**操作系统课程设计实验报告**

实验题目： 实验三进程通信

姓 名： 王一凡

学 号： 19061536

专 业： 软件工程

班 级： 19052713

老师姓名： 刘真

日 期： 2021 年 6 月 10 日

目 录

[一 题目介绍 1](#_Toc515881940)

[二 实验思路 1](#_Toc515881941)

[三 遇到问题及解决方法 1](#_Toc515881942)

[四 核心代码及实验结果展示 1](#_Toc515881943)

[五 个人实验改进与总结 1](#_Toc515881944)

[5.1 个人实验改进 1](#_Toc515881945)

[5.2 个人实验总结 1](#_Toc515881946)

[六 参考文献 2](#_Toc515881947)

八 源代码详单 2

# 一 题目介绍

实验一：实现一个管道通信程序。由父进程创建一个管道，然后再创建三个子进程，并由这三个子进程利用管道与父进程之间进行通信：子进程发送消息，父进程等待三个子进程全部发完消息后再接收信息。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，测试管道的默认大小，并且要求利用Posix信号量机制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。

实验二：利用linux的消息队列通信机制实现两个线程间的通信。编写程序创建三个线程：sender1线程、sender2线程和receive线程，三个线程的功能描述如下：①sender1线程：运行函数sender1()，它创建一个消息队列，然后等待用户通过终端输入一串字符，并将这串字符通过消息队列发送给receiver线程；可循环发送多个消息，直到用户输入“exit”为止，表示它不再发送消息，最后向receiver线程发送消息“end1”，并且等待receiver的应答，等到应答消息后，将接收到的应答信息显示在终端屏幕上，结束线程的运行。②sender2线程：运行函数sender2()，共享sender1创建的消息队列，等待用户通过终端输入一串字符，并将这串字符通过消息队列发送给receiver线程；可循环发送多个消息，直到用户输入“exit”为止，表示它不再发送消息，最后向receiver线程发送消息“end2”，并且等待receiver的应答，等到应答消息后，将接收到的应答信息显示在终端屏幕上，结束线程的运行。③Receiver线程：运行函数receive()，它通过消息队列接收来自sender1和sender2两个线程的消息，将消息显示在终端屏幕上，当收到内容为“end1”的消息时，就向sender1发送一个应答消息“over1”；当收到内容为“end2”的消息时，就向sender2发送一个应答消息“over2”；消息接收完成后删除消息队列，结束线程的运行。选择合适的信号量机制实现三个线程之间的同步与互斥。

实验三：利用linux的共享内存通信机制实现两个进程间的通信。编写程序sender，它创建一个共享内存，然后等待用户通过终端输入一串字符，并将这串字符通过共享内存发送给receiver；最后，它等待receiver的应答，收到应答消息后，将接收到的应答信息显示在终端屏幕上，删除共享内存，结束程序的运行。编写receiver程序，它通过共享内存接收来自sender的消息，将消息显示在终端屏幕上，然后再通过该共享内存向sender发送一个应答消息“over”（老师告知可以省略这步操作），结束程序的运行。选择合适的信号量机制实现两个进程对共享内存的互斥及同步使用。

# 二 实验思路

实验一：对于阻塞读的模拟，题目要求创建三个子进程。因此我设定在父进程输入要读出的字节数，由三个子进程来写，每写一轮都判断写入的总体的字节数是否满足父进程读出的字节数，若不满足则子进程while（1）继续写，若满足则由父进程终止所有程序的执行。那么如何分辨子进程和父进程呢？我在程序中通过if和创建子进程时返回的pid来区分三个子进程，以保证他们的执行。对于阻塞写的模拟，首先我把整个管道都写满了与此同时还可以测出了管道容量达到了题目的要求。写满之后由另一个进程去读出管道的内容，独出之后前一个进程就可以继续写了因为我设置的时while（1），要达到此效果无名管道很难实现因此我采用有名管道的方式来模拟。

实验二：此实验是用消息队列来实现进程的数据传输。再了解了消息队列的机制后，我才用了一个信号量来保证对消息队列写的互斥操作，因为其他的同步可以使用不同的消息类型来实现。

实验三：共享内存我用了三个信号量，一个是写的信号量，另外两个是接受者和发送者收消息的信号量来完成次实验。

# 三 遇到问题及解决方法

1. 对信号量的使用不够熟练，之前只会理论的信号量知识，对于在程序里实际使用不够熟练，通过查资料的方式掌握信号量的使用。
2. 对消息队列和共享内存的操作相关的函数不熟练，查阅相关的资料，掌握对消息队列和共享内存的创建、读写等操作

# 四 核心代码及实验结果演示

阻塞读核心代码：flag = pipe(fd); //no named pipe

if (flag == -1) //create defeat

{

printf("pipe create error\n");

return 1;

}

pid1 = fork();

if (pid1 > 0)

{

pid2 = fork();

if (pid2 > 0)

{

pid3 = fork();

}

}

if (pid1 < 0 || pid2 < 0 || pid3 < 0)

