**实验7 信号量应用——线程实现**

实验目的

1．理解操作系统中信号量同步的原理；

2. 掌握Linux中线程的同步和互斥方法。

实验内容

注意：程序放在自己学号文件夹下；gcc编译程序时，需要在最后加上参数 –pthread 主要头文件 pthread.h 、semaphore.h

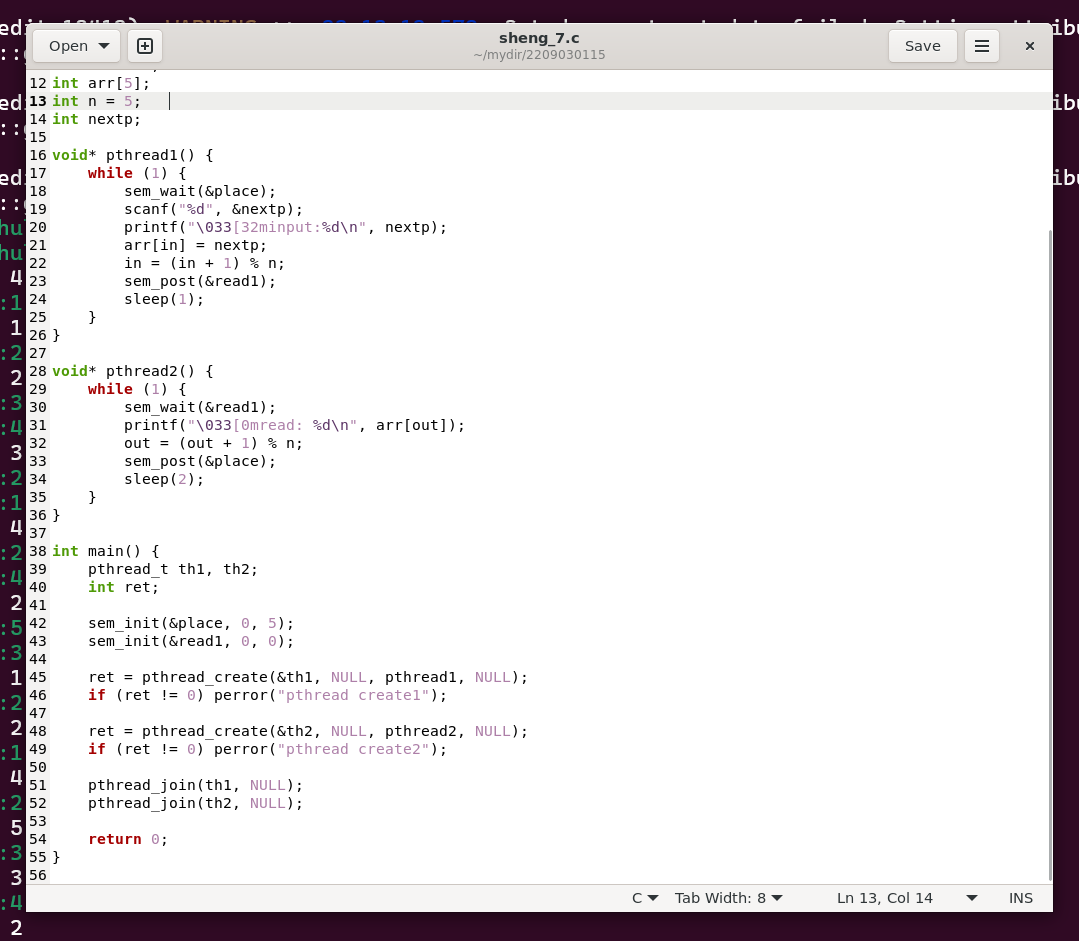
利用Linux中的线程创建和线程信号量同步的方法完成以下程序。

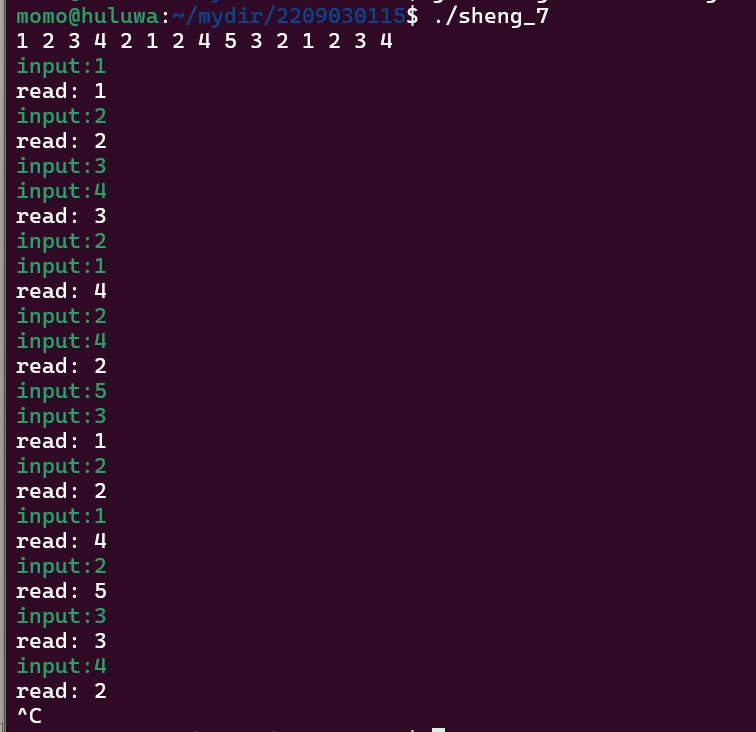
1. 用线程模拟实现生产者和消费者的程序。缓冲区大小为5（可以定义数组来作为存储产品的缓冲区，数组大小为5），生产者线程负责输入15个整数（每次输入一个）放入缓冲区，消费者线程取出生产者输入的整数（每次读出一个，读出就可以，读出后原数的位置可以存入新输入的数），消费者线程全部取出后结束。参考课本4.6.1（P123）的伪代码描述。

要求：1.先写出用wait操作（P操作）和signal操作（V操作）模拟表达。

|  |
| --- |
| semaphore place = 5;  semaphore read1 = 0;  void\* pthread1() {  while (1) {  P(place); // P操作：等待缓冲区有空位  // 输入数据  V(read1); // V操作：通知有数据可读取  sleep(1); // 模拟生产过程  }  } |
| void\* pthread2() {  while (1) {  P(read1); // P操作：等待缓冲区有数据  // 读取数据  V(place); // V操作：通知有空位可写入  sleep(2); // 模拟消费过程  }  } |

2.用Linux线程模拟实现以上程序。要有程序截图和运行结果。注意观察运行结果是否符合生产者消费者问题。





2.(选做，任选其一完成)

