lab_work

班级：2020219111　学号：2020212256　姓名：陈俊卉

Q1：什么是操作系统？操作系统在计算机系统中的主要作用是什么？

A1：

操作系统是管理计算机硬软件的程序。它还为应用程序提供基础，并且充当计算机用户与计算机硬件的中介。

操作系统是计算机系统中一个系统软件，集中了资源管理功能和控制程序执行的软件。是具有各种功能的、大量程序模块的集合。

大型机的操作系统主要用于优化硬件使用率，个人计算机的操作系统支持各种应用，移动计算机的操作系统为用户提供一个环境，以便与计算机进行交互及执行程序。因此操作系统设计主要关注点是**便捷**和**高效**，并根据情况有所倾斜。

- 用户视角：
  - 普通用户：大多数计算机用户坐在PC前，这种操作系统让单个用户单独使用资源，目的是优化用户进行的工作。**对于这种情况，操作系统设计的主要目的是使用方便，次要的是性能，不在乎的是资源利用。**
  - 使用大型机和小型机的用户：用户坐在大型机与小型机相连的终端，其他用户通过其他终端访问同一计算机，这些用户共享资源、交换信息。**这种操作系统的设计目标是优化资源利用率。**
  - 用户在工作站：**这类操作系统需要兼顾方便性和资源利用率。**
- 系统视角：

  从计算机的角度看，操作系统是与硬件相连的程序。所以可以将操作系统看作资源分配器。面对许多甚至冲突的资源请求，**操作系统应该考虑如何为各个程序和用户分配资源，以便计算机系统能高效、公平地运行。操作系统是个控制程序，它管理用户程序的执行，防止计算机资源的错误或不当使用。**

操作系统的主要作用：

- 进程管理：其工作主要是进程调度，在单用户单任务的情况下，处理器仅为一个用户的一个任务所独占， 进程管理的工作十分简单。但在多道程序或多用户的情况 下，组织多个作业或任务时，就要解决处理器的调度、 分配和回收等问题 。
- 存储管理：分为几种功能：存储分配、存储共享、存储保护 、存储扩张。
- 设备管理：分有以下功能：设备分配、设备传输控制 、设备独立性。
- 文件管理：文件存储空间的管理、目录管理 、文件操作管理、文件保护。
- 作业管理：是负责处理用户提交的任何要求。

Q2：操作系统有哪几大特征？它的最基本特征是什么？

A2：

- **并发性**：并行性是指两个或多个事件在同一时刻发生，而并发性是指两个或多个事件在同一时间间隔内发生;计算机系统中同时存在多个程序，宏观上看，这些程序是同时向前推进的。 在单CPU上，这些并发执行的程序是交替在CPU上运行的。 程序并发性体现在两个方面： 用户程序与用户程序之间的并发执行。 用户程序与操作系统程序之间的并发。
- **共享性**：指系统中的资源（硬件资源和信息资源）可以被多个并发执行的程序共同使用，而不是被其中一个独占。资源共享有两种方式：互斥访问和同时访问。
- **异步性**：操作系统允许多个并发进程共享资源，使得每个进程的运行过程受到其他进程制约，使进程的执行不是一气呵成，而是以停停走走的方式运行。
- **虚拟性**：虚拟性是一种管理技术，把物理上的一个实体变成逻辑上的多个对应物，或把物理上的多个实体变成逻辑上的一个对应物的技术。采用虚拟技术的目的是为用户提供易于使用、方便高效的操作环境。在OS中利用了多种虚拟技术，分别用来实现虚拟处理机,虚拟内存、虚拟外部设备和虚拟信道等。实现虚拟技术有两种形式：
    - 时分复用
    - 空分复用

共享和并发是操作系统的两个最基本的特征，虚拟以并发和共享为前提，异步是并发和共享的必然结果。

Q3：进程在系统中有哪几种基本状态,状态如何发生变化?

