习题 2

1-7 题，见教材相关章节内容。

8． Job1 与 Job2 同时到达，根据短作业优先算法，这时应该先执行 Job2，Job2 执行完毕时 9.0，系统中有 job1 与 job3 两个作业，job3 较短，所以先执行 job3，job3 执行完毕时 9.7，系统中有 job1 和 job4，job4 较短，先执行 job4，再执行 job1，所以他们四个作业的执行顺序为 Job2、Job3、Job4、Job1；

Job2 的开始时刻是 8.0，完成时刻是 9.0、周转时间 1，带权周转时间 1；

Job3 的开始时刻是 9.0，完成时刻是 9.7、周转时间 1.3，带权周转时间 1.86；

Job4 的开始时刻是 9.7，完成时刻是 10.0，周转时间 0.8，带权周转时间 2.7；

Job1 的开始时刻是 10.0，完成时刻是 12.0，周转时间 4.0，带权周转时间 2；

平均周转时间为 1.78；

平均带权周转时间为 1.89。

9．盘子是个互斥使用的通用类型缓冲区，可存放两种不同类型数据，父母=两个特定类型数据/产品生产者，子女=两个特定类型数据/产品消费者。

```
enum{apple, orange} plate;
semaphore   sCap，sOrg,  sApp;      //盘子里可以放水果数量, 盘子里有无桔子/苹果
sCap = 1;       // 盘子容量：一个水果
sOrg = 0;       // 盘子里没有桔子
sApp = 0;       // 盘子里没有苹果
cobegin
process father {
   while(true){
      削一个苹果；
      P(sCap);
      把苹果放入 plate；
      V(sApp);
   }
}
process mother {
   while(true){
      剥一个桔子；
      P(sCap);
      把桔子放入 plate；
      V(sOrg);
   }
}
process son{
   while(true){
      P(sOrg);
      从 plate 中取桔子；
      V(sCap);
      吃桔子；
   }
}
process daughter {
   while(true){
      P(sApp);
      从 plate 中取苹果；
      V(sCap);
      吃苹果；
   }
}
coend
```

10．

```
semaphore   S=1, SO=0, SS=0, SW=0;      //容器是否可用, 容器中是浓缩汁/糖/水
```

```
enum { sugar, water, orange } container;
cobegin
process Provider {
   while(true){
      P(S);
      将原料装入容器内;
      if (cantainer==orange) V(SO);
      else if (cantainer==sugar) V(SS);
      else V(SW);
   }
}
process P1 {
   while(true){
      P(SO);
      从容器中取浓缩汁;
      V(S);
      生产橙汁;
   }
}
process P2 {
   while(true){
      P(SS);
      从容器中取糖;
      V(S);
      生产橙汁;
   }
}
process P3 {
   while(true){
      P(SW);
      从容器中取水;
      V(S);
      生产橙汁;
   }
}
coend
```

11.

答：系统的剩余资源向量 A=(1,2,3,0)，各进程的剩余请求矩阵

$$R = \begin{pmatrix} 1 & 2 & 0 & 0 \\ 0 & 7 & 5 & 0 \\ 1 & 2 & 1 & 2 \\ 0 & 2 & 2 & 0 \\ 0 & 6 & 2 & 2 \end{pmatrix}$$

（1） 由于系统存在一个进程完成的安全序列 P4、P1、P2、P3、P5，故系统状态安全；

进程	可用资源	请求资源	占有资源	分 配 释 放 量	完成情况
P1		1,2,0,0	0,0,,1,2		
P2		0,7,5,0	1,0,0,0		
P3		1,2,1,2	1,1,4,4		
P4	1,,2,3,0	0,2,2,0	0,6,3,2		1 true
P5		0,6,2,2	0,0,1,4		

进程	可用资源	请求资源	占有资源	分 配 释 放 量	完成情况
P1	1,8,6,2	1,2,0,0	0,0,,1,2	1,8,7,4	2 true
P2		0,7,5,0	1,0,0,0		
P3		1,2,1,2	1,1,4,4		
P4	1,2,3,0	0,2,2,0	0,6,3,2	1,8,6,2	1 true
P5		0,6,2,2	0,0,1,4		

进程	可用资源	请求资源	占有资源	分 配 释 放 量	完成情况
P1	1,8,6,2	1,2,0,0	0,0,,1,2	1,8,7,4	2 true
P2	1,8,7,4	0,7,5,0	1,0,0,0	2,8,7,4	3 true
P3		1,2,1,2	1,1,4,4		
P4	1,2,3,0	0,2,2,0	0,6,3,2	1,8,6,2	1 true
P5		0,6,2,2	0,0,1,4		

进程	可用资源	请求资源	占有资源	分 配 释 放 量	完成情况
P1	1,8,6,2	1,2,0,0	0,0,,1,2	1,8,7,4	2 true
P2	1,8,7,4	0,7,5,0	1,0,0,0	2,8,7,4	3 true
P3	2,8,7,4	1,2,1,2	1,1,4,4	3,9,11,8	4 true
P4	1,2,3,0	0,2,2,0	0,6,3,2	1,8,6,2	1 true
P5		0,6,2,2	0,0,1,4		

