目录

[第一章 操作系统引论 2](#_Toc25145)

[1.1操作系统的目标和作用 2](#_Toc17091)

[1.1.1 引入操作系统的原因 2](#_Toc27699)

[1.1.2 引入操作系统的好处？ 3](#_Toc12775)

[1.2操作系统的发展过程 4](#_Toc28978)

[1.2.1 单道批处理系统的工作过程？ 4](#_Toc16075)

[1.2.2 多道批处理系统的工作过程？ 4](#_Toc15746)

[1.2.3 分时系统的工作过程？ 5](#_Toc11527)

[1.2.4 实时系统的工作过程？ 5](#_Toc29937)

[1.3操作系统的基本特性 5](#_Toc3447)

[1.3.1 操作系统的共同特征是什么？ 5](#_Toc6150)

[1.4 操作系统的主要功能 7](#_Toc19747)

[1.4.1 操作系统实现那些功能？ 7](#_Toc31715)

[第二章 进程的描述于控制 8](#_Toc24589)

[2.1 进程的描述 8](#_Toc4669)

[2.1.1 什么是进程？ 8](#_Toc4052)

[2.1.2 引入挂起操作的进程切换的样子？ 9](#_Toc2574)

[2.1.3 进程控制块里边包含的信息？ 10](#_Toc29710)

[2.2 进程同步 11](#_Toc22260)

[2.2.1 什么叫做进程同步？ 11](#_Toc2752)

[2.2.2 信号量有什么用？ 11](#_Toc23634)

[用wait-signal操作解决下图之同步问题: 13](#_Toc22888)

[2.4 线程的基本概念 13](#_Toc8880)

[2.4.1 线程与进程的区别？ 13](#_Toc876)

[第二章 处理机调度与死锁 15](#_Toc28911)

[3.1 作业与作业调度 15](#_Toc26861)

[3.1.2 作业调度算法有哪些？ 15](#_Toc12365)

[3.2 死锁概述 16](#_Toc14674)

[3.2.1 死锁产生的原因？ 16](#_Toc31048)

[3.2.2 安全状态与不安全状态？ 16](#_Toc32516)

[3.2.3 如何预防死锁？ 17](#_Toc22757)

[第四章 存储器管理 18](#_Toc5684)

[4.1 程序的装入和链接 18](#_Toc6375)

[4.1.1 程序的装入内存的方式？ 18](#_Toc20127)

[第五章 虚拟存储器 19](#_Toc27379)

[5.1 虚拟存储器概述 19](#_Toc14511)

[5.1.1 引入虚拟存储器的原因？ 19](#_Toc5568)

# 宋雅珍和王爱雯：

# 操作系统引论

## 1.1操作系统的目标和作用

### 引入操作系统的原因

在介绍进入操作系统的原因之前，先认识一下什么是操作系统。举一个简单的例子，我们将计算机抽象为一个国家，操作系统就是这个国家的国王，它管理这个国家的正常运行，没有它，那么这个国家就会陷入混乱。操作系统是配置在计算机上的第一层软件，其主要作用是管理好设备。

**1.方便性**

计算机的工作的本质就是，用户编写程序，将编好的程序存储到计算机，计算机执行程序，完成用户需要实现的功能。用户使用的语言与计算机处理的语言是不一致的，如果没有配置操作系统，用户不得不采用计算机的语言去编写命令，这样计算机才可以看懂，但是有了操作系统，利用操作系统就可以实现两个语言之间的交流。这大大的方便了用户。

1. **有效性**

一个国家的国王是非常熟悉自己的国度，为了不使自己国度的资源进行浪费，需要国王制定规则去约束自己的子民。如果没有国王的管理，那么打架斗殴抢占地盘的事情就会经常发生。所以有了操作系统，它就可以管理好计算机内部的资源分配。

1. **可扩充性**

从古至今，每一个国家的管理都是模块化管理的，不同的人负责不同的部分，对于操作系统来说也是一样的，由于有了这样的管理方式，当有地方出现错误的时候，只需要处理出错的地方，而不影响其他部分，同时若有新的模块需要建立的时候，也可以加入进来，这就是操作系统的可扩充性。

1. 开放性

每一个操作系统都可以适配这个计算机硬件。国王要能管理自己的国家，不然由他和没他是一样的。

### 引入操作系统的好处？

**1.OS作为用户与计算机硬件系统之间的接口**

用户与计算机之间交流的语言是不一样的，这时候就可以让国王代表国家与用户之间进行交流。然后国王再使用计算机听的懂的语言去操作计算机硬件，满足用户的要求。图1-1说明了用户可以使用系统调用/命令/图标-窗口与操作系统进行交流，然后操作系统理解意思后，就命令计算机硬件满足用户的需求。

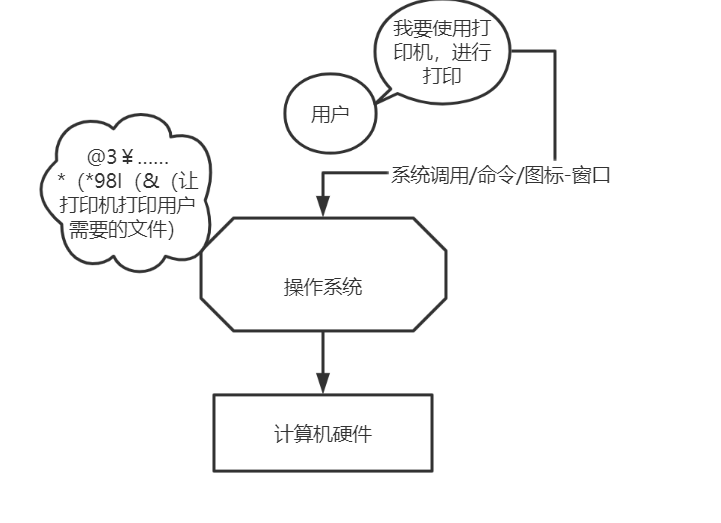


图1-1 OS作为接口示意图

1. **OS作为计算机系统资源的管理者**

在计算机系统中，通常都含有多种硬件和软件资源。可以将这些软件资源分为四类：处理机、存储器、I/O设备以及文件（数据和程序）。处理机管理是用于分配和控制处理机；存储器管理主要负责内存的分配与回收；I/O设备的分配（回收）与操纵；文件管理是用于实现对文件的存取、共享和保护。

1. **OS实现了对计算机资源的抽象**

用户使用计算机的时候，从来不会关心计算机内部是如何运行的，操作系统提供了三种方式去让用户操作计算机，这三种方式是比较简单易操作的，然而实际上计算机内部处理用户的要求是比较麻烦的，但是由于操作系统，用户不需要去关心。我们可以使用操作系统提供给我们的东西，操作计算机。

## 1.2操作系统的发展过程

### 1.2.1 单道批处理系统的工作过程？

早期的操作系统，只能一个接着一个的处理任务，就像排队一样，任务在对应的地方（磁带）上先进行排队，然后一个接着一个的进行处理。具体的工作过程是：首先由监督程序将队首的作业装入内存，并把控制权交给该作业；当该作业处理完成时，将控制权交给监督程序；一直这样重复，直到所有的任务完成。

### 1.2.2 多道批处理系统的工作过程？

由于单批道处理系统过于浪费时间，CPU的执行速度远远快于I/O设备，单批道处理系统中CPU执行结束之后，会浪费时间等待I/O设备，所以人们又提出了多道批处理系统，它可以处理多个任务，所有的任务先排队进入外村，然后利用一定的调度算法，选择一部分任务进入内存，使他们共享CPU和系统中的各种资源，让CPU充分忙碌起来，不需要让CPU去等待I/O设备。大大降低了运行时间。

### 1.2.3 分时系统的工作过程？

一台主机连接了若干个终端，每个终端有一个用户在使用。用户交互式地向系统提出命令请求，系统接受每个用户的命令，采用时间片轮转方式处理服务请求，并通过交互方式在终端上向用户显示结果。用户根据上步结果发出下道命令。****分时操作系统将CPU的时间划分成若干个片段，称为时间片****。操作系统以时间片为单位，轮流为每个终端用户服务。每个用户轮流使用一个时间片而使每个用户并不感到有别的用户存在。分时系统主要方便于人际交互。

### 1.2.4 实时系统的工作过程？

实时操作系统（RTOS）是指当外界事件或数据产生时，能够接受并以足够快的速度予以处理，其处理的结果又能在规定的时间之内来控制生产过程或对处理系统作出快速响应，并控制所有实时任务协调一致运行的操作系统。

其特点是及时响应和高可靠性。实时系统又分为硬实时系统和软实时系统，硬实时系统要求在规定的时间内必须完成操作，这是在操作系统设计时保证的；软实时则只要按照任务的优先级，尽可能快地完成操作即可。

## 1.3操作系统的基本特性

### 1.3.1 操作系统的共同特征是什么？

**1.并发**

操作系统引入进程（系统中能独立运行并作为资源分配的基本单位）实现并发执行。在未引入进程，操作系统之间的执行必须是按顺序执行的，例如在属于同一个的应用程序的计算程序和I/O程序，只要当计算程序执行结束之后，I/O程序才可以执行，但是有了并发之后，它们可以一起执行，提高了系统资源的利用率。

1. **共享**

由于计算机内部的资源是十分有限的，资源存在回收和复用。且供需关系之间供少于求，若没有制定资源复用的规则，就会发生资源之间的抢夺。资源复用的规则目前的方式有两种，互斥共享和同时访问。

1. **虚拟**

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。

1. **异步**

在多道程序环境下，允许多个程序并发执行，但由于资源有限，进程的执行不是一贯到底。而是走走停停，以不可预知的速度向前推进，这就是进程的异步性。异步性使得操作系统运行在一种随机的环境下，可能导致进程产生与时间有关的错误。但是只要运行环境相同，操作系统必须保证多次运行程序，都获得相同的结果。

## 1.4 操作系统的主要功能

### 1.4.1 操作系统实现那些功能？

**1.处理机管理功能**

处理机就相当于打仗时的将军，它控制它手底下的士兵（进程）完成老大（操作系统）分配给的任务，它管理进程的控制（进程的创建、进程的终止）、进程同步（进程的执行步骤）、进程通信（两个进程相互合作完成任务，它们需要进行交流）、调度（任务调度和人员调度（进程））

1. **存储器管理功能**

存储器主要就是进行管理内存，相当于房地产资源管理，它首先要做好的就是进行房屋（内存）分配，让程序“个得其所”,不要因为分配失误，让两个不熟的人住同一个屋子，其次要做到的就是让居民之间不要随意盗用别人的房间，（确保每一道用户程序都仅在自己的内存空间内运行，彼此互不干扰）。管理好自己内部的资源，不要让其余人知道（绝不允许用户访问操作系统的程序和数据，也不允许用户程序转移到非共享的其他用户程序中去执行）

1. **设备管理功能**

主要任务就是解决用户进程提出的I/O请求，为用户分配所需的I/O设备，然后提高CPU和I/O设备的利用率。由于CPU的速度远远快于I/O设备，为了解决速度之间的不匹配，提出了缓冲管理，将I/O输出的内容可以存储道缓冲区里边，减少了访问I/O设备的次数，提高速率，设备的分配也需要相应的管理，就如同图书借阅一样，需要登记等相关操作。设备的处理，主要设计设备的使用。

1. **文件管理功能**

文件管理主要是管理好用户文件和系统文件，方便用户使用，并保证文件的安全性。所以文件管理需要对文件存储空间的管理（文件的存储与回收），目录管理（方便用户查找文件），文件的读/写管理和保护（有些文件是比较重要的，需要提供权限才可以修改）

# 第二章 进程的描述于控制

## 2.1 进程的描述

### 2.1.1 什么是进程？

在用户的角度来说，就是正在运行的程序。打开WPS,使用它编辑文字，此时的WPS就是一个进程。对于操作系统来说，进程就是一个进程控制块（PCB)。

2.1.2 进程有几种状态

进程在并发执行时共享系统资源，由于资源有限，所以会呈现间断性，会出现多种状态。

**就绪状态。**

进程已经做好准备开始运行。士兵已经拿到作战所需要的武器（进程已经分配到除CPU以外的资源），只要将军下令（获得CPU),便可以立即执行。

**执行状态。**

进程执行任务（士兵正在作战）

**阻塞状态。**

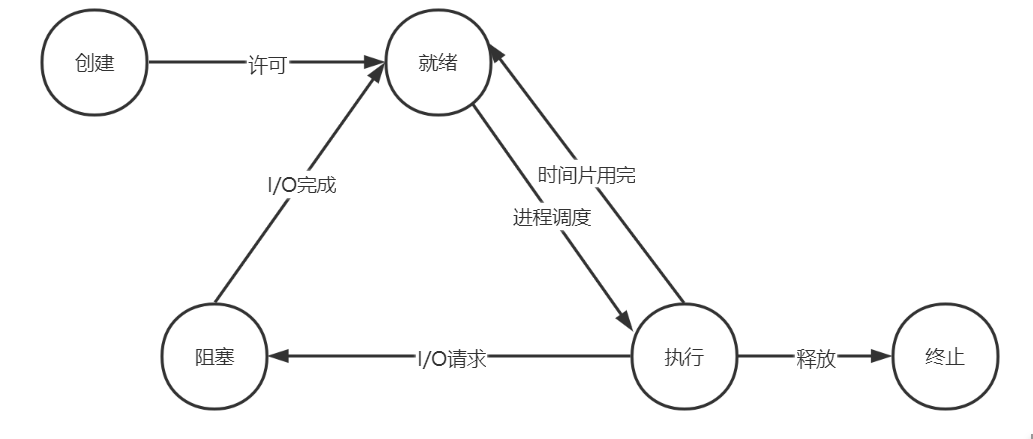
正在执行的进程由于发生某事件（如I/O请求、申请缓冲区失败等）暂时无法继续执行时的状态。（作战安营扎寨时）