A3：

进程可以分为五种基本状态：

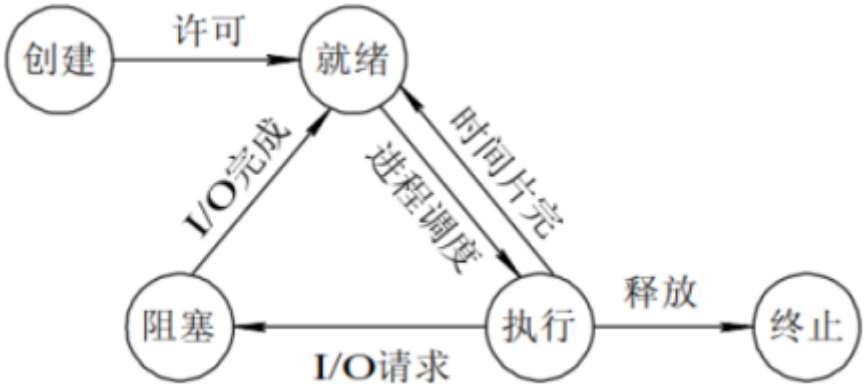
- 创建状态：进程在创建时需要申请一个空白PCB，向其中填写控制和管理进程的信息，完成资源分配。如果创建工作无法完成，比如资源无法满足，就无法被调度运行，把此时进程所处状态称为创建状态。
- 就绪状态：在创建状态完成之后，进程已经准备好，但是还未获得处理器资源，无法运行。
- 运行状态：获取处理器资源，被系统调度，开始进入运行状态。如果进程的时间片用完了就进入就绪状态。
- 阻塞状态：在运行状态期间，如果进行了阻塞的操作（I/O请求，申请缓存区失败、在程序中调用sleep(xx secs)函数 ），此时进程暂时无法操作就进入到了阻塞状态，在这些操作完成后就进入就

绪状态。

- 终止状态：进程结束，或出现错误，或被系统终止，进入终止状态。无法再执行。

如果**进程运行时间片使用完也会进入就绪状态，或者进程正在运行，但被更高优先级的进程抢先了，也会由running转为ready。**

状态变化图如下：



Q4：介绍信号量、互斥量和条件变量以及它们的区别

A4:

- 信号量(semaphore)：**是操作系统用来解决并发中的同步问题的一种方法。**信号量是一个与队列有关的整型变量。信号量的值n代表的意思为：
  - n>0：当前有数量为n的可用资源
  - n=0：资源都被占用，可用资源数量为0
  - n<0：资源都被占用，并且有|n|个进程在排队

  信号量有P（semWait）操作和V（semSignal）操作：

  P操作为申请资源。申请资源时，如果资源数在-1后<0，那么这个进程（线程）就会加入到等待队列，并进入休眠状态（根据条件变量）。如下图所示：

```
60      ----------  Semaphore . Down  ----------
61
62      method Down ()
63          var
64            oldIntStat: int
65          oldIntStat = SetInterruptsTo (DISABLED)
66          if count == 0x80000000
67            FatalError ("Semaphore count underflowed during 'Down' operation")
68          endIf
69          count = count - 1
70          if count < 0
71            waitingThreads.AddToEnd (currentThread)
72            currentThread.Sleep ()
73          endIf
74          oldIntStat = SetInterruptsTo (oldIntStat)
75        endMethod
76
77  endBehavior
```

  V操作为释放资源。释放资源时，如果资源数在+1后小于等于0，证明目前等待队列中还有进程排队，此时需要唤醒一个正在排队的进程。如下图所示：

```
41      ----------  Semaphore . Up  ----------
42
43      method Up ()
44          var
45            oldIntStat: int
46            t: ptr to Thread
47          oldIntStat = SetInterruptsTo (DISABLED)
48          if count == 0x7fffffff
49            FatalError ("Semaphore count overflowed during 'Up' operation")
50          endIf
51          count = count + 1
52          if count <= 0
53            t = waitingThreads.Remove ()
54            t.status = READY
55            readyList.AddToEnd (t)
56          endIf
57          oldIntStat = SetInterruptsTo (oldIntStat)
58        endMethod
```

个人理解：信号量本身并不强调互斥，强调的是事件的先后执行顺序，交错出现于两个合作进程内。只是二值化后可以引申出互斥的定义。

- 互斥量(mutex)：**是操作系统用来解决并发中互斥问题的一种方法。**互斥量**确保临界资源被互斥地访问**，防止同时访问可能出现的问题。**其上锁与解锁必须出现在同一进程内。**

