# 操作系统课程设计报告

1. 实验4：
   1. 实验题目：linux进程管理
   2. 小组成员：

组长：李秀峰 15058317

组员：李思琦 15058315

韦 鑫 15061323

* 1. 项目设计方案：

（1）成员分工描述；

李秀峰：完成shell设计、无名管道设计（第2/3小题），及文档撰写

韦 鑫：完成sender和receive线程创建（第4小题），及文档撰写

李思琦：完成sender程序（第5小题），及文档撰写

（2）本实验总体设计思路；

1. shell设计（第二题）

先写三个cmd程序，分别输出一句话，编译；然后再写一个主程序用来当做shell，根据输入的名称分别运行相应的程序，直到输入exit才退出程序。

2.无名管道（第三题）

父进程fork出3个子进程，然后子进程向管道写入数据，过程中要考虑到子进程的对管道的互斥访问，等子进程都结束后，父进程读取数据，程序完成。

3. sender和receive线程创建（第四题）

（韦 鑫）

4. sender程序（第五题）

（李思琦）

（3）本实验中主要函数的接口设计；

1. shell设计（第二题）

1. 三个cmd程序

输出语句代表是第几个cmd

printf("This is xf's cmd\*\*\*----\*\*\* !\n");

1. int getInput()函数

获得输入的数据，和shellCommand数组比对，直到返回正确的指令，如果输入无效，则返回-1

1. void runCmd(int i)函数

对输入的指令进行判断，执行相应步骤，退出程序或者运行子进程（cmd）

1. void cmdFork(int i)函数

对参数指令进行判断，fork子进程，然后运行相应的cmd程序

1. 主函数

提示信息，该输入什么指令，然后死循环运行cmd，直到遇到exit指令结束程序

2.无名管道（第三题）

1. void WritePipe(int id)函数

成功创建子进程之后，进行写操作，对输入的参数进行判断，对应相应的子进程，先关闭读管道，再锁死管道，保证互斥，接着再写入数据，写完数据后使进程沉睡3s,保证父进程可以从管道读出数据，接着解锁再退出子进程

