# 操作系统的内存管理

### 内存管理的概念

操作系统对内存的划分和动态分配，就是内存管理的概念。

### 内存管理的功能：

* 内存空间的分配与回收：由操作系统完成主存储器空间的分配和管理，使程序员摆脱存储分配的麻烦，提高编程效率。
* 地址转换：在多道程序环境下，程序中的逻辑地址与内存中的物理地址不可能一致，因此存储管理必须提供地址变换功能，**把逻辑地址转换成相应的物理地址**。
* 内存空间的扩充：利用虚拟存储技术或自动覆盖技术，从逻辑上扩充内存。
* 存储保护：保证各道作业在各自的存储空间内运行，互不干扰。

## 逻辑地址与物理地址

逻辑地址：CPU所生成的地址。逻辑地址是内部和编程使用的、并不唯一。

物理地址：物理地址空间是指内存中物理单元的集合，它是地址转换的最终地址，进程在运行时执行指令和访问数据最后都要通过物理地址从主存中存取。当装入程序将可执行代码装入内存时，必须通过地址转换将逻辑地址转换成物理地址，这个过程称为地址重定位。

## 分页存储

物理内存被划分为一小块一小块，每块被称为帧(Frame)。分配内存时，帧是分配时的最小单位，最少也要给一帧。在逻辑内存中，与帧对应的概念就是页(Page)。每个进程也以块为单位进行划分，进程在执行时，以块为单位逐个申请主存中的块空间。

用户程序的地址空间被划分成若干固定大小的区域称为页，相应内存空间分成若干物理块，页和块的大小相等，可将程序的任一页放入内存的任一块中，实现了离散分配，运行时一页一页读取。

### 逻辑地址

系统将程序的逻辑空间按照同样大小也划分成若干页面，逻辑页面从0开始依次编号，称为逻辑页号或相对页号，每个页面内从0开始编址称为页内地址。程序中逻辑地址有两部分：页号p和页内位移量w。

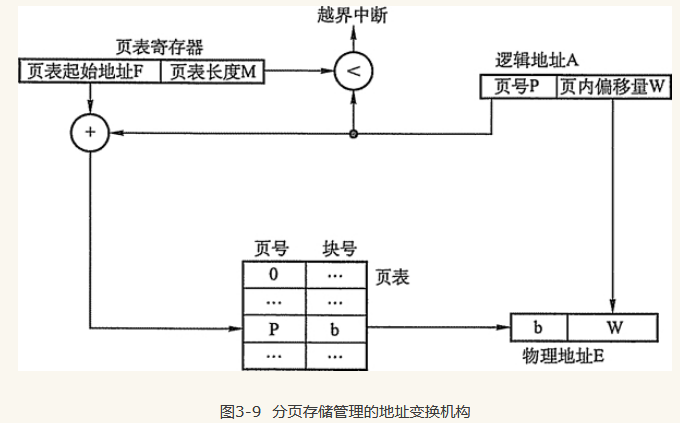
页表：为了便于在内存中找到进程的每个页面所对应的物理块，系统为每个进程建立一张页表，记录页面在内存中对应的物理块号，页表一般存放在内存中。在配置了页表后，进程执行时，通过查找该表，即可找到每页在内存中的物理块号。可见，页表的作用是实现从页号到物理块号的地址映射。

### 逻辑地址到物理地址的转换：

系统为每个进程建立一张页表，用于记录逻辑页面与内存物理页面的对应关系（页号到物理块号的地址映射）

页号：逻辑地址/页面大小   页内偏移：逻辑地址%页面大小

物理地址=物理快号\*页面大小+页内偏移



在系统中通常设置一个页表寄存器(PTR)，存放页表在内存的始址F和页表长度M。进程未执行时，页表的始址和长度存放在进程控制块中，当进程执行时，才将页表始址和长度存入页表寄存器。设页面大小为L，逻辑地址A到物理地址E的变换过程如下：