**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（士兵没有被选中去作战）

**终止状态**

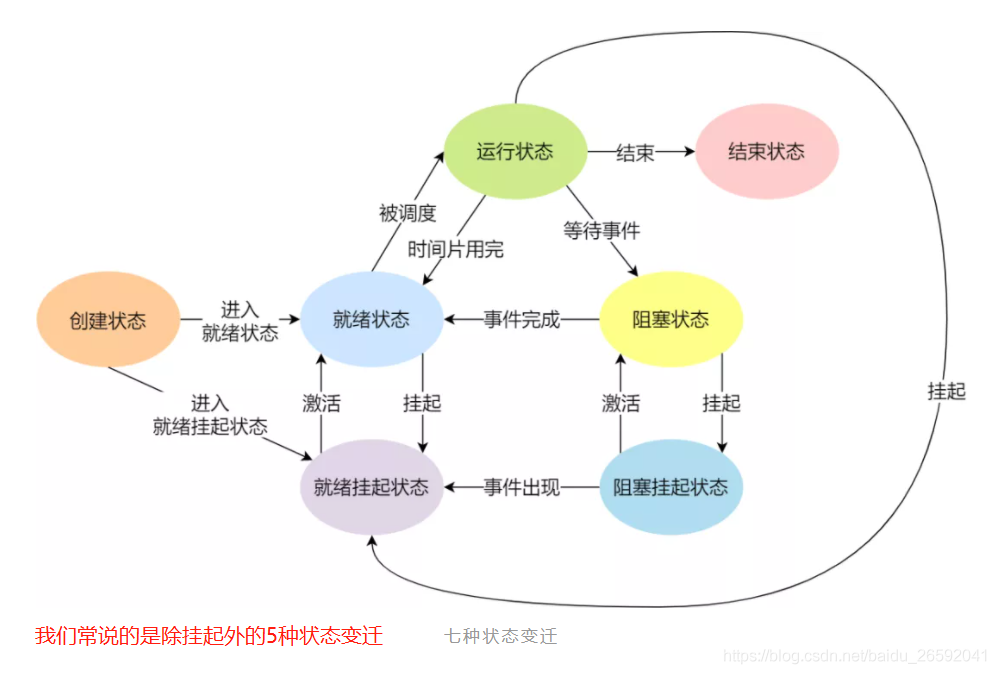
进程结束，或出现错误，或被系统终止，进入终止状态。无法再执行（战争结束）



**图2-1 进程状态的转换**

### 2.1.2 引入挂起操作的进程切换的样子？

挂起操作就是将运行的进程按了暂停键一样，变为静止的样子。



**图2-2 具有挂起操作的进程转换图**

### 2.1.3 进程控制块里边包含的信息？

**1.进程标识符**

它包含外部标识符，相当于对外的身份，和内部标识符，方便操作系统认识进程。

**2.处理机状态**

通用寄存器，暂存一些进程处理后的数据信息；指令计数器，存放下一条指令的地址，方便进程被切换回来之后可以继续执行；程序状态字PSW,存储状态信息，用户栈指针，存放系统调用参数以及调用地址。

1. **进程调度信息**

进程状态，此时进程处于的状态，方便后期进程进行状态的切换；进程优先级，进程被调用的优先级；进程调度所需的其它信息（方便后期使用调度算法，对进程进行调度）；事件，进程阻塞的原因。

1. **进程控制信息**

程序和数据的地址（以便再次调度到程序，可以能从PCB中找到其程序和数据）；进程同步和通信机制，包含消息队列指针，和信号量（方便进程可以互斥或者同步使用资源，以及实现进程交流）；资源清单（进程所需要的全部资源）链接指针（下一个PCB的地址）

## 2.2 进程同步

### 2.2.1 什么叫做进程同步？

多个进程之间并发执行时，按照一定规则共享系统资源，能很好的相互合作，使程序的执行具有可在现性。不按照规则的话，每一次执行可能都是不同的结果。进程之间的关系不同，那么遵照的规则也不同。进程之间有两类关系，一种是间接相互制约关系吗，多个进程不能同时访问临界资源，必须互斥访问；直接相互制约关系，两个进程访问临界资源的时候，必须要有先后顺序。临界资源就是如果多个进程需要访问相同的资源，那么该资源就被称为临界资源，访问临界资源代码的地方就叫做临界区。

### 2.2.2 信号量有什么用？

**1.利用信号量进行进程互斥**

**经典的生产者-消费者问题**

现在有两个进程，一个进程为生产者，用于生产产品，另一个进程为消费者，用户消费产品，生产者生产产品后，放到buff缓冲区，消费者从buff缓冲区里边取出产品，者两个进程必须互斥使用buff缓冲区，不可以生产者生产了产品还没有放进去，就被消费者消耗，所以这两个进程对应产品来说，他们是直接相互制约关系，对于缓冲区这个资源，它们的关系是间接相互制约关系。利用信号量empty和full 分别表示缓冲池中空缓冲区和满缓冲区的量，并用mutex 进行互斥访问缓冲区。

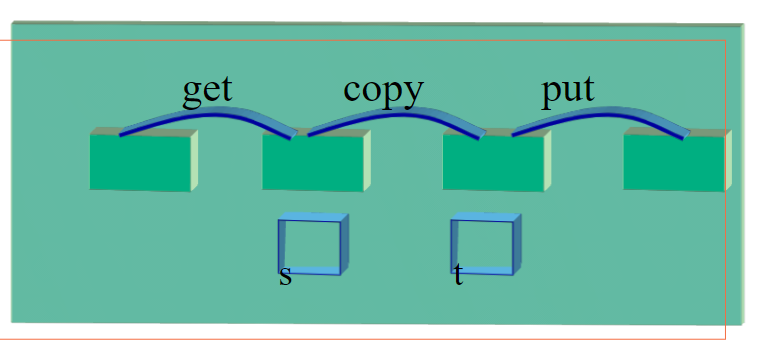
|  |
| --- |
| int in=0,out=0;  int buffer[n];  Semaphore mutex=1,empty=n,full=0; |

|  |
| --- |
| void Producer(){  wait(empty) ; //判断是否可以继续生产产品放入缓冲区  wait(mutex) ; //判断现在是否有其他进程访问该缓冲区  生产产品，放入缓冲区  signal(mutex);//访问临界区资源结束，释放锁  signal(full);//产品数加一  } |

|  |
| --- |
| void Consumer(){  wait(full) ; //判断是否可以继续生产产品放入缓冲区  wait(mutex) ; //判断现在是否有其他进程访问该缓冲区  生产产品，放入缓冲区  signal(mutex);//访问临界区资源结束，释放锁  signal(empty);//产品数加一} |

1. **利用信号量实现前驱关系**

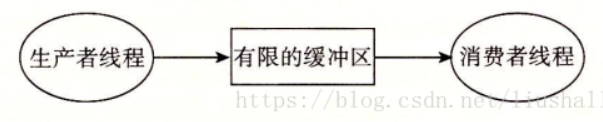
用wait-signal操作解决下图之同步问题:



|  |
| --- |
| S= 0 s1=0  void Get(){ void copy() { put:  从s中取出东西； wait(s): wait(s1);  Signal(s) } 进行复制； 将东西放到t中}  Signal(s1);|} |

## 2.3 经典进程同步问题

### 2.3.1生产者消费者问题



1. **问题分析**

* 在缓冲区为空时，消费者不能再进行消费
* 在缓冲区为满时，生产者不能再进行生产
* 在一个线程进行生产或消费时，其余线程不能再进行生产或消费等操作，即保持线程间的同步
* 注意条件变量与互斥锁的顺序

由于前两点原因，因此需要保持线程间的同步，即一个线程消费（或生产）完，其他线程才能进行竞争CPU，获得消费（或生产）的机会。对于这一点，可以使用条件变量进行线程间的同步：生产者线程在product之前，需要wait直至获取自己所需的信号量之后，才会进行product的操作；同样，对于消费者线程，在consume之前需要wait直到没有线程在访问共享区（缓冲区），再进行consume的操作，之后再解锁并唤醒其他可用阻塞线程。

在访问共享区资源时，为避免多个线程同时访问资源造成混乱，需要对共享资源加锁，从而保证某一时刻只有一个线程在访问共享资源。

1. **伪代码实现**

假设缓冲区大小为10，生产者、消费者线程若干。生产者和消费者相互等效,只要缓冲池未满,生产者便可将消息送入缓冲池;只要缓冲池未空,消费者便可从缓冲池中取走一个消息。

* items代表缓冲区已经使用的资源数，spaces代表缓冲区可用资源数
* mutex代表互斥锁
* buf[10] 代表缓冲区，其内容类型为item
* in、out代表第一个资源和最后一个资源

var items = 0, space = 10, mutex = 1;

var in = 0, out = 0;

item buf[10] = { NULL };

producer {

while( true ) {

wait( space ); // 等待缓冲区有空闲位置， 在使用PV操作时，条件变量需要在互斥锁之前

wait( mutex ); // 保证在product时不会有其他线程访问缓冲区

// product

buf.push( item, in ); // 将新资源放到buf[in]位置

in = ( in + 1 ) % 10;

signal( mutex ); // 唤醒的顺序可以不同

signal( items ); // 通知consumer缓冲区有资源可以取走

}

}

consumer {

while( true ) {

wait( items ); // 等待缓冲区有资源可以使用

wait( mutex ); // 保证在consume时不会有其他线程访问缓冲区

// consume

buf.pop( out ); // 将buf[out]位置的的资源取走

out = ( out + 1 ) % 10;

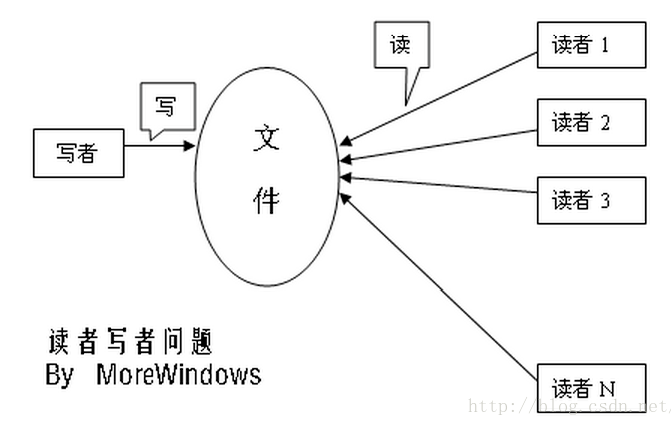
signal( mutex ); // 唤醒的顺序可以不同

signal( space ); // 通知缓冲区有空闲位置

}

}

### 2.3.2读者写者问题



1. **题目要求：**

①允许多个读者可以同时对文件执行读操作；  
②只允许一个写者往文件中写信息；  
③任一写者在完成写操作之前不允许其他读者或写者工作；  
④写者执行写操作前，应让已有的读者和写者全部退出。

1. **问题分析**

关系分析。由题目分析读者和写者是互斥的，写者和写者也是互斥的，而读者和读者不存在互斥问题。  
两个进程，即读者和写者。写者是比较简单的，它和任何进程互斥，用互斥信号量的P操作、V操作即可解决。读者的问题比较复杂，它必须实现与写者互斥的同时还要实现与其他读者的同步，因此，仅仅简单的一对P操作、V操作是无法解决的。那么，在这里用到了一个计数器，用它来判断当前是否有读者读文件。当有读者的时候写者是无法写文件的，此时读者会一直占用文件，当没有读者的时候写者才可以写文件。同时这里不同读者对计数器的访问也应该是互斥的。