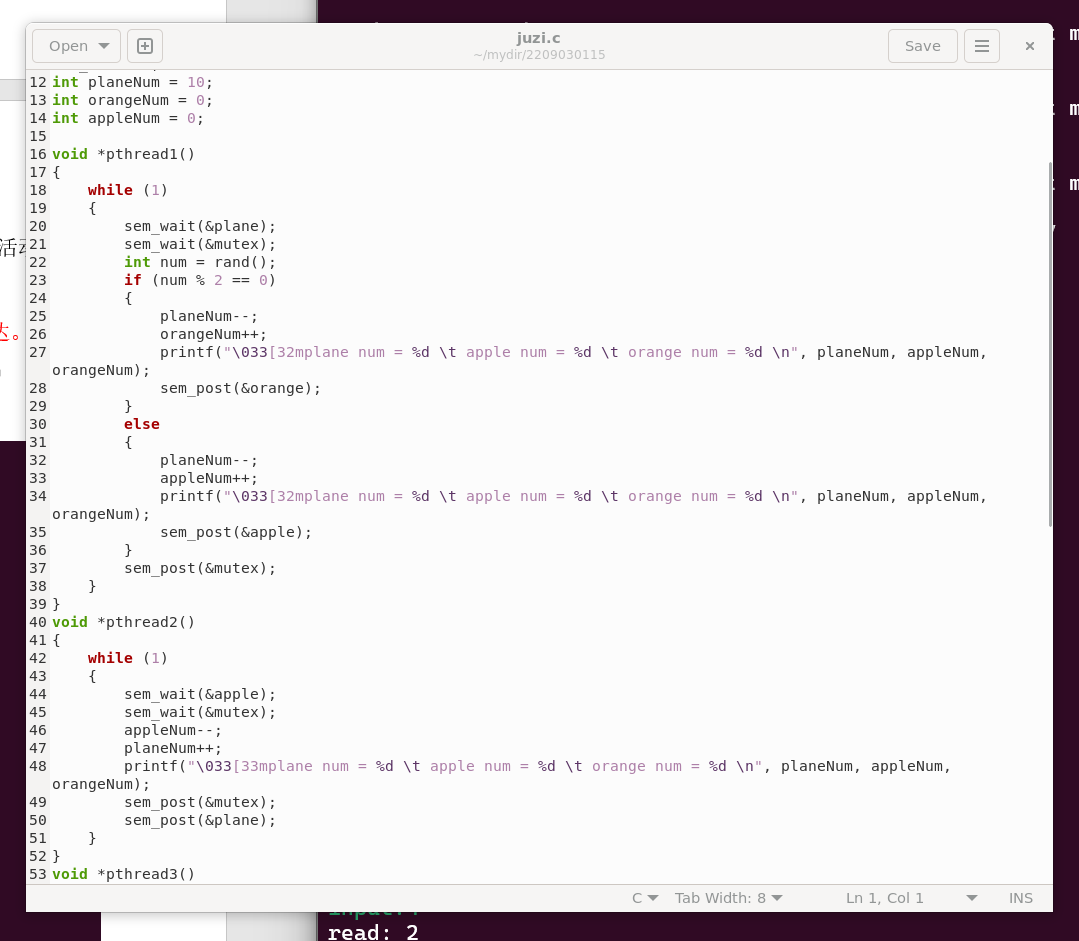
某医院的医生看病叫号的流程如下：候诊室有20个座位，当病人需要看病时先进入候诊室，若候诊室满了则在候诊室外等待；医生呼叫候诊室的病人进行看病，若候诊室没有病人则医生休息，医生看完一个则呼叫下一个病人；医院安排3个医生同时给病人看病。若把患者和医生看病的过程分别看作线程，用信号量机制描述这两个线