  一个进程在使用资源时，必须确保临界资源是可用的（mutex == 1）,且在使用时对资源上锁（mutex = 0）;同时，在使用完毕后，需要及时释放资源（mutex = 1）。

考虑一个同时使用信号量和互斥量的模型：生产者和消费者模型。

```
semaphore
  n=0, s=1, e=buf-size;
void producer(){
 while (true) {
   produce();
   semWait(e);
   semWait(s);
   append();
   semSignal(s);
   semSignal(n);
 }}
```

```
void consumer(){
 while (true) {
   semWait(n);
   semWait(s);
   take();
   semSignal(s);
   semSignal(e);
   consume();
 }}
```


① 必须先执行信号量（同步关系）的P操作，再执行互斥量的P操作。否则可能会因为先上锁，再尝试执行同步操作时发现需要排队而休眠，导致锁无法及时在该进程解开而死锁的情况。

② 而V操作可以任意次序执行，没有影响。

- 条件变量：**是一种相对复杂的线程同步的方法。条件变量允许线程睡眠，直到满足某种条件。**当满足条件的时候可以向该线程发送信号，通知唤醒。

- 与信号量的区别：不保存、更新数值；是因为某个事件而令线程进入等待，而不是量本身的大小。

Q5：描述你所了解的页面调度算法

A5：

先介绍页面调度的前序知识：虚拟存储器（虚存）：

只装入作业的部分信息就可以开始执行，当主存空间小于作业需求量时， 系统就可以接受该作业， 进而也就可以允许逻辑地址 空间大于实际的主存空间。 相对于系统而言，称之为 虚拟存储器， 简称虚存。

虚存的好处：

- 使主存空间能充分地被利用
- 从用户的角度来看，好像计算机系统提供了容量很大的主存储器

虚存工作原理：

- 把作业信息保存在磁盘上，当要求装入时， 只将其中一部分先装入主存储器。 作业执行过程中，若要访问的信息不在主存中，则再设法把这些信息装入主存。

实现虚存需要解决的问题：

- 怎样知道当前哪些信息已在主存储器中？ 哪些信息尚未装入主存储器中？
- 如果作业要访问的信息不在主存储器中， 怎样找到这些信息并把它们装到主存储器中？
- 在把欲访问的信息装入主存储器时， 发现主存中已无闲块又怎么办？

解答：

页式虚拟存储器：

- 以页为单位的虚拟存储器
- 它的虚拟空间跟主存空间都将被划分为相同大小的页，主存中的页，称为实页，虚存中的页，称为虚页。
- 虚拟地址被分为两个字段，虚页号和页内偏移。
- 为了对每个虚拟页的存放位置，存取位置，使用情况，修改情况等等进行等进行说明，操作系统在主存中给每个进程都生成一个页表。
- 因此，每个虚拟页表都对应有一个页表项。页表是一张存放在主存中的，虚拟页与实页对照的映射表。
- 其中存放位置字段用来建立虚拟页与实页之间的映射，进而进行地址转换。有效位用来表示对应页面是否存在。

页式虚拟存储器工作原理：

- 只要求将当前需要的一部分页面装入内存，便可以启动作业运行，倘若在执行的过程中，出现要访问的页面不在内存中，那么就通过页面调入功能将其调入，同时还可以**将暂时不用的页面换出到辅存（页面调度）**，以腾出空间。因为页是根据请求调入的，故称为请求页式存储管理。