1. 主函数

创建管道，再创建3个子进程，分别调用WritePipe函数进行写操作，接着用wait函数阻塞自己，确保3个子进程完结，再开始读操作，最后退出程序

3. sender和receive线程创建（第四题）

（韦 鑫）

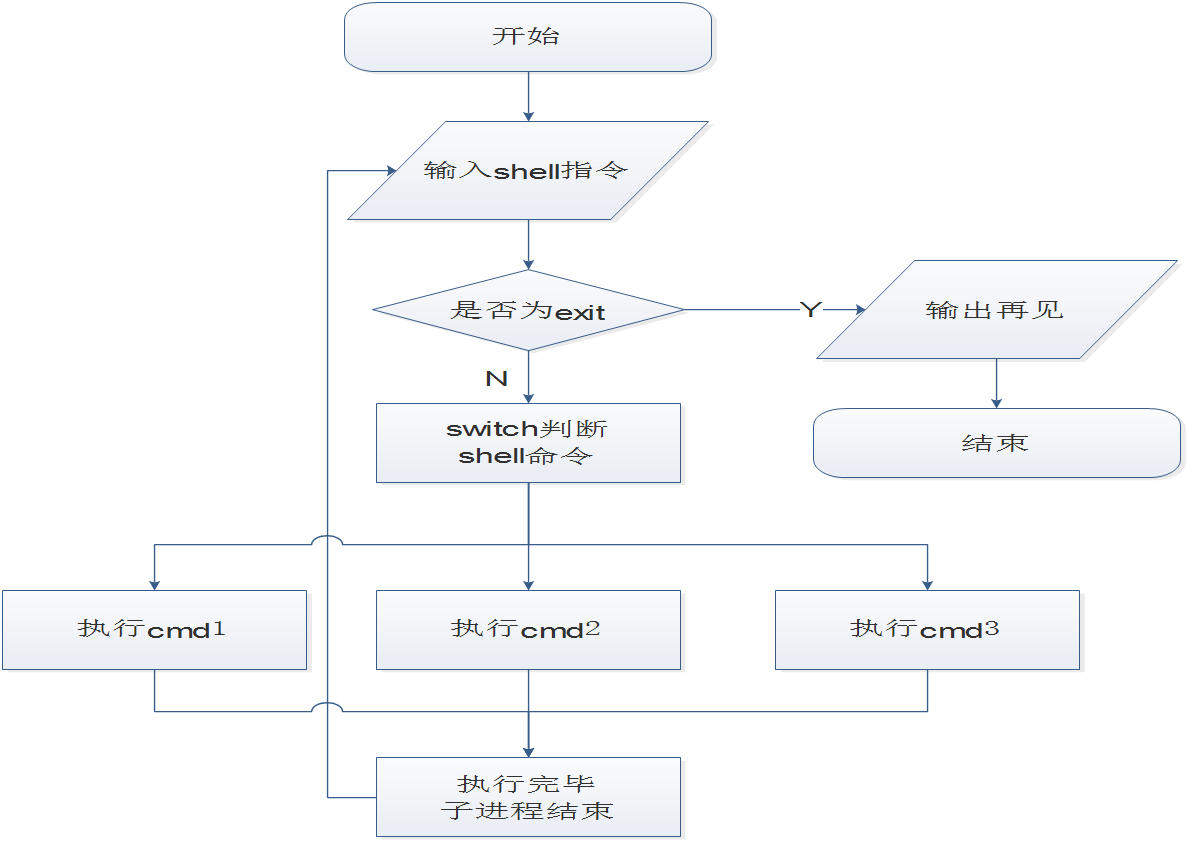
4. sender程序（第五题）

（李思琦）

（4）主要函数的程序设计思路，并画出程序设计流程图。

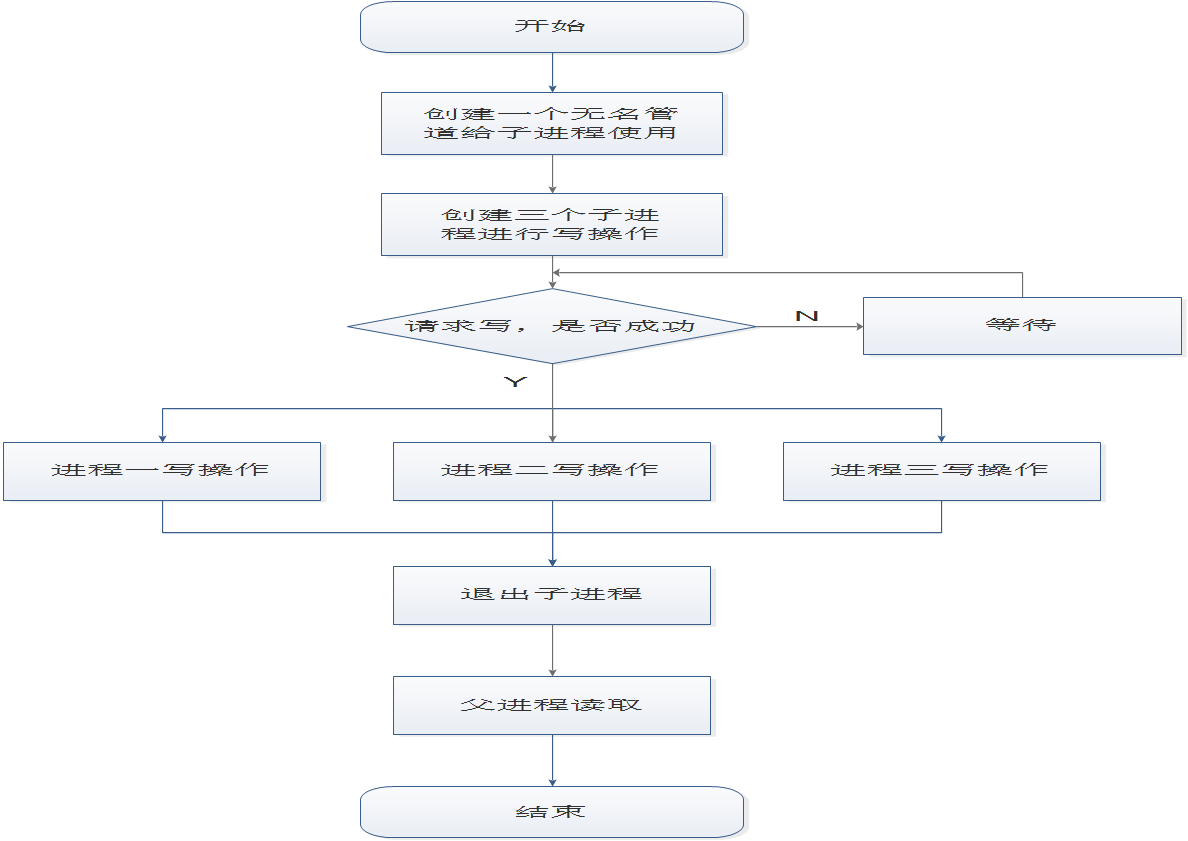
1. shell设计（第二题）

根据提示信息，死循环运行，输入指令，分别运行对应的cmd程序，直到遇到exit指令结束程序



2.无名管道（第三题）

父进程创建管道，再创建3个子进程，子进程进行写操作，中间注意互斥，接着父进程用wait函数阻塞自己，确保3个子进程完结，再开始读操作，最后退出程序



3. sender和receive线程创建（第四题）

（韦 鑫）

4. sender程序（第五题）

（李思琦）

（5）对设计方案的创新性分析

1. shell设计（第二题）

先对输入的指令进行判断，有效再运行Switch执行对应操作，加大了程序合理性

2.无名管道（第三题）

因为3个子进程操作一样，故使用自定义函数操作，简化代码量

用fprintf(stderr, "%s \n",strerror(errno) )来代替printf输出错误信息，因为stderr优先级比stdout高，故可以在程序出错后输出信息

3. sender和receive线程创建（第四题）

