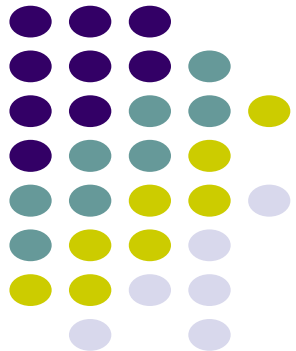
操作系统

第5章 虚拟存储器

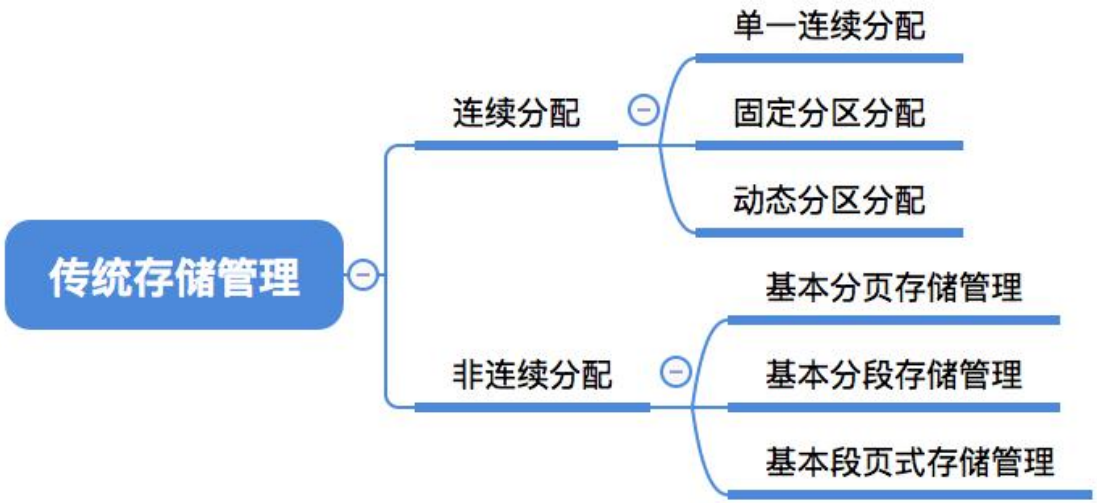


内容

- 4.1 程序的装入和链接
- 4.2 连续分配方式
- 4.3 基本分页存储管理方式
- 4.4 基本分段存储管理方式
- 4.5 虚拟存储器的基本概念
- 4.6 请求分页存储管理方式
- 4.7 页面置换算法
- 4.8 请求分段存储管理方式

传统存储管理方式的特征、缺点

很多暂时用不到的数 据也会长期占用内存， 导致内存利用率不高



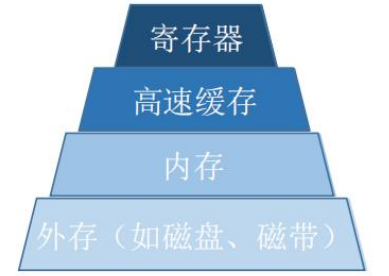
一次性：作业必须一次性全部装入内存后才能开始运行。这会造成两个问题：①作业很大时，不能全部装入内存，导致大作业无法运行；②当大量作业要求运行时，由于内存无法容纳所有作业，因此只有少量作业能运行，导致多道程序并发度下降。

驻留性：一旦作业被装入内存，就会一直驻留在内存中，直至作业运行结束。事实上，在一个时间段内，只需要访问作业的一小部分数据即可正常运行，这就导致了内存中会驻留大量的、暂时用不到的数据，浪费了宝贵的内存资源。

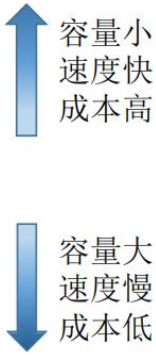
局部性原理

时间局部性：如果执行了程序中的某条指令，那么不久后这条指令很有可能再次执行；如果某个数据 被访问过，不久之后该数据很可能再次被访问。（因为程序中存在大量的循环）

空间局部性：一旦程序访问了某个存储单元，在不久之后，其附近的存储单元也很有可能被访问。（因为很多数据在内存中都是连续存放的，并且程序的指令也是顺序地在内存中存放的）



计算机中存储器的层次结构



快表机构就是将近期 常访问的页表项副本 放到更高速的联想寄 存器中

```
int i = 0;
int a[100];
while (i < 100) {
  a[i] = i;
  i++;
}
```

高速缓冲技术的思想：将近期会频繁访问到的数据放到更高速的存储器中，暂时用不到的数据放在更低速存储器中

如何应用局 部性原理？

虚拟内存的定义和特征

基于局部性原理，在程序装入时，可以将程序中很快会用到的部分装入内存，暂时用不到的部分留在外存， 就可以让程序开始执行。

在程序执行过程中，当所访问的信息不在内存时，由操作系统负责将所需信息从外存调入内存，然后继续执行程序。

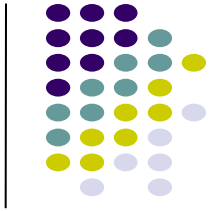
若内存空间不够，由操作系统负责将内存中暂时用不到的信息换出到外存。

在操作系统的管理下，在用户看来似乎有一个比实际内存大得多的内存，这就是虚拟内存

虚拟内存的最大容量是由计算机的地址结构（CPU寻址范围）确定的
虚拟内存的实际容量 =min（内存和外存容量之和，CPU寻址范围）

如：某计算机地址结构为32位，按字节编址，内存大小为512MB，外存大小为2GB。 则
虚拟内存的最大容量为 $2^{32}B = 4GB$
虚拟内存的实际容量 = min ($2^{32}B$, 512MB+2GB) = 2GB+512MB

