内蒙古农业大学

计算机与信息工程学院

实  验  报  告

实验三

课程名称：

班    级：  21计科3

姓    名：   周烨

学    号：    2021122156404

授课教师：张永安

实验三 线程编程

一、      实验目的

1.     掌握常用函数的使用；

2.     掌握嵌入式Linux多线程编程；

3.     掌握线程的同步与互斥机制。

二、      实验工具和环境

PC机、Linux Ubuntu操作系统。

三、      实验内容·

1、（1）创建线程文件pth.c，在主线程中利用pthread\_create()函数创建一个线程，线程号为id；线程函数名称为thread1()。在主线程中输出5条“This is main thread!”,在线程函数中输出5条“This is pthread1!”。主线程中利用线程阻塞函数pthread\_join()，使线程函数得以执行，并利用pthread\_self()分别输出显示主线程和线程thread1的线程号。该题完成后，将所有文件保存在文件夹expr1中，并保留linux输出结果的截屏。

（2）在上题的pth.c文件的基础上复制并修改程序，练习使用pthread\_cancel()函数取消线程的执行。即在线程程序中增加while(1)，让线程不断输出内容，在主程序中增加pthread\_cancel()，来终止线程的输出。该题完成后，将所有文件保存在文件夹expr2中，并保留linux输出结果的截屏。

（3）在上题的pth.c文件的基础上复制并修改程序，练习使用pthread\_create()函数的参数传递（传递字符型还是数值型自定），利用pthread\_create()函数传递参数到线程程序，并输出所传递参数的内容。该题完成后，将所有文件保存在文件夹expr3中，并保留linux输出结果的截屏。

（4）在上题的pth.c文件的基础上复制并修改程序，实现多线程操作。增加线程函数thread2()，每个线程函数输出5条信息，注意输出时增加sleep(1)的延时。查看输出结果。该题完成后，将所有文件保存在文件夹expr4中，并保留linux输出结果的截屏。

（5）在上题的pth.c文件的基础上复制并修改程序，实现多线程互斥操作。增加互斥锁，让每个得到互斥锁的线程能够锁定临界资源，完成该线程的所有操作。查看输出结果。该题完成后，将所有文件保存在文件夹expr5中，并保留linux输出结果的截屏。

（6）在上题的pth.c文件的基础上复制并修改程序，练习信号量互斥操作。增加两个信号量，完成类似互斥锁的线程互斥，让每个线程能够锁定临界资源，完成该线程的所有操作。查看输出结果。该题完成后，将所有文件保存在文件夹expr6中，并保留linux输出结果的截屏。

（7）在上题的pth.c文件的基础上复制并修改程序，练习信号量同步操作。让线程thread1()和thread2()能够交替执行。查看输出结果。该题完成后，将所有文件保存在文件夹expr7中，并保留linux输出结果的截屏。

（8）在上题的pth.c文件的基础上复制并修改程序，练习单个信号量同时并行多个任务的操作。创建一个信号量和一个线程函数，但该信号量可同时并行运行多个任务（具体任务个数自己决定，一般不小于3个）。在线程函数中输出线程申请资源的信息、申请成功和释放资源的信息，并随机延时一段时间（随机延时2-5秒）。查看输出结果。该题完成后，将所有文件保存在文件夹expr8中，并保留linux输出结果的截屏。

2、利用线程的同步与互斥实现“生产者消费者问题”：

      在主线程中创建生产者与消费者线程，生产者每次生产时锁定临界资源，向资源池添加一个资源（资源可以用一个公有变量代替）；消费者消费每次消费时锁定临界资源，并使资源池的数量减一。注意生产和消费时都需要一定的时间，可以用随机时间，或自定义生产消费时间。

四、      实验结果

将以上综合案例的代码复制到下面，并将执行的输出结果截图放到代码后面。

 1.#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

void\* thread1(){

int i;

printf("this thread id is %lu\n",pthread\_self());

for(i=0;i<5;i++){

printf("this is thread1\n");

}

}

int main(){

int i,ret;

pthread\_t id;

ret=pthread\_create(&id,NULL,thread1,NULL);

if(ret!=0){

printf("fail\n");