1. **伪代码实现**

首先设置信号量count为计数器，用来记录当前读者数量，初值为0;   
设置mutex为互斥信号量，用于保护更新count变量时的互斥；  
设置互斥信号量rw用于保证读者和写者的互斥访问。

int count=0; //用于记录当前的读者数量

semaphore mutex=1; //用于保护更新count变量时的互斥

semaphore rw=1; //用于保证读者和写者互斥地访问文件

writer () { //写者进程

while (1){

P(rw); // 互斥访问共享文件

Writing; //写入

V(rw) ; //释放共享文件

}

}

reader () { // 读者进程

while(1){

P (mutex) ; //互斥访问count变量

if (count==0) //当第一个读进程读共享文件时

P(rw); //阻止写进程写

count++; //读者计数器加1

V (mutex) ; //释放互斥变量count

reading; //读取

P (mutex) ; //互斥访问count变量

count--; //读者计数器减1

if (count==0) //当最后一个读进程读完共享文件

V(rw) ; //允许写进程写

V (mutex) ; //释放互斥变量 count

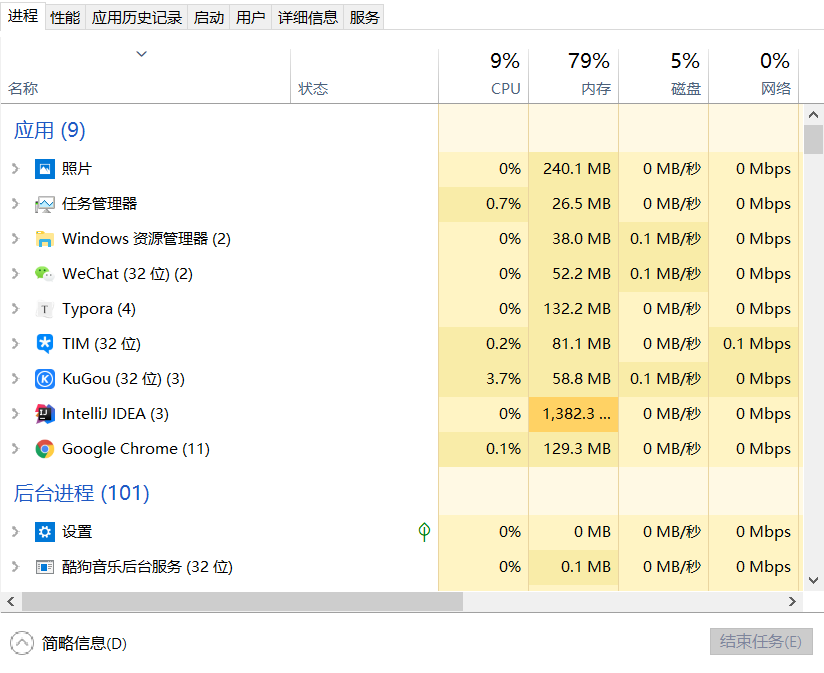
}

}

## 2.4 线程的基本概念

### 2.4.1 线程与进程的区别？

一个在[内存](https://so.csdn.net/so/search?q=%E5%86%85%E5%AD%98&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/ThinkWon/article/details/_blank)中运行的应用程序。每个进程都有自己独立的一块内存空间，一个进程可以有多个线程，比如在Windows系统中，一个运行的xx.exe就是一个进程。



线程就是进程中的一个执行任务（控制单元），负责当前进程中程序的执行。一个进程至少有一个线程，一个进程可以运行多个线程，多个线程可共享数据。

与进程不同的是同类的多个线程共享进程的堆和方法区资源，但每个线程有自己的程序计数器、虚拟机栈和本地方法栈，所以系统在产生一个线程，或是在各个线程之间作切换工作时，负担要比进程小得多，也正因为如此，线程也被称为轻量级进程。

最本质的区别就是进程是操作系统资源分配的基本单位，而线程是处理器任务调度和执行的基本单位。

# 第三章 处理机调度与死锁

## 作业与作业调度

### 3.1.2 作业调度算法有哪些？

**1.先来先服务（FCFS）【先到先得】**

FCFS即先来先服务的算法,是一种非常简单的调度算法,总是把当前处于就绪队列之首的那个进程调度到运行状态.只考虑进程进入就绪队列的先后,而不考虑它的下一个CPU周期的长短及其他因素.属于非抢占式策略,性能相较于其他算法有所不足。

**2.短作业优先（SJF）【永远最短的最优先】**

SJF即短作业优先。它也是一个非抢占的。根据服务时间的长短来调度进程,每次当当前进程运行完毕,就会对当前正在就绪队列中排队的进程进行挑选,**找出其中的运行时间最短的进程**,缺点是对长作业不友好,可能会造成较长运行时间的作业长时间停留在就绪队列.

1. 优先级调度算法（PSA)

基于作业的紧迫程度，人为控制作业的优先级，去进行调度。

4.高响应比优先调度算法（HRRN)

HRRN是介于FCFS与SJF之间的折中算法。所有在就绪队列等待的进程进行响应比的计算。**其响应比会越来越大,最终会被执行**。

**响应比：**（等待时间+运行时间）/运行时间

## 死锁概述

### 死锁的定义

多个进程在执行过程中，因争夺同类资源且资源分配不当而造成的一种互相等待的现象，若无外力作用，它们都将永远无法继续执行，这种状态称为死锁，这些处于等待状态的进程称为死锁进程。

### 产生死锁的必要条件

1. 互斥条件：进程对所分配到的资源进行排他性使用，即一段时间内，某资源只能被一个进程占用。如果此时还有其他进程请求该资源，则进程只能等待，直至占有资源的进程用毕释放。
2. 请求和保持条件：进程已经保持了至少一个资源，但又提出了新的资源请求，而该资源已经被其他进程占有，此请求进程被阻塞，但对自己已获得的资源保持不放。
3. 不可抢占条件：进程已获得的资源在未使用完成之前不能被抢占，只能在进程使用完成时由自己释放。
4. 循环等待条件：在发生死锁时，必存在一个进程——资源的循环链，即进程的集合{P0，P1，P2，…Pn}中的P0正在等待P1占用的资源，P1正在等待P2占用的资源，……Pn正在等待P0占用的资源。

### 处理死锁的方法

1. **预防死锁：**

这是一种较简单和直观的实现预防方法。该方法通过设置某些限制条件，去破坏产生死锁的四个必要条件中的一个或几个来预防产生死锁。预防死锁是一种较易实现的方法，已被广泛使用。

2. **避免死锁：**

同样是属于事先预防策略，但它并不是事先采取各种限制措施，去破坏产生死锁的四个必要条件，而是在资源的动态分配过程中，用某种方法防治系统进入不安全状态，从而可以避免发生死锁。

**3. 检测死锁：**

这种方法无需事先采取任何限制性措施，允许进程在运行过程中发生死锁。但可以通过检测机构及时检测出死锁的发生，然后采取相应的措施，把进程从死锁中解脱出来。

4. **解除死锁：**

当系统检测到系统中已经发生了死锁时，就采取相应的措施，将进程从死锁状态中解脱出来。常用的方法就是撤消一些进程，回收他们的资源，将他们分配给已处于阻塞状态的进程，使其能继续运行。

### 如何预防死锁

**（1）破坏互斥条件**

方法：

允许系统资源都能共享使用，则系统不会进入死锁状态。

缺点：

有些资源根本就不能被同时使用，如打印机资源，所以，破坏互斥条件而预防死锁的方法不太可行，而且在有的场合应该保护这种互斥性。

**（2）破坏请求保持条件**

方法：

釆用预先静态分配方法，即进程在运行前一次申请完它所需要的全部资源，在它的资源未满足前，不把它投入运行。一旦投入运行后，这些资源就一直归它所有，也不再提出其他资源请求，这样就可以保证系统不会发生死锁。

缺点：

系统资源被严重浪费，其中有些资源可能仅在运行初期或运行快结束时才使用，甚至根本不使用。而且还会导致“饥饿”现象，当由于个别资源长期被其他进程占用时，将致使等待该资源的进程迟迟不能开始运行。

**（3）破坏不可抢占条件**

方法：

当一个已保持了某些不可剥夺资源的进程，请求新的资源而得不到满足时，它必须释放已经保持的所有资源，待以后需要时再重新申请。这意味着，一个进程已占有的资源会被暂时释放，或者说是被剥夺了，或从而破坏了不可剥夺条件。

缺点：

该策略实现起来比较复杂，释放已获得的资源可能造成前一阶段工作的失效，反复地申请和释放资源会增加系统开销，降低系统吞吐量。这种方法常用于状态易于保存和恢复的资源，如CPU的寄存器及内存资源，一般不能用于打印机之类的资源。

**（4）破坏循环等待条件**

方法：

为了破坏循环等待条件，可釆用顺序资源分配法。首先给系统中的资源编号，规定每个进程，必须按编号递增的顺序请求资源，同类资源一次申请完。也就是说，只要进程提出申请分配资源Ri，则该进程在以后的资源申请中，只能申请编号大于Ri的资源。

缺点：

这种方法存在的问题是，编号必须相对稳定，这就限制了新类型设备的增加；尽管在为资源编号时已考虑到大多数作业实际使用这些资源的顺序，但也经常会发生作业使用资源的顺序与系统规定顺序不同的情况，造成资源的浪费；此外，这种按规定次序申请资源的方法，也必然会给用户的编程带来麻烦。

### 死锁产生的原因？

**1、竞争资源**

当系统中供多个进程共享的资源如打印机、公用队列等，其数目不足以满足产生诸进程的需要时，会引起诸进程对资源的竞争而产生死锁。

**2、进程间推进顺序非法**

进程在运行过程中，请求和释放资源的顺序不当，也同样会导致产生死锁。

### 安全状态与不安全状态？

安全状态，是指系统能按某种进程顺序(P1,P…Pn)(称〈P1,P…Pn〉序列为安全序列)，来为每个进程Pi分配其所需资源，直至满足每个进程对资源的最大需求，使每个进程都可顺利地完成。如果系统无法找到这样一个安全序列，则称系统处于不安全状态。

假定系统中有三个进程P1、P2和P3, 共有12台磁带机。进程P1总共要求10台磁带机，P2和P3分别要求4台和9台。假设在T0时刻，进程P1、P2和P3已分别获得5台、2台和2台磁带机，尚有3台空闲未分配，此时就是安全状态，因为可以满足进程的需求，

|  |  |  |  |
| --- | --- | --- | --- |
| 进程 | 最大需求 | 已分配 | 可用 |
| P1  P2  P3 | 10  4  9 | 5  2  2 | 3 |

但是在T0时刻以后，P3又请求1台磁带机，若此时系统把剩余3台中的1台分配给P3,则系统便进入不安全状态。因为，此时也无法再找到一个安全序列，例如，把其余的2台分配给P2,这样，在P2完成后只能释放出4台，既不能满足P1尚需5台的要求，也不能满足P3尚需6台的要求，致使它们都无法推进到完成，彼此都在等待对方释放资源，即陷入僵局，结果导致死锁。

### 如何避免死锁？

避免死锁的基本思想就是，当有进程请求资源时，系统需要对该进程的请求进行计算，若将该资源分配给该进程后，进程是否可以安全的推进。若是安全的，就将该资源分配给进程。

银行家算法：

Requseti[j] 是进程Pi对资源的请求，Requsesti[j]=k,表示，Pi请求需要K个j资源。Need[i,j]表示，Pi进程对资源的j的需求数，Available[j]表示分配给进程Pi需求的资源之后，剩余的资源数，Allocation[i,j]，表示已经分配给进程Pi j资源的个数。

1. 如果Requesti[j]<=Need[i,j],执行步骤二
2. 若Requesti[j]<=Avaiable[j],执行步骤三
3. 假设将资源分配给进程Pi，修改值

Avaiable[j]=Avaiable[j]-Requesti[j];

Allocation[i,j]=Allocation[i,j]+Request[j]

Need[i,j]=Need[i,j]-Requesti[j]

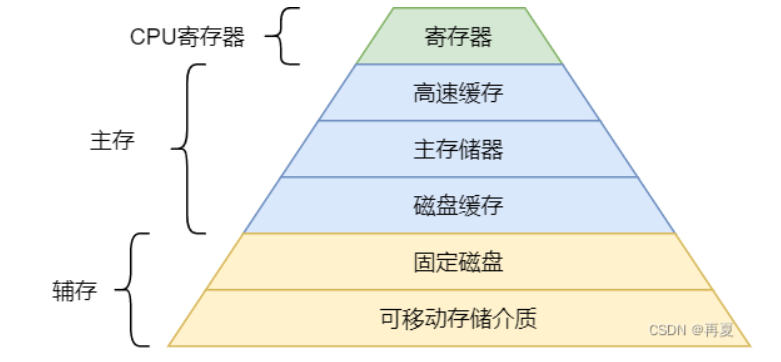
1. 执行安全算法，检测按照资源分配后系统是否处于安全状态。

# 第四章 存储器管理

## 存储器的层次结构

### 4.1.1存储器结构示意图和各层次解释？

存储器的层次结构如图所示：



1. **寄存器**

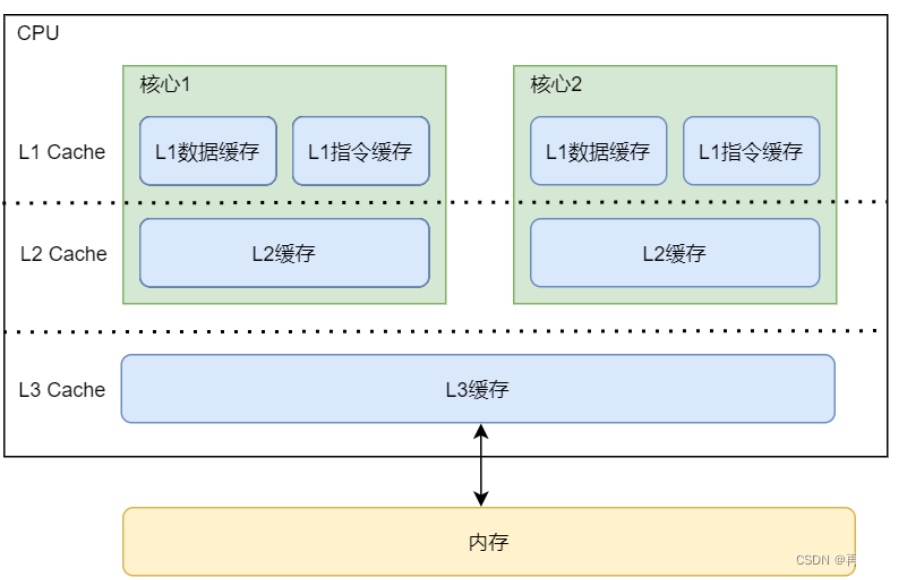
最靠近CPU的控制单元的存储器，就是寄存器了，它使用的材料速度也是最快的，因此价格也是最贵的，那么数量就没很多。