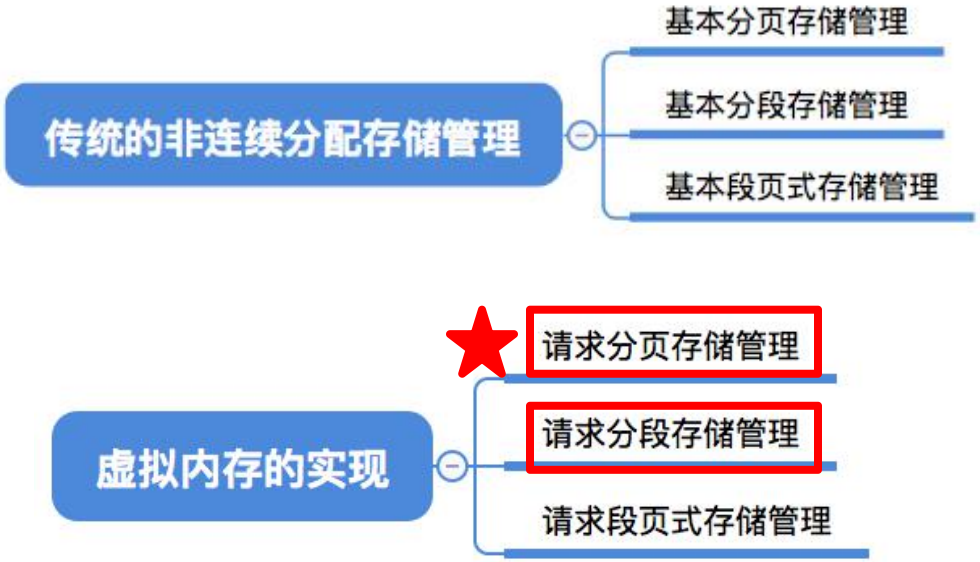
虚拟内存的定义和特征



- 虚拟存储器：是指具有请求调入功能和置换功能，能从逻辑上对内存容量加以扩充的一种存储器系统。其逻辑容量由内存容量和外存容量之和所决定，其运行速度接近于内存速度，而每位的成本却接近于外存。
- 主要特征：
  - 多次性：无需在作业运行时一次性全部装入内存，而是允许被分成多次调入内存。
  - 对换性：在作业运行时无需一直常驻内存，而是允许在作业运行过程中，将作业换入、换出。
  - 虚拟性：从逻辑上扩充了内存的容量，使用户看到的内存容量，远大于实际的容量。

如何实现虚拟内存技术

虚拟内存技术，允许一个作业分多次调入内存。如果采用连续分配方式，会不方便实现。因此， 虚拟内存的实现需要建立在离散分配的内存管理方式基础上。

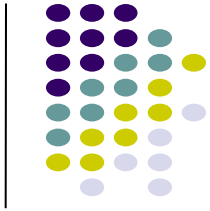


操作系统要提供请求调页（或请求调段）功能

主要区别：
在程序执行过程中，当所访问的信息不在内存时，由操作系统负责将所需信息从外存调入内存，然后继续执行程序。 若内存空间不够，由操作系统负责将内存中暂时用不到的信息换出到外存。

操作系统要提供页面置换（或段置换）的功能

请求分页系统



- 在分页系统的基础上，增加了请求调页功能和页面置换功能所形成的页式虚拟存储系统。它允许只装入部分页面的程序(及数据)，便启动运行。以后再通过调页功能及页面置换功能，陆续将即将要运行的页面调入内存，同时把暂不运行的页面换出到外存上。置换时以页面为单位。
- 为实现请求调页和置换功能，系统必须提供必要的支持：
  - 硬件支持：①请求分页的页表机制 ②缺页中断机构
    ③地址变换机构
  - 软件支持：①实现请求调页的软件　②实现页面置换的软件

请求分段系统

- 在分段系统的基础上，增加了请求调段功能和分段置换功能所形成的段式虚拟存储系统。它允许只装入若干段的程序(及数据)，便启动运行。以后再通过调段功能及段的置换功能，把暂不运行的段调出，同时调入即将要运行的段。置换时以段为单位
- 为实现请求调段和置换功能，系统必须提供必要的支持：
  - 硬件支持：①请求分段的段表机制 ②缺段中断机构
    ③地址变换机构
  - 软件支持：①实现请求调段的软件 ②实现段的置换的软件

知识回顾


虚拟内存的基本概念
传统存储管理方式的特征、缺点
一次性：作业数据必须一次全部调入内存
驻留性：作业数据在整个运行期间都会常驻内存
局部性原理
时间局部性：现在访问的指令、数据在不久后很可能会被再次访问
空间局部性：现在访问的内存单元周围的内存空间，很可能在不久后会被访问
高速缓存技术：使用频繁的数据放到更高速的存储器中
虚拟内存的定义和特征
程序不需全部装入即可运行，运行时根据需要动态调入数据，若内存不够，还需换出一些数据
特征
多次性：无需在作业运行时一次性全部装入内存，而是允许被分成多次调入内存。
对换性：无需在作业运行时一直常驻内存，而是允许在作业运行过程中，将作业换入、换出。
虚拟性：从逻辑上扩充了内存的容量，使用户看到的内存容量，远大于实际的容量。
如何实现虚拟内存技术
访问的信息不在内存时，由操作系统负责将所需信息从外存调入内存（请求调页功能）
内存空间不够时，将内存中暂时用不到的信息换出到外存（页面置换功能）
虚拟内存的实现
请求分页存储管理
请求分段存储管理
请求段页式存储管理

内容

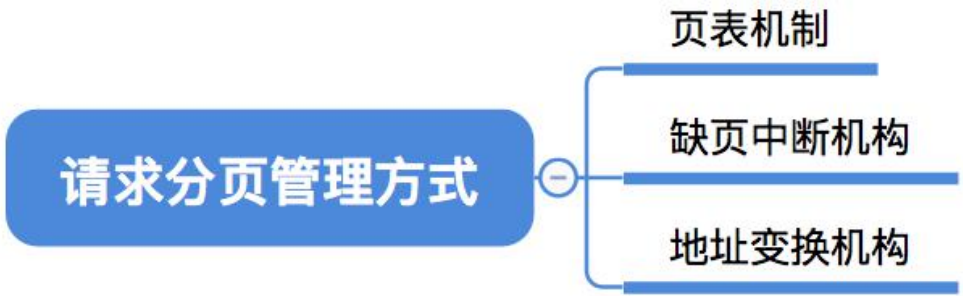
- 4.1 程序的装入和链接
- 4.2 连续分配方式
- 4.3 基本分页存储管理方式
- 4.4 基本分段存储管理方式
- 4.5 虚拟存储器的基本概念
- 4.6 请求分页存储管理方式
- 4.7 页面置换算法
- 4.8 请求分段存储管理方式

请求分页存储管理方式

- 请求分页中的硬件支持
- 内存分配策略和分配算法
- 请求分页策略

请求分页中的硬件支持

请求分页存储管理与基本分页存储管理的主要区别： 在程序执行过程中，当所访问的信息不在内存时，由操作系统负责将所需信息从外存调入内存，然后继续执行程序。 若内存空间不够，由操作系统负责将内存中暂时用不到的信息换出到外存。



请求页表机制

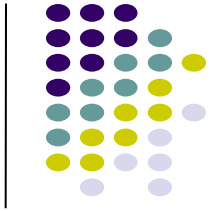
与基本分页管理相比，请求分页管理中，为了实现“请求调页”，操作系统需要知道每个页面是否已经调入内存；如果还没调入，那么也需要知道该页面在外存中存放的位置

当内存空间不够时，要实现“页面置换”，操作系统需要通过某些指标来决定到底换出哪个页面；有的页面没有被修改过，就不用再浪费时间写回外存。有的页面修改过，就需要将外存中的旧数据覆盖，因此，操作系统也需要记录各个页面是否被修改的信息

页号	物理块号	状态位P	访问字段A	修改位M	外存地址

(1) 状态位P：指示该页是否已调入内存。
(2) 访问字段A：用于记录本页在一段时间内被访问的次数，或记录本页最近已有多长时间未被访问。
(3) 修改位M：表示该页在调入内存后是否被修改过。
(4) 外存地址：用于指出该页在外存上的地址。