}

printf("main thread id is %lu\n",pthread\_self());

pthread\_join(id,NULL);

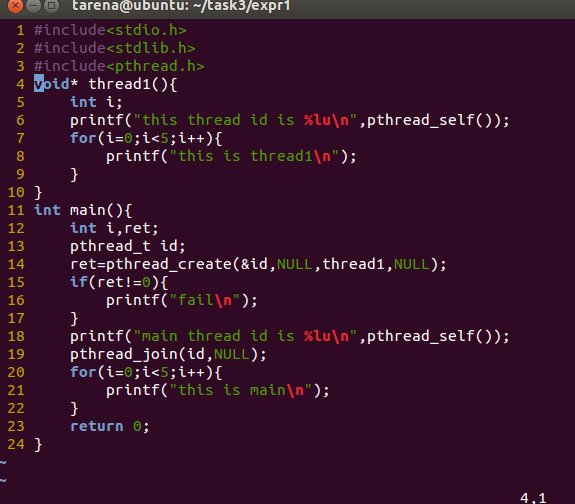
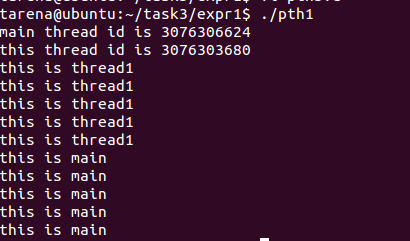
for(i=0;i<5;i++){

printf("this is main\n");

}

return 0;

}

 2.#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

void\* thread1(){

int i;

printf("this thread id is %lu\n",pthread\_self());

while(1){

printf("this is thread1\n");

}

}

int main(){

int i,ret;

pthread\_t id;

ret=pthread\_create(&id,NULL,thread1,NULL);

if(ret!=0){

printf("fail\n");

}

printf("main thread id is %lu\n",pthread\_self());

// pthread\_join(id,NULL);

pthread\_cancel(id);

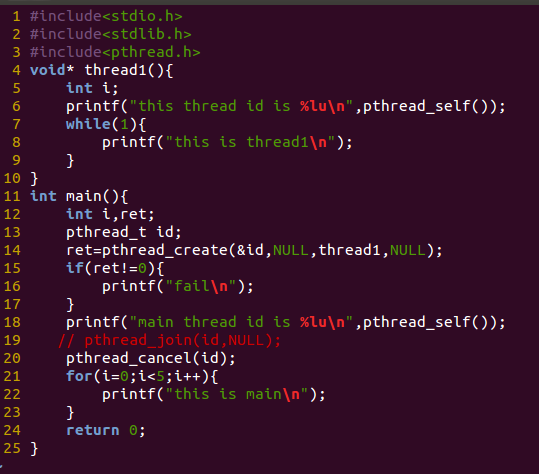
for(i=0;i<5;i++){

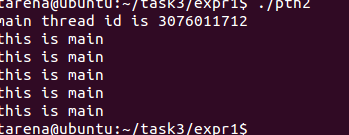
printf("this is main\n");

}

return 0;

}





3.#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

void\* thread1(void \*arg){

int i;

int \*val=arg;

printf("this thread id is %lu\n",pthread\_self());

for(i=0;i<5;i++){

printf("this is thread1 argment is%d\n",\*val);

}

}

int main(){

int i,ret,t\_arg=100;

pthread\_t id;

ret=pthread\_create(&id,NULL,thread1,&t\_arg);

if(ret!=0){

printf("fail\n");

}

printf("main thread id is %lu\n",pthread\_self());

pthread\_join(id,NULL);