程的同步与互斥活动，并用Linux线程信号量来模拟实现。

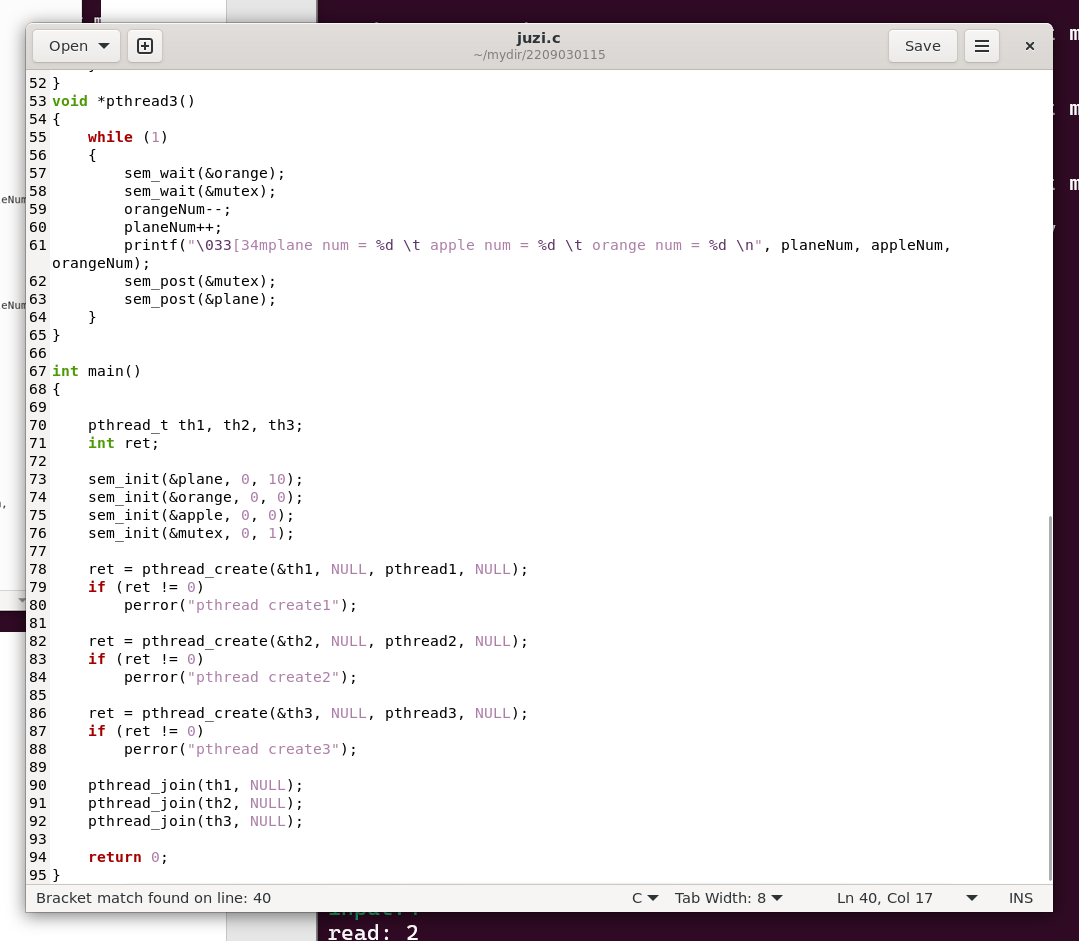
桌上有一个能盛得下10个水果的空盘子。爸爸不停的向盘中放苹果和橘子，儿子不停的从盘中取出橘子享用，女儿不停的从盘中取出苹果享用。规定3人不能同时从（向）盘子中放（取）水果。用信号量机制实现这三个线程的同步与互斥活动，并用线程来实现。