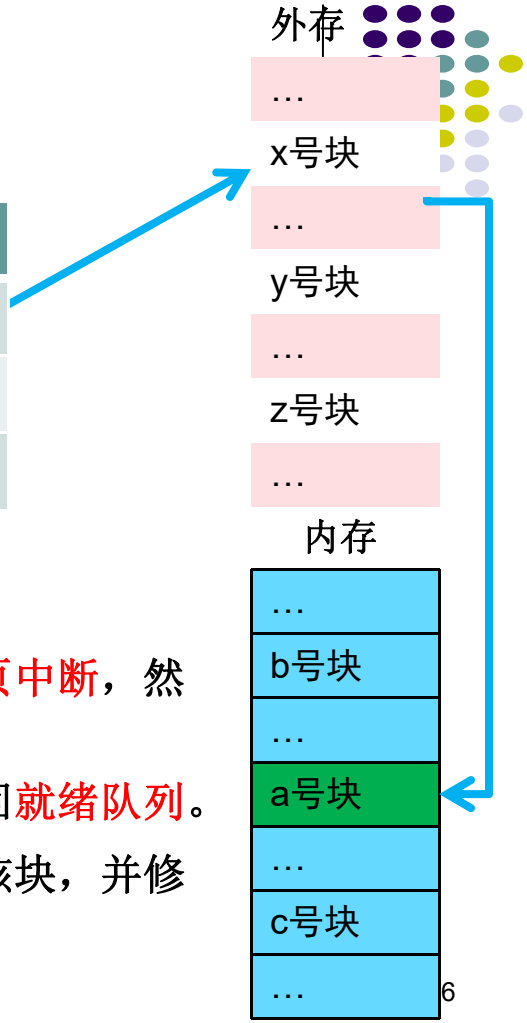
缺页中断机构



- 在请求分页系统中，每当要访问的页面不在内存时，便产生一缺页中断，请求OS将所缺之页调入内存
- 缺页中断作为中断，同样需要经历诸如保护CPU环境、分析中断原因、转入缺页中断处理程序进行处理、恢复CPU环境等几个步骤。但缺页中断是一种特殊的中断，与一般中断有明显区别：
  - 在指令执行期间产生和处理中断信号。
  - 一条指令在执行期间，可能产生多次缺页中断

缺页中断机构

页号	物理块号	状态位P	访问字段A	修改位M	外存地址
0	a	1	0	0	x
1	b	1	10	0	y
2	c	1	6	1	z



假设此时要访问逻辑地址=（页号，页内偏移量）=（0, 1024）

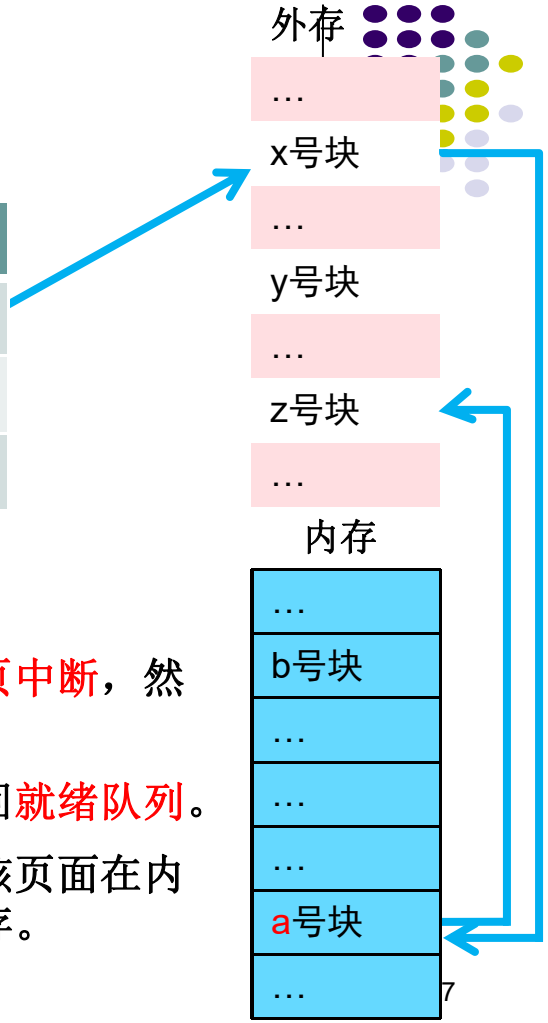
在请求分页系统中，每当要访问的页面不在内存时，便产生一个缺页中断，然后由操作系统的缺页中断处理程序处理中断。

