**《操作系统》**

**实 验 指 导 手 册**

**授课教师： 刘知青**

**实验指导教师： 刘知青**

**教学对象： 二年级本科生**

**开课时间： 春季学期**

**学生姓名： 裴仪瑶**

**学号： 2014212052**

**成绩：**

**北京邮电大学软件学院**

**2016年3月30日**

1. 实验目的

了解进程的基本特性和创建过程与方法。熟悉进程之间的相互调用关系以及进程间相互通信的机制。同时熟悉fork函数以及execv 函数的基本使用方法。

1. 实验内容
   * + 1. 创建Unix下的shell文件
       2. 利用fork创建子进程
       3. 用execv函数执行相应的命令
2. 实验环境

Linux操作系统，Code::Blocks

1. 实验要求
   * + 1. 按照要求修改main函数和setup函数
       2. 能按Ctrl+C调出10个历史指令列表，重写handle\_SIGINT函数
       3. 输入r，执行最近一条指令。输入r x命令，执行以x为第一个字符的指令
2. 实验代码

#include<stdio.h>

#include<unistd.h>

#include<signal.h>

#define MAXLINE 100

char\* args[100][MAXLINE];

int line\_num = 0;//输入命令行数组

int wrong = 0;//输入错误标志

int index(char a) {//根据字符查询数组下标

int begin;//开始

if (line\_num - 10 < 0)//少于10行

begin = 0;

else//少于10行

begin = line\_num - 10;

while (begin <= line\_num) {

if (args[begin][0][0] == a)

return begin;

begin++;

}

return -1;

}

void setup(char \*args[]) {//处理命令行字符串，拆分

char a;

int i = 0;

while (1) {

args[i] = (char\*)malloc(10);

scanf("%s", args[i]);

a = getchar();

if (a == '\n')

break;

else

ungetc(a, stdin);

i++;

}

args[i + 1] = NULL;

}

void handler\_SIGINT() {//输入control C时

line\_num--;

int begin;

write(STDOUT\_FILENO, "History：\n", strlen("History：\n"));

if (line\_num - 10 < 0)

begin = 0;

else

begin = line\_num - 9;

while (begin <= line\_num) {//写出近10行命令

int i = 0;

while (args[begin][i] != NULL) {//写命令

write(STDOUT\_FILENO, args[begin][i], strlen(args[begin][i]));

write(STDOUT\_FILENO, " ", 1);

i++;

}

begin++;

write(STDOUT\_FILENO, "\n", 1);

}

//重写历史命令

int num;//命令在数组中的下标

while (1) {

write(STDOUT\_FILENO, "\nEnter 'r x' to run command in history, x is the first letter of command\nEnter r to run the last command\n", strlen("\nEnter 'r x' to run command in history, x is the first letter of command\nEnter r to run the last command\n"));

char order[] = { ' ',' ',' ' };

read(STDIN\_FILENO, order, 3);

if (order[0] == 'r' && order[2] == ' ') {//输入r

num = line\_num;

line\_num++;

int j = 0;

do{

args[line\_num][j] = (char\*)malloc(80);

args[line\_num][j]= args[num][j];

j++;

}while(args[num][j] != NULL);

break;

}

else if (order[0] == 'r' && index(order[2]) >= 0) {//输入r x

num = index(order[2]);

line\_num++;

int j = 0;

args[line\_num][0] = (char\*)malloc(80);

do{

args[line\_num][j] = args[num][j];

j++;

}while(args[num][j]!= NULL);

break;

}

else {//输入错误

write(STDOUT\_FILENO, "wrong\n", strlen("wrong\n"));

wrong =1;//错误标识

break;

}

}

}

int main() {

int background; //判定所有子进程是否结束

struct sigaction handler;

handler.sa\_handler = handler\_SIGINT;

sigaction(SIGINT, &handler, NULL);

while (1) {//输入命令行

wrong = 0;

write(STDOUT\_FILENO, "COMMAND->", 9);

setup(args[line\_num]);

background = fork();

if (background > 0) {//父进程

wait();

}

else if (background == 0) {//子进程

if(wrong ==0){

execvp(args[line\_num][0], args[line\_num]);

}

exit(0);

}

line\_num++;

}

}

1. 实验结果