（韦 鑫）

4. sender程序（第五题）

（李思琦）

4、项目实现过程：

（1）详细记录项目实现过程中遇到的问题、原因及解决方法（可用截图加文字说明的方法进行编写）

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 问题 | 原因 | 解决方案 |
| 1  （第二题） | 刚开始不知道如何运行cmd | 对fork及excel不太了解 | Fork出子进程，然  后excel运行cmd程序 |
| 2  （第三题） | 无名管道写入数据时，要将参数id写入string，刚开始用  itoa函数报错 | Linux没有iota这个函数 | 百度之后，知道linux使用sprintf将整数转换成字符串 |
| 3  （第三题） | 刚开始本来连续3个fork，这样不止产生了3个子进程 | Fork函数，会把之后的代码都运行一遍，那子进程也会fork子进程，这样就不止3个子进程了 | 用if else来判断，将  fork写在父进程里面 |
| 4  （第三题） | 没等子进程运行完毕，父进程就开始运行，导致读取失败 | 刚开始子进程没有sleep确保运行完，父进程也没有阻塞自己 | 子进程加上sleep（3），沉睡3s，父进程加上3个wait（0）阻塞自己 |
| 5  （第四题） | （韦 鑫） |  |  |
| 6  （第四题） | （韦 鑫） |  |  |
| 7  （第五题） | （李思琦） |  |  |
| 8  （第五题） | （李思琦） |  |  |