此时缺页的进程阻塞，放入阻塞队列，调页完成后再将其唤醒，放回就绪队列。

如果内存中有空闲块，则为进程分配一个空闲块，将所缺页面装入该块，并修改页表中相应的页表项。

缺页中断机构

页号	物理块号	状态位P	访问字段A	修改位M	外存地址
0	a	1	0	0	x
1	b	1	10	0	y
2	无	0	0	0	z



假设此时要访问逻辑地址=（页号，页内偏移量）=（0, 1024）

在请求分页系统中，每当要访问的页面不在内存时，便产生一个缺页中断，然后由操作系统的缺页中断处理程序处理中断。

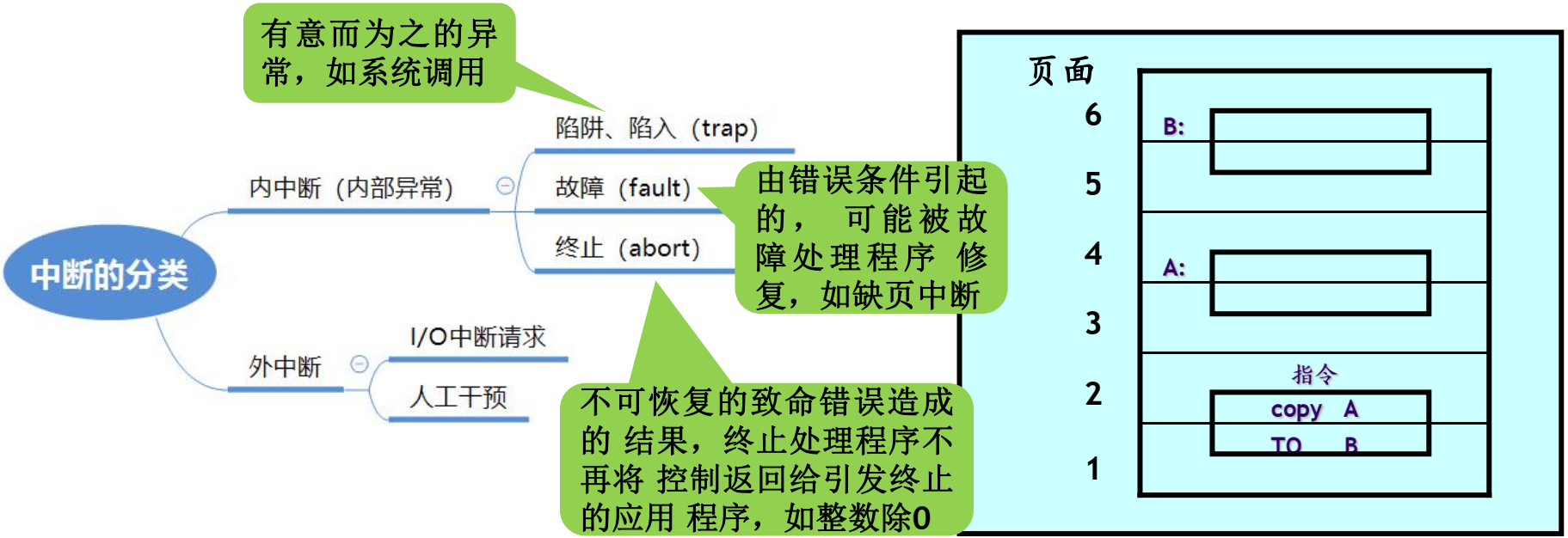
此时缺页的进程阻塞，放入阻塞队列，调页完成后再将其唤醒，放回就绪队列。

如果内存中没有空闲块，则由页面置换算法选择一个页面淘汰，若该页面在内存期间被修改过，则要将其写回外存。未修改过的页面不用写回外存。

缺页中断机构

缺页中断是因为当前执行的指令想要访问的目标页面未调入内存而产生的，因此属于内中断

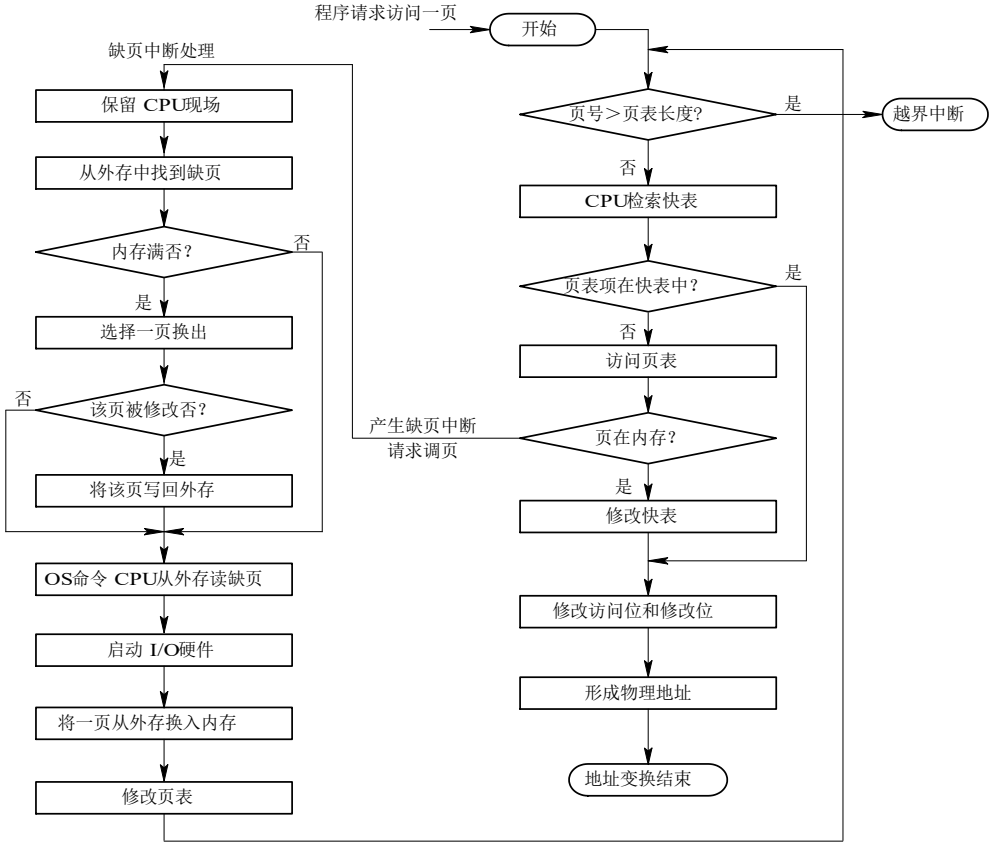
一条指令在执行期间，可能产生多次缺页中断。（如：copy A to B，即将逻辑地址A中的数据复制到 逻辑地址B，而A、B属于不同的页面，则有可能产生多次中断）



地址变换机构

- 与分页管理方式类似，增加虚拟存储器功能
- 情况一：
  - 首先检索快表，若找到，修改页表项中的访问位，然后利用页表项中给出的物理块号和页内地址，形成物理地址。
- 情况二：
  - 如果在快表中未找到相应的页表项，检索内存中的页表，查看页表中的状态位，若该页已经调入内存，填写快表，当快表满时，应淘汰一个页表项；若该页尚未调入内存，产生缺页中断，请求OS把该页调入

请求分页地址变化过程


程序请求访问一页
开始
缺页中断处理
保留 CPU现场
从外存中找到缺页
内存满否？
否
是
选择一页换出
该页被修改否？
否
是
将该页写回外存
OS命令 CPU从外存读缺页
启动 I/O硬件
将一页从外存换入内存
修改页表
产生缺页中断
请求调页
页号＞页表长度？
是
越界中断
否
CPU检索快表
页表项在快表中？
是
否
访问页表
页在内存？
是
修改快表
修改访问位和修改位
形成物理地址
地址变换结束

内存分配策略和分配算法

- 为进程分配内存时，涉及三个问题：
- ①最小物理块数的确定
- ②物理块的分配策略
- ③物理块的分配算法

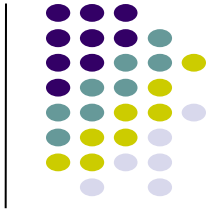
最小物理块数的确定

- 最小物理块数，指能保证进程正常运行所需的最小物理块数。当系统为进程分配的物理块数少于此值时，进程将无法运行。
- 进程应获得的最小物理块数与计算机的硬件结构有关，取决于指令的格式、功能和寻址方式。

物理块的分配策略

- 固定分配局部置换
  - 为进程分配的物理块数在整个运行期间都不再改变。若某个进程发生缺页，则只能将自己的某个内存页换出
  - 困难：应为每个进程分配多少个物理块难以确定。
- 可变分配全局置换
  - 为每个进程分配一定数目的物理块，当进程发生缺页，若系统中有空闲的物理块，则分配一个物理块并装入缺页；页面的置换范围是任一个进程
  - 被选中进程缺页率增加
- 可变分配局部置换
  - 为每个进程分配一定数目的物理块后，若某个进程发生缺页，则只能将自己的某个内存页换出。OS根据缺页率进行物理块分配的调整