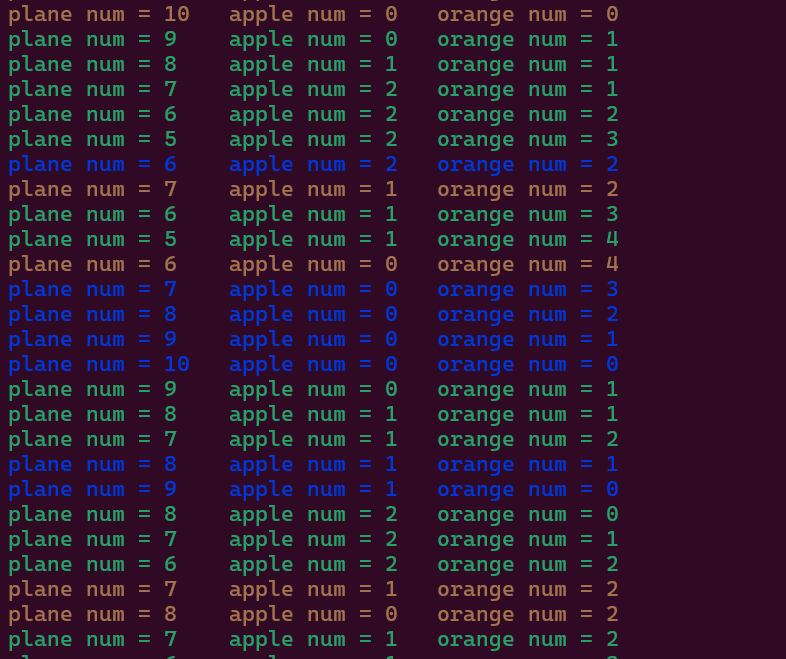
要求：1.先写出用wait操作（P操作）和signal操作（V操作）模拟表达。

|  |
| --- |
| semaphore mutex = 1, empty = 10, apple= 0, orange = 0;  void father  {      do      {          wait(empty);          wait(mutex);          if (放苹果)              signal(apple);          else              signal(orange);          signal(mutex);      } while (1)  }  void daughter  {      do      {          wait(apple);          wait(mutex);          //吃苹果          signal(mutex);          signal(empty);      } while (1)  }  void son  {      do      {          wait(orange);          wait(mutex);          //吃橘子          signal(mutex);          signal(empty);      } while (1)  } |
|  |

2.用Linux线程模拟实现以上程序。要有程序截图和运行结果。







附：教材中的生产者消费者伪代码

int in=0;out=0;

item buffer[n];

semaphore mutex=1,empty=n,full=0;

void producer()

{

do {

生产产品；

wait(empty)；

wait(mutex);

buffer[in]=nextp;

in=(in+1)%n;

signal(mutex);

signal(full);

}

While(true);

}

Void consumer()

{

do{

wait(full);

wait(mutex);

nextc=buffer[out];

out=(out+1)%n;

signal(mutex);

signal(empty);

}

while(true);

}

void main()

{ cobegin

producer(); consumer();

coend

}