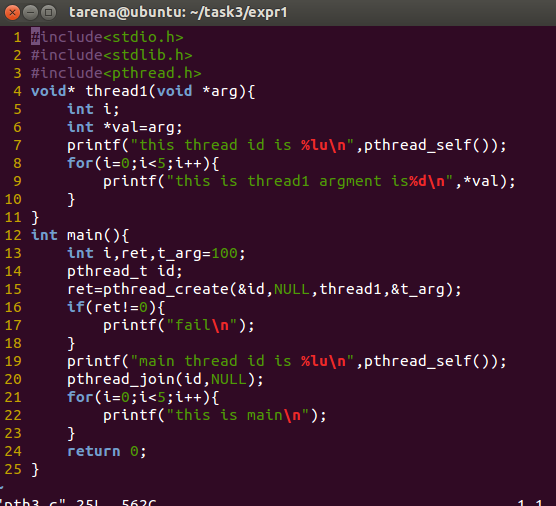
for(i=0;i<5;i++){

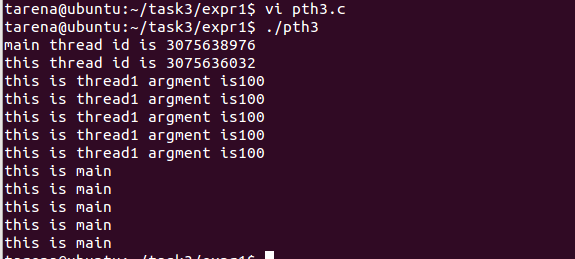
printf("this is main\n");

}

return 0;

}





4.#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

void\* thread1(){

int i;

printf("this thread id is %lu\n",pthread\_self());

for(i=0;i<5;i++){

printf("this is thread1\n");

}

}

void\* thread2(){

int i;

printf("this thread id is %lu\n",pthread\_self());

for(i=0;i<5;i++){

printf("this is thread2\n");

}

sleep(1);

}

int main(){

int i,ret1,ret2;

pthread\_t id1,id2;

ret1=pthread\_create(&id1,NULL,thread1,NULL);

if(ret1!=0){

printf("fail\n");

}

ret2=pthread\_create(&id2,NULL,thread2,NULL);

if(ret2!=0){

printf("fail\n");

}

printf("main thread id is %lu\n",pthread\_self());

// pthread\_join(id1,NULL);

// pthread\_join(id2,NULL);

sleep(3);

for(i=0;i<5;i++){

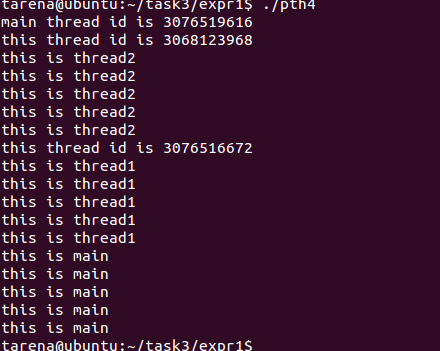
printf("this is main\n");

}

return 0;

}





5.#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

pthread\_mutex\_t mutex;

void\* thread1(){

pthread\_mutex\_lock(&mutex);

int i;

printf("this thread id is %lu\n",pthread\_self());

for(i=0;i<5;i++){

printf("this is thread1\n");

}

pthread\_mutex\_unlock(&mutex);

}

void\* thread2(){

pthread\_mutex\_lock(&mutex);

int i;

printf("this thread id is %lu\n",pthread\_self());

for(i=0;i<5;i++){

printf("this is thread2\n");

}

sleep(1);

pthread\_mutex\_unlock(&mutex);

}

int main(){

pthread\_mutex\_init(&mutex,NULL);

int i,ret1,ret2;

pthread\_t id1,id2;

ret1=pthread\_create(&id1,NULL,thread1,NULL);

if(ret1!=0){

printf("fail\n");

}

ret2=pthread\_create(&id2,NULL,thread2,NULL);

if(ret2!=0){

printf("fail\n");

}

printf("main thread id is %lu\n",pthread\_self());

// pthread\_join(id1,NULL);

// pthread\_join(id2,NULL);

sleep(3);

for(i=0;i<5;i++){

printf("this is main\n");

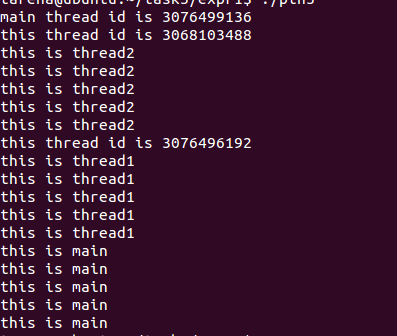
}

pthread\_mutex\_destroy(&mutex);

return 0;