（2）分析程序运行结果

1. shell设计（第二题）

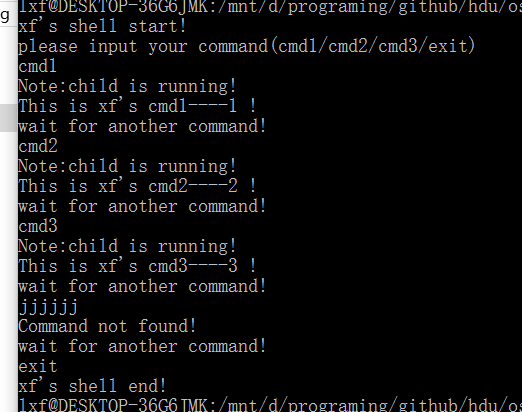
输入 exit 退出程序

输入 cmd1 运行cmd1程序，输出This is xf's cmd1

输入 cmd2 运行cmd2程序，输出This is xf's cmd2

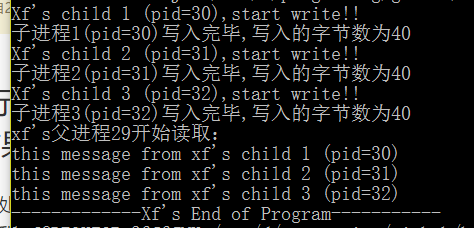
输入 cmd3 运行cmd3程序，输出This is xf's cmd3

输入 无效字符 提示 command not found



2.无名管道（第三题）

先分别输出3个子程序的内容，再停顿，然后父程序读取内容



3. sender和receive线程创建（第四题）

（韦 鑫）

4. sender程序（第五题）

（李思琦）

（3）分析程序实现中的创新性；

1. shell设计（第二题）

暂无

2.无名管道（第三题）

写入数据的时候，用了函数，一起解决，且输出时都带上了pid，方便了解进程间的关系

用fprintf(stderr, "%s \n",strerror(errno) );来代替printf，可以在程序运行出错后有效的输出报错信息

3. sender和receive线程创建（第四题）

（韦 鑫）

4. sender程序（第五题）

（李思琦）

5、对项目的进一步思考：

（1）分析你所实现的程序的性能，包括优缺点；

1. shell设计（第二题）

优点：充分考虑到各种输入，个性化报错提示

缺点：3个cmd程序都要先编译运行，太麻烦了

2.无名管道（第三题）

优点：提示多，且输出时带上了pid，好了解父子进程的关系

缺点：因为sleep，运行时间长

3. sender和receive线程创建（第四题）

（韦 鑫）

4. sender程序（第五题）

（李思琦）

（2）考虑的改进思路；

1. shell设计（第二题）

可以写个Makefile，来一起编译运行程序

Fork出错时错误信息用的printf输出，可以用标准错误流stderr输出，因为stdout优先级比stderr低，当fork出错程序崩溃时，stderr能输出，而printf不行

2.无名管道（第三题）

总觉得连续3个创建子进程可以写个函数，减少代码量

3. sender和receive线程创建（第四题）

（韦 鑫）

4. sender程序（第五题）

（李思琦）

（3）你认为教师对本实验的设计是否合理？难易程度如何？有什么改进建议？

较难，合理

6、参考文献（含阅读书籍、论文、网络资源等）

<http://blog.csdn.net/l_xrui/article/details/71505094?utm_source=gold_browser_extension>

<http://blog.csdn.net/y_23k_bug/article/details/9885185>

<http://blog.csdn.net/u014285517/article/details/50443557>

<http://www.doc88.com/p-6061500862556.html>

7、程序完整代码

1. shell设计（第二题）

**cmd1.c**

#include<stdio.h>

int main()

{

printf("This is xf's cmd1----1 !\n");

return 0;

}

**Cmd2.c**

#include<stdio.h>

int main()

{

printf("This is xf's cmd2----2 !\n");

return 0;

}

**cmd3.c**

#include<stdio.h>

int main()

{

printf("This is xf's cmd3----3 !\n");

return 0;

}

**Xf\_shell.c**

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

#include<string.h>

#include<sys/types.h>

#include<sys/wait.h>

#define cmdWrong -1

#define cmdExit 0

#define cmdCMD1 1

#define cmdCMD2 2

#define cmdCMD3 3

char \*shellCommand[]={"exit","cmd1","cmd2","cmd3"};

//获得输入并返回指令

int getInput()

{

char \*in;

int i;

scanf("%s",in);

for(i=0;i<4;i++)

{

if(strcmp(in,shellCommand[i])==0)

return i;

}

return -1;

}

//创建子进程，运行程序

void cmdFork(int i)

{

pid\_t pid;

if((pid=fork())<0)

{

printf("Error:fork error! xf's shell end!\n");

exit(0);

}

else if(pid==0)

{

int execl\_status=-1;

printf("Note:child is running!\n");

switch(i)

{

case cmdCMD1:

execl\_status=execl("./cmd1","cmd1",NULL);

break;

case cmdCMD2:

execl\_status=execl("./cmd2","cmd2",NULL);

break;

case cmdCMD3:

execl\_status=execl("./cmd3","cmd3",NULL);

break;

default:

printf("Command not found!\n");

break;

}

if (execl\_status<0)

{

printf("Error:fork error! xf's shell end!\n");

exit(0);

}

printf("child is running!");

exit(0);

}

}

//对输入的指令进行判断，执行相应步骤

void runCmd(int i)

{

switch(i)

{

case cmdWrong:

printf("Command not found!\n");

break;

case cmdExit:

printf("xf's shell end!\n");

exit(0);

break;

default:

cmdFork(i);

break;

}

}

int main()