物理块的分配算法



- 平均分配算法
  - 将系统中所有可供分配的物理块，平均分配给各个进程。
  - 缺点：未考虑各进程本身的大小
- 按比例分配算法
  - 根据进程的大小按比例分配物理块。
  - 设系统中共有n个进程，每个进程的页面数为Si，则系统中各进程页面数的总和为：

$$S = \sum_{i=1}^{n} S_i$$

  - 又假定系统中可用的物理块总数为m，则每个进程所能分到的物理块数为bi，将有

$$b_i = \frac{S_i}{S} * m$$

bi应该取整，必须大于最小物理块数

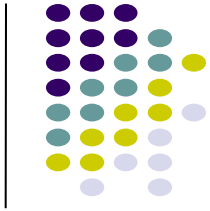
物理块的分配算法

- 考虑优先权的分配算法
  - 在实际应用中，为了照顾重要的、急迫的作业尽快完成，应为它分配较多的内存空间
  - 把内存中可供分配的物理块分为两部分：
    - 一部分按比例分配给各进程；
    - 一部分则根据各进程的优先权，适当地增加其相应份额，分配给各进程

知识回顾


请求分页管理方式
页表机制
在基本分页的基础上增加了几个表项
状态位：表示页面是否已在内存中
访问字段：记录最近被访问过几次，或记录上次访问的时间，供置换算法选择换出页面时参考
修改位：表示页面调入内存后是否被修改过，只有修改过的页面才需在置换时写回外存
外存地址：页面在外存中存放的位置
缺页中断机构
找到页表项后检查页面是否已在内存，若没在内存，产生缺页中断
缺页中断处理中，需要将目标页面调入内存，有必要时还要换出页面
缺页中断属于内中断，属于内中断中的“故障”，即可能被系统修复的异常
一条指令在执行过程中可能产生多次缺页中断
地址变换机构（重点关注与基本分页不同的地方）
找到页表项是需要检查页面是否在内存中
若页面不再内存中，需要请求调页
若内存空间不够，还需换出页面
页面调入内存后，需要修改相应页表项

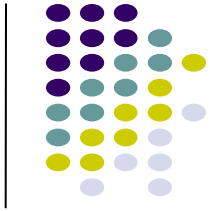
页面调入策略



- 何时调入页面
- 为了确定系统将进程运行时所缺页面调入内存的时机，可采取预调页策略或请求调页策略。

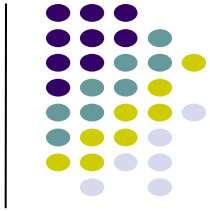
- 预调页策略
  - 采用以预测为基础的预调页策略，将那些预计在不久之后便会被访问的页面，预先调入内存。
  - 主要用于进程的首次调入时，由程序员指出应该先调入哪些页。

页面调入策略



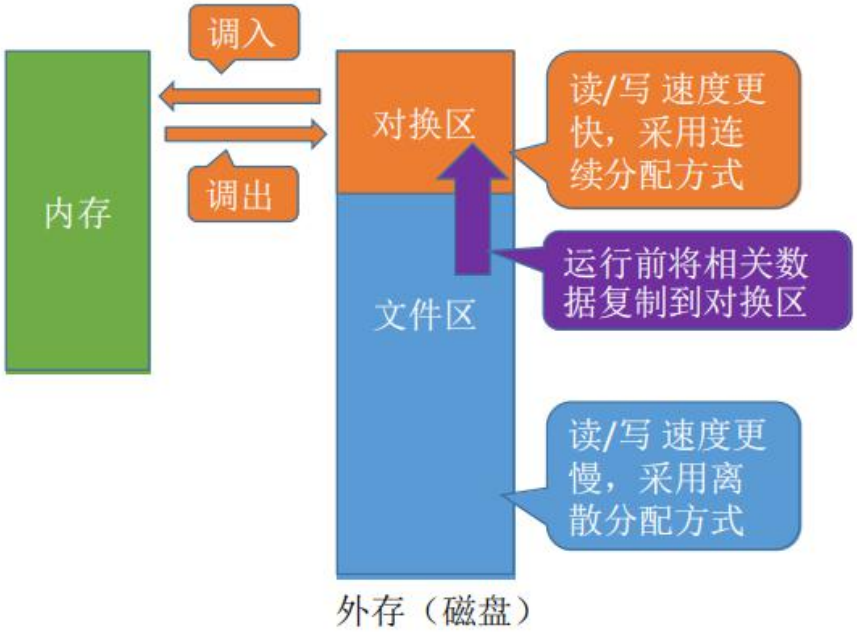
- 请求调页策略
  - 当程序在运行中需要访问某部分程序和数据时，若发现其所在的页面不在内存，便立即提出请求，由OS将其所需要的页面调入内存。
  - 优点：由请求调页策略所确定调入的页，一定会被访问；请求调页策略比较容易实现。
  - 缺点：每次仅调入一页，需花费较大的系统开销，增加了磁盘 I/O的启动频率。

页面调入策略



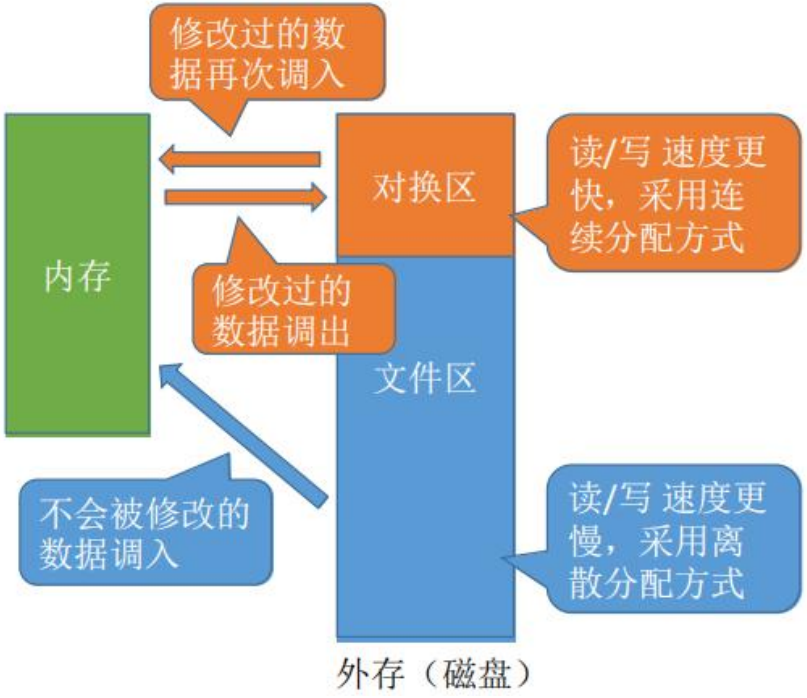
- 从何处调入页面
- 将请求分页系统中的外存分为文件区和对换区。每当发生缺页时，系统应从何处将缺页调入内存，可分为：
- 系统拥有足够的对换区空间：
  - 可以全部从对换区调入所需页面，以提高调页速度
- 系统缺少足够的对换区空间：
  - 凡不会被修改的文件，直接从文件区调入；当换出这些页面时，因为未修改不用换出，再调入时仍从文件区调入。可能被修改的部分，换出时需调到对换区，换入时从对换区调入。
- Unix

