简答题：

1. 当进程调度是抢占式的时候，Peterson 针对图 2-24 所示的互斥问题的解决方案是否有效？当进程调度是非抢占式的时候又是怎样的？

   答：

   对于抢占式调度并且存在两个不同优先级进程同时访问临界区时，Peterson 算法是不能解决互斥的。原因在下一道题目之中已经给出，此时可能会发生优先级反转的现象，即：当低优先级进程在运行并同时也占有了临界区资源时，高优先级进程抢占 CPU，但临界区资源被低优先级进程占有，所以高优先级进程将会陷入忙等中，但基于抢占式调度，高优先级进程未结束前，低优先级进程永远不会恢复访问并且一直占有临界区资源，变相造成死锁。Peterson 算法对于抢占式调度中用来相同优先级的进程间是可以的。

   对于非抢占式调度，每一个进程在临界区中停留的时间总是有限的，所以我们可以认为任何一个进程都不会进入永久的等待，同时 Peterson 算法符合临界区的设计，能解决互斥问题。

2. 在 2.3.4 节描述了具有高优先级进程 H 和低优先级进程 L 的情况，这种情况导致 H 永远循环。 如果使用时间片轮转调度而不是优先级调度，是否会发生相同的问题？

   答：若采用轮转调度算法而不是优先级调度算法，不会发生这样的问题。因为在轮转调度算法下，H 的时间片总会运行完，总会在一段时间后退出循环，此时轮到进程 L 运行，L 将会离开临界区，则等到再回到 H 进程时，H 则能成功进入临界区。

3. 考虑以下涉及两个进程 P0 和 P1 的互斥问题的解决方案。 假设将变量 turn 初始化为 0。进程 P0 的代码如下所示。 对于进程 P1，在上面的代码中将 0 替换为 1。 确定解决方案是否满足正确的互斥解决方案的所有必需条件。

   答：(1) 无空等待。 当 P0 进程进入临界区时，turn==0，否则 P0 进程会卡在第一个 while 处死循环无法 进入临界区，则此时 turn 始终为 0，P1 始终满足 turn!=1，卡在死循环无法进入临界区； P1 进程先进入同理。因此该算法满足无空等待。

   (2) 有空让进。

   当 P0 进程退出临界区时，turn=0，此时 P1 始终满足 turn!=1，卡在死循环无法进入临界区。因此不满足有空让进。

   (3) 有限等待。

   当不满足有空让进原则时，显然会陷入死循环，不满足有限等待。

应用题：

3. 有两个优先级相同的进程 P1 和 P2，各自执行的操作如下，信号量 S1 和 S2 初值均为 0。试问 P1、P2 并发执行后，x、y、z 的值各为多少？

```
P1( ) {                    P2( ) {
y=1;                        x=1;
y=y+3;                      x=x+5;
V(S1);                      P(S1);
z=y+1;                      x=x+y;
P(S2);                      V(S2);
y=z+y;                      z=z+x;
}                           }
```

答：现对进程语句进行编号，以方便描述。

```
y=1;        ①              x=1;        ⑤
y=y+3;      ②              x=x+5;      ⑥
V(S1);                     P(S1);
z=y+1;      ③              x=x+y;      ⑦
P(S2);                     V(S2);
y=z+y;      ④              z=z+x;      ⑧
```

①、②、⑤和⑥是不相交语句，可以任何次序交错执行，而结果是唯一的。接着无论系统如何调度进程并发执行，当执行到语句⑦时，可以得到 x=10，y=4。按 Bernstein 条件，语句③的执行结果不受语句⑦的影响，故语句③执行后得到 z=5。最后，语句④和⑧并发执行，这时得到了两种结果为：

语句④先执行：x=10，y=9，z=15。

语句⑧先执行：x=10，y=19，z=15。

此外，还有第三种情况，语句③被推迟，直至语句⑧后再执行，于是依次执行以下三个语句：

```
z=z+x;
z=y+1;
y=z+y;
```

这时 z 的值只可能是 y+1=5，故 y=z+y=5+4=9，而 x=10。

第三种情况为：x=10，y=9，z=5。

第四种情况为：先执行语句⑧，再执行语句③，此时 x=10，y=4，而 z 的值是未定义的。

22. 今有 k 个进程，它们的标号依次为 1、2、…、k，如果允许它们同时读文件 file，但必须满足条件：参加同时读文件的进程的标号之和需小于 M（k<M），请使用：1)信号量与 P、V 操作，2)管程，编写出协调多进程读文件的程序。

