1. 画出下面四条语句的前趋图:

S1：a=x+y;　S2：b=z+1;　S3：c=a – b；S4：w=c+1;

2. 程序并发执行时为什么会失去封闭性和可再现性？

3. 比较进程和程序。

4. 试说明 PCB 的作用，为什么说 PCB 是进程存在的惟一标志？

5. 画出进程五状态的转换图。

6. 试说明进程在三个基本状态之间转换的典型原因。

7. 为什么要引入挂起状态？该状态有哪些性质？

8. 在创建一个进程时所要完成的主要工作是什么？

9. 进程在运行时存在哪两种形式的制约？并举例说明之。

10. 写出记录型信号量 wait 和 signal 的实现。

11. 在生产者消费者问题中，如果缺少了 signal(full)或 signal(empty),对执行结果有何影响？

12. 在生产消费者问题中，如果将两个 wait 操作即 wait(full)和 wait(mutex)互换位置，或者将 signal(mutex)与 signal（full）互换位置，结果如何？

13. 试利用记录型信号量写出一个不会出现死锁的哲学家进餐问题的算法。

14. 为什么要在 OS 中引入线程？

15. 对进程和线程进行比较。

16. 某火车订票系统，可共多个用户同时共享一个订票数据库。规定允许多个用户同时查询该数据库，有查询者时，用户不能订票；有用户订票而需要更新数据库时，不可以有其他用户使用数据库。 请用 P、V 操作写出查询者和订票者的同步执行程序。

【解析】本题是一个典型的读者-写者问题，查询者是读者，订票者是写者。读者-写者问题的主要要求是：①允许多个读者共享对象；②不允许写者和其他读者或写者同时访问共享对象。为了达到上述控制，引入一个变量 readcount，用于记录当前正在运行的读者进程数以及读互斥信号量 rmutex 和写互斥信号量 wmutex。每个读者进程进入系统后需对 readcount 加 1。当 readcount 的值由 0 变为 1 时，说明是第一个读者进程进入，因此需要该读者进程对控制写者进程的信号量 wmutex 进行 P 操作，以便与写者进程互斥运行；当 readcount 的值由非 0 值增加时，说明不是第一个读者进程，此时控制写者进程的信号量已进行过 P 操作，已经禁止写者进程进入，因此不需要再次对该信号量进行 P 操作。当读者进程退出时，需对 readcount 减 1。如发现减 1 后 readcount 的值变为 0，说明是最后一个读者进程退出，因此需要该读者进程对控制写者进程的信号量 wmutex 进行 V 操作，以便写者进程能够进入。同步程序描述如下：

```
Semaphore rmutex=1,wmutex=1;
Int readcount=0;
Inquirer()
{                                    //查询者进程
   While(true)
   {
      P(rmutex);
      If(readcount==0) P(wmutex);
         //如果有查询者，不允许订票
      readcount=readcount+1;
      V(rmutex);
      查询数据库;
      P(rmutex);
      readcount=readcount-1;
      If(readcount==0) V(wmutex);
         //最后一个查询者退出后允许订票
      V(rmutex);
   }
}
Booker()                             //订票者进程
{
   While(true)
   {
      P(wmutex);
      使用数据库，订票;
      V(wmutex);
```

```
    }
}
```

下面改进要求，规定允许多个用户同时查询数据库，当有订票者到达时，不允许后续查询者查询数据库，且多个订票者可以互斥使用数据库（即写者优先算法）。描述如下：

```
Semaphore rmutex=wmutex=r=w=1;
    //加入 r、w 两个信号量实现订票者优先
Int Readcount=0;
Int Writecount=0;
Inquirer()
{
    While(true)
    {
        P(r);                          //需先检查有无订票者进程存在
        P(rmutex);
        If(readcount==0) P(w);
        Readcount=readcount+1;
        V(rmutex);
        V(r);
        查询数据库;
        P(rmutex);
        Readcount=readcount-1;
        If(readcount==0) v(w);    //无查询者进程存在
        V(rmutex);
    }
}
Booker()
{
    While(true)
    {
        P(wmutex);
        If(writecount==0) P(r);
            //第一个订票者进程进入，不允许后续查询者进程进入
        Writecount=writecount+1;
        V(wmutex);
        P(w);
        使用数据库，订票;
        V(w);
        P(wmutex);
        Writecount=writecount-1;
        If (writecount==0) v(r);
```

```
            //无订票者时允许查询者进入
            V(wmutex);
        }
    }
```

这里 r 信号量用来控制读者进程的进入，若有写者存在，则占用该信号量，阻止后续读者进入临界区；而 w 信号量则表示对临界区进行写操作的权力，当读者在临界区时，占用 w 信号量以阻止写者进行写操作，这里 w 的作用类似于刚才未添加新条件的解法中的 wmutex 信号量。本解法中，rmutex 和 wmutex 信号量变为对读者、写者计数器进行互斥操作控制的信号量。