页面调入策略



系统拥有足够的对换区空间：
页面的调入、调 出都是在内存与对换区之间进行，这样可以保 证页面的调入、调出速度很快。在进程运行前，需将进程相关的数据从文件区复制到对换区

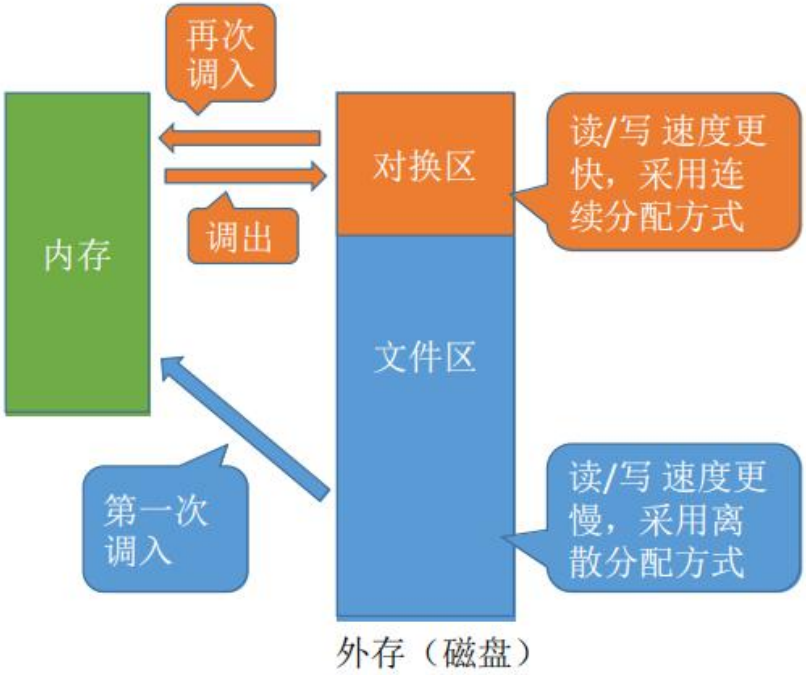
页面调入策略



系统缺少足够的对换区空间：

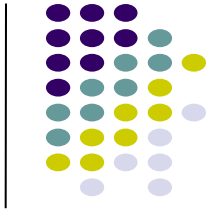
凡是不会被修改 的数据都直接从文件区调入，由于这些页面不 会被修改，因此换出时不必写回磁盘，下次需 要时再从文件区调入即可。对于可能被修改的 部分，换出时需写回磁盘对换区，下次需要时 再从对换区调入

页面调入策略



UNIX 方式：
运行之前进程有关的数据全部放在文件区，故未使用过的页面，都可从文件区调 入。若被使用过的页面需要换出，则写回对换 区，下次需要时从对换区调入

页面调入过程



- 每当程序所要访问的页面未在内存时，便向CPU发出一缺页中断，中断处理程序首先保护CPU环境，分析中断原因后，转入缺页中断处理程序
- 程序通过查找页表，得到该页在外存上的物理块后，如果此时内存能容纳新页，则启动磁盘I/O将所缺之页调入内存，然后修改页表
- 如果内存已满，则需按照某种置换算法从内存中选出一页准备换出；如果该页未被修改过，可不必写回磁盘；但如果此页已被修改，则必须将它写回磁盘，然后把所缺的页调入内存，并修改页表中的相应表项，置其存在位为“1”，并将此页表项写入快表
- 在缺页调入内存后，利用修改后的页表，形成所要访问的物理地址，再去访问内存数据。整个页面调入过程对用户是透明的

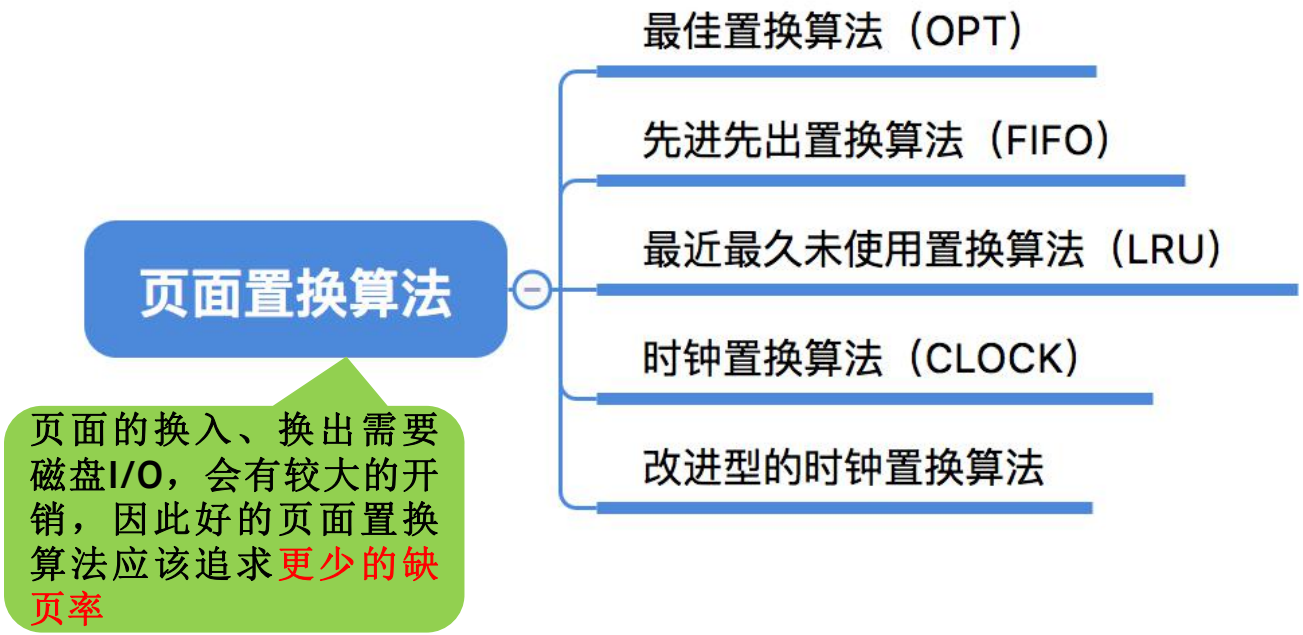
内容

- 4.1 程序的装入和链接
- 4.2 连续分配方式
- 4.3 基本分页存储管理方式
- 4.4 基本分段存储管理方式
- 4.5 虚拟存储器的基本概念
- 4.6 请求分页存储管理方式
- 4.7 页面置换算法
- 4.8 请求分段存储管理方式