{

sem\_unlink(Mutex);

sem\_unlink(send1);

sem\_unlink(send2);

sem\_unlink(send3);

sem\_unlink(receive1);

sem\_unlink(receive2);

sem\_unlink(receive3);

printf("fork error\n");

return 2;

}

sleep(1);

if (pid1 == 0)

{

while (1)

{

close(fd[0]);

sem\_wait(send1);

sem\_wait(Mutex);

printf("pid:%d pthread 1 input data:",getpid());

scanf("%[^\n]%\*c",buf);

write(fd[1],buf,strlen(buf));

buflen+=strlen(buf);

printf("total read %d bytes,present son pthread give %d bytes\n",readbuflen,strlen(buf));

sleep(1);

sem\_post(Mutex);

sem\_post(receive1);

sleep(5);

}

}

else if (pid2 == 0)

{

while (1)

{

close(fd[0]);

sem\_wait(send2);

sem\_wait(Mutex);

printf("pid:%d pthread 2 input data:",getpid());

scanf("%[^\n]%\*c",buf);

write(fd[1],buf,strlen(buf));

buflen+=strlen(buf);

printf("total read %d bytes,present son pthread give %d bytes\n",readbuflen,strlen(buf));

sleep(1);

sem\_post(Mutex);

sem\_post(receive2);

sleep(5);

}

}

else if (pid3 == 0)

{

while (1)

{

close(fd[0]);

sem\_wait(send3);

sem\_wait(Mutex);

printf("pid:%d pthread 3 input data:",getpid());

scanf("%[^\n]%\*c",buf);

write(fd[1],buf,strlen(buf));

buflen+=strlen(buf);

printf("total read %d bytes,present son pthread give %d bytes\n",readbuflen,strlen(buf));

sleep(1);

sem\_post(Mutex);

sem\_post(receive3);

sleep(5);

}

}

else

{

while (1)

{

printf("father pthread is reading:\n");

close(fd[1]);

sem\_wait(receive1);

sem\_wait(receive2);

sem\_wait(receive3);

sem\_wait(Mutex);

read(fd[0], buf, 1024);

strcat(readbuf,buf);

if (strlen(readbuf)<readbuflen)

{

printf("read waiting,,the pipe's data is not enough,read blocked.\n");

sem\_post(Mutex);

sem\_post(send1);

sem\_post(send2);

sem\_post(send3);

}

else

{

printf("pid:%d parents pthread get data:",getpid());

for(int i=0;i<=readbuflen;i++)

printf("%c",readbuf[i]);

printf("\n");

sem\_post(Mutex);

sem\_post(send1);

sem\_post(send2);

sem\_post(send3);

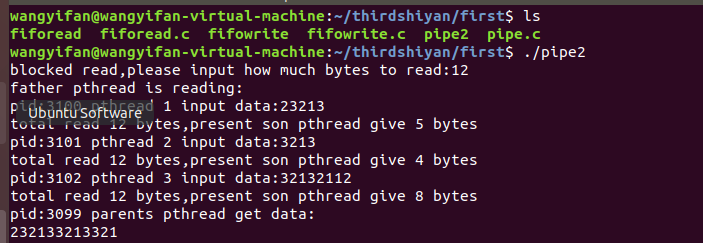
break;

}

}

}

}

****

图表 1阻塞读

阻塞写核心代码： if((mkfifo(FIFO\_NAME,0777)<0)&&(errno!=EEXIST))

{

printf("cannot create fifo...\n");

exit(1);

}

//open pipefifo by only read and block

fd=open(FIFO\_NAME,O\_RDONLY);

if(fd==-1)

{

printf("open %s for read error\n");

exit(1);

}

while(1){

printf("please input how much KB you want to read(0<x<65),0KB represent exit\n");

scanf("%d",&num);

if(num>0 && num<=64)

{

printf("------------will read %d KB---------------------------------\n",num);

while(num--){

r\_num=read(fd,r\_buf,1024);

printf("already read %d bytes, the content read is :%s\n",r\_num,r\_buf);

}

printf("-----------------------read over--------------------------------------\n");

}

else

break;

}

if((mkfifo(FIFO\_NAME,0777)<0)&&(errno!=EEXIST))

{

printf("cannot create fifo...\n");

exit(1);

}

//以阻塞型只写方式打开fifo

fd=open(FIFO\_NAME,O\_WRONLY);

int count = 0;

while (1)

