云南大学数学与统计学实验教学中心

实验报告

课程名称：操作系统实验	**学期**：2015~2016 学年上学期	**成绩**：
指导教师： 彭程	**学生姓名**：金洋	**学号**：20131910023
实验名称：进程间的通信		
实验编号：二	**实验日期**： 9 月 22 日	**实验学时**： 1
学院： 数学与统计学院	**专业**： 信息与计算科学	**年级**： 2013 级

一、实验目的

使用 c 语言实现创建和终止进程；

二、实验内容

1. 分析一段 Linux 下的进程代码如何实现了创建和终止进程的功能；

三、实验环境

平台：Visual C++ 6.0

语言：C 语言

四、实验过程

1.Linux 下的进程原代码

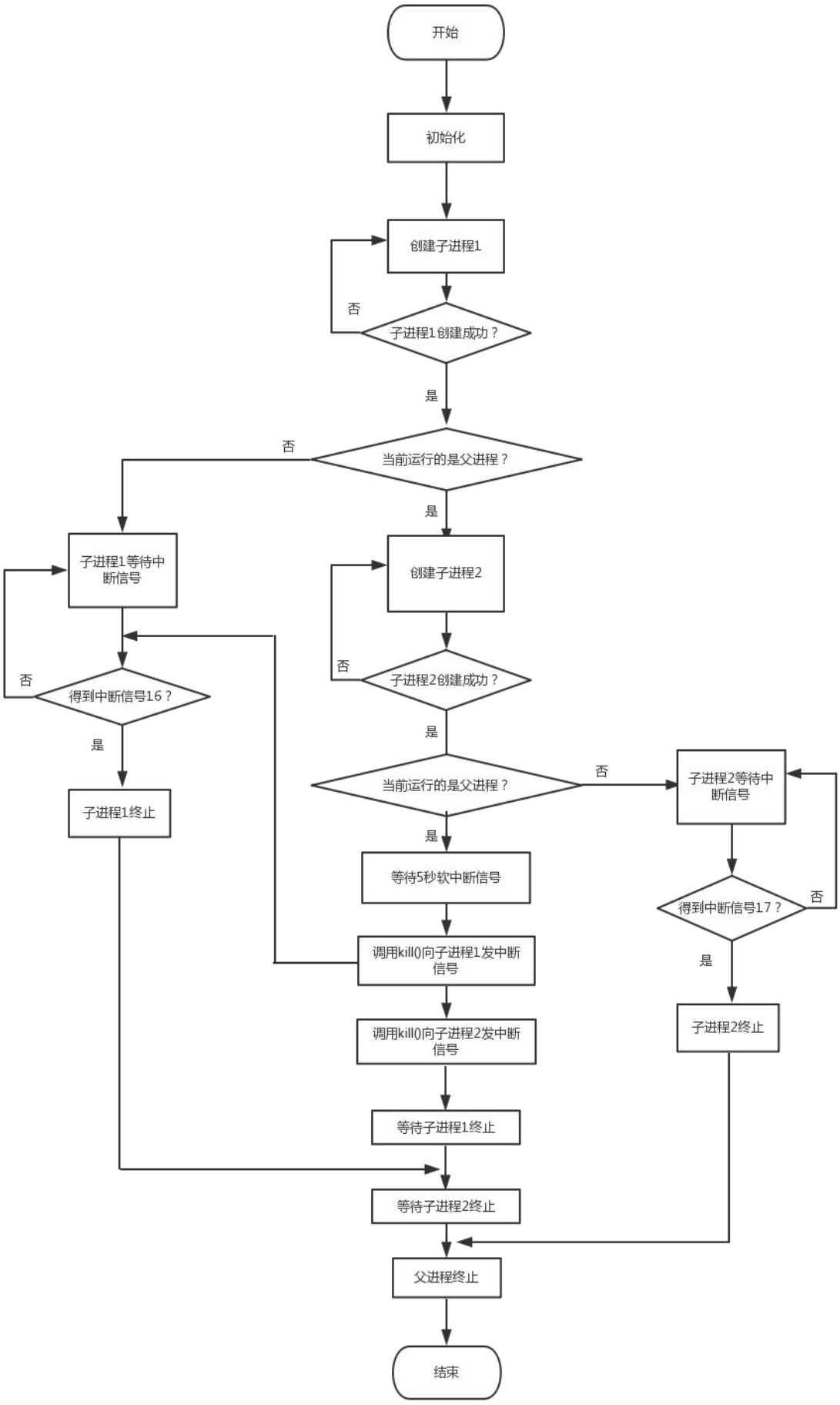
```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <sys/types.h>
int wait_flag;

void stop(){
    wait_flag=0;
}

void main(){
```

```
    int pid1,pid2;
    signal(3,stop); //或  signal(14,stop);

    while ( (pid1=fork())==-1);
    if (pid1>0){

        while    ((pid2=fork())==-1);
        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");
            exit(0);
        }
        else{
            wait_flag=1;
            signal(17,stop);
            printf("\n Child process 2 is killed by parent!!\n");
            exit(0);
        }

    else{
        wait_flag=1;
        signal(16,stop);
        printf("\n Child process 1 is killed by parent!!\n");
        exit(0);
    }
}
```

