习题六

班级： 姓名： 学号

1. 什么是临界区？对临界区的访问应该遵循什么准则？

临界区：设计到临界资源的代码段叫临界区。1、临界区是代码片段。2、临界区是进程内的代码。3、每个进程有一个或多个临界区。4、临界区的设置方法由程序员确定。若能保证各个进程互斥进入具有相同临界资源的临界区，可实现对临界资源的互斥访问。

访问准则：

1. 互斥准则：假定进程Pi在其临界区内执行，其他任何进程将被排斥在自己的临界区之外，有相同临界资源的临界区都需互斥；无相同临界资源的临界区不需互斥
2. 有空让进准则：临界区无进程执行，不能无限期地延长下一个要进入临界区进程的等待时间。
3. 有限等待准则：每个进程进入临界区前的等待时间必须有限，不能无限等待。
4. 请谈谈同步信号量的值有什么含义。

同步信号量是一种用于同步的数据类型，通常在多线程环境中使用。信号量具有一个整数值，这个值被称为信号量的计数。

信号量的计数有以下几种含义：

资源计数：信号量的计数可以用来记录可用的资源数量。比如，假设你有一个打印店，你可以设置信号量的计数为打印店中剩余的打印机数量。如果信号量的计数为0，则意味着所有的打印机都被占用了，没有可用的打印机。

线程同步：信号量的计数还可以用来同步线程的执行。比如，假设你有两个线程，你可以设置信号量的计数为1，并让第一个线程先执行。当第一个线程执行完毕后，它会将信号量的计数减1，这样第二个线程就可以执行了。这种方法可以用来保证多个线程按照预定的顺序执行。

互斥：信号量的计数还可以用来实现互斥。比如，假设你有两个线程，你可以设置信号量的计数为1，并让第一个线程先执行。当第一一个线程执行完毕后，它会将信号量的计数加1，这样第二个线程就可以执行了。这种方法可以用来保证在任意时刻，只有一个线程在访问共享资源。

1. 有四个进程S1、R1、R2和R3，其中S1向缓冲区BUFF发送消息，R1、R2和R3从缓冲区中接收消息。发送和接收的规则如下：
2. 缓冲区BUFF任何时候只能存放1个消息；
3. R1、R2和R3每次可取S1存放在缓冲区中的消息；
4. 每个存放在缓冲区中的消息必须被R1、R2和R3均接收后才能清除。

请用信号量机制来实现这4个进程间的同步。

创建两个信号量：

信号量S：记录缓冲区中是否有消息。初始值为0。

信号量N：记录消息是否被R1、R2和R3均接收。初始值为0。

然后，我们可以让这4个进程按照如下方式运行：

S1进程：

S1进程在向缓冲区BUFF发送消息之前，需要先使用wait操作等待信号量S的值为0。这样可以保证缓冲区中没有消息。

当信号量S的值为0时，S1进程向缓冲区发送消息，并使用signal操作将信号量S的值设为1。这样可以告诉R1、R2和R3进程，缓冲区中有消息可以被接收。

R1、R2和R3进程：

R1、R2和R3进程在接收消息之前，需要先使用wait操作等待信号量S的值为1。这样可以保证缓冲区中有消息。

当信号量S的值为1时，R1、R2和R3进程接收消息。然后，使用wait操作等待信号量N的值为2。这样可以保证消息已经被R1和R2接收。

当信号量N的值为2时，R3进程接收消息，并使用signal操作将信号量N的值设为3。这样可以告诉S1进程，消息已经被R1、R2和R3均接收。

最后，R3进程使用signal操作将信号量S的值设为0，并使用signal操作将信号量N的值设为0。这样可以告诉S1进程，缓冲区中的消息已经被接收完毕，可以发送新的消息，并告诉R1、R2和R3进程，缓冲区中的消息已经被清除。

4、下面是用wait/signal机制解决写者优先的Readers-Writers Problem的伪代码。在下面空格处填空，并回答该题目最后面的问题。

所谓写者优先的Readers-Writers Problem定义如下：

int rcount, wcount; (initial value = **0**  )

semaphore m1, m2, m3, w, r ; (initial value = 1)

READER：

  wait(m1); //所有的readers竞争抢w信号量的权利

    wait(r); //所有抢readers竞争w信号量的过程是互斥的

      wait(m2);

        rcount := **rcount+1** ;

        if rcount = 1 then wait( **w** );  //只有第一个reader能够有资格抢w信号量

      ? ;

    signal(r);

  signal( **m1** );

**reading**

   wait(m2);

    rcount := **rcount-1** ;

    if rcount = 0 then signal( **w** );//还有readers在等，就不放w信号量

  signal(m2);

WRITER：

    wait(m3);

      wcount := wcount + 1;

      if wcount = 1 then **wait(m1)** ;

    signal(m3);

  wait(w);

**writing**

  signal( **w** );

  wait(m3);

    wcount := wcount - 1;

    if wcount = 0 then signal( **m1** );

  signal(m3);

在上面这个解决方案中，有饥饿现象发生，例如可以发生在这样的情况下：

**(1)如果连续出现大量的writer进程，那么可能会出现饥饿现象。例如，假设当前有10个writer进程在等待m1信号量，并且一直没有reader进程在运行，那么这10个writer进程就会一直被阻塞，无法继续执行，///////////这就是饥饿现象。**

(2)如果有连续的写者进程在等待，并且没有读者进程在运行，则写者进程可能会一直等待