操作系统课程设计报告

学院：计算机学院

专业班级：15052316

姓名：鞠畅

学号:15051407

# 实验4 Linux进程管理

## 1 实验要求

（1） 熟悉 linux 常用命令： pwd， useradd， passwd, who, ps, pstree, kill, top, ls, cd, mkdir,rmdir, cp, rm, mv, cat, more, grep 等。

（2） 实现一个模拟的 shell

（3） 实现一个管道通信程序

（4） 利用 linux 的消息队列通信机制实现两个线程间的通信

（5） 利用 linux 的共享内存通信机制实现两个进程间的通信：

## 2 核心思路及代码

（1）模拟shell

Main()函数

1.获得当前进程的PID号并打印

2.循环请求输入命令，若命令为exit，则退出，若不是则执行myshell函数

Myshell(cmd)

1.创建子进程，若失败则报错并退出

2.若成功则判断该命令是否存在，若不存在则返回报错

2.若存在则调用execl函数执行命令

void myshell(char\* cmdstr)

{

pid\_t pid=fork();

if(pid<0)

{

printf("fork error\n");

exit(0);

}

if(pid==0)

{

int i=0;

int execl\_status=-1;

for(;i<CMD\_LEN;i++)

{

if(strcmp(cmdstr,cmd[i])==0)

{

printf("child pid:%d is running\n",getpid());

execl\_status=execl(path[i],cmd[i],NULL);

exit(0);

break;

}

}

if(execl\_status<0)

{

printf("command not found\n");

exit(0);

}

}

}

（2）管道通信

1.创建信号量write\_psx ,read\_psx以及管道

2.创建process\_num个子进程

3.如果是子进程（pid=0）则请求write\_psx信号量，关闭pipe的读段，从写段写入数据，完成后释放read\_psx和write\_psx信号量

4.如果是父进程请求read\_psx，关闭写端，从管道中读出子进程发送的消息

代码（子进程和父进程互斥读写）

if (pid == 0) {

P(write\_psx);

close(file\_descriptors[0]);

printf("child proc %d write data\n", sid);

write(file\_descriptors[1], "hey",3);

V(read\_psx);

V(write\_psx);

exit(0);

} else {

// only father could reach here

//sleep(1);

while (1)

{

P(read\_psx);

char buf[1024];

int ret;

close(file\_descriptors[1]);

ret=read(file\_descriptors[0], buf, 1024);

while (ret > 0)

{

printf("father read %d bytes data: %s\n", ret, buf);

ret = read(file\_descriptors[0], buf, 1024);

}

}

}

（3）消息队列

1.创建信号量full, empty ,mutex并初始化为0,2,1。创建消息队列。

2.创建写线程，请求信号量empty，mutex。输入待写入的字符串，如果字符串为exit，则写入end并发送，否则写入字符串并发送，最后释放mutex和full

3.创建读进程，请求full和mutex，接收消息队列中国的信息，如果接收到的信息为end则发送over给sender并退出循环释放empty和mutex，如果不是则打印接收的信息，则释放empty和mutex继续循环。

4.销毁信号量

代码：

void \* ReadProcess(void \*arg)

{

msgbuf msg;

msg.mtype = 1;

while(1)

{

sem\_wait(&full);

sem\_wait(&mutex);

//sleep(5);

msgrcv(msgid,&msg,sizeof(msgbuf),1,0);

if(strcmp(msg.mtext,"end") == 0)

{

msg.mtype = 2;

strncpy(msg.mtext,"over",BUF\_SIZE);

msgsnd(msgid,&msg,sizeof(msgbuf),0);

//printf("%s\n",msg.mtext);

break;

}

printf("receive message:%s\n",msg.mtext);

sem\_post(&empty);

sem\_post(&mutex);

}

sem\_post(&empty);

sem\_post(&mutex);

//exit(EXIT\_SUCCESS);

}

void \* WriteProcess(void \*arg)

{

char input[50];

msgbuf msg;

msg.mtype = 1;

while (1)

{

sem\_wait(&empty);

sem\_wait(&mutex);

scanf("%s",input);

if(strcmp(input,"exit") == 0)

{

strncpy(msg.mtext,"end",BUF\_SIZE);

msgsnd(msgid,&msg,sizeof(msgbuf),0);

sem\_post(&full);

sem\_post(&mutex);

break;

}

strncpy(msg.mtext,input,BUF\_SIZE);

msgsnd(msgid,&msg,sizeof(msgbuf),0);

sem\_post(&mutex);

sem\_post(&full);

}

// memset(&msg,'\0',sizeof(msgbuf));

//msgrcv(msgid,&msg,sizeof(msgbuf),2,0);

printf("sender\_rec:%s\n",msg.mtext );

msgctl (msgid,IPC\_RMID,0);

exit(EXIT\_SUCCESS);

}

（4）共享内存

1.创建共享内存，声明信号量full , empty, mutex并初始化为0,1,1

2.创建sender线程，请求empty，mutex，输入待发送的字符串，写入共享内存中并释放mutex，empty。如果字符串为exit则释放后退出循环，请求empty，等读进程读完后。

3.创建receiver进程，请求full，mutex，将共享内存中的消息读出，如果消息是exit，则向共享内存中写入over并退出循环，释放mutex，empty。否则打印消息，释放mutex，empty。

4.销毁信号量

代码：

void \* SenderProcess()

{

while(1)

{

P(empty);

P(mutex);

scanf("%s",input);

if(strcmp(input,"exit") == 0)

{

strcpy(shmptr,input);

V(mutex);

V(full);

break;

}

strcpy(shmptr,input);

V(mutex);

V(full);

}

P(empty);

SemDestroy();

printf("sender end\n");

exit(0);

}

void \* ReceiverProcess()

{

while(1)

{

P(full);

P(mutex);

sleep(2);

strcpy(result,shmptr);

if(strcmp(result,"exit")==0)

{

strcpy(shmptr,"over");

break;

}

else

printf("Receiver:%s\n",result);

V(mutex);

V(empty);

}

V(mutex);

V(empty);

}

## 3 心得体会

这次实验的收获挺大的，把课堂上学过的很多知识都综合在了一起，比如进程与线程，进程间的互斥，信号量，Linux的三种通信机制等。实验过程中遇到的最大困难是读写实现异步，因为不太熟悉关于信号量的内核函数，以为程序结束时信号量就会自动销毁，导致之前的运行结果影响到了后面的测试。还有就是关于键盘输入的问题，尝试了scanf，gets等多个方法后终于写入成功，但输出时有时会有一些乱码，这个问题一直都没找到原因。