寄存器的访问速度非常快，一般要求在半个CPU时钟内完成读写，CPU时钟周期跟CPU主频息息相关，比如2GHz的主频的CPU，那么它的时钟周期就是1/2G，也就是0.5ns。

CPU处理一条指令的处理周期，除了读写寄存器，还主要解码指令、控制指令执行和计算。

1. **高速缓存**

CPU的高速缓存，通常可以分为L1、L2、L3这样的三层高速缓存，也称为一级缓存、二级缓存、三级缓存。如图所示：



* L1高速缓存

L1高速缓存的访问速度几乎和寄存器一样快，通常只需要2~4个时钟周期，而大小在几十KB到几百KB不等。

每个CPU核心都有一块属于自己的L1高速缓存，指令和数据在L1是分开存放的，所以L1高速缓存通常分成指令缓存和数据缓存。

* L2高速缓存

L2高速缓存同样每个CPU核心都有，但是L2高速缓存位置比L1高速缓存距离CPU核心更远，它大小比L1高速缓存更大，CPU型号不同大小也就不同，通常大小在几百KB到几MB不等，访问速度则更慢，速度在10~20个时钟周期。

* L3高速缓存

L3高速缓存通常是多个CPU可信共用的，位置比L2高速缓存距离CPU可信更远，大小也会更大些，通常大小在几MB到几十MB不等，具体值根据CPU型号而定。访问速度相对也比较慢一些，访问速度在20~60个时钟周期。

1. **主存储器**

简称内存或主存，也称可执行存储器。内存用的芯片和CPU Cache有所不同，他使用的是一种叫做DRAM（Dynamic Random Access Memory）的芯片。

相比SRAM，DRAM的密度更高，功耗更低，有更大的容量，而且造价比SRAM芯片便宜很多。

DRAM存储一个bit数据，只需要一个晶体管和一个电容就能存储，但是因为数据会被存储在电容里，电容会不断漏电，所以需要定时刷新电容，才能保证数据不会丢失，这就是DRAM之所以被称为动态存储器的原因，只有不断刷新，数据才能被存储起来。

DRAM的数据访问电路和刷新电路都比SRAM更复杂，所以访问的速度会更慢，内存速度大概在200~300个时钟周期之间。

1. **磁盘缓存**

* 用来缓和磁盘I/O操作与主存速度的不匹配。
* 用于暂时存放频繁使用的磁盘数据。
* 并不是实物，而是利用主存的部分存储空间。

1. **SSD，HDD硬盘**

SSD（Solid-state disk）就是我们常说的固体硬盘，结构和内存类似，但是它相比内存的优点是断电后数据还是存在的，而内存、寄存器、高速缓存断电后数据都会丢失。内存的读写速度比SSD快10~1000倍。

## 4.2 程序的装入和链接

### 4.2.1 程序的装入内存的方式？

**1.绝对装入方式**

物理地址和逻辑地址一样

1. **可重定位装入方式**

用户编译后形成的目标模块，我们可以得到它们的相对于起始地址的偏移地址，然后根据内存，选择合适的内存将目标模块放入，此时放入的地址（物理地址）和我们知道的地址（逻辑地址）是不一样的，此时程序和数据的地址就需要修改。

1. **动态运行时的装入方式**

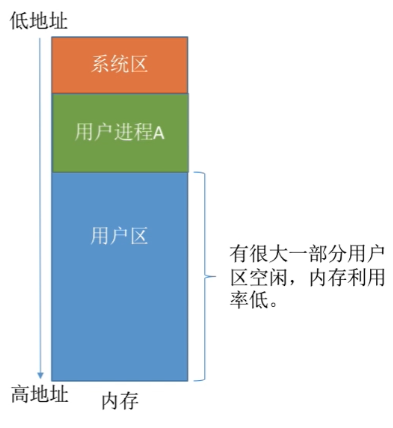
动态运行时的装入程序在把装入模块装入内存时，并不立即把装入模块中的逻辑地址转化为物理地址，而是推迟到程序真正执行的时候才进行，为了避免程序装入内存的地址发生变化。

## 连续分配储存管理方式

### 连续分配方式的分类和概念

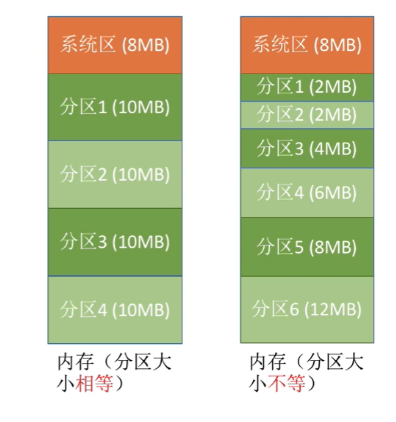
1. ****单一连续分配****

将内存分为系统区（内存低端，分配给OS用）和用户区（内存高端，分配给用户用）。采用静态分配方式，即作业一旦进入内存，就要等待它运行结束后才能释放内存



1. ****固定分区分配****

* 将内存空间划分为若干个固定大小的分区，除OS占一区外，其余的一个分区装入一道程序。分区的大小可以相等，也可以不等，但事先必须确定，在运行时不能改变。即分区大小及边界在运行时不能改变。



* 系统需建立一张分区说明表或使用表，以记录分区号、分区大小、分区的起始地址及状态（已分配或未分配）。



1. ****动态分区分配****

不事先将内存划分成一块块的分区，而是在作业进入内存时，根据作业的大小动态地建立分区，并使分区的大小正好适应作业的需要。因此系统中分区的大小是可变的，分区的数目也是可变的。

动态分区分配算法包含2种类型：****基于顺序搜索的动态分区分配算法。基于索引搜索的动态分区分配算法。****

1. ****动态重定位分区分配****

动态重定位分区分配算法与动态分区分配算法基本上相同，差别仅在于：在这种分配算法中，增加了紧凑的功能。通常，当该算法不能找到一个足够大的空闲分区以满足用户需求时，如果所有的小的空闲分区的容量总和大于用户的要求，这时便须对内存进行“紧凑”，将经“紧凑”后所得到的大空闲分区分配给用户。如果所有的小的空闲分区的容量总和仍小于用户的要求，则返回分配失败信息。

### 基于顺序搜索的动态分区分配算法

1. 首次适应算法

空闲分区（链）按地址递增的次序排列。在进行内存分配时,从空闲分区表/链首开始顺序查找,直到找到第一个满足其大小要求的空闲分区为止。然后再按照作业大小，从该分区中划出一块内存空间分配给请求者，余下的空闲分区仍留在空闲分区表（链）中。



1. 循环首次适应算法

循环首次适应算法又称为下次适应算法，由首次适应算法演变而来。在为作业分配内存空间时,不再每次从空闲分区表/链首开始查找,而是从上次找到的空闲分区的下一个空闲分区开始查找,直到找到第一个能满足其大小要求的空闲分区为止。然后，再按照作业大小，从该分区中划出一块内存空间分配给请求者，余下的空闲分区仍留在空闲分区表/链中。



1. 最佳适应算法

空闲分区表/链按容量大小递增的次序排列。在进行内存分配时，从空闲分区表/链的首开始顺序查找，直到找到第一个满足其大小要求的空闲分区为止。

1. 最坏适应算法

空闲分区表/链按容量大小递减的次序排列。在进行内存分配时，从空闲分区表/链的首开始顺序查找，直到找到第一个比之大的空闲分区为止。剩下的空闲仍留在空闲分区表/链中。

# 虚拟存储器

## 5.1 虚拟存储器概述

### 5.1.1 引入虚拟存储器的原因？

有的作业很大，其要求的内存空间超过了内存总容量，做也不能全部装入内存而运行；多道作业要求运行，但由于内存容量不足以容纳所有这些作业，导致大量作业在外存等待。这种一次性装入，以及作业装入后，即使部分模块执行完成但整个作业没有完成，还是会一直驻留在内存中，占用了大量内存空间。虚拟内存就是解决这种大作业的运行问题，主要目的还是解决内存不足的问题。

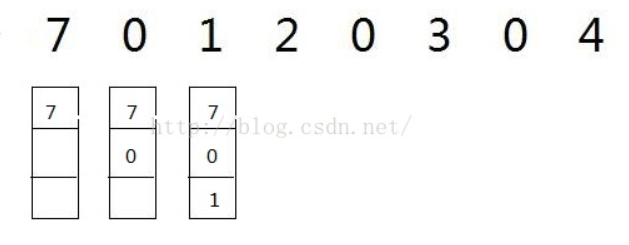
## 5.2 页面置换算法

### 5.2.1 LUR置换算法

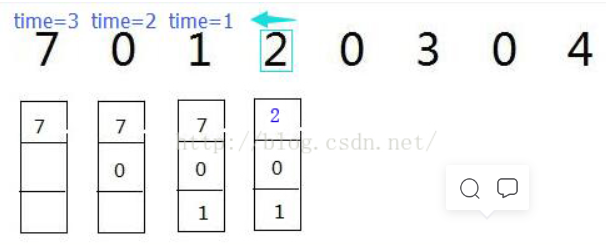
最近最少使用置换法。顾名思义，就是当需要置换一页时，在[内存](https://so.csdn.net/so/search?q=%E5%86%85%E5%AD%98&spm=1001.2101.3001.7020" \t "_blank)块中找到一个最久没有使用的页面予以淘汰并进行置换的算法。



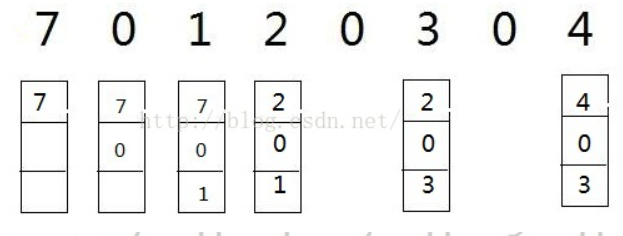
1. 先把所有的内存块装满。



1. 我们先设请求块的数组为a[ ]，则a[0]=7,a[1]=0,a[2]=1,a[3]=2 ... ...    此时，a[3]=2在内存块中不存在，这时候就要进行页面置换。从a[3]往前看，找到time=3这个最大时间值对应的内存块为：7，表示内容为“”7“”的内存块停留的时间最久，需要替换掉。

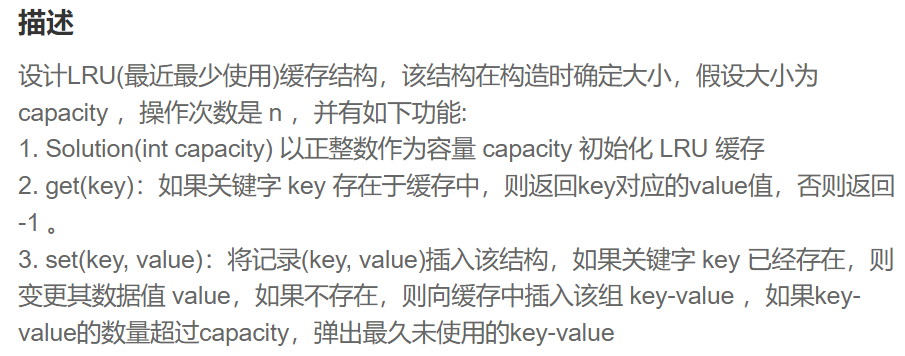


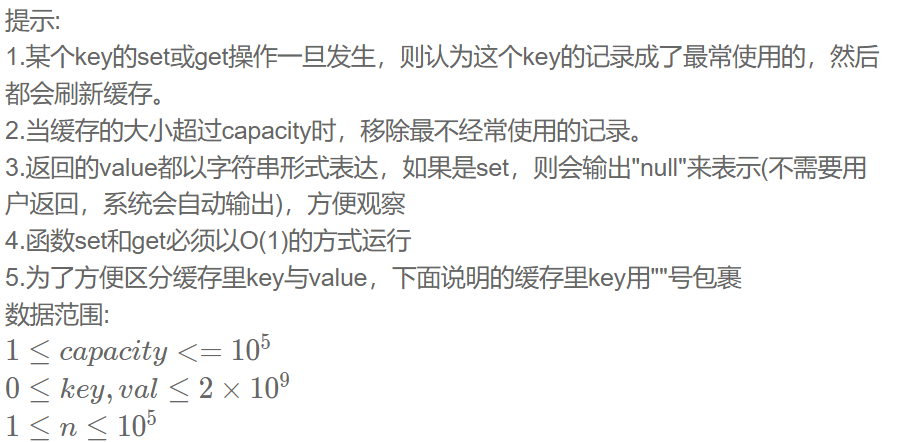
1. 如果请求块的当前数据在内存块中已存在，则跳过；如果不存在，则进行页面置换。置换的后面步骤以此类推。置换结束后的最终结果如下：