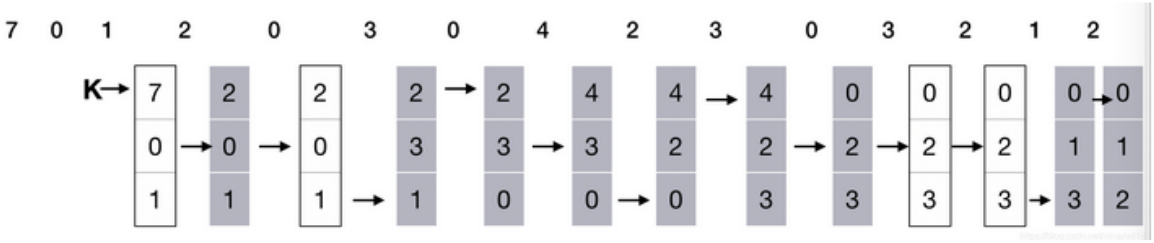
页面调度算法：

- **最佳调度算法（OPT）**：是一种理想的调度算法。当要装入一个新页而必须调出一个旧页时， 所调出的页应该是以后不再访问的页或距当前最长时间后再访问的页。
    - **注： 最佳调度算法在实现时有难度， 因为对运行中的程序无法精确判断以后要访问的页面。这个理想算法只是被用做衡量其他算法的标准。**

- **先进先出调度算法（FIFO）**：总是调出最先进入主存储器的那一页。 算法简单，易实现，一种实现方法是把装入主存储器的那些页的页号按进入的先后次序排成队列， 称它为页号队列， 用指针K 指示当前调入新页时应调出的那一页在页号队列中的位置， 最初应指向队首位置。 每当调入一个新页后， 在指针指示的位置上填上新页页号， 然后指针K加1， 指向下一个应调出的页。假定页号队列中有n个页号， 每次调出一页后， 执行

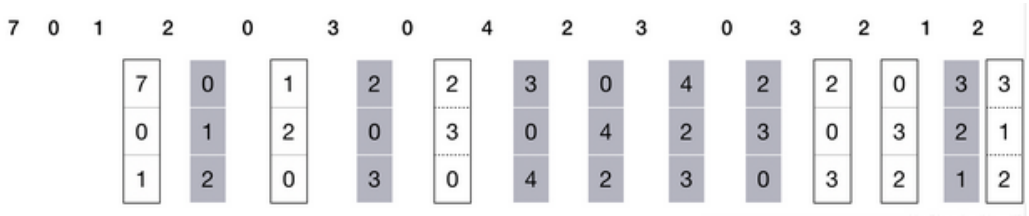
$$K := (K + 1)\ mod\ n$$

  例子（阴影为产生缺页中断【主存中没有该页】）：



- **最近最久未使用调度算法（LRU）**：最近经常被使用到的页很可能很快还要被访问， 因此不能把它调用， 相反如果在过去一段时间里没有被访问过的页，在最近的将来也可能暂时不会被访问。 所以需要装入新页时， 应选择把在最近一段时间里最久没有被使用过的页调出。

  例子（每次把最近使用的页放在最下方，缺页时排出最上面的页，下述为堆栈实现，也可以通过计数器实现）：



FIFO与LRU容易混淆，注意区别。主要区别是LRU在使用主存中存在的页时会将其放在队列的最下方（确保下方是最近使用过的），而FIFO不会。

类LRU算法：

- **第二次机会算法（又称为时钟算法）**：采用循环队列。指针指示接下来要置换哪个页面。当需要一个帧时，指针向前移动直到找到一个引用位为0的页面。在向前移动时，它会将引用位为1的页面设为0. 一旦找到引用位为0的牺牲页面，就置换该页面。

