云南大学数学与统计学院
上机实践报告

课程名称： 操作系统实验	**年级：** 2015 级	**上机实践成绩：**
指导教师： 李源	**姓名：** 刘鹏	
上机实践名称： 实现软中断通信	**学号：** 20151910042	**上机实践日期：** 2017-10-27
上机实践编号： No.03	**创建时间：**	**最近一次修改时间：** 17:22

一、实验目的

1. 掌握进程的概念，明确进程的含义；
2. 认识并了解并发执行的实质。

二、实验环境

OS： Red Hat Enterprise Linux 7.4
CPU： Intel Core i3 550
RAM： 三星 Trident Z 16GB 3400MHz
主硬盘： 东芝 SSD

三、实验内容

编制一段程序，实现软中断通信：

使用系统调用 `fork()`创建两个子进程，再用系统调用 `signal()`让父进程捕捉键盘上来的中断信号（即按 `Del` 键），当父进程接受到这两个软中断的其中某一个后，父进程用系统调用 `kill()`向两个子进程分别发送整数值为 `16` 和 `17` 软中断信号，子进程获得对应软中断信号后，分别输出下列信息后终止：

```
Child process 1 is killed by parent!!
Child process 2 is killed by parent!!
```

父进程调用 `wait()`函数等待两个子进程终止后，输出以下信息后终止：

```
Parent process is killed!!
```

四、实验要求

1、写出运行结果
2、多运行几次程序，看结果是否相同，简略分析出现不同结果的原因。

五、实验内容与步骤

题 1

程序代码

```
1   // filename: soft_break.c
2
```

```c
#include<stdio.h>
#include<signal.h>
#include<unistd.h>
#include<sys/types.h>

int wait_flag;

void stop()
{
    wait_flag = 0;
}

int main()
{
    int pid1, pid2;

    signal(3, stop);
    pid1 = fork();
    // Create a new process

    if(pid1 == -1)
    {
        printf("Failed Creating New Process!");
        return 0;
    }
    if(pid1 > 1)
    {
        pid2 = fork();

        if(pid2 == -1)
        {
            printf("Failed Creating New Process!");
            return 0;
        }

        if(pid2 > 0)
        {
            wait_flag = 1;
            sleep(5);
            kill(pid1, 16);
            kill(pid2, 17);
            wait(0);
            wait(0);
            printf("\n Parent process is killed!!\n");
            return 0;
        }
        else
        {
            wait_flag = 1;
```

```
52              signal(17, stop);
53              printf("\n Child process 2 is killed by parent!!\n");
54              return 0;
55          }
56      }
57      else
58      {
59          wait_flag = 1;
60          signal(16, stop);
61          printf("\n Child process 1 is killed by parent!!\n");
62          return 0;
63      }
64      return 0;
65  }
```

Code Box 1

五、实验总结

C 语言在 Windows 下与 Linux 之下，有很多区别，所以有几个函数的功能还需要仔细学习。

`fork` 函数：这个函数应该是生成一个新的进程，同时把新进程的 pid 返回给父进程，同时把数字 0 返回给子进程。当然，当返回 0 的时候代表创建失败。这里需要明确一点，`fork` 函数创建的新进程，与父进程并不完全一样。新进程应该是从 fork 语句之后开始的。设想如果子进程会重新执行这个程序，那么当到了 fork 这个节点之后，就会再创建进程，这直接导致无限循环直到资源用尽。

signal 函数：`void ( *signal( int sig, void (* handler)( int ))) ( int )`，函数声明很长。

`wait` 函数：如其名，等待。`wait` 会暂时停止目前进程的执行，直到有信号来到或者子进程结束。函数定义是 `pid_t wait(int *status)`，当子进程结束之后，wait 就会把子进程的状态值返回给本进程，存储在 status 里面。

kill 函数：`int kill(pid_t pid, int sig)`，将 sig 这个信号传给进程 id 为 pid 的进程。

sleep 函数

global 变量在多线程下的意义：编译完一个含有多线程的 C 代码，得到二进制文件，是否子进程也会拷贝过除了 fork 返回值之外的所有变量，甚至把全局变量也独立拷贝一份运行？这是错误的，实验告诉我，

六、参考文献