所以，依次发生置换的页面号是：7、0、1、2、3、4 ，缺页数为：5

**设计LRU缓存结构**

****

****

**解析**

使用到的数据结构：hashMap+ 双向链表

运用Map是因为get 和set都是O(1),运用双向链表是因为插入和删除都是O(1)

同时还需要维护两个指针，一个指向最近一次调用的节点，一个指向最久未调用的节点。

维护最近一次掉用节点，是为了在有新的get/set执行时，能迅速找到要查入的位置

维持最久未调用，则是为了当超出节点数量时删除

/\*\*

\* @param {number} capacity

\*/

function ListNode (val){

this.val = val;

this.key = null;

this.next = null;

this.pre = null;

}

var Solution = function(capacity) {

// write code here

this.map = new Map();

this.max = capacity;

this.latestUsedPtr = null; //最近调用

this.oldestUnusedPtr = null; //最久未调用

};

/\*\*

\* @param {number} key

\* @return {number}

\*/

Solution.prototype.get = function(key) {

// write code here

if(this.map.has(key)){

let valueNode = this.map.get(key);

// 如果当前被get的节点不是最近一次调用过的，则将其提到第一个。

if(valueNode.pre){

// 如果当前节点是最久未调用，将最久未调用设为前一个

if(!valueNode.next && valueNode.pre){

this.oldestUnusedPtr = valueNode.pre;

}

//将该节点的前后节点连接起来，以维持链表结构

valueNode.pre.next = valueNode.next;

if(valueNode.next)

valueNode.next.pre = valueNode.pre;

//将该节点插入链表头

valueNode.next = this.latestUsedPtr;

this.latestUsedPtr.pre = valueNode;

valueNode.pre = null;

this.latestUsedPtr = valueNode;

}

return valueNode.val;

}

else{

return -1;

}

};

/\*\*

\* @param {number} key

\* @param {number} value

\* @return {void}

\*/

Solution.prototype.set = function(key, value) {

// write code here

let nodeToInsert = new ListNode(value);

nodeToInsert.key = key;

// 超出限额，拿掉最久未调用节点

if(this.map.size === this.max && !this.map.has(key)){

let preNode = this.oldestUnusedPtr.pre;

preNode.next = null;

this.map.delete(this.oldestUnusedPtr.key);

this.oldestUnusedPtr = preNode;

}

// 已经存在该key

if(this.map.has(key)){

let curNode = this.map.get(key);

//将旧节点从链表中移除

if(curNode.pre){

curNode.pre.next = curNode.next;

}

//将旧节点从表中移除

this.map.delete(key);

}

// 将其添加到第一个，为最近掉用节点

this.map.set(key,nodeToInsert);

nodeToInsert.pre = null;

nodeToInsert.next = this.latestUsedPtr;

if(this.latestUsedPtr)

this.latestUsedPtr.pre = nodeToInsert;

if(nodeToInsert.next)

nodeToInsert.next.pre = nodeToInsert;

this.latestUsedPtr = nodeToInsert;

// 如果只剩一个节点。

if(!this.oldestUnusedPtr)

this.oldestUnusedPtr = nodeToInsert;

return

};

**常思佳：**

1. **详述操作系统的发展过程**

**单道批处理系统**

工作原理：

首先由监督程序将磁带上的第一个作业装入内存，并把运行控制权交给该作业；当该作业处理完成时，又把控制权交还给监督程序，再由监督程序把磁带上的第二个作业调入内存。计算机系统就这样自动地一个作业紧接一个作业地进行处理，直至磁带上的所有作业全部完成，这样便形成了早期的批处理系统。虽然系统对作业的处理是成批进行的，但在内存中始终只保持一道作业，故称为单道批处理系统。

目标：

（1）提高系统资源的使用效率；

（2）提高作业吞吐量。

优点：

1. 实现了自动性、顺序性、单道性，执行比较快；
2. 减少了人工操作，解决了作业的自动续接。

缺点：

（1）系统资源达不到充分利用,内存仅有一道程序。每逢该程序在运行中发出I/O请求后，CPU便处于等待状态，必须在其I/O完成后才继续 运行。又因IO设备的低速性，更使CPU的利用率显著降低；

（2）平均周转时间长，没有交互能力。

**多道批处理系统**

工作原理：

在该系统中，用户所提交的作业先存放在外存上，并排成一个队列，称为“后备队列”。然后由作业调度程序按一定的算法，从后备队列中选择若干个作业调入内存，使它们共享CPU和系统中的各种资源。由于同时在内存中装有若干道程序,这样便可以在运行程序A时,利用其因I/O操作而暂停执行时的CPU空档时间,再调度另一道程序B运行，同样可以利用程序B在I/O操作时的CPU空档时间，再调度程序C运行，使多道程序交替地运行，这样便可以保持CPU处于忙碌状态。

目标：

（1）提高系统资源的使用效率；

（2）提高作业吞吐量。

优点：

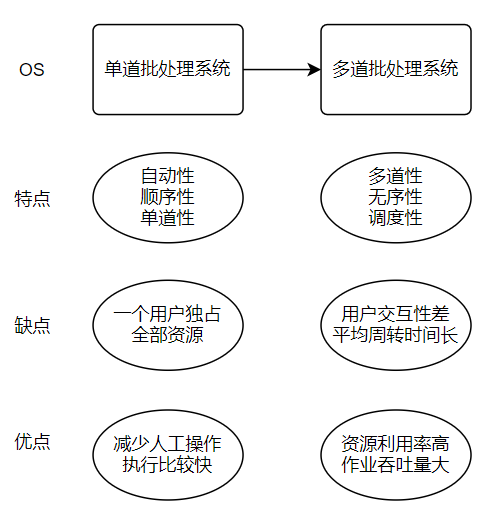
（1）资源利用率高：多道程序交替运行，cpu处于忙碌状态,提高了内存和I/O的利用率；

（2）系统吞吐量大：cpu与其他资源保持忙碌状态,仅当作业完成或运行受阻才切换,开销小。

缺点：

（1）平均周转时间长：作业需要排队等待时间长；

（2）无交互能力：用户一旦把作业提交给系统后，直至作业完成，用户都不能与自己的作业进行交互，修改和调试程序极不方便。



**分时操作系统**

工作原理：

一台主机连接了若干个终端，每个终端有一个用户在使用。用户交互式地向系统提出命令请求，系统接受每个用户的命令，采用时间片轮转方式处理服务请求，并通过交互方式在终端上向用户显示结果。用户根据上步结果发出下道命。分时操作系统将CPU的时间划分成若干个片段，称为时间片。操作系统以时间片为单位，轮流为每个终端用户服务。每个用户轮流使用一个时间片而使每个用户并不感到有别的用户存在。分时系统具有多路性、交互性、“独占”性和及时性的特征。多路性指，伺时有多个用户使用一台计算机，宏观上看是多个人同时使用一个CPU，微观上是多个人在不同时刻轮流使用CPU。交互性是指，用户根据系统响应结果进一步提出新请求(用户直接干预每一步)。“独占”性是指，用户感觉不到计算机为其他人服务，就像整个系统为他所独占。及时性指，系统对用户提出的请求及时响应。它支持位于不同终端的多个用户同时使用一台计算机，彼此独立互不干扰，用户感到好像一台计算机全为他所用。

目标：

(1)对用户的请求及时响应；

(2)尽量提高系统资源的利用率；

(3)支持人机交互与资源共享。

特点：

（1）多路性。该特性是指系统允许将多台终端同时连接到一台主机上，并按分时原则为每个用户服务。多路性允许多个用户共享一台计算机，显著地提高了资源利用率，降低了使用费用，从而促进了计算机更广泛的应用。

（2）独立性。该特性是指系统提供了这样的用机环境，即每个用户在各自的终端上进行操作，彼此之间互不干扰，给用户的感觉就像是他一人独占主机进行操作。

（3）及时性。及时性是指用户的请求能在很短时间内获得响应。这一时 间间隔是根据人们所能接受的等待时间确定的，通常仅为1～3秒钟。

（4）交互性。交互性是指用户可通过终端与系统进行广泛的人机对话。 其广泛性表现在：用户可以请求系统提供多方面的服务，如进行文件编 辑和数据处理，访问系统中的文件系统和数据库系统，请求提供打印服 务等。



**实时操作系统**

工作原理：

当外界事件或数据产生时，能够接受并以足够快的速度予以处理，其处理的结果又能在规定的时间之内来控制生产过程或对处理系统做出快速响应，调度一切可利用的资源完成实时任务，并控制所有实时任务协调一致运行的操作系统。

目标：

提高系统的响应时间，对随机发生的外部事件作出及时响应并对其进行处理。

优点：

（1）及时性。快速的响应时间，不同的系统要求不一样，控制对象必须在截止时间内完成；

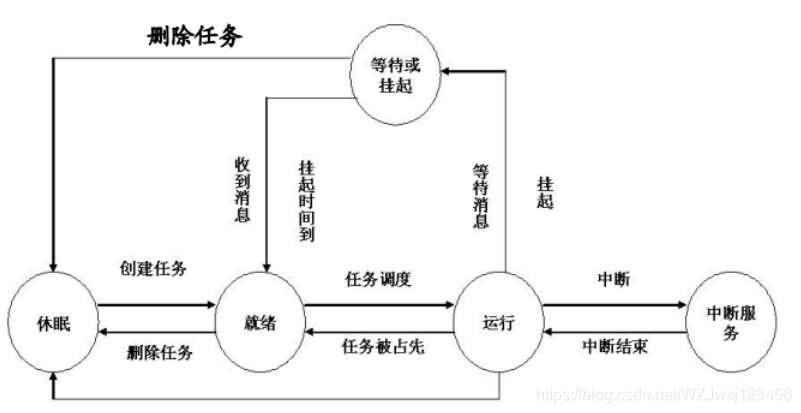
（2）高可靠性。具有过载防护能力；

（3）多路性。能对多个对象进行控制；

（4）独立性。独立运行，彼此之间互不干扰，不混淆，不破坏；

缺点：

有限的交互能力。仅限于访问系统中某些特定的专用服务程序。



**分时和实时的区别**

|  |  |  |
| --- | --- | --- |
|  | 分时系统 | 实时系统 |
| 多路性 | 按分时原则为多个终端用户服务 | 对多路的现场信息进行采集及对多个对象或多个执行机构进行控制 |
| 独立性 | 每个终端用户在向实时系统提出服务请求时，是彼此独立的操作，互不干扰 | 每个终端用户在向实时系统提出服务请求时，是彼此独立的操作，互不干扰；  对信息的采集和对象的控制也是彼此互不干扰的 |
| 及时性 | 用户的请求能在短时间（用户能接受的时间）内获得响应 | 以控制对象所要求的开始截止时间或完成截止时间来确定的，一般为秒级、百毫秒直至毫秒级，甚至有的要低于100μs |
| 交互性 | 能向终端用户提供数据处理服务、资源共享等服务 | 仅限于访问系统中某些特定的专用服务程序 |
| 可靠性 | 可靠程度低 | 要求系统高度可靠，采取了多级容错措施来保证系统的安全及数据的安全 |

**网络操作系统**

网络操作系统把计算机网络中的各台计算机有机结合，提供一种统一，经济并且有效的使用各台计算机的方法，实现各个计算机之间的互相传送数据。



**分布式操作系统**

分布式计算机系统是以计算机网络为基础的计算机系统，包含多台处理机，每台处理机完成系统中指定的一部分功能。从硬件上讲，它与计算机局域网没有任何区别，关键是软件 。

特征：

（1）以计算机网络为基础的统一操作系统；

（2）处理上的分布，即功能和任务的分布。所有系统任务可在系统中任何处理机上运行，自动实现全系统范围内的任务分配并自动调度各处理机的工作负载。

（3）①资源共享、透明

②自治性

③高可靠性

**分布式和网络的区别**

（1) 分布式系统的各个计算机之间处于平等地位无主从关系；网络有一些主从关系；

（2) 分布式系统资源为所有用户共享；而网络有限制地共享；

（3) 分布式系统中一项任务可分给若干处理单元相互协作共同完成，而网络往往是各司其责。

**多处理操作系统**

多处理机系统是由多台处理器组成的计算机系统。多处理操作系统的出现是为了提高计算机系统性能和可靠性。提高性能有两条途径：提高各个组成部分的速度、增大处理的并行程度。