页面置换算法

在程序执行过程中，当所访问的信息不在内存时，由操作系统负责将所需信息从外存调入内存，然后继续执行程序。

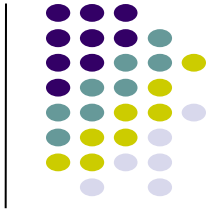
若内存空间不够，由操作系统负责将内存中暂时用不到的信息换出到外存



用页面置换算法决定应该换出哪个页面

页面的换入、换出需要磁盘I/O，会有较大的开销，因此好的页面置换算法应该追求更少的缺页率

最佳置换算法（OPT）



- 最佳置换算法是一种理想化的算法，具有最好的性能，但难于实现。先进先出置换算法最直观，但可能性能最差，故应用极少
- 其所选择的被淘汰页面，将是以后永不再用的，或许是在最长(未来)时间内不再被访问的页面。
- 优点：保证获得最低的缺页率
- 缺点：无法预知一个进程在内存的若干个页面，哪个在未来最长时间内不再被访问。
- 算法无法实现，但可评价其他算法。

例题

- 设作业分配3个物理块：

访问页面	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
内存块1	7	7	7	2		2		2			2			2				7		
内存块2		0	0	0		0		4			0			0				0		
内存块3			1	1		3		3			3			1				1		
是否缺页	√	√	√	√		√		√			√			√				√		

整个过程缺页中断发生了9次，页面置换发生了6次。 注意：缺页时未必发生页面置换。若还有可用的空闲内存块， 就不用进行页面置换。

缺页率 = 9/20 = 45%

先进先出置换算法（FIFO）

- 算法总是淘汰最先进入内存的页面，即选择在内存中驻留时间最久的页面予以淘汰。
- 算法实现简单，只需把一个进程已调入内存的页面，按先后次序链接成一个队列，并设置一个指针(替换指针)，使它总是指向最老的页面。
- 算法与进程的实际运行规律不相适应，因为进程中的某些页面经常被访问，但先进先出置换算法不能保证这些页面不被淘汰。

先进先出置换算法（FIFO）

假设系统为某进程分配了三个内存块，并考虑到有以下页面号引用串

访问页面	3	2	1	0	3	2	4	3	2	1	0	4
内存块1	3	3	3	0	0	0	4			4	4	
内存块2		2	2	2	3	3	3			1	1	
内存块3			1	1	1	2	2			2	0	
是否缺页	√	√	√	√	√	√	√			√	√	

分配三个内存块时，缺页次数：9次

假设系统为某进程分配了四个内存块，并考虑到有以下页面号引用串

访问页面	3	2	1	0	3	2	4	3	2	1	0	4
内存块1	3	3	3	3			4	4	4	4	0	0
内存块2		2	2	2			2	3	3	3	3	4
内存块3			1	1			1	1	2	2	2	2
内存块4				0			0	0	0	1	1	1
是否缺页	√	√	√	√			√	√	√	√	√	√

分配四个内存块时，缺页次数：10次
分配三个内存块时，缺页次数：9次

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

最近最久未使用置换算法（LRU）

最近最久未使用置换算法（LRU，least recently used）：每次淘汰的页面是最近最久未使用的页面

实现方法：赋予每个页面对应的页表项中，用访问字段记录该页面自上次被访问以来所经历的时间t。 当需要淘汰一个页面时，选择现有页面中 t 值最大的，即最近最久未使用的页面

该算法的实现需要专门的硬件支持，虽然算法性能好，但是实现困难，开销大

假设系统为某进程分配了四个内存块，并考虑到有以下页面号引用串

访问页面	1	8	1	7	8	2	7	2	1	8	3	8	2	1	3	1	7	1	3	7
内存块1	1	1		1		1					1						1			
内存块2		8		8		8					8						7			
内存块3				7		7					3						3			
内存块4						2					2						2			
缺页否	√	√		√		√					√						√			

LRU置换算法的硬件支持（实现）

- LRU置换算法虽然较好，但需较多的硬件支持，为了了解一个进程在内存中的各个页面各有多少时间未被进程访问，以及如何快速地知道哪一页是最近最久未使用的页面，需有以下两类硬件之一的支持：
- 寄存器
- 栈

LRU置换算法的硬件支持（实现）

- 寄存器
  - 为每个在内存中的页面配置一个移位寄存器，用来记录某进程在内存中各页的使用情况。移位寄存器表示为

$$R=R_{n-1}R_{n-2}R_{n-3}\cdots R_2R_1R_0$$

  - 当进程访问某物理块时，要将相应寄存器的R_{n-1}位置成1。此时，定时信号将每隔一定时间将寄存器右移一位。如果把n位寄存器的数看作一个整数，那么具有最小数值的寄存器所对应的页面，就是最近最久未使用的页面

LRU置换算法的硬件支持（实现）

某进程具有8个页面时的LRU访问情况

实质 \ R	R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0
1	0	1	0	1	0	0	1	0
2	1	0	1	0	1	1	0	0
3	0	0	0	0	0	1	0	0
4	0	1	1	0	1	0	1	1
5	1	1	0	1	0	1	1	0
6	0	0	1	0	1	0	1	1
7	0	0	0	0	0	1	1	1
8	0	1	1	0	1	1	0	1

LRU置换算法的硬件支持（实现）

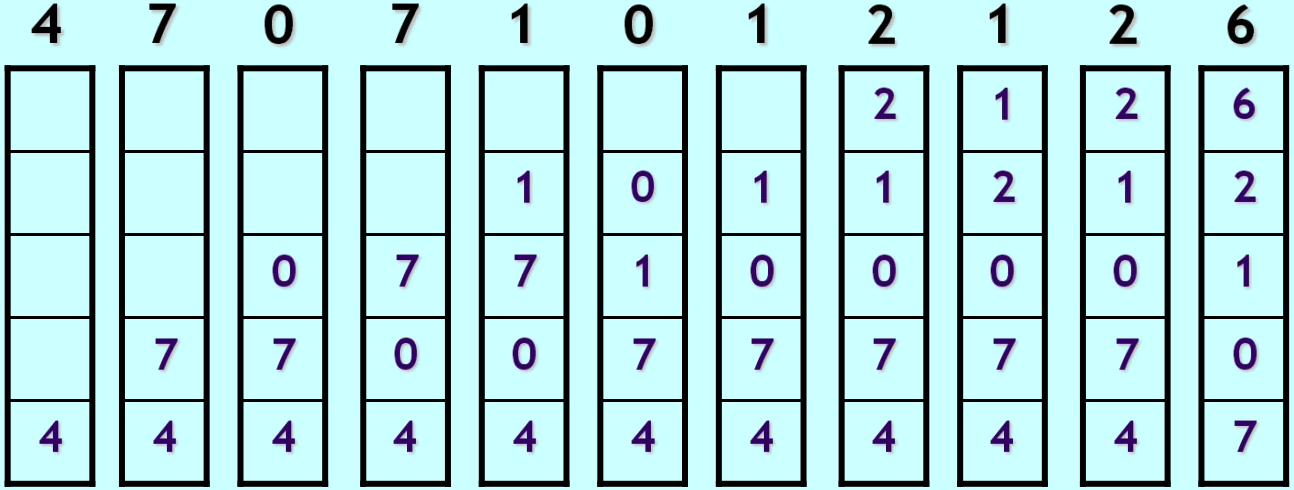
- 栈
  - 利用一个特殊的栈来保存当前使用的各个页面的页面号。每当进程访问某页面时，便将该页面的页面号从栈中移出，将它压入栈顶。因此，栈顶始终是最新被访问页面的编号，而栈底则是最近最久未使用页面的页面号