1. 计算页号P(P=A/L)和页内偏移量W (W=A%L)。
2. 比较页号P和页表长度M，若P >= M，则产生越界中断，否则继续执行。
3. 页表中页号P对应的页表项地址 = 页表起始地址F + 页号P \* 页表项长度，取出该页表项内容b，即为物理块号。
4. 计算E=b\*L+W，用得到的物理地址E去访问内存。

以上整个地址变换过程均是由硬件自动完成的。  
例如，若页面大小L为1K字节，页号2对应的物理块为b=8，计算逻辑地址A=2500 的物理地址E的过程如下：P=2500/1K=2，W=2500%1K=452，查找得到页号2对应的物理块的块号为 8，E=8\*1024+452=8644。

页面管理的优点：

* 没有外碎片
* 一个程序不必连续存放
* 便于改变程序占用空间大小

页面管理的缺点：

* 无论数据量大小都只能按照页面大小分配，容易产生内部碎片（页面可能填不满造成浪费）
* 不能体现程序逻辑
* 页长与程序的逻辑大小不相关
* 不利于编程时的独立性，并给换入换出处理、存储保护和存储共享等操作造成麻烦

## 段式存储

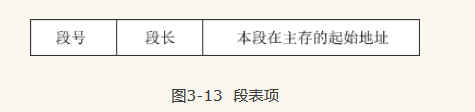
分页管理方式是从计算机的角度考虑设计的，以提高内存的利用率，提升计算机的性能, 且分页通过硬件机制实现，对用户完全透明；而分段管理方式的提出则是考虑了用户和程序员，以满足方便编程、信息保护和共享、动态增长及动态链接等多方面的需要。

段式管理方式按照用户进程中的自然段划分逻辑空间。例如，用户进程由主程序、两个子程序、栈和一段数据组成，于是可以把这个用户进程划分为5个段，每段从0 开始编址，并分配一段连续的地址空间（段内要求连续，段间不要求连续，因此整个作业的地址空间是二维的）。其逻辑地址由段号S与段内偏移量W两部分组成。

在页式系统中，逻辑地址的页号和页内偏移量对用户是透明的，但在段式系统中，段号和段内偏移量必须由用户显示提供，在髙级程序设计语言中，这个工作由编译程序完成。

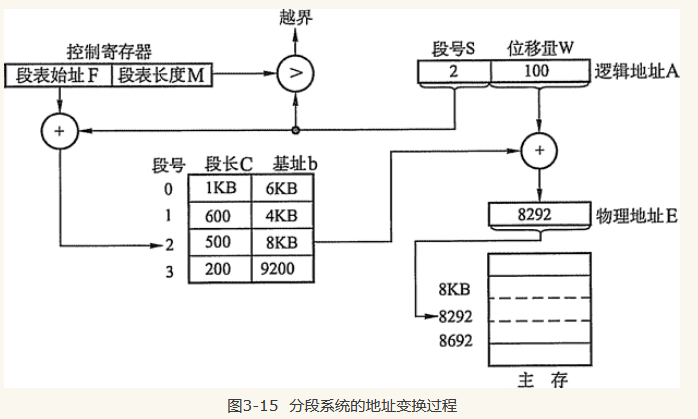
### 段表

每个进程都有一张逻辑空间与内存空间映射的段表，其中每一个段表项对应进程的一个段，段表项记录该段在内存中的起始地址和段的长度。



为了实现进程从逻辑地址到物理地址的变换功能，在系统中设置了段表寄存器，用于存放段表始址F和段表长度M。其从逻辑地址A到物理地址E之间的地址变换过程如下：

* 从逻辑地址A中取出前几位为段号S，后几位为段内偏移量W。
* 比较段号S和段表长度M，若S多M，则产生越界中断，否则继续执行。
* 段表中段号S对应的段表项地址 = 段表起始地址F + 段号S \* 段表项长度，取出该段表项的前几位得到段长C。若段内偏移量>=C，则产生越界中断，否则继续执行。
* 取出段表项中该段的起始地址b，计算 E = b + W，用得到的物理地址E去访问内存。