答 1：1) 使用信号量与 P、V 操作

```
   semaphore waits,mutex;
    int numbersum=0;
    waits=0;mutex=1;
cobegin
     process readeri(int number) {       //i=1,2,…
              P(mutex);
               while(numbersum+number>=M)
                        {V(mutex);P(waits);}
               numbersum=numbersum+number;
              V(mutex);
              Read file;
              P(mutex);
              numbersum=numbersum-number;
              V(waits);
              V(mutex);
     }
coend
```

2) 使用管程：

```
TYPE   sharefile =MONITOR
          int numbersum;
          cond SF;
          numbersum=0;
DEFINE startread,endread;
USE wait,signal;
procedure startread(int number) {
            while(number+numbersum>=M)    {wait(SF);}
            numbersum=numbersum+number;
     }
procedure endread(int number) {
              numbersum=numbersum-number;
              signal(SF);
     }
cobegin
       process-i ( );
coend

process-i ( ) {
      int number;
       number=进程读文件编号;
```

```
        sharefile.startread(number);
            read F;
        sharefile.endread(number);
    }
```

24. 系统有 A、B、C、D 共 4 种资源，在某时刻进程 P0、P1、P2、P3 和 P4 对资源的占有和需求情况如表，试解答下列问题：

Process	Allocation				Claim				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	0	3	2	0	0	4	4	1	6	2	2
P_1	1	0	0	0	2	7	5	2				
P_2	1	3	5	4	3	6	10	10				
P_3	0	3	3	2	0	9	8	4				
P_4	0	0	1	4	0	6	6	10				

1)系统此时处于安全状态吗？

2)若此时 P2 发出 request2(1、2、2、2)，系统能分配资源给它吗？为什么？

答：(1)系统处于安全状态，存在安全序列：P0，P3，P4，P1，P2。

(2)不能分配，否则系统会处于不安全状态。

29. 进程 A1、A2、…、An1 通过 m 个缓冲区向进程 B1、B2、…、Bn2 不断地发送消息。发送和接收工作符合以下规则：

(1) 每个发送进程每次发送一个消息，写进一个缓冲区，缓冲区大小与消息长度相等；

(2) 对每个消息，B1、B2、…、Bn2 都需接收一次，并读入各自的数据区内；

1. 当 M 个缓冲区都满时，则发送进程等待，当没有消息可读时，接收进程等待。

试用信号量和 PV 操作编制正确控制消息的发送和接收的程序。

答：本题是生产者－消费者问题的一个变形，一组生产者 A1，A2，…An1 和一组消费者 B1，B2，…Bn2 共用 m 个缓冲区，每个缓冲区只要写一次，但需要读 n2 次。因此，可以把这一组缓冲区看成 n2 组缓冲区，每个发送者需要同时写 n2 组缓冲区中相应的 n2 个缓冲区，而每一个接收者只需读它自己对应的那组缓冲区中的对应单元。

应设置一个信号量 mutex 实现诸进程对缓冲区的互斥访问；两个信号量数组 empty[n2] 和 full[n2]描述 n2 组缓冲区的使用情况。其同步关系描述如下：

```
semaphore mutex,empty[n2],full[n2];
int i;mutex=1;
for(i=0;i<n2;i++) {
   empty[i]=m; full[i]=0;
   }
   main( ) {
   cobegin
   A1( );
   A2( );
   ┇
  An1 ();
  B1 ();
  B2 ();
   ┇
  Bn2( );
coend
  }

procedure send (  )   {        /*进程 Ai 发送消息*/
  for (int i=0;i<n2;i++)
       p(empty[i]);
       p(mutex);
       /*将消息放入缓冲区*/
       V（mutex）;
  for(int i=0;i<n2;i++)
       V(full[i]);
  }
procedure receive(i)  {  /*进程 Bi 接收消息*/
     p(full[i]);
     p(mutex);
     /*将消息从缓冲区取出*/
     V(mutex);
```

```
        V(emtpy[i]);
}
Ai (    )   {           /*发送进程 A1,A2,…An1 的程序类似，这里给出进程 Ai 的描述*/
while (true) {
     send (    );
        ┇
     }
}
Bi (    )   {        /*接收进程 B1,B2,…Bn2 的程序类似，这里给出进程 Bi 描述*/
while (true)    {
        receive(i);
        ┇
     }
}
```