{

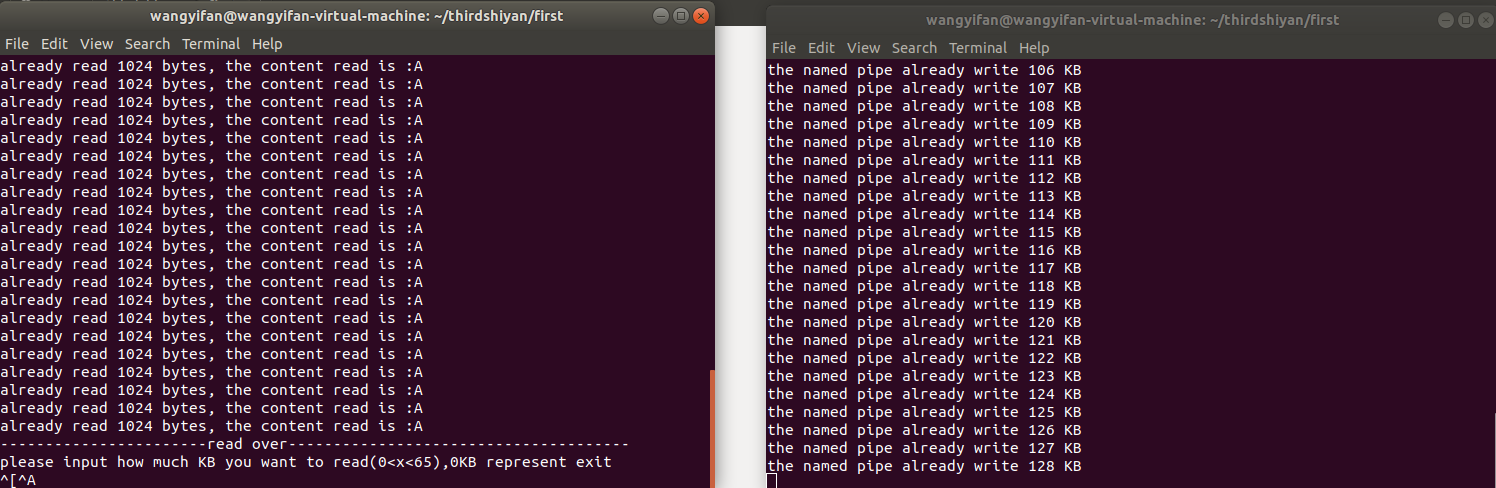
char buf[1024] = "A";

write(fd,buf,sizeof(buf));

count++;

printf("the named pipe already write %d KB\n",count);

}



图表 2阻塞写

消息队列核心代码：

void \*sender1(){

int n;

struct mymsg message;

char buf[MAX];

sem\_t \*mutex = sem\_open("mutex", O\_CREAT | O\_RDWR, 0666, 0);

key\_t key = ftok("tmp",66);

int msqid = msgget(key, 0666 | IPC\_CREAT);

if( msqid == -1){

printf("create failed");

exit(-1);

}

while(1){

sem\_wait(mutex);

printf("sender1 wirte :");

scanf("%s", &buf);

printf("\n");

message.mtype = 1;

if(strcmp(buf,"exit") == 0){

strcpy(message.mtext,"end1");

n = msgsnd(msqid, (void \*)&message, 100, 0);

n = msgrcv(msqid, (void \*)&message, 100, 2, 0);

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

sem\_post(mutex);

sleep(1);

return 0;

}

else{

strcpy(message.mtext,buf);

n = msgsnd(msqid, (void \*)&message, 100, 0);

sem\_post(mutex);

sleep(1);

}

}

}

void \*sender2(){

int n;

struct mymsg message;

char buf[MSG\_MAX];

sem\_t \*mutex = sem\_open("mutex", O\_CREAT | O\_RDWR, 0666, 0);

sem\_t \*sender2\_over = sem\_open("sender2\_over", O\_CREAT | O\_RDWR, 0666, 0);

sem\_t \*receive2\_over = sem\_open("receive2\_over", O\_CREAT | O\_RDWR, 0666, 0);

int msqid = msgget((key\_t)8088, 0666 | IPC\_CREAT);

if( msqid == -1){

printf("create failed");

exit(-1);

}

while(1){

sem\_wait(mutex);

printf("sender2 wirte :");

scanf("%s", &buf);

printf("\n");

message.mtype = 1;

if(strcmp(buf,"exit") == 0){

strcpy(message.mtext,"end2");

n = msgsnd(msqid, (void \*)&message, 100, 0);

n = msgrcv(msqid, (void \*)&message, 100, 3, 0);

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

sem\_post(mutex);

sleep(1);

return 0;

}

else{

strcpy(message.mtext,buf);

n = msgsnd(msqid, (void \*)&message, 100, 0);

sem\_post(mutex);

sleep(1);

}

}

}

void \*receive(){

int n;

int over1=0;

int over2=0;

struct mymsg message;

char buf[MAX];

key\_t key = ftok("tmp",66);

int msqid = msgget(key, 0666 | IPC\_CREAT);

if( msqid == -1){

printf("create failed");

exit(-1);

}

sleep(5);

while(1){

n = msgrcv(msqid, (void \*)&message, 100, 0, 0);

if(n > 0){

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

if( strcmp(message.mtext,"end1") == 0 ){

message.mtype = 2;

strcpy(message.mtext,"over1");

n = msgsnd(msqid, (void \*)&message, 100, 0);

sleep(3);

over1 = 1;

}else if( strcmp(message.mtext,"end2") == 0 ){

message.mtype = 3;

strcpy(message.mtext,"over2");

n = msgsnd(msqid, (void \*)&message, 100, 0);

sleep(3);

over2 = 1;

}

}

if(over1==1 && over2==1){

msgctl(msqid, IPC\_RMID, 0);

exit(0);

}

sleep(1);