  最坏情况：当所有位置都已经被设置，指针会循环遍历整个队列，给每个页面第二次机会。第二次机会置换会退化为FIFO替换。

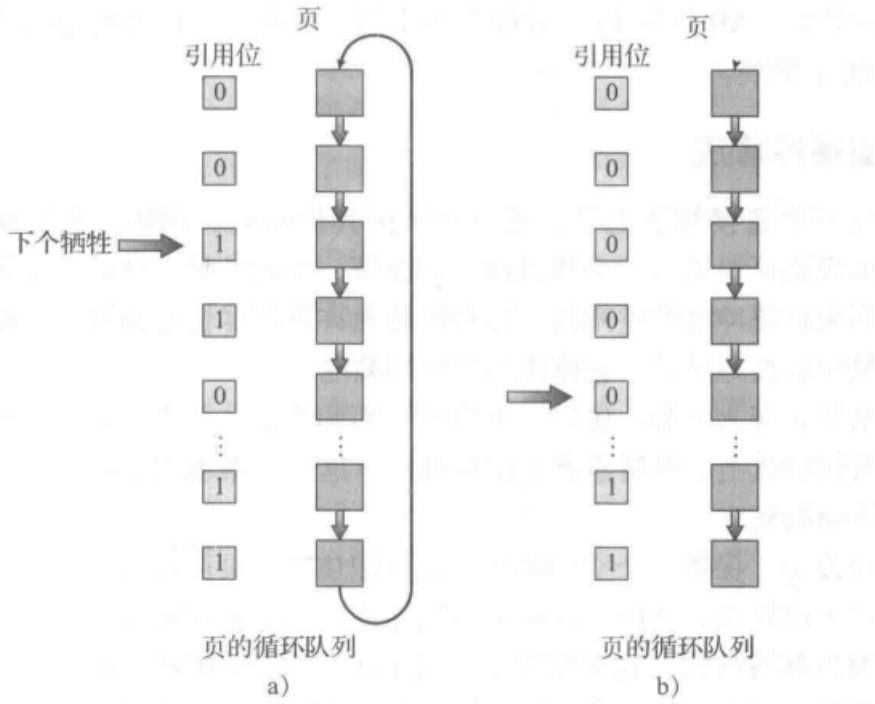


图 9-17　第二次机会（时钟）页面置换算法

增强型第二次机会算法：

9.4.5.3　增强型第二次机会算法

通过将引用位和修改位（参见 9.4.1 节的描述）作为有序对，可以改进二次机会算法。有了这两个位，就有下面四种可能的类型：

- （0，0）最近没有使用且没有修改的页面，最佳的页面置换。
- （0，1）最近没有使用但修改过的页面，不太好的置换，因为在置换之前需要将页面写出。
- （1，0）最近使用过但没有修改的页面，可能很快再次使用。
- （1，1）最近使用过且修改过，可能很快再次使用，并且在置换之前需要将页面写出到磁盘。

每个页面都属于这四种类型之一。当需要页面置换时，可使用与时钟算法一样的方案；但不是检查所指页面的引用位是否设置，而是检查所查页面属于哪个类型。我们替换非空的最低类型中的第一个页面。请注意，可能需要多次扫描循环队列，才会找到要置换的页面。

这种算法与更为简单的时钟算法的主要区别在于：这里为那些已修改页面赋予更高级别，从而降低了所需 I/O 数量。

基于计数的算法（不常用，实现昂贵，且不能很好近似OPT置换）：

- **最近最不经常使用调度算法（LFU）**：在**过去一段时间里**被访问次数多的页可能是经常需要用的页， 所以应调出被访问次数少的页。
- **最近最经常使用调度算法（MFU）**：在**过去一段时间里**被访问次数少的页可能刚刚引入且尚未使用。

Q6：描述你所了解的磁盘调度算法

A6：

先介绍前置知识：磁盘读写时间

- 寻找时间（寻道时间）：在读写数据前，需要将磁头移动到指定磁道所花费的时间
  - 寻道时间Ts = 启动磁头臂时间s + 移动磁头时间m * n
  - m：跨越一个磁道消耗时间为m，共跨越n条磁道
- 延迟时间：通过旋转磁盘，使磁头定位到目标扇区的时间。设转速为r转/秒：
  - 平均延迟时间（转半圈的时间） = 1/2r
- 传输时间：从磁盘读出或向磁盘写入数据经历的时间
  - 设磁盘转速为r，读写的字节数为b，每个磁道上的字节数为N，则
  - 传输时间 = (b/N) * (1/r) = b/(rN)
- 总平均时间 = Ts + 1/2r + b/rN