分段存储的优点

* 段的逻辑独立性使其易于编译、管理、修改和保护，也便于多道程序共享。
* 段长可以根据需要动态改变，允许自由调度，以便有效利用主存空间。
* 方便编程，分段共享，分段保护，动态链接，动态增长

分段存储的缺点

* 主存空间分配比较麻烦。
* 容易在段间留下许多碎片（外部碎片），造成存储空间利用率降低。
* 由于段长不一定是2的整数次幂，因而不能简单地像分页方式那样用虚拟地址和实存地址的最低若干二进制位作为段内地址，并与段号进行直接拼接，必须用加法操作通过段起址与段内地址的求和运算得到物理地址。因此，段式存储管理比页式存储管理方式需要更多的硬件支持。

## 段页式存储

段页式存储组织是分段式和分页式结合的存储组织方法，这样可充分利用分段管理和分页管理的优点。它首先将程序按其逻辑结构划分为若干个大小不等的逻辑段，然后再将每个逻辑段划分为若干个大小相等的逻辑页。主存空间也划分为若干个同样大小的物理页。辅存和主存之间的信息调度以页为基本传送单位，每个程序段对应一个段表，每页对应一个页表。

段页式地址结构:程序员按照分段系统的地址结构将地址分为段号与段内位移量，地址变换机构将段内位移量分解为页号和页内位移量。为实现段页式存储管理，系统应为每个进程设置一个段表，包括每段的段号，该段的页表始址和页表长度。每个段有自己的页表，记录段中的每一页的页号和存放在主存中的物理块。段页式系统中，作业的地址结构包含三部分的内容：段号，页号，页内位移量。CPU访问时，段表指示每段对应的页表地址，每一段的页表确定页所在的主存空间的位置，最后与页表内地址拼接，确定CPU要访问单元的物理地址。段页存储管理方式综合了段式管理和页式管理的优点，但需要经过两级查表才能完成地址转换，消耗时间多。

地址映射：

* 进行地址变换时，首先利用段号S，将它与段表长TL进行比较。若S<TL，表示未越界。
* 于是利用段表始址和段号来求出该段所对应的段表项在段表中的位置，从中得到该段的页表始址。
* 利用逻辑地址中的段内页号P来获得对应页的页表项位置，从中读出该页所在的物理块号b再利用块号b和页内地址来构成物理地址。

段页式存储的优点：

* 它提供了大量的虚拟存储空间。
* 能有效地利用主存，为组织多道程序运行提供了方便。

段页式存储的缺点：

* 增加了硬件成本、系统的复杂性和管理上的开消。
* 存在着系统发生抖动的危险。
* 存在着内碎片。
* 还有各种表格要占用主存空间。

# 进程调度算法

### 进程调度的原因：

在操作系统中，由于进程总数多于处理机，它们必然竞争处理机。为了充分利用

计算机系统中的CPU资源，让计算机系统能够多快好省地完成我们让它做的各种

任务，所以需要进行进程调度。

### 进程调度的定义

进程调度（也称CPU调度）是指按照某种调度算法（或原则）从就绪队列中选取

进程分配CPU，主要是协调对CPU的争夺使用。

### 通常有两种进程调度的方式：

非剥夺调度方式，又称非抢占方式：是指当一个进程正在处理机上执行时，即使有某个更为重要或紧迫的进程进入就绪队列，仍然让正在执行的进程继续执行，直到该进程完成或发生某种事件而进入阻塞状态时，才把处理机分配给更为重要或紧迫的进程。在非剥夺调度方式下，一旦把CPU分配给一个进程，那么该进程就会保持CPU直到终止或转换到等待状态。这种方式的优点是实现简单、系统开销小，适用于大多数的批处理系统，但它不能用于分时系统和大多数的实时系统。