特点：

（1）增加系统的吞吐量：N个处理器加速比达不到N倍（额外的调度开销，算法的并行化）；

（2）提高系统可靠性：故障时系统降级运行。

**嵌入式操作系统**

以应用为中心，以计算机技术为基础，软硬件可裁剪，适应应用系统对功能、可靠性、成本、体积、功耗等严格要求的专用计算机系统。通常，嵌入式系统是一个控制程序存储在ROM中的嵌入式处理器控制板，如手表、微波炉、录像机、汽车等。有些嵌入式系统还包含操作系统，但大多数嵌入式系统都是由单个程序实现整个控制逻辑。

1. **进程通信的三种方式**

**一．共享存储**

1，什么是共享存储？

操作系统提供了一个共享空间可以被进程1,2互斥的访问

2，分类

1）基于数据结构的共享：比如共享空间只能存放一个长度为10的数组。这种共享方式速度慢，限制多，是一种低级的通信方式

2）基于存储区的共享：在内存中画出一块共享存储区，数据的形式，存放的位置都由进程控制，而不是操作系统。相比之下它的速度更快，是一种高级通信方式

**二．消息传递**

1，什么是消息传递？

进程中的数据交换以“格式化消息“为单位，进程通过操作系统提供的”发送消息、接收消息“两个源语进行数据交换”

2，格式化消息的结构：

1）消息头：包括发送进程的ID，接收进程的ID，消息类型，消息长度等格式化信息

2）消息体：记录被发送的信息

3，分类：

1）直接通信方式：比如进程1，给进程2发信息，就把信息挂到进程2的信息缓存队列上

2）间接通信方式：进程1把消息放到一个中间实体上，等待需要它的进程2取走

**三．管道通信**

1，什么是管道：是指用于连接读写进程的一个共享文件，是在内存中开辟了一个固定大小的缓冲区

2，特点：

1）管道只能采用半双工通信，在一段时间只能单向传输，如果要实现双向传输，就需要设置两个管道

2）各个进程需要互斥的访问管道

3）数据以字符流的形式写入管道，当管道写满时，写进程将被堵塞，等待数据被取走。当数据全部被取出后，管道变空，此时写进程的系统调用将被堵塞。

4）如果没写满，就不允许读，没有读空，就不允许写

5）速度一旦读出，就从管道中被抛弃，这就意味着读进程只能有一个，否则可能会发生读错数据的情况

**3.利用AND信号量解决生产者-消费者问题**

用Swait(empty,mutex)来代替Wait(empty)和Wait(mutex);

用Ssignal(mutex,full)来代替Signal(empty)和Signal(mutex);

用Swait(full,mutex)来代替Wait(full)和Wait(mutex);

用Ssignal(mutex,empty)来代替Signal(mutex)和Signal(empty);

**生产者producer类**

int in=0,out=0;

item buffer[n];

semaphore mutex=1,empty=n,full=0;

void producer(){

do{

producer an item nextp;

...

**Swait(empty,mutex);**

//当empty小于等于0，也就是没有空缓冲区时

//或者信号量小于0，则阻塞当前进程

//同时对两个信号量进行操作

buffer[in]=nextp; //缓冲区，nextp暂存当前生产的产品

in=(in+1)%n;

//因为是缓冲区是循环队列

//所以in指针进行取模运算来判断队列是否已满

**Ssignl(mutex,full);**

//唤醒等待队列中因wait(mutex,full)阻塞的进程

//释放资源

}while{TRUE};

}

**消费者consumer类**

void consumer(){

do{

**Swait(full,mutex);**

//如果缓冲区中没有产品可分配

//或者信号量小于0，则阻塞当前进程

//同时对两个信号量进行操作

nextc=buffer[out]; //缓冲区，nextc暂存当前要消费的产品

out=(out+1)%n;

//因为是缓冲区是循环队列

//所以out指针进行取模运算

**Ssignal(mutex,empty);** //释放资源

consumer the item in nextc;

}

**}**

**4.有两个并发执行的进程 P1 和 P2，共享初值为 1 的变量 x。P1 对 x 加 1，P2 对 x 减 1。加1 和减 1 操作的指令序列分别如下所示。**



**两个操作完成后，x 的值A．可能为（C）**

**A. 可能为-1 或 3**

**B. 只能为 1**

**C. 可能为 0、1 或 2**

**D. 可能为-1、0、1 或 2**

**解析：**由于对两个并发进程P1与P2没有先后进行的要求，因此会出现三种情况：

①P1先执行，P2后执行。

P1进程：x→R1=1，x=x+1=2，R1→x=2

P2进程：x→R2=2，x=x-1=1，R2→x=1

所以，两个操作完成后，x的值为1；

②P2先执行，P1后执行。

P2进程：x→R2=1，x=x-1=0，R2→x=0

P1进程：x→R1=0，x=x+1=1，R1→x=1

所以，两个操作完成后，x的值也为1；

③P1、P2并发进行。

x→R1=1，x→R2=1

x=x+1=2，x=x-1=0

以上是两个进程并发而进行的步骤，而进程的最后一步无法并发进行，因而会出现：

R1→x=2，R2→x=0

所以，两个操作完成后，x的值可能为2，也可能为0；

所以综合以上三种情况可知x的值可能为0、1或2。

**知识点：**

****并发执行的问题****  
 由于进程的相对执行速度不可预测，因而  
   1. 全局资源的共享很危险。多个进程都对同一全局变量读写，后果很 麻烦。  
   2. 操作系统很难对资源进行最优分配。比如被占用I/O的进程被阻塞。  
   3. 定位程序设计的错误很困难。因为结果的不确定性和不可再现性。

**5.中断处理和子程序调用都需要压栈以保护现场，中断处理一定会保存而子程序调用不需要保存其内容的是（ B ）**

**A. 程序计数器**

**B. 程序状态字寄存器**

**C. 通用数据寄存器**

**D. 通用地址寄存器**

**解析：**

当系统有中断信号产生时，CPU立即响应并开始执行中断处理程序。中断处理程序执行完后，返回程序断点处继续执行。为了保证程序的正确执行，在中断处理前后必须保存和恢复被中断的程序现场。需要保存和恢复的现场信息包括：程序计数器、通用寄存器以及程序状态字寄存器等关键寄存器的内容。

程序状态字(PSW)一般包括条件码、中断允许/禁止位、内核/用户态位。程序状态字的内容在进入中断处理程序时可能会发生改变，而在子程序调用时不会发生改变。所以为了能够恢复现场，进入中断处理程序前必须保存程序状态字，而且一般由硬件保存。

程序状态字寄存器用于记录当前处理器的状态和控制指令的执行顺序，并且保留与运行程序相关的各种信息，主要作用是实现程序状态的保护和恢复。所以中断处理程序要将PSW保存，子程序调用在进程内部执行，不会更改PSW，所以不需要保存程序状态字寄存器的内容，只需要将局部变量、参数以及返回地址压入堆栈。

**知识点：**

**保护现场和恢复现场:**

如果主程序中断时，有些寄存器还存放着中间结果，而子程序很有可能要占用这些寄存器，那么就要将主程序的中间结果存入其他地方(比如堆栈)，把寄存器“让给”子程序，这个过程就是保护现场。待子程序/中断服务函数完毕，再将主程序的中间结果放回寄存器，这个过程就好似恢复现场。

**子程序/中断调用过程:**

当主程序转去执行子程序/中断时，首先将端口地址压入堆栈保存，然后将子程序的入口地址送入PC。进入子程序后首先保护现场，然后执行子程序，子程序执行完毕后，恢复现场，然后将端口地址从栈顶送回PC。

**6.安全性算法过程描述**

（1)设置两个向量：

①工作向量Work：它表示系统可提供给进程继续运行所需的各类资源数目，它含有m个元素，在执行安全算法开始时，Work=Available；

②Finish：它表示系统是否有足够的资源分配给进程，使之运行完成。开始时先做Finish[i]=false；当有足够资源分配给进程时，再令Finish[i]=true。

（2）从进程集合中找到一个能满足下述条件的进程：

① Finish[i]=false;

② Need[i,j]≤Work[j]；若找到，执行步骤（3)，否则，执行步骤（4)。

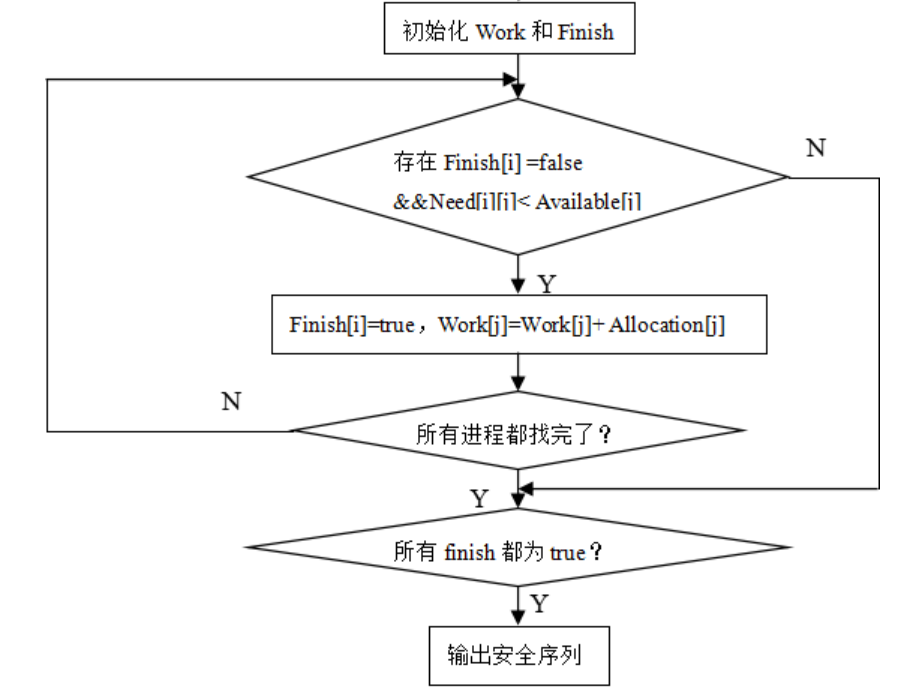
（3）当进程Pi获得资源后，可顺利执行，直至完成，并释放出分配给它的 资源，故应执行：

Work[j]=Work［i］+Allocation[i,j];

Finish[i]=true;

go to step 2;

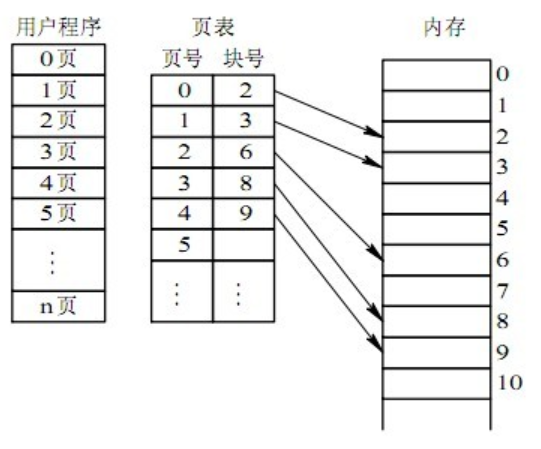
（4）如果所有进程的Finish[i]=true都满足，则表示系统处于安全状态； 否则，系统处于不安全状态。



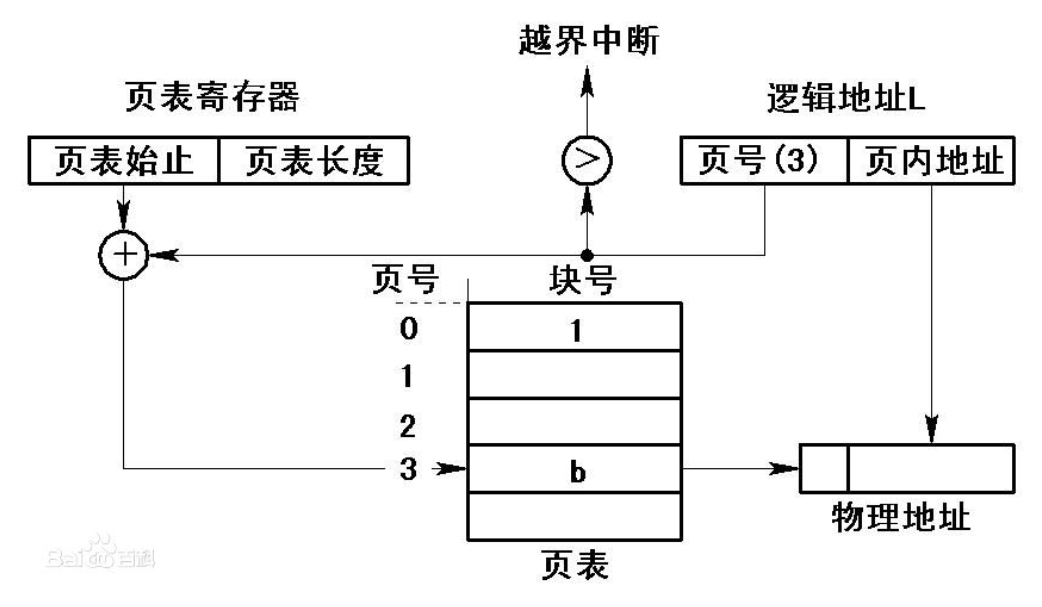
**7.非连续分配管理方式**

**基本分页储存管理方式**

进程会被固定单位的空间划分成块，一个块称之为页，内存也被这个单位划分成块，一个块称之为页框，外存也以同样单位划分成块，称之为块。进程在执行时需要申请主存空间，也就是为每个页面分配主存中可用的页框，这就产生了页和页框一一对应的关系。于是为了方便在内存中找到进程的每个页面所对应的物理块号，系统为每个进程建立一张页表，记录页面在内存中对应的物理块号，页表一般存放在内存中。