17. 设有两个优先级相同的进程 P1 和 P2，共享 x、y、z 三个变量，执行代码见下表。信号量 s1 和 s2 的初值均为 0。试问 P1、P2 并发执行后，x、y、z 的值各是多少?给出解题过程。

进程 P1	进程 P2
y=1;	x=1;
y=y+2;	x=x+2;
V(S1);	P(S1);
z=y+1;	x=x+y;
P(S2);	V(S2);
y=z+y;	z=x+z;

【解析】可以将上述进程分解成以下 6 个程序段：

PS_1：y=1； y=y+2；	PS_2：z=y+1；	PS_3：y=z+y；
PS_4：x=1； x=x+2；	PS_5：x=x+y；	PS_6：z=x+z；

并将它们的并发执行关系用前趋图（见图 2-22）描述出来。根据 Bernstein 条件（见【解释】），程序 PS_1 和 PS_4 的确是能并发执行的，程序段 PS_2 与 PS_5 也能并发执行，而程序段 PS_3 和 PS_6 则不能并发执行，或者说它们的并发执行具有不可再现性。若先执行 PS_3，再执行 PS_6，则最后 x、y、z 的值分别为 6、7、10；若先执行 PS_6，再执行 PS_3，则最后 x、y、z 的值分别为 6、13、10。

【解释】**Bernstein 条件**是说两个过程如果有数据冲突（Data Hazard），那么就没法并行执行，例如，过程 A 生成数据 d，而过程 B 需要输入数据 d，那么 B 就需要 A 的输入，它们就无法并行执行（写后读问题，RAW）。如果二者会影响后续过程需要的数据，尤其是该数据和它们执行的顺序关系密切，那么它们同样也不能并行执行（写后写问题，WAW）。

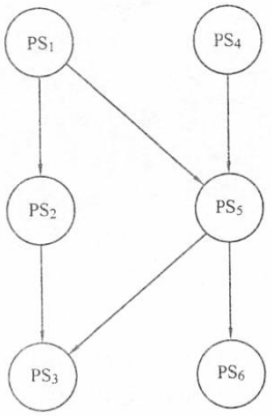
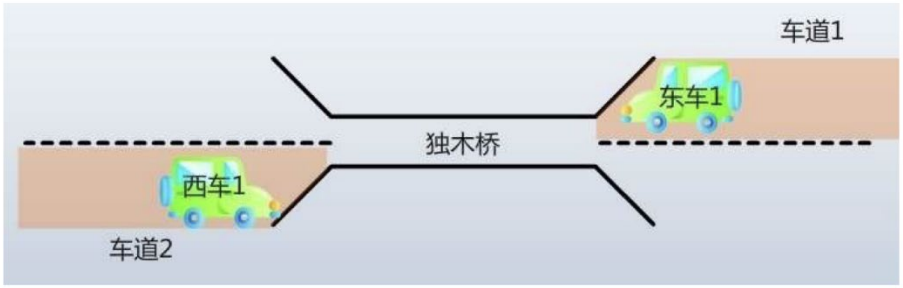


图 2-22　前趋图

18. 现有一东西向的独木桥，现要设计一个自动管理系统，管理规则如下：
1>当独木桥上有车辆在行驶时同方向的车也可以跟着驶入独木桥，但另一方向的车必须在独木桥外等待；
2>当独木桥上无车时，到达两个桥头的车辆都可以进入，但不能从同时驶入；
3>当在独木桥上朝某方向行驶的车辆驶出了独木桥且无后续车辆进入驶入，应让另一方向等待的车辆驶入独木桥。

请用记录型信号量对独木桥通行实现正确管理。



问题分析：

* 1> 应写两段程序分别代表桥的东、西两个方向上的车辆，设为east(i)、west(j)；
* 2> 对east(i)、west(j)进程来说独木桥是临界资源，因此需要设置一个互斥信号量，记为bmutex，初值为1；
* 3> 对每一车辆来说，如果桥上有同方向的车存在，则可跟着驶上桥，因此需要分别设置两个共享变量统计东、向西方向行驶的车辆数量供后续车辆参考是否可以过桥，记为ecount、wcount，初值均为0；
* 4> 由于同方向可能同时有多个车辆过桥，这些车辆都可能修改ecount或者 wcount，因此ecount、wcount对东、西方向的车辆来说均是临界资源，故需设置两个互斥信号量，分别记为emutex、wmutex，初值均为1。

* 利用记录型信号量描述独木桥问题

```
var bmutex，emutex，wmutex：semaphore：=1，1，1；
ecount，wcount： integer： =0，0；
```

```
east (i)
    P(emutex);
    ecount++;
    if(ecount==1)  P(bmutex);
    V(emutex);
    过桥;
    P(emutex);
    ecount--;
    if(ecount==0)  V(bmutex);
    V(emutex);
```

```
west (j)
    P(wmutex);
    wcount++;
    if(wcount==1)  P(bmutex);
    V(wmutex);
    过桥;
    P(wmutex);
    wcount--;
    if(wcount==0)  V(bmutex);
    V(wmutex);
```