}

}

int main(){

key\_t key = ftok("tmp",66);

int msqid = msgget(key, 0666 | IPC\_CREAT);

msgctl(msqid, IPC\_RMID, 0);

sem\_unlink("mutex");

sem\_t \*mutex = sem\_open("mutex", O\_CREAT | O\_RDWR, 0666, 0);

pthread\_t pt1,pt2,pt3;

pthread\_create(&pt1, NULL, sender1, NULL);

pthread\_create(&pt2, NULL, sender2, NULL);

pthread\_create(&pt3, NULL, receive, NULL);

sem\_post(mutex);

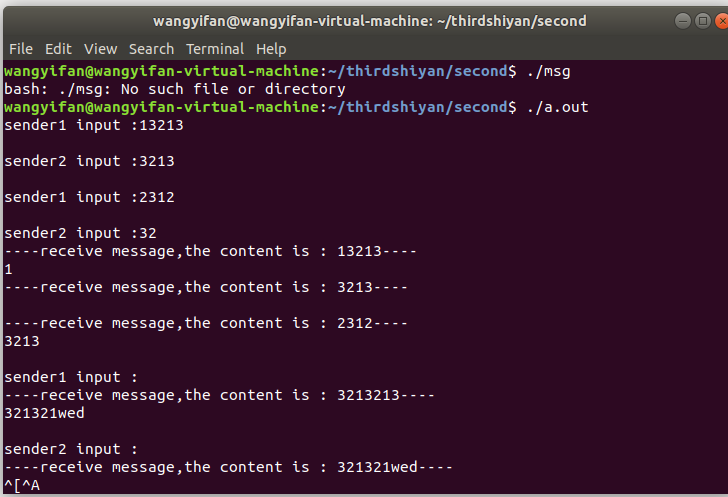
pthread\_join(pt1, NULL);

pthread\_join(pt2, NULL);

pthread\_join(pt3, NULL);

return 0;

}



图表 3消息队列

共享内存核心代码：#include "common.h"

pthread\_t r\_thread, s\_thread;

void \*send(void \*arg)

{

char temp[100], s\_str[100];

while(1)

{

printf("receiver send message:");

fflush(stdout);

scanf("%[^\n]%\*c",s\_str);

sem\_wait(sem\_sender);

sprintf(temp,"%d:%s",\*((pid\_t \*)arg),s\_str);

strcpy((char \*)shmp, temp);

sem\_post(sem\_receiver1);

printf("OK\n");

if(!strcmp(s\_str,"over"))

{

pthread\_cancel(r\_thread);

shmdt(shmp);

break;

}

}

}

void \*receive(void \*arg)

{

char r\_str[100], \*p;

while(1)

{

sem\_wait(sem\_receiver2);

strcpy(r\_str, (char \*)shmp);

p = strchr(r\_str,':');

\*(p++) = '\0';

printf("\rget message from %s:%s\n", r\_str,p);

if(strcmp(p, "over") == 0)

{

pthread\_cancel(s\_thread);

shmdt(shmp);

shmctl(shmid, IPC\_RMID, NULL);

sem\_unlink("sender");

sem\_unlink("receiver1");

sem\_unlink("receiver2");

break;

}

printf("receiver send message:");

fflush(stdout);

sem\_post(sem\_sender);

}

}

int main()

{

pid\_t pid = getpid();

int p1 = 0,p2 = 0;

init\_signal();

printf("pthread %d start!\n",pid);

p1 = pthread\_create(&s\_thread,NULL,send,&pid);

p2 = pthread\_create(&r\_thread,NULL,receive,&pid);

if(p1 || p2)

{

printf("create pthread failed\n");

return 1;

}

pthread\_join(s\_thread, NULL);

pthread\_join(r\_thread, NULL);

printf("pthread %d game over!\n",pid);

}

#include "common.h"

pthread\_t r\_thread, s\_thread;

void \*send(void \*arg)

{

char temp[100], s\_str[100];

while(1)

{

printf("sender send message:");

fflush(stdout);

scanf("%[^\n]%\*c",s\_str);

sem\_wait(sem\_sender);

sprintf(temp,"%d:%s",\*((pid\_t \*)arg),s\_str);

strcpy((char \*)shmp, temp);

sem\_post(sem\_receiver2);

printf("OK\n");

if(!strcmp(s\_str,"over"))

{

pthread\_cancel(r\_thread);

shmdt(shmp);

break;

}

}

}

void \*receive(void \*arg)

{

char r\_str[100], \*p;

while(1)

{

sem\_wait(sem\_receiver1);

strcpy(r\_str, (char \*)shmp);

p = strchr(r\_str,':');

\*(p++) = '\0';

printf("\rget message from %s：%s\n", r\_str,p);

if(strcmp(p, "over") == 0)

{

pthread\_cancel(s\_thread);

shmdt(shmp);

shmctl(shmid, IPC\_RMID, NULL);

sem\_unlink("sender");

sem\_unlink("receiver1");

sem\_unlink("receiver2");

break;

}

printf("sender send message:");

fflush(stdout);

sem\_post(sem\_sender);

}

}

int main()