剥夺调度方式，又称抢占方式：是指当一个进程正在处理机上执行时，若有某个更为重要或紧迫的进程需要使用处理机，则立即暂停正在执行的进程，将处理机分配给这个更为重要或紧迫的进程。

### 进程调度的常见算法：

### 1)先来先服务调度算法

先来先服务(FCFS)调度算法是一种最 简单的调度算法，该算法既可用于作业调度，也可用于进程调度。当在作业调度中采用该算法时，每次调度都是从后备作业队列中选择一个或多个最先进入该队列的 作业，将它们调入内存，为它们分配资源、创建进程，然后放入就绪队列。在进程调度中采用FCFS算法时，则**每次调度是从就绪队列中选择一个最先进入该队列的进程**，为之分配处理机，使之投入运行。该进程一直运行到完成或发生某事件而阻塞后才放弃处理机。

FCFS调度算法的特点是算法简单，但效率低；对长作业比较有利，但对短作业不利（相对SJF和高响应比）；有利于CPU繁忙型作业，而不利于I/O繁忙型作业。

### 2)短作业(进程)优先调度算法

短作业(进程)优 先调度算法SJ(P)F，是指对短作业或短进程优先调度的算法。它们可以分别用于作业调度和进程调度。短作业优先(SJF)的调度算法是从后备队列中选择 一个或若干个**估计运行时间最短的作业**，将它们调入内存运行。而短进程优先(SPF)调度算法则是**从就绪队列中选出一个估计运行时间最短的进程**，将处理机分 配给它，使它立即执行并一直执行到完成，或发生某事件而被阻塞放弃处理机时再重新调度。

SJF又分为两种：

（1）SRTF抢占式：又称最短剩余优先，当新进来的进程的CPU区间比当前执行的进程所剩的CPU区间短，则抢占。

（2）非抢占：称为下一个最短优先，即为在就绪队列中选择最短CPU区间的进程放在队头。

SJF调度算法的特点是吞吐率高，平均等待时间、平均周转时间最少；但算法对长作业十分不利，也完全未考虑作业的紧迫程度。

### 3）时间片轮转法（RR，Round Robin）

在早期的时间片轮转法中，系统将所有的就绪进程按先来先服务的原则排成一个队列，每次调度时，把CPU 分配给队首进程，并令其执行一个时间片。时间片的大小从几ms 到几百ms。当执行的时间片用完时，由一个计时器发出时钟中断请求，调度程序便据此信号来停止该进程的执行，并将它送往就绪队列的末尾；然后，再把处理机 分配给就绪队列中新的队首进程，同时也让它执行一个时间片。这样就可以保证就绪队列中的所有进程在一给定的时间内均能获得一时间片的处理机执行时间。换言 之，系统能在给定的时间内响应所有用户的请求。

在时间片轮转调度算法中，时间片的大小对系统性能的影响很大。如果时间片足够大，以至于所有进程都能在一个时间片内执行完毕，则时间片轮转调度算法就退化为先来先服务调度算法。如果时间片很小，那么处理机将在进程间过于频繁 切换，使处理机的开销增大，而真正用于运行用户进程的时间将减少。因此时间片的大小应选择适当。

时间片的长短通常由以下因素确定：系统的响应时间、就绪队列中的进程数目和系统的处理能力。

时间片q =  系统对相应时间的要求RT /   最大进程数N（经验表明，时间片的取值，应该使得80%的进程在时间片内完成所需的一次CPU运行活动。）

### 4）多级反馈队列调度算法（MLFQ，Multi-Level Feedback Queue）

(1) 应设置多个就绪队列，并为各个队列赋予不同的优先级。第一个队列的优先级最高，第二个队列次之，其余各队列的优先权逐个降低。该算法赋予各个队列中进程执 行时间片的大小也各不相同，在优先权愈高的队列中，为每个进程所规定的执行时间片就愈小。例如，第二个队列的时间片要比第一个队列的时间片长一倍，……， 第i+1个队列的时间片要比第i个队列的时间片长一倍。