由于延迟时间和传输时间都是与磁盘转速有关的，而转速固有，所以只能通过操作系统优化寻找时间。

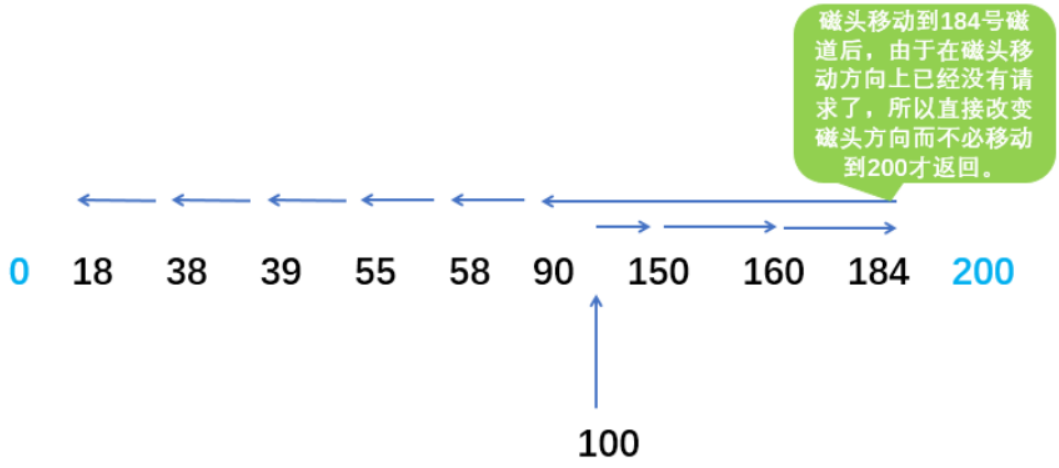
磁盘调度算法

- **先来先服务算法（FCFS）**：根据进程请求访问磁盘的先后顺序进行调度。
  - 优点：公平。如果请求访问的磁道比较集中，算法性能还算可以。
  - 缺点：大量进程竞争使用磁盘，磁道分散时性能很差。
- **最短寻找时间优先（SSTF）**：优先处理与当前磁头最近的磁道。可以保证每次寻道时间最短，但是不能保证总的寻道时间最短。（本质是贪心算法，总选择眼前最优，但总体未必最优）
  - 缺点：**磁头可能再一小块区域来回移动，产生饥饿现象。**

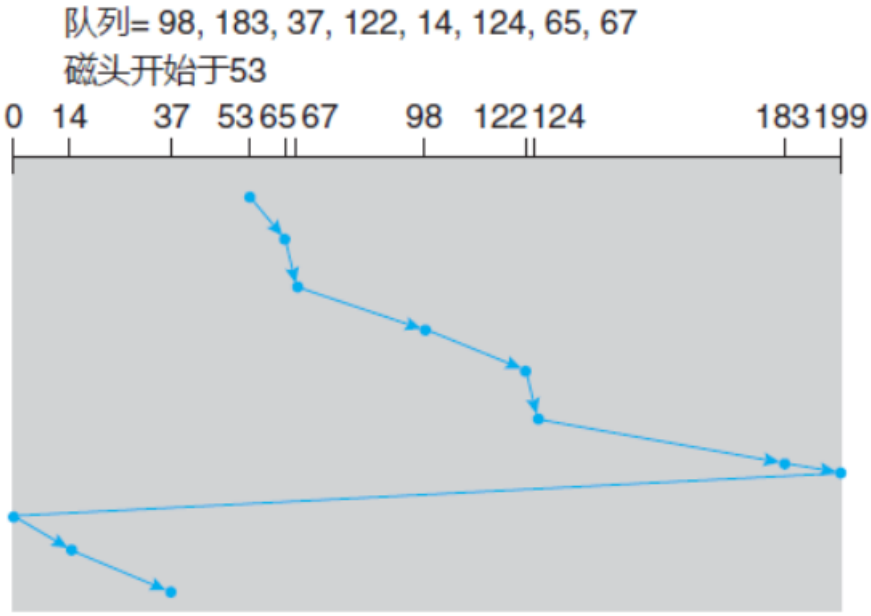
扫描算法（SCAN）

- **电梯算法（look算法）**：从起始位置先扫到最大请求的磁道，再往里扫描。

  假设某磁盘的磁道为0~200号，磁头的**初始位置是100号磁道**，且此时磁头正在往磁道号增大的方向移动，有多个进程先后陆续的访问55、58、39、18、90、160、150、38、184号磁道。



- 扫描算法（一扫到底，然后迅速回到首柱面，再依次往后扫）



Q7：描述银行家算法

A7:

为了避免死锁，出现了银行家算法：系统必须确保是否有足够的资源分配给一个进程。若有，再计算分配后系统是否会处于不安全状态，若安全才分配。

数据结构:

- 可利用资源向量Available：初值是系统中配置的该类全部可用资源数目，available[j]=k，代表现有R_j类资源K个
- 最大需求矩阵Max ： Max[i,j]=K，代表进程i需要R_j类资源最大数目为K
- 分配矩阵Allocation：当前已分配给每个进程的资源数
- 需求矩阵Need (Need[i, j] = Max[i, j] - Allocation[i. j])

步骤流程:

$Request_i$是进程P_i的请求向量。P_i发出资源请求后，

1. 如果$Request_i[j] \leqslant Need[i,j]$，继续；否则出错，超出了所需最大值
2. 如果$Request_i[j] \leqslant Available[j]$，继续；否则$P_i$等待，无足够资源
3. 试分配，判断新时刻是否安全

$$Available[j] = Available[j] - Request_i[j];$$
$$Allocation[i,j] = Allication[i,j] + Request_i[j];$$
$$Need[i,j] = Need[i,j] - Request_i[j];$$

   执行安全性算法，检查是否安全

安全性算法：尝试执行，如果进程执行完毕后资源够用，则安全

(1) 设置两个向量。①工作向量Work，开始执行安全性算法时，Work=Available；

②Finish，开始时Finsh[i]=flase；足够资源时Finish[i]=true

(2) 在进程中找到一个满足以下条件的进程：找到后执行步骤3

①$Finish[i] = false;$

②$Need[i,j] \leqslant Work[j];$

(3) 当P_i获得资源后，顺利执行直到结束，并释放资源。返回步骤2

$Work[j] = Work[j] + Allocation[i,j];$

$Finish[i] = true;$

(4) 当所有进程的Finish[i]=true都满足，则处于安全状态。

Q8：设有 3 个并发执行的进程：输入进程 Pi、计算进程 Pc 和输出进程 Po。其中进程 Pi 不断地从键盘读入整数，放入缓冲区 Buf1，Pc 按输入顺序从 Buf1 中取数据，每次取出 2 个整数，计算其和，将结果放入缓冲区 Buf2。Po 负责将Buf2 中的数据按顺序输出。设缓冲区 Buf1、Buf2 可存放的整数个数分别为m、n（m、n>0）。要求利用信号量的 P、V 操作写出进程 Pi、Pc、Po 的算法。

A8:

```
// 设信号量为e1,f1,e2,f2 e1 = m, f1 = 0, e2 = n, f2 = 0
int e1 = m;
int f1 = 0;
int e2 = n;
int f2 = 0;
int Buf1[m];
int Buf2[n];

// 因为没有临界资源互斥访问的需求，所以没有mutex

//以下四个子函数略写
// 向buf1输入 (可以cin输入)
void input_Buf1(){
    ...
}

//从Buf1中获取一个数字
int get_num_from_Buf1(){
    ...
}

// 向buf2输入
void input_Buf2(int z){
    ...
}

//从Buf2中获取一个数字
int get_num_from_Buf2(){
    ...
}
```

```
31
32  // 输入进程Pi的操作
33  void Pi(){
34      while (true) {
35          // num为将要输入的数
36          // 进行e1的P操作
37          P(e1);
38          // 向Buf1输入一个数
39          input_Buf1()
40          // 进行f1的V操作
41          V(f1);
42      }
43  }
44
45  // 计算进程Pc的操作
46  void Pc(){
47      while (true) {
48          P(f1);
49          // 从Buf1中取数据
50          int x = get_num_from_Buf1();
51          // 释放e1
52          V(e1);
53
54          P(f1);
55          // 从Buf1中取数据
56          int y = get_num_from_Buf1();
57          // 释放e1
58          V(e1);
59
60          // 进行相加
61          int z = x + y;
62
63          P(e2);
64          // 将z送入Buf2中
65          input_Buf2(z);
66          V(f2);
67      }
68  }
69
70
71  void Po(){
72      while (true) {
73          P(f2);
74          int res = get_num_from_Buf2();
75          cout << res << endl;
76          V(e2);
77      }
78  }
79
```