{

pid\_t pid = getpid();

int p1 = 0,p2 = 0;

sem\_unlink("sender");

sem\_unlink("receiver1");

sem\_unlink("receiver2");

init\_signal();

printf("pthread %d start\n",pid);

p1 = pthread\_create(&s\_thread,NULL,send,&pid);

p2 = pthread\_create(&r\_thread,NULL,receive,&pid);

if(p1 || p2)

{

printf("create pthread failed\n");

return 1;

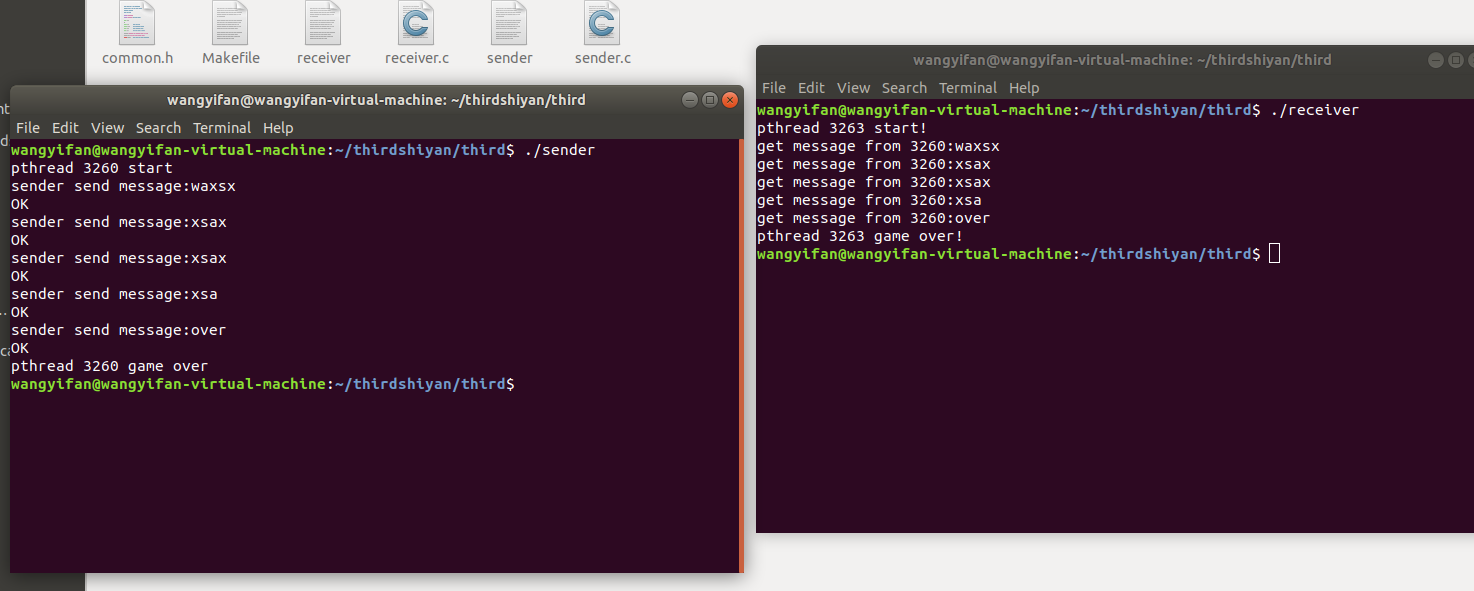
}

pthread\_join(s\_thread, NULL);

pthread\_join(r\_thread, NULL);

printf("pthread %d game over\n",pid);

}



图表 4共享内存

# 五 个人实验改进与总结

## 5.1 个人实验改进

首先我对阻塞写的模拟进行了改进，我设置了写进程while（1）一直在写，另外一边还有一个读进程接收用户输入的数据，可以自己设定读出多少kb，这边读完之后，写进程立马就继续往管道里写，并打印出来。在消息队列中，我灵活的运用了消息队列的机制，只设置了一个信号量，通过不同的消息类型来完成同步与互斥的部分操作。如果不设置不同的消息类型或者加信号量的话发送着发送的消息很快的就会被自己接收，针对这个问题我设置了三种不同的消息，避免此类情况的发生。

## 5.2 个人实验总结

通过本次实验的练习，我知道了Linux管道通信机制、消息队列通信机制、共享内存通信机制的应用，加深了对不同类型的进程通信方式的理解。对Linux的posix信号量和ipc信号量的应用也更加的熟练。本次实验历经曲折，从刚开始的无从下手到最后的成过那么多代码很有成就感，令我印象最深刻的是学习消息队列和共享内存的操作的时候，可能因为这部分比较简单，当时在网上找到对他们操作的资料之后，比如创建、映射、发送、接收这些函数，拿过来就能直接用，自己也能很快的看懂就很有成就感。

# 六 参考文献

1. 操作系统课本
2. <https://www.kernel.org/>

# 七 源代码详单

Pipe.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <semaphore.h>

#include <signal.h>

#include <fcntl.h>

#include <wait.h>

int main()