2.分析程序功能

功能为：fork()创建两个子进程，系统调用 signal()函数，让父进程捕捉键盘上来的中断信号，当父进程接收到中断信号中的一个，父进程调用 kill()向两个子进程分别发送中断信号

程序流程图如下：


开始
初始化
创建子进程1
否
子进程1创建成功？
是
否
当前运行的是父进程？
是
子进程1等待中断信号
创建子进程2
否
得到中断信号16？
否
子进程2创建成功？
是
是
子进程1终止
当前运行的是父进程？
否
子进程2等待中断信号
是
等待5秒软中断信号
得到中断信号17？
否
是
调用kill()向子进程1发中断信号
子进程2终止
调用kill()向子进程2发中断信号
等待子进程1终止
等待子进程2终止
父进程终止
结束

各函数功能

(1)signal()函数

signal()函数是允许调用进程控制软中断信号的处理。

函数原型 ：void (*signal(int signum,void(* handler)(int)))(int);

参数说明:第一个参数 signum 指明了所要处理的信号类型，它可以取除了 SIGKILL 和 SIGSTOP 外的任何一种信号。第二个参数 handler 描述了与信号关联的动作。

函数说明:signal()会依参数 signum 指定的信号编号来设置该信号的处理函数。当指定的信号到达时就会跳转到参数 handler 指定的函数执行。当一个信号的信号处理函数执行时,如果进程又接收到了该信号，该信号会自动被储存而不会中断信号处理函数的执行,直到信号处理函数执行完毕再重新调用相应的处理函数。但是如果在信号处理函数执行时进程收到了其它类型的信号，该函数的执行就会被中断。

返回值： 返回先前的信号处理函数指针，如果有错误则返回 SIG_ERR(-1)。

sig 的取值如下：

信号	功能	值
SIGHUP	挂起	1
SIGINT	键盘中断，键盘按 Delete 键或 Break 键	2
SIGQUIT	键盘按 Quit 键	3
SIGILL	非法指令	4
SIGTRAP	跟踪中断	5
SIGIOT	IOT 指令	6
SIGBUS	总线错	7
SIGFPE	浮点运算溢出	8
SIGKILL	要求终止进程	9
SIGUSR1	用户定义信号#1	10
SIGSEGV	段违法	11
SIGUSR2	用户定义信号#2	12
SIGPIPE	向没有读进程的管道上写	13
SIGALRM	定时器告警，时间到	14
SIGTERM	kill 发出的软件结束信号	15

SIGCHLD	子进程死	17
SIGCONT	若已停止则继续	18
SIGPWR	电源故障	30

(2)fork()函数

fork()仅仅被调用一次，却能够返回两次，它可能有三种不同的返回值：

1）在父进程中，fork 返回新创建子进程的进程 ID；

2）在子进程中，fork 返回 0；

3）如果出现错误，fork 返回一个负值；

在 fork 函数执行完毕后，如果创建新进程成功，则出现两个进程，一个是子进程，一个是父进程。在子进程中，fork 函数返回 0，在父进程中，fork 返回新创建子进程的进程 ID。我们可以通过 fork 返回的值来判断当前进程是子进程还是父进程。

（3）sleep()函数

在 Linux C 语言中，sleep(n)函数的作用是延时，程序暂停若干时间。参数 n 即为暂停秒数。

（4）kill()函数

定义函数 int kill(pid_t pid,int sig);kill 送出一个特定的信号 sig 给进程号为 pid 的进程程根据该信号而做特定的动作,若没有指定，预设是送出终止的信号。

参数 pid 有几种情况：

pid>0 将信号传给进程识别码为 pid 的进程。

pid=0 将信号传给和当前进程相同进程组的所有进程

pid=-1 将信号广播传送给系统内所有的进程

pid<0 将信号传给进程组识别码为 pid 绝对值的所有进程

（5）wait()函数

进程一旦调用了 wait，就立即阻塞自己，由 wait 自动分析是否当前进程的某个子进程已经退出，如果让它找到了这样一个已经变成僵尸的子进程，wait 就会收集这个子进程的信息，并把它彻底销毁后返回；如果没有找到这样一个子进程，wait 就会一直阻塞在这里，直到有一个出现为止。

有时候，父进程要求子进程的运算结果进行下一步的运算，或者子进程的功能是为父进程提供了下一步执行的先决条件（如：子进程建立文件，而父进程写入数据），此时父进程就必须在某一个位置停下来，等待子进程运行结束，而如果父进程不等待而直接执行下去的话，可以想见，会出现极大的混乱。

