

**课 程 实 验 报 告**

**课程名称： 操 作 系 统 原 理**

**专业班级： 计算机科学与技术1609**

**学 号： U201614758**

**姓 名： 梅朝瑞**

**指导教师： 谢 夏**

**报告日期： 2019/1/1**

**计算机科学与技术学院**

# 实验一：进程控制

## 实验目的

1. 加深对进程的理解,进一步认识并发执行的实质；
2. 分析进程争用资源现象,学习解决进程互斥的方法；
3. 掌握Linux进程基本控制；
4. 掌握Linux系统中的软中断和管道通信。

## 实验内容

1. **程序要求**

编写程序，演示多进程并发执行和进程软中断、管道通信。

• 父进程使用系统调用pipe( )建立一个管道,然后使用系统调用 fork()创建两个子进程，子进程1和子进程2；

• 子进程1每隔1秒通过管道向子进程2发送数据: I send you x times. (x初值为1，每次发送后做加一操作） 子进程2从管道读出信息，并显示在屏幕上。

• 父进程用系统调用signal()捕捉来自键盘的中断信号（即按Ctrl+C 键）；当捕捉到中断信号后，父进程用系统调用Kill()向两个子进 程发出信号，子进程捕捉到信号后分别输出下列信息后终止：

Child Process l is Killed by Parent! Child Process 2 is Killed by Parent! • 父进程等待两个子进程终止后，释放管道并输出如下的信息后终止

Parent Process is Killed!

1. **运行环境**

**软件配置：**

主机：Windows 10 专业版 Build 17763.195

虚拟机：VMware Workstation 14 Pro，Deepin 15.8 Linux/GNU

开发环境：Eclipse IDE for C/C++ Developers Oxygen Release (4.7.0)

**硬件：**

AMD Ryzen 5 1600 Six-Core Processor 3.20 GHz

Kinston HyperX Fury DDR4 2400 8G

NVIDIA GeForce GTX 1060 6GB

1. **源程序**

//头文件，包含了各种系统调用的库

#include <stdio.h>

#include <unistd.h>

#include <string.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <signal.h>

//全局变量声明，包括一些输出信息和管道指针等。

pid\_t pid1;

pid\_t pid2;

char killed1[40] = "Child Process l is Killed by Parent!\n";

char killed2[40] = "Child Process 2 is Killed by Parent!\n";

char buf[30];

int n=1;

int fd[2];

int \*write\_fd = &fd[1];

int \*read\_fd = &fd[0];

int status;

//主进程收到SIGINT信号后的处理函数

void shutall(int sig\_no){

kill(pid1,SIGUSR1); //向两个子进程发送自定义SIGUSR1信号

//usleep(1000); //略微错开时间，以使子进程1先结束

//（实际上上面一行导致了子进程2没能输出信息，故注释掉）

kill(pid2,SIGUSR1);

}

//子进程1收到SIGUSR1信号后的处理函数

void down1(int sig\_no){

printf("%s",killed1); //输出进程结束信息

usleep(300);

kill(pid1,SIGKILL); //发送自杀信号，结束进程

}

//子进程2收到SIGUSR1信号后的处理函数

void down2(int sig\_no){

printf("%s",killed2); //输出进程结束信息

usleep(300);

kill(pid2,SIGKILL); //发送自杀信号，结束进程

}

//子进程1行为

int child1(){

signal(SIGINT,SIG\_IGN);

signal(SIGUSR1,down1);

while(1){

sprintf(buf,"I send you %d times.\n",n);

write(fd[1],buf,30); //将字符串送入管道

n++;

sleep(1); //休眠1秒

}

}

//子进程2行为

int child2(){

signal(SIGINT,SIG\_IGN);

signal(SIGUSR1,down2);

while(1){

usleep(300000); //休眠0.3秒后进行read

read(fd[0],buf,30);

printf("%s",&buf);

usleep(700000); //休眠0.7秒，则一周期为1秒

}

}

int main(void){

signal(SIGINT,shutall);

if(pipe(fd)==-1){

perror("make pipe");

exit(-1);

}

pid1=fork(); //fork两个子进程

pid2=fork();

if(pid1==0&&pid2!=0) child1(); //判定是哪个子进程，以执行相应行为函数

if(pid2==0&&pid1!=0) child2();

waitpid(pid1,&status,0); //等待子进程结束

waitpid(pid2,&status,0);

close(fd[0]); //关闭管道

close(fd[1]);