　　 (2) 当一个新进程进入内存后，首先将它放入第一队列的末尾，按FCFS原则排队等待调度。当轮到该进程执行时，如它能在该时间片内完成，便可准备撤离系统；如 果它在一个时间片结束时尚未完成，调度程序便将该进程转入第二队列的末尾，再同样地按FCFS原则等待调度执行；如果它在第二队列中运行一个时间片后仍未 完成，再依次将它放入第三队列，……，如此下去，当一个长作业(进程)从第一队列依次降到第n队列后，在第n 队列便采取按时间片轮转的方式运行。

 　　(3) 仅当第一队列空闲时，调度程序才调度第二队列中的进程运行；仅当第1～(i-1)队列均空时，才会调度第i队列中的进程运行。如果处理机正在第i队列中为 某进程服务时，又有新进程进入优先权较高的队列(第1～(i-1)中的任何一个队列)，则此时新进程将抢占正在运行进程的处理机，即由调度程序把正在运行 的进程放回到第i队列的末尾，把处理机分配给新到的高优先权进程。

### 5）高响应比优先调度算法（HRRF，Highest Response Ratio First）

高响应比优先调度算法主要用于作业调度，该算法是对FCFS调度算法和SJF调度算法的一种综合平衡，同时考虑每个作业的等待时间和估计的运行时间。在每次进行作业调度时，先计算后备作业队列中每个作业的响应比，从中选出响应比最高的作业投入运行。

响应比   =   （等待时间+要求服务时间）   /   要求服务时间   = 响应时间   /   执行时间。

### 6）高优先权优先调度算法

 优先权调度算法的类型

　为了照顾紧迫型作业，使之在进入系统后便获得优先处理，引入了最高优先权优先(FPF)调度算法。此算法常被用于批处理系统中，作为作业调度算法，也作为多种操作系统中的进程调度算法，还可用于实时系统中。当把该算法用于作业调度时，系统将从后备队列中选择若干个优先权最高的作业装入内存。当用于进程调度时，该算法是把处理机分配给就绪队列中优先权最高的进程，这时，又可进一步把该算法分成如下两种。

**1) 非抢占式优先权算法**

　　在这种方式 下，系统一旦把处理机分配给就绪队列中优先权最高的进程后，该进程便一直执行下去，直至完成；或因发生某事件使该进程放弃处理机时，系统方可再将处理机重 新分配给另一优先权最高的进程。这种调度算法主要用于批处理系统中；也可用于某些对实时性要求不严的实时系统中。

**2) 抢占式优先权调度算法**

在这种方式 下，系统同样是把处理机分配给优先权最高的进程，使之执行。但在其执行期间，只要又出现了另一个其优先权更高的进程，进程调度程序就立即停止当前进程(原 优先权最高的进程)的执行，重新将处理机分配给新到的优先权最高的进程。因此，在采用这种调度算法时，是每当系统中出现一个新的就绪进程i 时，就将其优先权Pi与正在执行的进程j 的优先权Pj进行比较。如果Pi≤Pj，原进程Pj便继续执行；但如果是Pi>Pj，则立即停止Pj的执行，做进程切换，使i 进程投入执行。显然，这种抢占式的优先权调度算法能更好地满足紧迫作业的要求，故而常用于要求比较严格的实时系统中，以及对性能要求较高的批处理和分时系 统中。

# 页面置换算法

当发生缺页中断时，操作系统必须在内存中选择一个页面将其换出内存，以便为即将调入的页面腾出空间。如果要换出的页面在内存驻留期间已经被修改过，就必须把它写回磁盘已更新该页面在磁盘上的副本。如果该页面没有被修改过，则说明，它在磁盘上的副本已经是最新的，不需要回写。