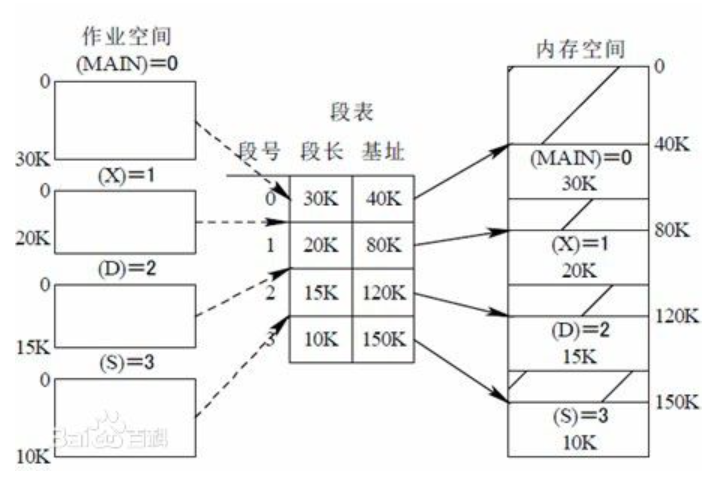


基本地址变换机构

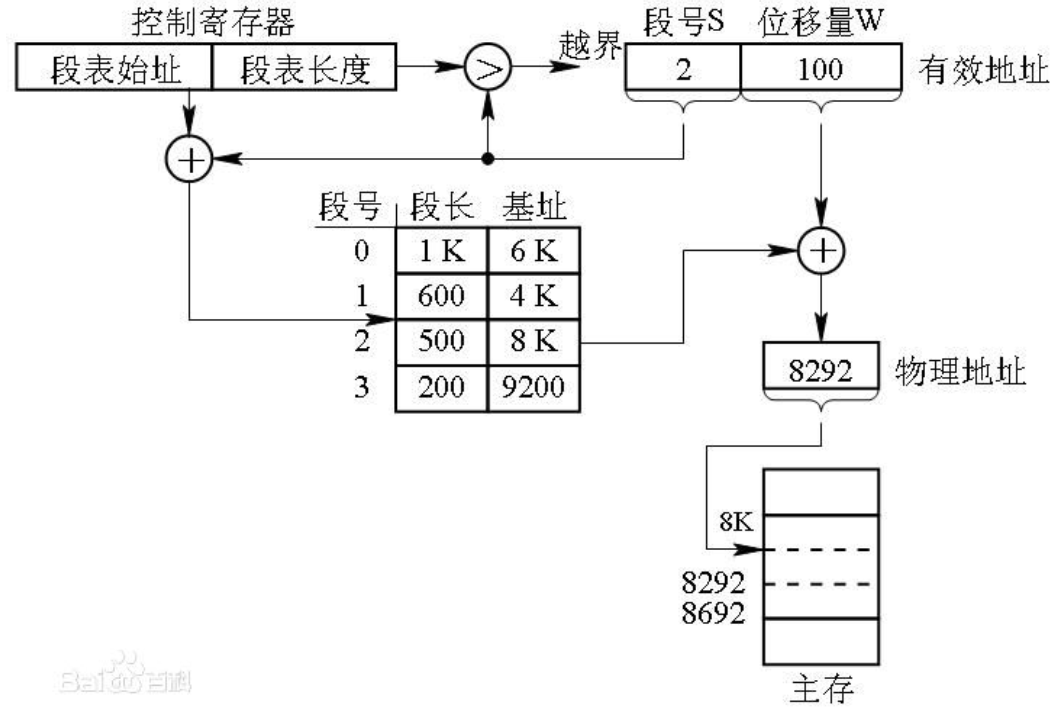


**基本分段储存管理方式**

段式管理方式按照用户进程中的自然段划分逻辑空间，如一用户进程由主程序，两个子程序，桟和一段数据组成，可以分为5段，每段从0开始编址，并分配一段连续连续地址空间（段内要求连续，段间不要求连续）。每个进程都有一张逻辑空间与内存映射的段表。段号和段内偏移都必须有由用户显示提供，高级语言中由编译程序完成这个操作。



基本地址变换机构



**优缺点对比**

|  |  |  |
| --- | --- | --- |
| 存储管理方式 | 优点 | 缺点 |
| 分页 | 内存空间利用率高，不会产生外部碎片，只会有少量的页内碎片 | ①不方便按照逻辑模块实现信息的共享和保护  ②程序全部装入内存，要求有相应的硬件支持 |
| 分段 | 可以分别编写和编译，可以针对不同类型的段采取不同的保护，可以按段为单位进行共享，包括通过动态链接进行代码共享（很方便按照逻辑模块实现信息的共享和保护） | ①如果段长过大，为其分配很大的连续空间会很不方便 ②会产生外部碎片 |

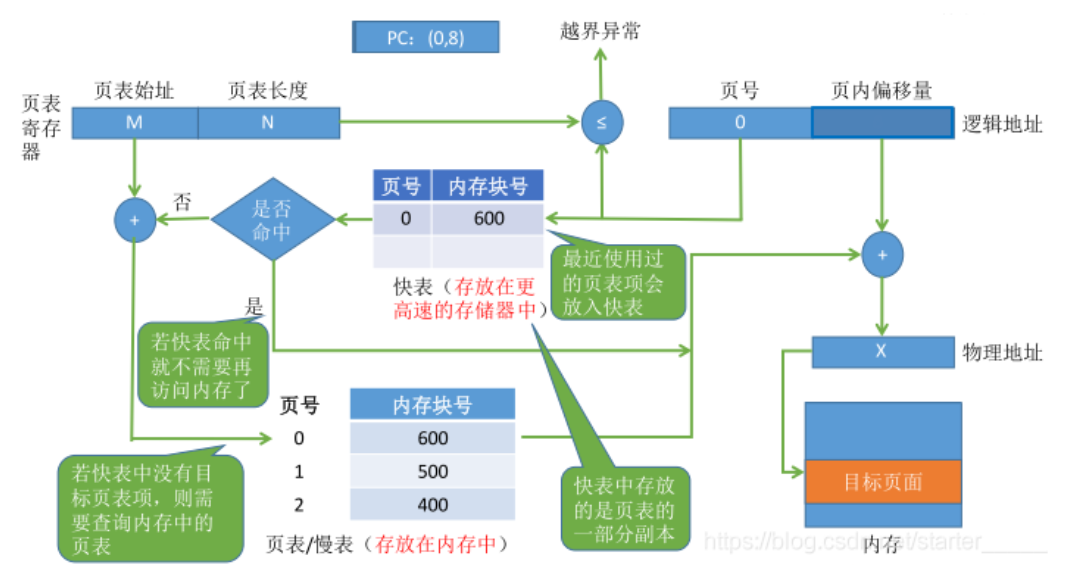
**段页式存储管理方式**

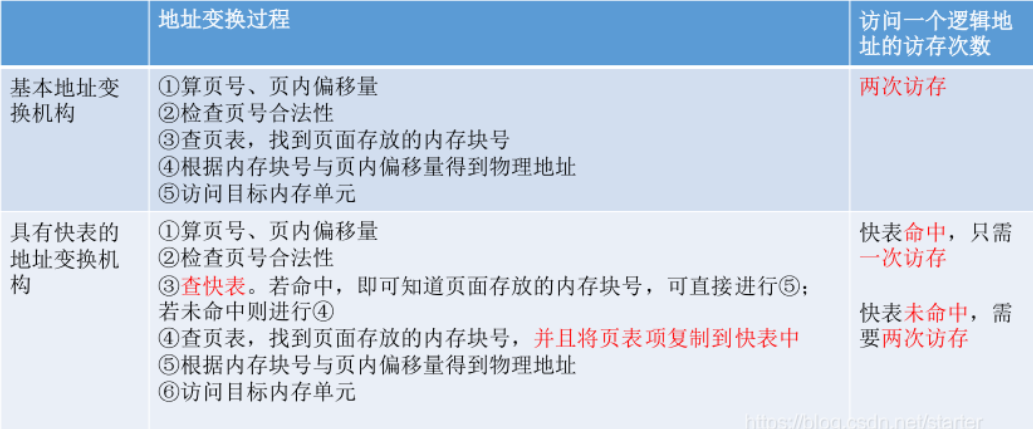
作业的地址空间首先被分成若干个逻辑段，每段都有自己的段号，然后再将每一段分成若干个大小固定的页。



**快表（TLB）**

快表又成为联想寄存器（TLB），是一种访问速度比内存块很多的高速缓冲存储器，用来存放当前访问的若干页表项, 以加速地址变换的过程. 与此对应的，内存中的页表常称为慢表。

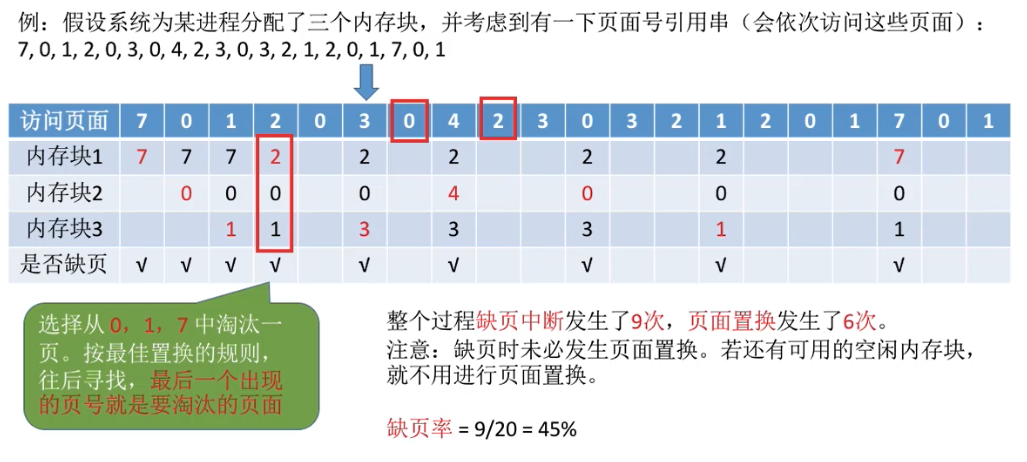




**8.最佳置换算法（OPT）**

最佳置换算法每次选择淘汰的页面将是以后永不使用，或者在最长时间内不再被访问的页面，这样可以保证最低的缺页率。

最佳置换算法可以保证最低的缺页率，但实际上，只有在进程执行的过程中才能知道接下来会访问到的是哪个页面。操作系统无法提前预判页面访问序列。因此，最佳置换算法是无法实现的。

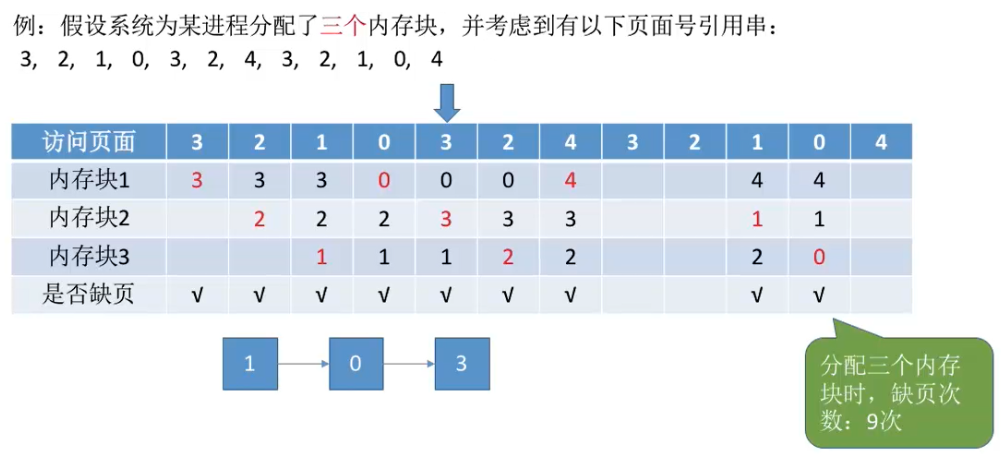


只有内存块已经都满的情况下，才需要页面置换。因此，在刚开始访问7、0、1这三个页面的时候，虽然它们都没在内存当中，但是由于刚开始那些有空闲的内存块，所以虽然发生了缺页中断，虽然会发生掉页，但是并不会发生页面置换这件事，只有所有的内存块都已经占满之后，再发生缺页的话，那才需要进行页面置换这件事情，因此缺页中断总共发生了9次，但页面置换只发生了6次，前面那三次只是发生了缺页，并没有页面置换，那缺页率的计算也很简单，只需要将缺页中断发生的次数除以总共访问了多少次的页面，就可以得到缺页率。