}





6.#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

#include<semaphore.h>

pthread\_mutex\_t mutex;

sem\_t sem;

void\* thread1(){

// pthread\_mutex\_lock(&mutex);

sem\_wait(&sem);

int i;

printf("this thread id is %lu\n",pthread\_self());

for(i=0;i<5;i++){

printf("this is thread1\n");

}

sem\_post(&sem);

// pthread\_mutex\_unlock(&mutex);

}

void\* thread2(){

// pthread\_mutex\_lock(&mutex);

sem\_wait(&sem);

int i;

printf("this thread id is %lu\n",pthread\_self());

for(i=0;i<5;i++){

printf("this is thread2\n");

}

sleep(1);

sem\_post(&sem);

// pthread\_mutex\_unlock(&mutex);

}

int main(){

pthread\_mutex\_init(&mutex,NULL);

sem\_init(&sem,0,1);

int i,ret1,ret2;

pthread\_t id1,id2;

ret1=pthread\_create(&id1,NULL,thread1,NULL);

if(ret1!=0){

printf("fail\n");

}

ret2=pthread\_create(&id2,NULL,thread2,NULL);

if(ret2!=0){

printf("fail\n");

}

printf("main thread id is %lu\n",pthread\_self());

// pthread\_join(id1,NULL);

// pthread\_join(id2,NULL);

sleep(3);

for(i=0;i<5;i++){

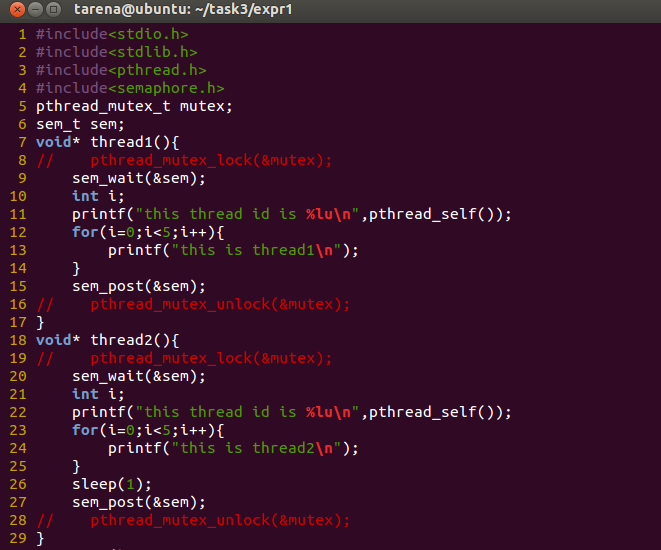
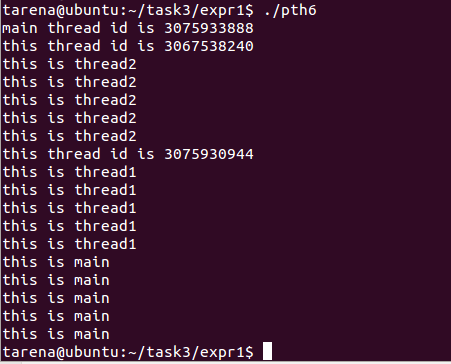
printf("this is main\n");

}

pthread\_mutex\_destroy(&mutex);

return 0;

}



7.#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

pthread\_mutex\_t mutex;

pthread\_cond\_t cond;

void\* thread1(){

pthread\_mutex\_lock(&mutex);

int i;

printf("this thread id is %lu\n",pthread\_self());

for(i=0;i<5;i++){

printf("this is thread1\n");

}

pthread\_cond\_signal(&cond);

pthread\_mutex\_unlock(&mutex);

}

void\* thread2(){

pthread\_mutex\_lock(&mutex);

pthread\_cond\_wait(&cond,&mutex);

int i;

printf("this thread id is %lu\n",pthread\_self());

for(i=0;i<5;i++){

printf("this is thread2\n");