### 1. 最优页面置换算法

在缺页中断发生时。在内存中的页面有的可能需要10条指令后才被访问，有的可能100条，或者1000条指令后才被访问。最优置换算法总是将需要指令最多的被置换。但这个算法是无法实现的，因为无法知道页面下一次将在什么时候被访问。但是可以先在仿真程序上运行，跟踪页面访问情况，在第二次运行时利用第一次运行收集到的信息来实现。

### ****2.最近未使用页面置换算法（NRU）****

系统为每一个页面设置两个标志位：当页面被访问时设置R位，当页面（修改）被写入时设置M位。当发生缺页中断时，OS检查所有的页面，并根据它们当前的R和M位的值，分为四类：

（1）！R&！M，没有被访问，没有被修改；

（2）！R&M ，没有被访问，已被修改

（3）R&！M ，有被访问，没有被修改；

（4）R&M，有被访问，已被修改

编号越小的类，越被优先换出。即在最近的一个时钟滴答内，淘汰一个没有被访问但是已经被修改的页面，比淘汰一个被频繁使用但是“clean”的页面要好。

### 3.先进先出页面置换算法（FIFO）及其改进

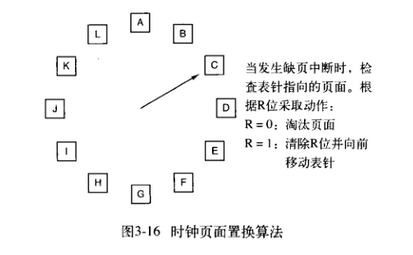
这种算法的思想和队列是一样的，OS维护一个当前在内存中的所有页面的链表，最新进入的页面在尾部，最久的在头部，每当发生缺页中断，就替换掉表头的页面并且把新调入的页面加入到链表末尾。

这个算法的问题，显然是太过于“公正了”，没有考虑到实际的页面使用频率。

一种合理的改进，称为第二次机会算法。即给每个页面增加一个R位，每次先从链表头开始查找，如果R置位，清除R位并且把该页面节点放到链表结尾；如果R是0，那么就是又老又没用到，替换掉。

### ****4.时钟页面置换算法（clock）****

这种算法只是模型像时钟，其实就是一个环形链表的第二次机会算法，表针指向最老的页面。缺页中断时，执行相同的操作，包括检查R位等。



### ****6. 最近最少使用页面置换算法****

基于观察，在前面几条指令中频繁使用的页面在后面的几条指令中被使用的几率很大。而很久没有使用的页面在未来较长的一段时间内仍然不会被使用。所以可以置换最长未使用的页面。该方法被称为(LRU-Latest Recently Used)。

这个算法代价很高，需要在内存中维护一个所有页面的链表，最近最多使用的页面在表头。每次访问内存都需要更新链表，在链表中找到一个页面，删除它，然后移动到表头，非常浪费时间。