{

sem\_t \*Mutex;

sem\_t \*send1, \*send2, \*send3;

sem\_t \*receive1, \*receive2, \*receive3;

pid\_t pid1, pid2, pid3;

char readbuf[1024];

int readbuflen;

int buflen = 0;

int fd[2], flag; //fd[0]-->read,fd[1]-->write

char buf[1024];

printf("blocked read,please input how much bytes to read:");

scanf("%d", &readbuflen);

getchar();

memset(buf, 0, sizeof(buf));

//delete before fuzhi

sem\_unlink("Mutex");

sem\_unlink("send1");

sem\_unlink("send2");

sem\_unlink("send3");

sem\_unlink("receive1");

sem\_unlink("receive2");

sem\_unlink("receive3");

//0666-->read and write

Mutex = sem\_open("Mutex", O\_CREAT, 0666, 1);

send1 = sem\_open("send1", O\_CREAT, 0666, 1);

send2 = sem\_open("send2", O\_CREAT, 0666, 1);

send3 = sem\_open("send3", O\_CREAT, 0666, 1);

receive1 = sem\_open("receive1", O\_CREAT, 0666, 0);

receive2 = sem\_open("receive2", O\_CREAT, 0666, 0);

receive3 = sem\_open("receive3", O\_CREAT, 0666, 0);

flag = pipe(fd); //no named pipe

if (flag == -1) //create defeat

{

printf("pipe create error\n");

return 1;

}

pid1 = fork();

if (pid1 > 0)

{

pid2 = fork();

if (pid2 > 0)

{

pid3 = fork();

}

}

if (pid1 < 0 || pid2 < 0 || pid3 < 0)

{

sem\_unlink(Mutex);

sem\_unlink(send1);

sem\_unlink(send2);

sem\_unlink(send3);

sem\_unlink(receive1);

sem\_unlink(receive2);

sem\_unlink(receive3);

printf("fork error\n");

return 2;

}

sleep(1);

if (pid1 == 0)

{

while (1)

{

close(fd[0]);

sem\_wait(send1);

sem\_wait(Mutex);

printf("pid:%d pthread 1 input data:",getpid());

scanf("%[^\n]%\*c",buf);

write(fd[1],buf,strlen(buf));

buflen+=strlen(buf);

printf("total read %d bytes,present son pthread give %d bytes\n",readbuflen,strlen(buf));

sleep(1);

sem\_post(Mutex);

sem\_post(receive1);

sleep(5);

}

}

else if (pid2 == 0)

{

while (1)

{

close(fd[0]);

sem\_wait(send2);

sem\_wait(Mutex);

printf("pid:%d pthread 2 input data:",getpid());

scanf("%[^\n]%\*c",buf);

write(fd[1],buf,strlen(buf));

buflen+=strlen(buf);

printf("total read %d bytes,present son pthread give %d bytes\n",readbuflen,strlen(buf));

sleep(1);

sem\_post(Mutex);

sem\_post(receive2);

sleep(5);

}

}

else if (pid3 == 0)

{

while (1)

{

close(fd[0]);

sem\_wait(send3);

sem\_wait(Mutex);

printf("pid:%d pthread 3 input data:",getpid());

scanf("%[^\n]%\*c",buf);

write(fd[1],buf,strlen(buf));

buflen+=strlen(buf);

printf("total read %d bytes,present son pthread give %d bytes\n",readbuflen,strlen(buf));

sleep(1);

sem\_post(Mutex);

sem\_post(receive3);

sleep(5);

}

}

else

{

while (1)

{

printf("father pthread is reading:\n");

close(fd[1]);

sem\_wait(receive1);

sem\_wait(receive2);

sem\_wait(receive3);

sem\_wait(Mutex);

read(fd[0], buf, 1024);

strcat(readbuf,buf);

if (strlen(readbuf)<readbuflen)

{

printf("read waiting,,the pipe's data is not enough,read blocked.\n");

sem\_post(Mutex);

sem\_post(send1);

sem\_post(send2);

sem\_post(send3);

}

else

{

printf("pid:%d parents pthread get data:",getpid());

for(int i=0;i<=readbuflen;i++)

printf("%c",readbuf[i]);

printf("\n");

sem\_post(Mutex);

sem\_post(send1);

sem\_post(send2);

sem\_post(send3);

break;

}

}

}

}

Fiforead.c

#include <sys/types.h>

#include <sys/stat.h>

#include <errno.h>

#include <fcntl.h>

#include <string.h>

#define FIFO\_NAME "/tmp/testfifo"

main()

{

char r\_buf[1024];

int fd;

int r\_num;

int num=0;

// create or use pipe

if((mkfifo(FIFO\_NAME,0777)<0)&&(errno!=EEXIST))

{

printf("cannot create fifo...\n");

exit(1);

}

//open pipefifo by only read and block

fd=open(FIFO\_NAME,O\_RDONLY);

if(fd==-1)

{

printf("open %s for read error\n");

exit(1);

}

while(1){

printf("please input how much KB you want to read(0<x<65),0KB represent exit\n");

scanf("%d",&num);

if(num>0 && num<=64)

{

printf("------------will read %d KB---------------------------------\n",num);

while(num--){

r\_num=read(fd,r\_buf,1024);

printf("already read %d bytes, the content read is :%s\n",r\_num,r\_buf);

}

printf("-----------------------read over--------------------------------------\n");

}

else

break;

}

}

Fifowrite.