**9.先进先出置换算法（FIFO）**

先进先出置换算法每次选择淘汰的页面是最早进入内存的页面

实现方法：把调入内存的页面根据调入的先后顺序排成一个队列，需要换出页面时选择队头页面即可。队列的最大长度取决于系统为进程分配了多少个内存块。





Belady异常：当为进程分配的物理块数增大时，缺页次数不减反增的异常现象。

只有FIFO算法会产生Belady异常。另外，FIFo算法虽然实现简单，但是该算法与进程实际运行时的规律不适应，因为先进入的页面也有可能最经常被访问。因此，算法性能差。

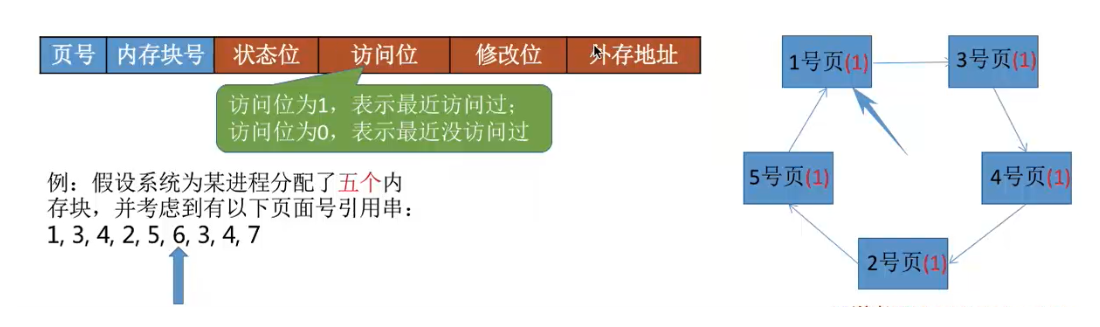
**10.时钟置换算法（CLOCK)**

最佳置换算法性能最好，但无法实现；先进先出置换算法实现简单，但算法性能差；最近最久未使用置换算法性能好，是最接近OPT算法性能的，但是实现起来需要专门的硬件支持，算法开销大。

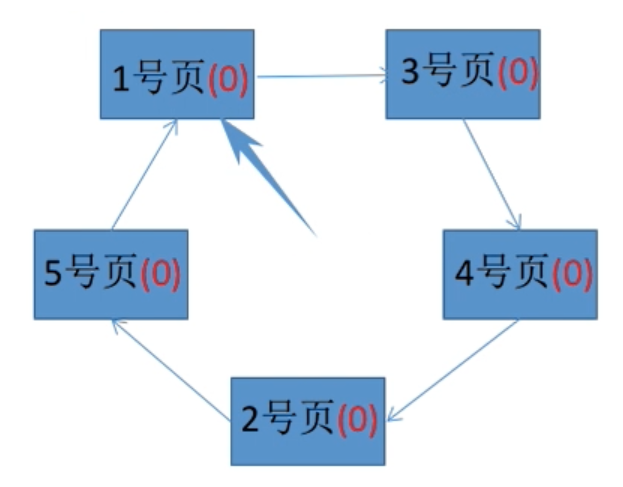
时钟置换算法是一种性能和开销较均衡的算法，又称CLOCK算法，或最近未用算法(NRU)。

简单的CLOCK算法实现方法：为每个页面设置一个访问位，再将内存中的页面都通过链接指针链接成一个循环队列。当某页被访问时，其访问位置为1。当需要淘汰一个页面时，只需检查页的访问位。如果是0，就选择该页换出；如果是1，则将它置为0，暂不换出，继续检查下一个页面，若第一轮扫描中所有页面都是1，则将这些页面的访问位依次置为0后，再进行第二轮捆描（第二轮扫描中一定会有访问位为0的页面，因此简单的CLOCK算法选择一个淘汰页面最多会经过两轮扫描)。

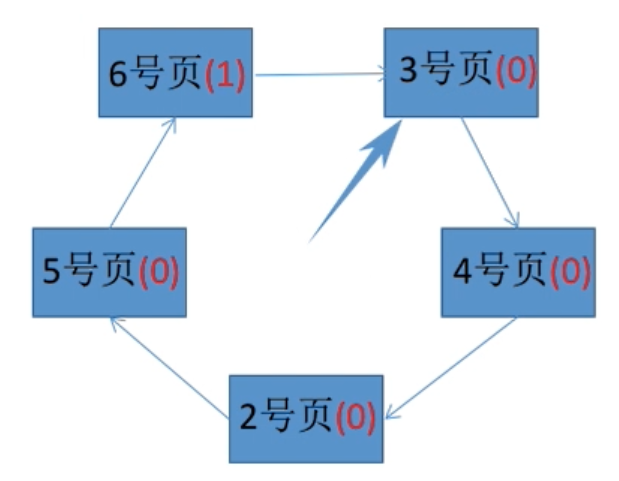
刚开始由于有五个空闲的内存块，所以前五个访问的这个页号13425都可以顺利地放入内存当中，只有在访问到6号页的时候，才需要考虑淘汰某个页面，那么在内存当中的13425这几个页面，会通过链接指针的方式连接成一个这样的循环队列。



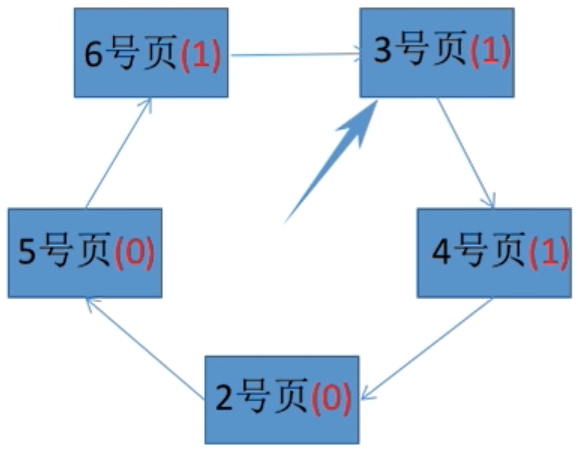
那由于接下来要访问六号页面，但是把五个内存块都已经满了，所以需要用clock算法来选择淘汰其中的某个页面。于是会从这个循环队列的队首开始扫描，尝试找到一个访问位为零的页面，并且被扫描过的页面需要把访问为1改为0，所以在经过第一轮的扫描之后，所有的这些页面的访问位都由1置为了0。



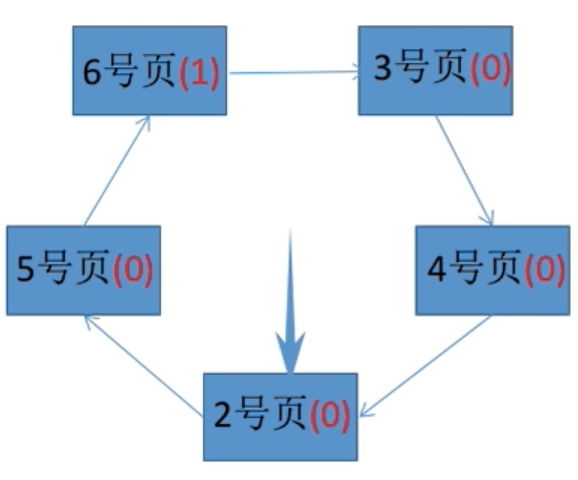
在进行第二轮扫描的时候，1号页面的访问位为0，所以会选择淘汰1号页面，于是6号页会装入到1号以前占有的这个内存块当中，并且6号页的访问位会被置为1，然后这个扫描的指针指向下一个页面。



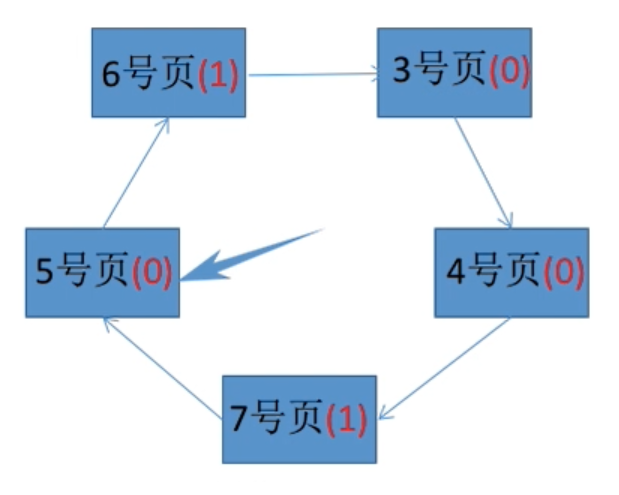
那接下来的访问当中会依次访问到3号和4号页面，那在访问了3号页面你们之后，3号页的访问位需要由0变为1，同样的，在访问了四号页面的时候，需要把4号页的访问位由0变为1。



在之后需要访问7号页，由于此时7号也没有在内存中，所以需要选择淘汰其中的某一个页面，然后把7号也放到内存中，那同样的需要从此时扫描到的这个位置依次的扫描，找到第一个访问位为0的页面，并且扫描过的那些页面的这个访问位需要由1变为0，所以3号和4号在扫描过后，访问位会变为0。



再扫描到2号页面的时候，我发现2号页面的访问位给本来就是0了，因此会选择淘汰2号页面，然后让7号页面放到这个位置，访问位置为1，然后这个扫描的指针再指向下一个页面的位置。

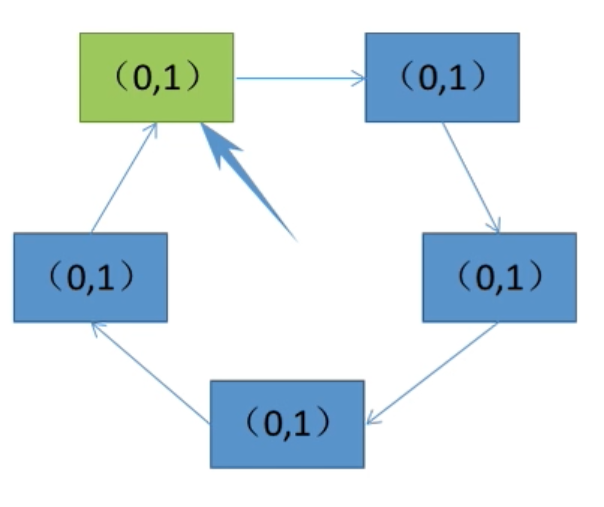


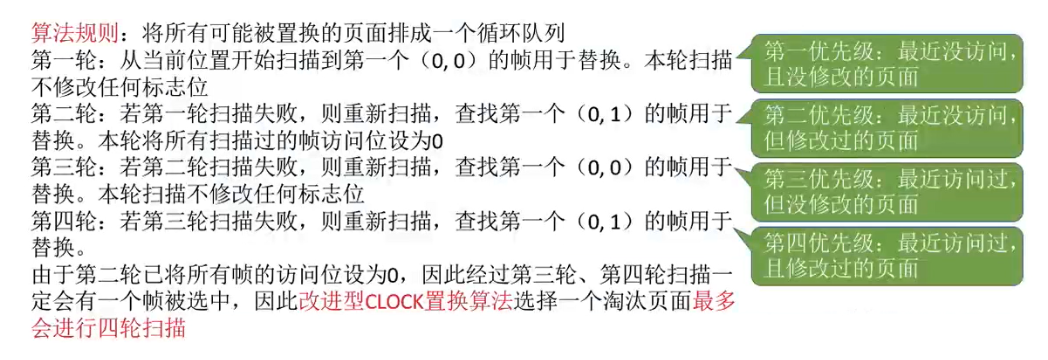
**改进型的时钟置换算法**

简单的时钟置换算法仅考虑到一个页面最近是否被访问过。事实上，如果被淘汰的页面没有被修改过，就不需要执行I/O操作写回外存。只有被淘汰的页面被修改过时，才需要写回外存。

因此，除了考虑一个页面最近有没有被访问过之外，操作系统还应考虑页面有没有被修改过。在其他条件都相同时，应优先淘汰没有修改过的页面，避免I/O操作。这就是改进型的时钟置换算法的思想。修改位=0，表示页面没有被修改过；修改位=1，表示页面被修改过。

为方便讨论，用==（访问位，修改位）==的形式表示各页面状态。如(1，1）表示一个页面近期被访问过，且被修改过。





**页面置换算法总结**