}

sleep(1);

pthread\_mutex\_unlock(&mutex);

}

int main(){

pthread\_mutex\_init(&mutex,NULL);

pthread\_cond\_init(&cond,NULL);

int i,ret1,ret2;

pthread\_t id1,id2;

ret1=pthread\_create(&id1,NULL,thread1,NULL);

if(ret1!=0){

printf("fail\n");

}

ret2=pthread\_create(&id2,NULL,thread2,NULL);

if(ret2!=0){

printf("fail\n");

}

printf("main thread id is %lu\n",pthread\_self());

// pthread\_join(id1,NULL);

// pthread\_join(id2,NULL);

sleep(3);

for(i=0;i<5;i++){

printf("this is main\n");

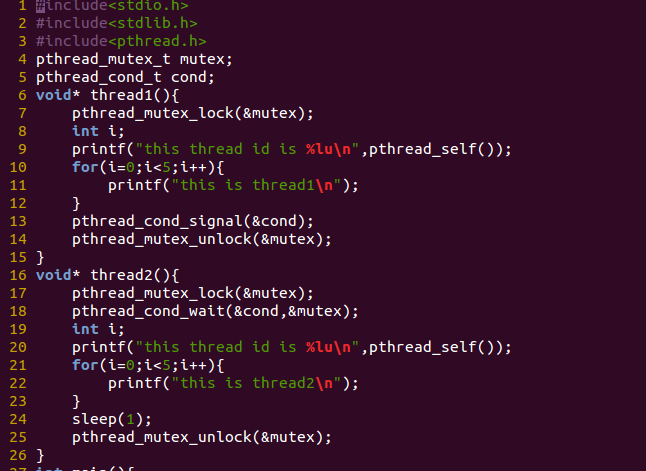
}

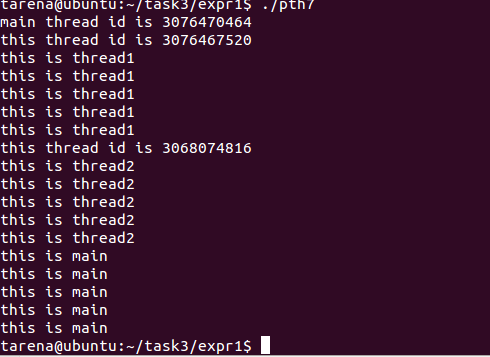
pthread\_cond\_destroy(&cond);

pthread\_mutex\_destroy(&mutex);

return 0;

}





8.

#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

#include<semaphore.h>

pthread\_mutex\_t mutex;

pthread\_cond\_t cond;

sem\_t semaphore;

void\* thread1(){

// pthread\_mutex\_lock(&mutex);

sem\_wait(&semaphore);

int i;

printf("this thread id is %lu\n",pthread\_self());

for(i=0;i<2;i++){

printf("this is thread1\n");

}

// pthread\_cond\_signal(&cond);

sleep(2);

sem\_post(&semaphore);

// pthread\_mutex\_unlock(&mutex);

}

void\* thread2(){

// pthread\_mutex\_lock(&mutex);

// pthread\_cond\_wait(&cond,&mutex);

sem\_wait(&semaphore);

int i;

printf("this thread id is %lu\n",pthread\_self());

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

printf("this is thread2\n");

}

sleep(2);

sem\_post(&semaphore);

// pthread\_mutex\_unlock(&mutex);

}

void\* thread3(){

// pthread\_mutex\_lock(&mutex);

// pthread\_cond\_wait(&cond,&mutex);

sem\_wait(&semaphore);

int i;

printf("this thread id is %lu\n",pthread\_self());

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

printf("this is thread3\n");

}

sleep(2);

sem\_post(&semaphore);

// pthread\_mutex\_unlock(&mutex);

}

int main(){

pthread\_mutex\_init(&mutex,NULL);

// pthread\_cond\_init(&cond,NULL);

sem\_init(&semaphore,0,2);

int ret1,ret2,ret3;

pthread\_t id1,id2,id3;

ret1=pthread\_create(&id1,NULL,thread1,NULL);

if(ret1!=0){

printf("fail\n");