其调用格式为：

#include <sys/type.h>

#include <sys/wait.h>
(pid_t) wait(int *statloc)；
正确返回：大于 0：子进程的进程 ID 值；
等于 0：其它；
错误返回：等于－1：调用失败；

3. 代码注释后如下：

```
#include <stdio.h>
#include <signal.h>//头文件<signal.h>中提供了一些用于处理程序运行期间所引发的异常条件的功能，如处理来源于外部的中断信号或程序执行期间出现的错误等事件
#include <unistd.h>
#include <sys/types.h>
int wait_flag;//全局变量 wait_flag,用来标识进程状态
void stop(){
    wait_flag=0;
}

void main(){
    int pid1,pid2;//定义两个进程号参数
    signal(3,stop); //或  signal(14,stop);signal()函数允许调用进程控制软中断信号的处理

    while ( (pid1=fork())==-1);//程序等待  成功创建子进程的事件发生

    if (pid1>0){
    //  子进程创建成功,pid1 为进程号
        while   ((pid2=fork())==-1);   //  创建子进程 2
        if (pid2>0){
            wait_flag=1;//标记为阻塞状态
            sleep(5);//  父进程等待 5 秒
            kill(pid1,16);//杀死进程 1
            kill(pid2,17);//杀死进程 2
            wait(0);//等待子进程 1 结束的信号
            wait(0);//等待子进程 2 结束的信号
            printf("\n Parent process is killed!!\n");
            exit(0); //  父进程结束
        }
```

```
        else{
            wait_flag=1;
            signal(17,stop);//等待进程 2 被杀死的中断号 17
            printf("\n Child process 2 is killed by parent!!\n");
            exit(0);
        }

    else{
        wait_flag=1;
        signal(16,stop);//等待进程 1 被杀死的中断号 16
        printf("\n Child process 1 is killed by parent!!\n");
        exit(0);
    }
}
```

五、实验结果

Child process 1 is killed by parent !!
Child process 2 is killed by parent !!
Parent process is killed !!
或者多次运行，并且 Delete 键后，会出现如下结果：
Child process 2 is killed by parent !!
Child process 1 is killed by parent !!
Parent process is killed !!

六、总结

简要分析

1.signal 函数

上述程序中，调用函数 signal()都放在一段程序的前面部位，而不是在其他接收信号处。这是因为 signal()的执行起的作用只是为进程指定信号量 16 和 17，以及分配相应的与 stop()过程连接的指针。因而 signal()函数必须在程序前面部分执行。

2.wait 函数

在父进程中调用第 1 个 wait(0)后，则父进程被阻塞。进入等待第一个子进程运行结束的队列，等待子进程结束。当子进程结束后，会产生一个终止状态字，系统会向父进程发出 SIGCHLD 信号。当接到信号后，父进程提取子进程的终止状态字，从 wait()返回继续执行原程序。同样的方式，父进程继续执行第二个 wait(0)，并再次阻塞，等待第 2 个子进程运行结束。当第二个子进程运行结束后父进程继续执行剩余的语句。

3.关于 exit 函数

该函数中每个进程退出时都用了语句 exit(0)，这是进程的正常终止。在正常终止时，exit()函数返回进程结束状态。进程终止时，则由系统内核产生一个代表异常终止原因的终止状态，该进程的父进程都能用 wait()得到其终止状态。在子进程调用 exit()后，子进程的结束状态会返回给系统内核，由内核根据状态字生成终止状态，供父进程在 wait()中读取数据。若子进程结束后，父进程还没有读取子进程的终止状态，则子进程就变成了“孤儿进程”，系统进程 init 会自动“收养”该子进程，成为该子进程的父进程，即父进程标识号变成 1，当子进程结束时，init 会自动调用 wait()读取子进程的遗留数据，从而避免在系统中留下大量的垃圾。

4.结果显示

上述结果中“Child process 1 is killed by parent !!” 和“Child process 2 is killed by parent !!”相继出现，当运行几次后，谁在前谁在后是随机的。这是因为：从进程调度的角度看，子进程被创建后处于就绪态。此时，父进程和子进程作为两个独立的进程，共享同一个代码段，分别参加调度、执行，直至进程结束。但是谁会先被调度程序选中执行，则与系统的调度策略和系统当前的资源状态有关，是不确定的。因此，谁先从 fork()函数中返回继续执行后面的语句也是不确定的。

七、参考文献

[1]汤小丹，梁红兵，哲凤屏，汤子瀛.计算机操作系统[M]（第三版）. 西安：西安电子科技大学出版社，2007 年 5 月；

[2]谭浩强著.c 程序设计[M]（第三版）.北京：清华大学出版社.2005.7；

八、教师评语