进程	可用资源	请求资源	占有资源	分 配 释 放 量	完成情况
P1	1,8,6,2	1,2,0,0	0,0,,1,2	1,8,7,4	2 true
P2	1,8,7,4	0,7,5,0	1,0,0,0	2,8,7,4	3 true
P3	2,8,7,4	1,2,1,2	1,1,4,4	3,9,11,8	4 true
P4	1,2,3,0	0,2,2,0	0,6,3,2	1,8,6,2	1 true
P5	3,9,11,8	0,6,2,2	0,0,1,4	3,9,12,12	5 true

（2） 进程 P3 提出对资源 R3 的剩余请求为 1，由于系统剩余资源向量 A=(1, 2, 2, 0)，所以假定分配给它，此时仍能找到一个与（1）相同的安全序列，即可以分配；

（3） 系统初始配置的各类资源分别为（3，9，12，12）。

补充（二）

一、分析 P、V 操作执行期间为什么要屏蔽中断？试举例说明。

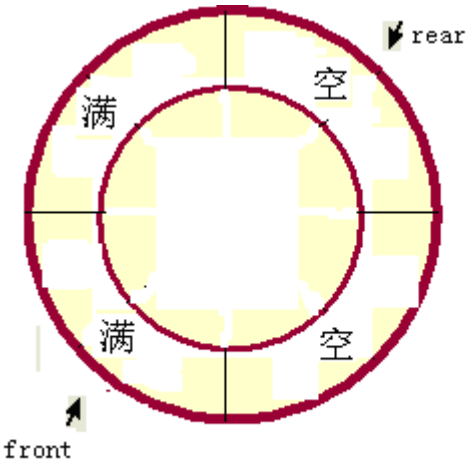
参考解答：

假设现有 A，B 两进程要互斥进入临界区 CS，互斥信号量 S.value 初值为 1。假设 P 操作不采用原语方式，如果此时 A 进程欲进入临界区 CS，那么它首先作 P（S）操作。在执行完 S.value：=S.value -1 后 S.value 值变为 0，此时由于某种原因,分配给 A 进程的时间片结束，停止 A 进程运行，系统转而调用 B 进程。由于此时 S.value =0，B 进程作 P（S）操作，S.value=S.value -1，S.Value 的值变为-1，B 进程被封锁，无法进入临界区。下次 A 进程重新轮到执行时，它从断点处开始，A 进程直接作 if S.Value <0 then Block(S.blocklist)由于此时 S 值为-1，因此 A 进程也被封锁，无法进入临界区 CS，此时临界区 CS 是处于空闲的状态，但 A、B 进程都不能进入此临界区，造成错误。由此可以知道 P 操作需要采用原语方式，即在它执行过程中不允许中断。对于 V 操作，请读者自己举例说明也必须是原语。

由上面分析， P、V 操作实现临界区进程互斥，是用简短的 P、V 操作屏蔽中断，即采

用原语的方式执行，从而使临界区可以开放中断，同时实现进程互斥。

二、m 个生产者进程和 n 个消费者进程共享缓冲区 Buffer[i]（i=1，2，3…k）。生产者进程循环地生产产品、把产品依次送入缓冲区，消费者循环地依次从中取出产品消费。缓冲区构成一个环形缓冲池。如下图所示，“满”的单元表示该缓冲区单元放有产品，“空”的单元表示该缓冲区单元为空。rear 指向生产者下次存入产品的单元，front 表示消费者下次取出产品的单元。利用 p、v 操作，实现他们的同步（设置信号量、设置其初值、画出同步流程）。说明您设置的信号量可能的最大值和可能的最小值，何时出现可能的最大值和可能的最小值的情况。



答：设缓冲区空信号量 empty，　empty.value=*k*;
设缓冲区满信号量 full，　full.value=0;
设进程互斥信号量 mutex，　mutex.value=1;

生产进程	消费进程
Loop :P(empty)	loop:　p(full)
P(mutex)	p(mutex)
向缓冲区存产品	从缓冲区取产品
V(mutex)	V(mutex)
V(full)	V(empty)
Goto loop	goto loop

Empty 可能最大值为 *k*，最小值为-*m*。
生产进程都没有向缓冲区装入产品，缓冲区全空，Empty 取得可能的最大值 *k*。
缓冲区全满，*m* 个生产进程继续要求向缓冲区装入产品，一个个都做了 P(empty)，empty 取得可能的最小值-*m*。

Full 可能最大值为 *k*，最小值为-*n*
生产进程向缓冲区装满产品，缓冲区全满，full 取得可能的最大值。

缓冲区全空，*n* 个消费进程要求消费，一个个都做了 P(full)，full 取得可能的最小值-*n*。

mutext 最大值为 1，如果 m+n>=k, mutext 最小值为*-(k-1)*. 如果 m+n<k, mutext 最小值为*-(m+n-1)*

无进程进入缓冲区时，mutext 有最大值 1。