}

ret2=pthread\_create(&id2,NULL,thread2,NULL);

if(ret2!=0){

printf("fail\n");

}

ret3=pthread\_create(&id3,NULL,thread3,NULL);

if(ret3!=0){

printf("fail\n");

}

// printf("main thread id is %lu\n",pthread\_self());

// pthread\_join(id1,NULL);

// pthread\_join(id2,NULL);

// pthread\_join(id3,NULL);

sleep(3);

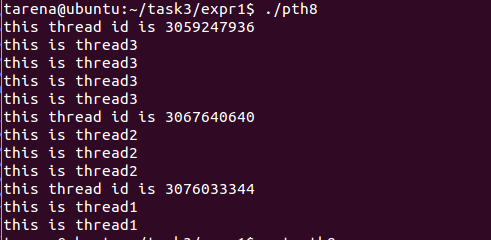
// pthread\_cond\_destroy(&cond);

sem\_destroy(&semaphore);

pthread\_mutex\_destroy(&mutex);

return 0;

}



二，

1 #include<stdio.h>

2 #include<stdlib.h>

3 #include<pthread.h>

4 #include<unistd.h>

5 #include<semaphore.h>

6 int buf[5];

7 int in=0;

8 int out=0;

9 pthread\_mutex\_t mutex;

10 sem\_t full,empty;

11 void \*producer(void \*arg){

12 int item;

13 while(1){

14 item=rand()%100+1;

15 sem\_wait(&empty);

16 pthread\_mutex\_lock(&mutex);

17 printf("producer produced item%d\n",item);

18 buf[in]=item;

19 in=(in+1)%5;

20 pthread\_mutex\_unlock(&mutex);

21 sem\_post(&full);

22 sleep(3);

23 }

24 }

25 void \*consumer(void \*arg){

26 int item;

27

28 while(1){

29

30 sem\_wait(&full);

31 pthread\_mutex\_lock(&mutex);

32

33 item=buf[out];

34 printf("consumer consumed item%d\n",item);

35 out=(out+1)%5;

36 pthread\_mutex\_unlock(&mutex);

37 sem\_post(&empty);

38 sleep(3);

39 }

40 }

41 int main(){

42 pthread\_t pro,con;

43 pthread\_mutex\_init(&mutex,NULL);

44 sem\_init(&full,0,0);

45 sem\_init(&empty,0,5);

46 pthread\_create(&pro,NULL,producer,NULL);

47 pthread\_create(&con,NULL,consumer,NULL);

48

49 pthread\_join(pro,NULL);

50 pthread\_join(con,NULL);

51

52 pthread\_mutex\_destroy(&mutex);

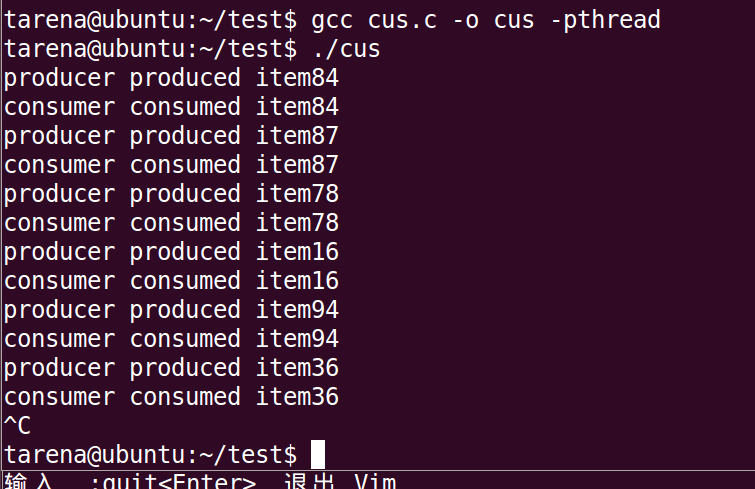
53 sem\_destroy(&full);

54 sem\_destroy(&empty);

55

56 return 0;

57 }



五、      实验总结

简述案例设计思路，并给出程序设计和调试过程中遇到问题的解决思路和方法。

