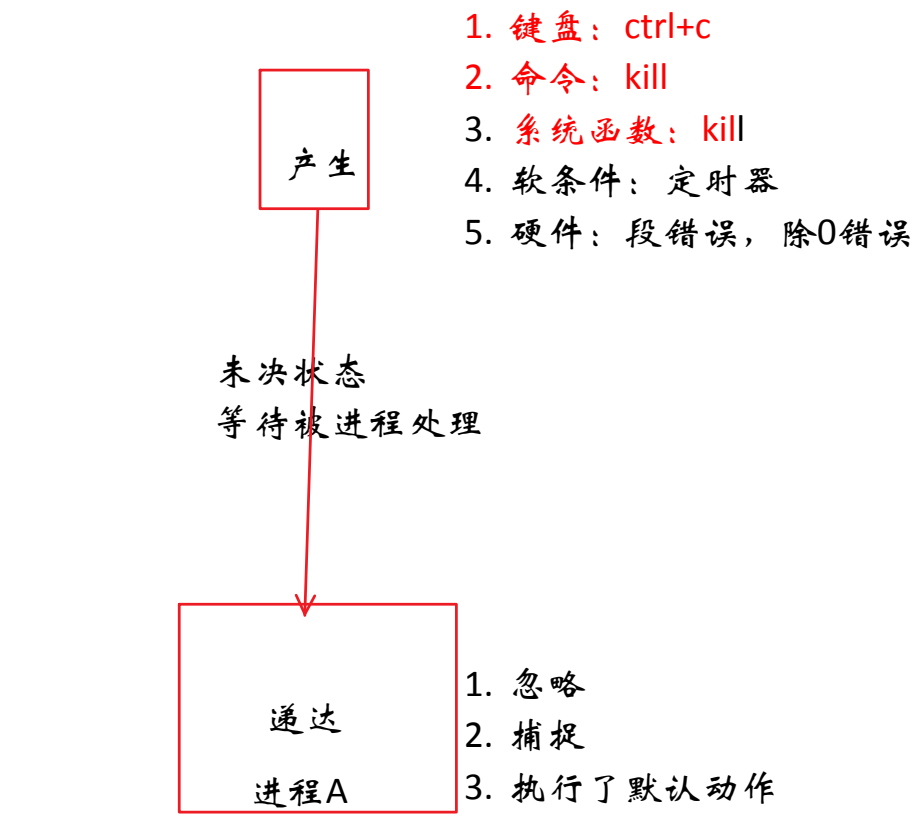
1. 产生的条件
    a. 子进程结束的时候
    b. 子进程挂起
    c. 子进程重新恢复运行状态
2. 使用该信号回收子进程
3. 使用时的注意事项
    a. 进程创建的时候，父子进程共享
        i. 阻塞信号集
        ii. 信号的处理动作
        iii. 极端问题：
            1) 父进程信号处理函数注册成功之前子进程死亡
            2) 解决思路：
                a) 先将SIGCHLD信号阻塞
                b) 父进程的信号处理函数注册成功之后，解除阻塞

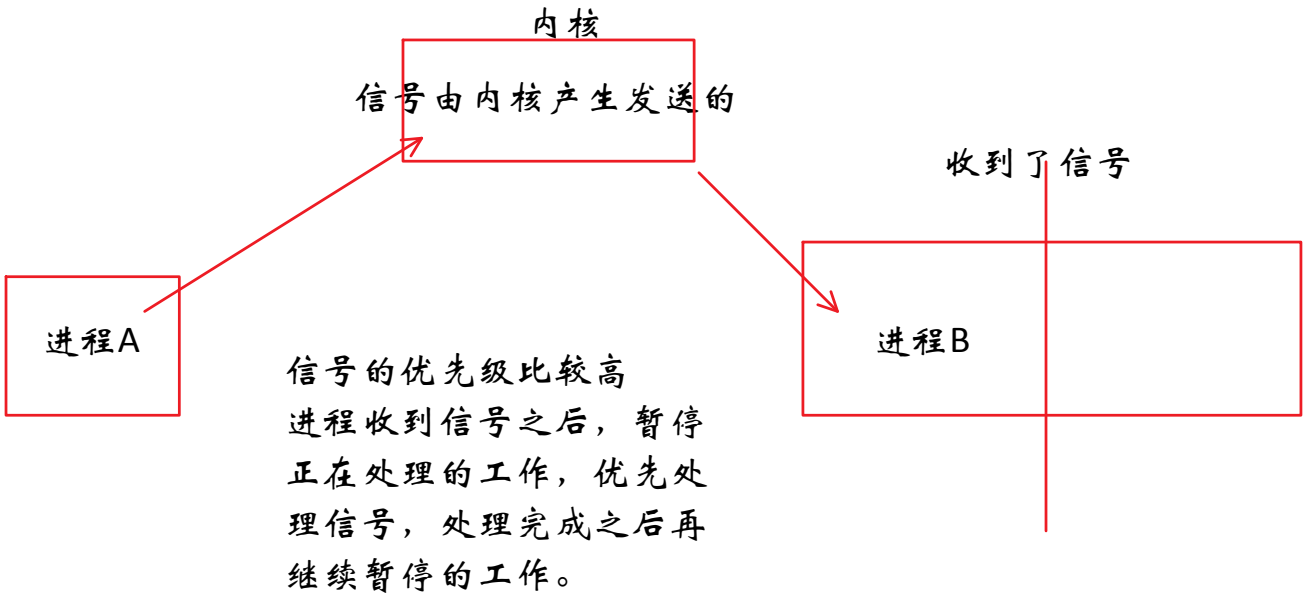
作业

1. 使用setitimer实现每个一秒打印一次hello, world
2. 编程实现对SIGINT信号实施捕捉。在捕捉函数执行期间，再有SIGQUIT信号递达将其阻塞。模拟捕捉函数长时间执行，SIGQUIT信号在此期间多次递达，查验对该信号处理几次。
    - 信号处理的时候是否支持排队
3. 利用SIGUSR1和SIGUSR2在父子进程之间进行消息传递。实现父子进程交替报数。
    - kill()

使用信号进行进程间通信

- 双方使用约定的信号
- 双方注册信号捕捉函数
    - signal
    - sigacton
- 编写对应的信号处理函数


产生
1. 键盘：ctrl+c
2. 命令：kill
3. 系统函数：kill
4. 软条件：定时器
5. 硬件：段错误，除0错误
未决状态
等待被进程处理
递达
进程A
1. 忽略
2. 捕捉
3. 执行了默认动作


内核
信号由内核产生发送的
收到了信号
进程A
进程B
信号的优先级比较高
进程收到信号之后，暂停
正在处理的工作，优先处
理信号，处理完成之后再
继续暂停的工作。