{

int command;

printf("xf's shell start!\n");

printf("please input your command(cmd1/cmd2/cmd3/exit)\n");

while(1)

{

command=getInput();

runCmd(command);

wait(0); //父进程要等到子进程结束，才接收下一条命令

printf("wait for another command!\n");

}

return 0;

}

2.无名管道（第三题）

**xf\_pipe.c**

/\*

\*@author：lxf

\*@time：2017/5/29 12:30

\*@title：os实验四第三题

\*@subject：由父进程创建一个管道，然后再创建3个子进程，

并由这3个字进程利用管道与父进程之间进行通信：

子进程发送消息，父进程等三个子进程全部发完消

息后再接受信息。通信的具体内容可根据自己的需

要随意设计，要求能试验阻塞型读写过程中的各种

情况，并且要实现进程间对管道的互斥访问。运行

程序，观察各种情况下，进程实际读写的字节数以

及进程阻塞唤醒的情况。

\*

\*/

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <string.h>

#include <errno.h>

#include <unistd.h>

#define size 40

int fd[2]; //fd[0]:读管道，fd[1]:写管道

//成功创建子进程，进行写操作

void WritePipe(int id){

//子进程进行写入前先关闭读取管道

close(fd[0]);

lockf(fd[1],1,0); //对写管道加锁

printf("Xf's child %d (pid=%d),start write!!\n",id+1,getpid());

char mesg[100];

// itoa(id+1, str, 10); int转字符串 linux没这个函数

// 使用sprintf将整数转换成字符串

sprintf(mesg, "this message from xf's child %d (pid=%d) !!\n", id+1,getpid());

//write函数写入失败时返回-1，成功时返回写入的字节数

int re = (int)write(fd[1], mesg, size);

if( re != -1 ){

printf("子进程%d(pid=%d)写入完毕,写入的字节数为%d\n",id+1,getpid(),re);

}

else{

printf("子程序%d(pid=%d)写入失败\n",id,getpid());

}

sleep(3); //子进程沉睡3s,保证父进程可以从管道读出数据

lockf(fd[1],0,0); //解锁

exit(0);

}

int main()

{

pid\_t pid[3]; //3个子进程

char getMesg[40];

int i;

if(pipe(fd)<0) //创建管道失败

{

//标准错误流，内核在处理stdout和stderr时优先级不一

//样，后者的优先级要高一些，因此有时候如果程序异常退出时，

//stderr能得到输出，而stdout就不行。

//printf(...)实际上相当于fprintf(stdout, ...)

//strerror(errno) 错误信息

fprintf(stderr, "Xf's create pipe error : %s\n",strerror(errno));

}

if ((pid[0]=fork())<0) //创建子进程1失败

{

fprintf(stderr, " Xf's Fork 1 Error : %s \n",strerror(errno) );

//EXIT\_FAILURE为1，表示没有成功地执行一个程序,

exit(EXIT\_FAILURE);

}

if (pid[0]==0) //子进程1，写操作

{

WritePipe(0);

}

else

{

if ((pid[1]=fork())<0) //创建子进程2失败

{

fprintf(stderr, " Xf's Fork 2 Error : %s \n",strerror(errno) );

//EXIT\_FAILURE为1，表示没有成功地执行一个程序,

exit(EXIT\_FAILURE);

}

if (pid[1]==0)

{

WritePipe(1); //子进程2，写操作

}

else

{

if ((pid[2]=fork())<0) //创建子进程3失败

{

fprintf(stderr, " Xf's Fork 3 Error : %s \n",strerror(errno) );

//EXIT\_FAILURE为1，表示没有成功地执行一个程序,

exit(EXIT\_FAILURE);

}

if (pid[2]==0)

{

WritePipe(2); //子进程3，写操作

}

else //父进程

{

wait(0); //阻塞自己，等3个子进程完结

wait(0);

wait(0);

//读取前关闭写入管道

close(fd[1]);

printf("xf's父进程%d开始读取：\n",getpid());

for ( i=0; i<3; i++)

{

if ( read(fd[0], getMesg, size) != -1 )

printf("%s\n",getMesg);

else

printf("读取子进程%d的失败！！\n",i);

}

printf("-------------Xf's End of Program-----------\n");

return 0;

}

}

}

return 0;

}

3. sender和receive线程创建（第四题）

（韦 鑫）

4. sender程序（第五题）

（李思琦）