1.

In a system, there are multiple producer processes which produce numbers to a buffer and multiple consumer processes which consume numbers from the buffer, where the buffer is shared among all producers and consumers. The following variables are shared among all processes:

int nextc=0, nextp=0, buf[10];  
      semaphore full,empty,mutex;

Producer and consumer processes are given in the following C++-like pseudo programs

 Producer Process:                             Consumer Process:  
int itemp;                                                        int itemc;  
while(1){                                                             while(1){  
1     itemp = rand(); // Generate a number          1     P(full);  
2     P(empty);                                                     2     P(mutex);  
3     P(mutex);                                                     3     itemc=buf[nextc];  
4     buf[nextp]=itemp;                                         4     nextc=(nextc+1)%10;  
5     nextp=(nextp+1)%10;                                  5     V(mutex);  
6     V(mutex);                                                     6     V(empty);  
7     V(full);                                                          7     cout << itemc << endl;  
}                                                                              }

**Note: P()**and **V()**operations are same as **wait()**and **signal()**respectively**.**

**(1)**  What are the critical sections in the given producer and consumer processes? *(4 marks)*

**(2)**  How should the semaphores **full,** **empty**, and **mutex** be initialized? *(3 marks)*

**(3)**  If we switch the order of 2 and 3 in the producer process and the order of 1 and 2 in the consumer process, would the system still work properly? Justify your answer. *(3 marks)*

Producer Process                                       Consumer Process  
…                                                                      …

1 itemp = rand(); // Generate a number        1 P(mutex);  
2 P(mutex);                                                      2 P(full);  
3 P(empty);                                                      3 itemc=buf[nextc];  
…                                                                      …

ANS:

（1）Producer最重要的部分是  itemp = rand();         buf[nextp]=itemp;       nextp=(nextp+1)%10;

         Consumer最重要的部分是         itemc=buf[nextc];          nextc=(nextc+1)%10;

（2）full:   0             empty: 10               mutex:   1

  (3) 系统运作将出现问题

2. 

自旋锁在有进程处于临界区时，采用连续的循环进行忙等待，而在实际的多道程序系统之中，当多个进程共享同一CPU时，这种忙等待浪费了CPU周期， 而被浪费的CPU周期原本可以用于其他进程 ，且需要打破一个进程的自旋锁的状态时，需要通过执行另一个进程来获得打破的条件，在单处理器中无法利用闲置处理器进行另一个进程的运行进行。这种由自旋锁带来的问题可以在多处理器系统中得到缓解，在多处理器系统中，可以允许一个线程在一个处理器上忙等待，而其他线程在其他处理器上进行临界区的执行，之后获得处于忙等待状态的进程的打破条件。

3.理发师问题

The Sleeping-Barber Problem. A barber shop consists of a waiting room with n chairs and a barber room with one barber chair. If there are no customers to be served, the barber goes to sleep. If a customer enters the barber shop and all chairs are occupied, then the customer leaves the shop. If the barber is busy but chairs are available, then the customer sits in one of the free chairs. If the barber is asleep, the customer wakes up the barber. Write a program (pseudo code) to coordinate the barber and the customers.

ANS：

semaphore mutex = 1;//用于保护椅子变量

semaphore customers = 0;

semaphore barbers = 0;

int EmptyChair= N; //所有椅子的数量

理发师：

void Barber()

{

while(true){

wait(customers)；//无顾客理发时，理发师睡觉

wait(mutex); //用于互斥

EmptyChair ++; //空椅子数量增加一个

signal(barbers); //理发师理发

signal（mutex）; ﻿﻿//释放椅子保护

babering; //理发操作

}

}

顾客:

﻿void Customer ()

{

wait(mutex);

if(EmptyChair > 0){

﻿//店内仍有空椅子

EmptyChair --;

signal(customers);//叫醒理发师

signal(mutex);//释放椅子互斥量

wait(barber);//等待正在忙的理发师

PrepareForBarber;

}﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿

else

//店内无空椅子

signal(mutex)﻿; //释放椅子互斥量

//顾客离开理发店

}﻿﻿﻿

﻿﻿﻿﻿﻿﻿﻿

﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