return 0; //主进程结束

}

1. **实验结果**

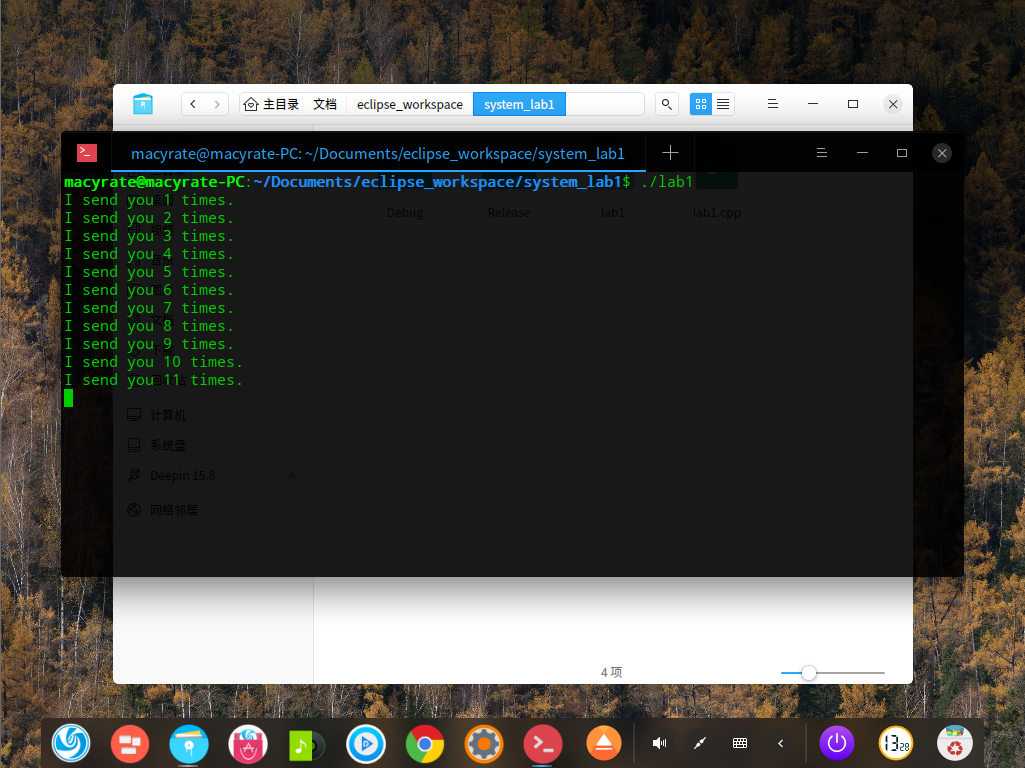
****

图1-1 程序执行中

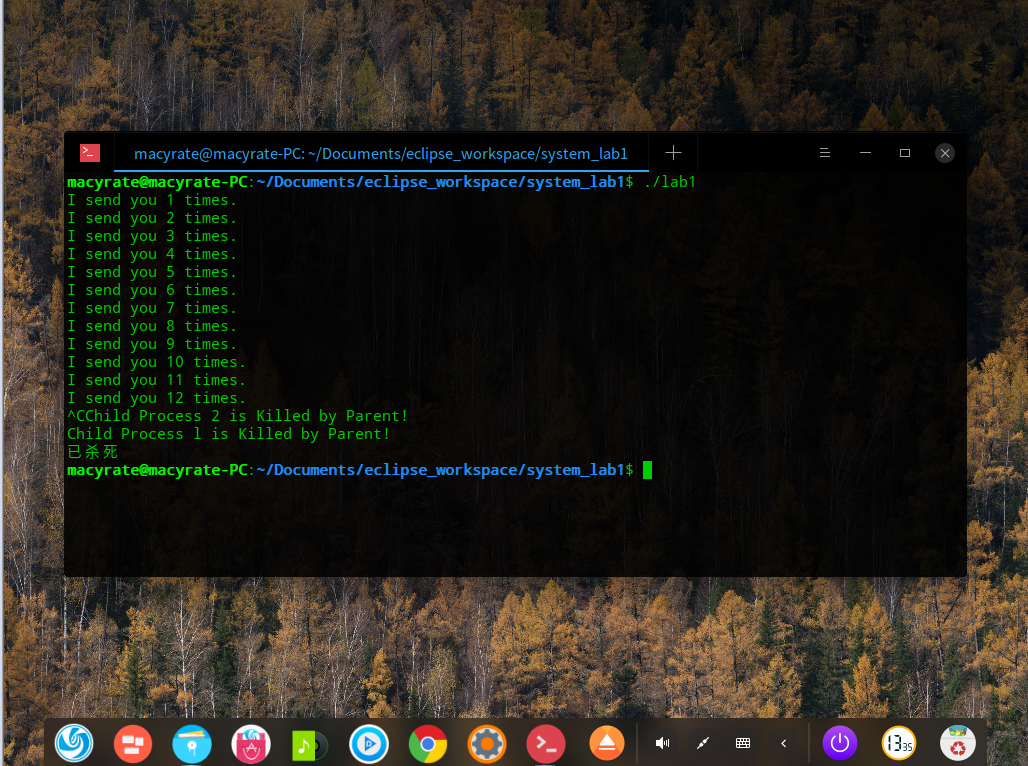
****

图1-2 Ctrl+C终止程序

## 实验心得

这是操作系统的第一次实验，其实重点应该在Linux环境的搭建上，实验内容不算复杂，但由于之前没有认真研究实验文档，而是直接阅读实验指导教材，走了不少弯路。

最为典型的就是那个signal调用，其在实验指导教材上给出的是第二个参数处写处理函数的函数名，但并未给出任何实例解释应该用怎样的方式定义处理函数。为此面向Google编程，花了很长时间都没有弄明白。因为这个调用按照规范使用起来虽然简单，但实际上这里的定义确实过于复杂，难以理解。

还有就是kill调用，直觉上本以为是用来杀死进程的，但实际上是用来向进程发送信号，只不过包含了杀死进程这个可能选项（并且也确实最为常用）所以才这么叫。

总之，一切的问题都在阅读了实验文档之后豁然开朗，也亏得这次的文档相对详细，实验做起来还算爽快。