#include <sys/types.h>

#include <sys/stat.h>

#include <errno.h>

#include <fcntl.h>

#include <string.h>

#include <stdio.h>

#define FIFO\_NAME "/tmp/testfifo"

main()

{

int fd;

if((mkfifo(FIFO\_NAME,0777)<0)&&(errno!=EEXIST))

{

printf("cannot create fifo...\n");

exit(1);

}

//以阻塞型只写方式打开fifo

fd=open(FIFO\_NAME,O\_WRONLY);

int count = 0;

while (1)

{

char buf[1024] = "A";

write(fd,buf,sizeof(buf));

count++;

printf("the named pipe already write %d KB\n",count);

}

}

Msg.c

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<pthread.h>

#include<semaphore.h>

#include<sys/types.h>

#include<sys/ipc.h>

#include<sys/msg.h>

#include<sys/sem.h>

#include<sys/shm.h>

#include <fcntl.h>

#include<sys/stat.h>

#include <unistd.h>

#include <signal.h>

#define MAX 100

struct mymsg{

long int mtype;

char mtext[MAX];

};

void \*sender1(){

int n;

struct mymsg message;

char buf[MAX];

sem\_t \*mutex = sem\_open("mutex", O\_CREAT | O\_RDWR, 0666, 0);

key\_t key = ftok("tmp",66);

int msqid = msgget(key, 0666 | IPC\_CREAT);

if( msqid == -1){

printf("create failed");

exit(-1);

}

while(1){

sem\_wait(mutex);

printf("sender1 wirte :");

scanf("%s", &buf);

printf("\n");

message.mtype = 1;

if(strcmp(buf,"exit") == 0){

strcpy(message.mtext,"end1");

n = msgsnd(msqid, (void \*)&message, 100, 0);

n = msgrcv(msqid, (void \*)&message, 100, 2, 0);

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

sem\_post(mutex);

sleep(1);

return 0;

}

else{

strcpy(message.mtext,buf);

n = msgsnd(msqid, (void \*)&message, 100, 0);

sem\_post(mutex);

sleep(1);

}

}

}

void \*sender2(){

int n;

struct mymsg message;

char buf[MSG\_MAX];

sem\_t \*mutex = sem\_open("mutex", O\_CREAT | O\_RDWR, 0666, 0);

sem\_t \*sender2\_over = sem\_open("sender2\_over", O\_CREAT | O\_RDWR, 0666, 0);

sem\_t \*receive2\_over = sem\_open("receive2\_over", O\_CREAT | O\_RDWR, 0666, 0);

int msqid = msgget((key\_t)8088, 0666 | IPC\_CREAT);

if( msqid == -1){

printf("create failed");

exit(-1);

}

while(1){

sem\_wait(mutex);

printf("sender2 wirte :");

scanf("%s", &buf);

printf("\n");

message.mtype = 1;

if(strcmp(buf,"exit") == 0){

strcpy(message.mtext,"end2");

n = msgsnd(msqid, (void \*)&message, 100, 0);

n = msgrcv(msqid, (void \*)&message, 100, 3, 0);

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

sem\_post(mutex);

sleep(1);

return 0;

}

else{

strcpy(message.mtext,buf);

n = msgsnd(msqid, (void \*)&message, 100, 0);

sem\_post(mutex);

sleep(1);

}

}

}

void \*receive(){

int n;

int over1=0;

int over2=0;

struct mymsg message;

char buf[MAX];

key\_t key = ftok("tmp",66);

int msqid = msgget(key, 0666 | IPC\_CREAT);

if( msqid == -1){

printf("create failed");

exit(-1);

}

sleep(5);

while(1){

n = msgrcv(msqid, (void \*)&message, 100, 0, 0);

if(n > 0){

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

if( strcmp(message.mtext,"end1") == 0 ){

message.mtype = 2;

strcpy(message.mtext,"over1");

n = msgsnd(msqid, (void \*)&message, 100, 0);

sleep(3);

over1 = 1;

}else if( strcmp(message.mtext,"end2") == 0 ){

message.mtype = 3;

strcpy(message.mtext,"over2");

n = msgsnd(msqid, (void \*)&message, 100, 0);

sleep(3);

over2 = 1;

}

}

if(over1==1 && over2==1){

msgctl(msqid, IPC\_RMID, 0);

exit(0);

}

sleep(1);

}

}

int main(){

key\_t key = ftok("tmp",66);

int msqid = msgget(key, 0666 | IPC\_CREAT);

msgctl(msqid, IPC\_RMID, 0);

sem\_unlink("mutex");

sem\_t \*mutex = sem\_open("mutex", O\_CREAT | O\_RDWR, 0666, 0);

pthread\_t pt1,pt2,pt3;

pthread\_create(&pt1, NULL, sender1, NULL);

pthread\_create(&pt2, NULL, sender2, NULL);

pthread\_create(&pt3, NULL, receive, NULL);

sem\_post(mutex);

pthread\_join(pt1, NULL);

pthread\_join(pt2, NULL);

pthread\_join(pt3, NULL);

return 0;