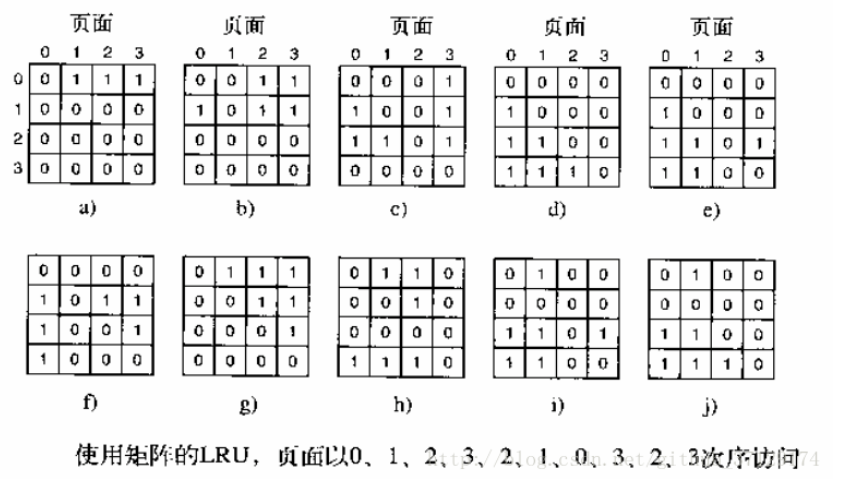
但是，还是有一些硬件实现的LRU算法。

### 64位计数器

要求硬件有一个64位计数器C，每条指令执行完后自动加1，每个页表项必须有一个足够容纳这个计数器值的域。在每次访问内存后，将当前的C值保存到被访问页面的页表项中。一旦发生缺页中断，操作系统就检查所有页表项中计数器的值，找到值最小的一个页面。就是最近最少使用的页面。

**n x n 矩阵**

在一个有n个页框的机器中，LRU硬件可以维持一个初值n\*n位的矩阵。当访问到页框k时，硬件首先把k行的位都设置成1，再把k列的位置都设置成0。在任何时候二进制数值最小的行就是最近最少使用的。

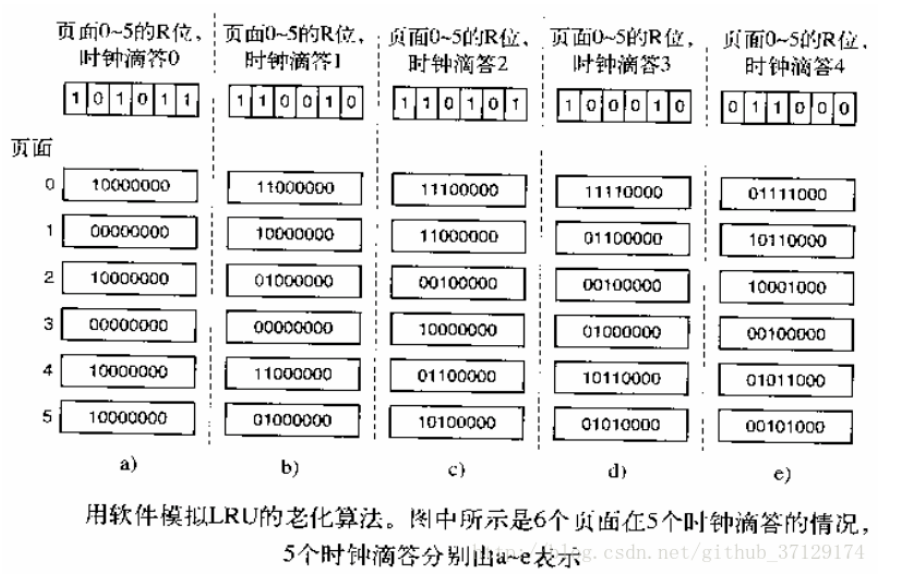


### 软件模拟LRU

要使用软件实现，一种可能的方法称为**最不常用算法(NFU-Not Frequently Used)**该算法将每个页面与一个软件计数器相关联，计数器初值为0.每次时钟中断时，由操作系统扫描内存中所有的页面，将每个页面的R位(访问位,它的值是0或1)加到它的计数器上。这个计数器大体上跟踪了各个页面被访问的频繁程度。发生缺页中断时，则置换计数器值最小的页面。但是NFU不会忘记任何事情，比如一个页面可能开始被频繁使用，计数器值很高。但是过一段时间后，它不在经常被访问了，但是被经常访问的其他页面计数器总是比他小，结果是将有用的页面置换走了。

所以，我们需要对此进行改进，修改以后的算法称为老化算法。首先，在R位被加进之前先将计数器右移一位；其次，将R位加到计数器最左端而不是最右端。

如图所示，假设在第一个时钟滴答后，页面0到页面5的R值分别为1、0、1、0、1、1，即在第一次时钟滴答前，程序访问了以免0，2，4，5，它们的R位被设置为1。第一次时钟滴答到二次时钟滴答之间，程序访问了页面0，1，4，于是得到图中的b。



老化算法只能采用有限的位数，所以可能在一定程度上精度会有所损失。