}

Commo.h

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/types.h>

#include <sys/msg.h>

#include <sys/stat.h>

#include <unistd.h>

#include <signal.h>

#include <fcntl.h>

#include <pthread.h>

#include <semaphore.h>

#include <glob.h>

#define SIZE 1024

sem\_t \*sem\_sender;

sem\_t \*sem\_receiver1;

sem\_t \*sem\_receiver2;

int shmid;

void \*shmp;

void init\_signal()

{

sem\_sender = sem\_open("sender", O\_CREAT, 0666, 1);

sem\_receiver1 = sem\_open("receiver1", O\_CREAT, 0666, 0);

sem\_receiver2 = sem\_open("receiver2", O\_CREAT, 0666, 0);

key\_t key = ftok("/tmp",666);

shmid = shmget(key, SIZE, 0666|IPC\_CREAT);

if(shmid < 0)

{

printf("shmget error！\n");

exit(-1);

}

shmp = shmat(shmid, NULL, 0);

if((long long int)shmp == -1)

{

printf("shmp error！\n");

exit(-1);

}

}

Receiver.c

#include "common.h"

pthread\_t r\_thread, s\_thread;

void \*send(void \*arg)

{

char temp[100], s\_str[100];

while(1)

{

printf("receiver send message:");

fflush(stdout);

scanf("%[^\n]%\*c",s\_str);

sem\_wait(sem\_sender);

sprintf(temp,"%d:%s",\*((pid\_t \*)arg),s\_str);

strcpy((char \*)shmp, temp);

sem\_post(sem\_receiver1);

printf("OK\n");

if(!strcmp(s\_str,"over"))

{

pthread\_cancel(r\_thread);

shmdt(shmp);

break;

}

}

}

void \*receive(void \*arg)

{

char r\_str[100], \*p;

while(1)

{

sem\_wait(sem\_receiver2);

strcpy(r\_str, (char \*)shmp);

p = strchr(r\_str,':');

\*(p++) = '\0';

printf("\rget message from %s:%s\n", r\_str,p);

if(strcmp(p, "over") == 0)

{

pthread\_cancel(s\_thread);

shmdt(shmp);

shmctl(shmid, IPC\_RMID, NULL);

sem\_unlink("sender");

sem\_unlink("receiver1");

sem\_unlink("receiver2");

break;

}

printf("receiver send message:");

fflush(stdout);

sem\_post(sem\_sender);

}

}

int main()

{

pid\_t pid = getpid();

int p1 = 0,p2 = 0;

init\_signal();

printf("pthread %d start!\n",pid);

p1 = pthread\_create(&s\_thread,NULL,send,&pid);

p2 = pthread\_create(&r\_thread,NULL,receive,&pid);

if(p1 || p2)

{

printf("create pthread failed\n");

return 1;

}

pthread\_join(s\_thread, NULL);

pthread\_join(r\_thread, NULL);

printf("pthread %d game over!\n",pid);

}

Sender.c

#include "common.h"

pthread\_t r\_thread, s\_thread;

void \*send(void \*arg)

{

char temp[100], s\_str[100];

while(1)

{

printf("sender send message:");

fflush(stdout);

scanf("%[^\n]%\*c",s\_str);

sem\_wait(sem\_sender);

sprintf(temp,"%d:%s",\*((pid\_t \*)arg),s\_str);

strcpy((char \*)shmp, temp);

sem\_post(sem\_receiver2);

printf("OK\n");

if(!strcmp(s\_str,"over"))

{

pthread\_cancel(r\_thread);

shmdt(shmp);

break;

}

}

}

void \*receive(void \*arg)

{

char r\_str[100], \*p;

while(1)

{

sem\_wait(sem\_receiver1);

strcpy(r\_str, (char \*)shmp);

p = strchr(r\_str,':');

\*(p++) = '\0';

printf("\rget message from %s：%s\n", r\_str,p);

if(strcmp(p, "over") == 0)

{

pthread\_cancel(s\_thread);

shmdt(shmp);

shmctl(shmid, IPC\_RMID, NULL);

sem\_unlink("sender");

sem\_unlink("receiver1");

sem\_unlink("receiver2");

break;

}

printf("sender send message:");

fflush(stdout);

sem\_post(sem\_sender);

}

}

int main()

{

pid\_t pid = getpid();

int p1 = 0,p2 = 0;

sem\_unlink("sender");

sem\_unlink("receiver1");

sem\_unlink("receiver2");

init\_signal();

printf("pthread %d start\n",pid);

p1 = pthread\_create(&s\_thread,NULL,send,&pid);

p2 = pthread\_create(&r\_thread,NULL,receive,&pid);

if(p1 || p2)

{

printf("create pthread failed\n");

return 1;

}

pthread\_join(s\_thread, NULL);

pthread\_join(r\_thread, NULL);

printf("pthread %d game over\n",pid);

}