LRU置换算法的硬件支持（实现）

设一进程访问页面的页面号序列为：

4，7，0，7，1，0，1，2，1，2，6

随着进程的访问，栈中页面号的变化情况：

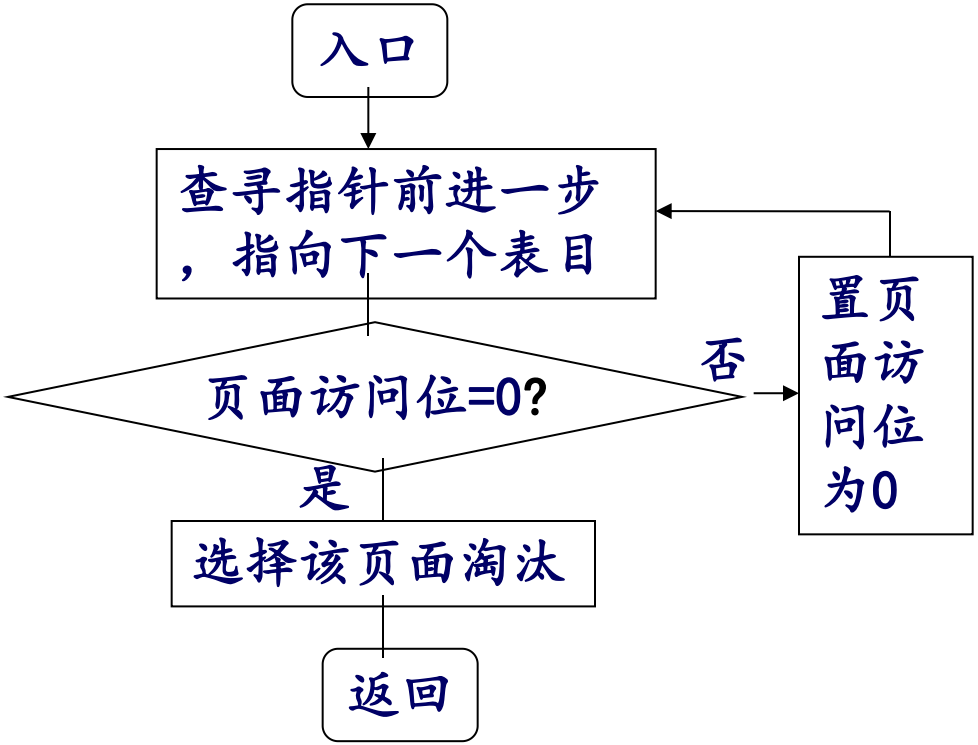


时钟置换算法（CLOCK）

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，Not Recently Used）

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，再进行第二轮扫描（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 ）

时钟置换算法（CLOCK）



块号	页号	访问位	指针
0			
1			
2	4	0	
3			
4	2	0	
5			
6	5	0	
7	1	0	

替换指针

改进型的时钟置换算法

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M=0，表示页面没有被修改过；修改位M=1，表示页面被修改过。

访问位A、修改位M，共同表示一个页面的状态

四种状态：

00： (A=0;M=0)最近未被访问也未被修改

01： (A=0;M=1)最近未被访问但已被修改

10： (A=1;M=0)最近已被访问但未被修改

11： (A=1;M=1)最近已被访问且被修改

改进型的时钟置换算法

第一优先级：最近没访问，且没修改的页面

第二优先级：最近没访问，但修改过的页面

三轮扫描：

第一轮：查找00页面。如果未找到则进入下一步；

第二轮：查找01页面。将所有扫描过的页面访问位复位为“0”。如果未找到则进入下一步；

第三轮：将所有页的访问位复位为“0”，从头开始，重复第一轮，必要时再重复第二轮。

第三优先级：最近访问过，但没修改的页面

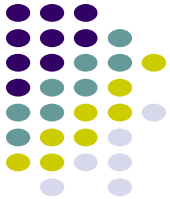
第四优先级：最近访问过，且修改过的页面

由于第二轮已将所有帧的访问位设为0，因此经过第三轮、第四轮扫描一定会有一个帧被选中，因此改进型CLOCK置换算法选择一个淘汰页面最多会进行四轮扫描

知识回顾

	算法规则	优缺点
OPT	优先淘汰最长时间内不会被访问的页面	缺页率最小，性能最好； 但无法实现
FIFO	优先淘汰最先进入内存的页面	实现简单；但性能很差， 可能出现Belady异常
LRU	优先淘汰最近最久没访问的页面	性能很好；但需要硬件 支持，算法开销大
CLOCK（NRU）	循环扫描各页面第一轮淘汰访问位=0的，并将扫描 过的页面访问位改为1。若第一轮没选中，则进行第二轮扫描	实现简单，算法开销小； 但未考虑页面是否被修 改过。
改进型CLOCK（改进型NRU）	若用（访问位, 修改位）的形式表述，则 第一轮：淘汰（0, 0） 第二轮：淘汰（0, 1），并将扫描过的页面访问位 都置为0 第三轮：淘汰（0, 0） 第四轮：淘汰（0, 1）	算法开销较小，性能也 不错

最少使用置换算法LFU



- 选择在最近时期使用最少的页面淘汰。（频率）
- 为每个页面配一个计数器。
- 需为在内存中的每个页面设置一个移位寄存器，用来记录该页面被访问的频率。
- 每次访问某页时，便将该移位寄存器的最高位置1，此后每隔一定时间自动右移一位。
- 最近一段时间最少使用的页面是∑Ri 最小的页

与LRU的区别：R1=10000000

R2=01110100

LRU----------淘汰R2

LFU----------淘汰R1

最少使用置换算法（LFU）

实页 \ R	R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0
1	0	1	0	1	0	0	1	0
2	1	0	1	0	1	1	0	0
3	0	0	0	0	0	1	0	0
4	0	1	1	0	1	0	1	1
5	1	1	0	1	0	1	1	0
6	0	0	1	0	1	0	1	1
7	0	0	0	0	0	1	1	1
8	0	1	1	0	1	1	0	1

抖动与工作集

- 请求分页系统性能优越，但若在系统中运行的进程太多，使缺页现象频繁发生，就会对系统的性能产生很大影响，因此，需对分页系统的性能做简单分析
  - 多道程序度与“抖动”
  - 工作集
  - 抖动的预防方法

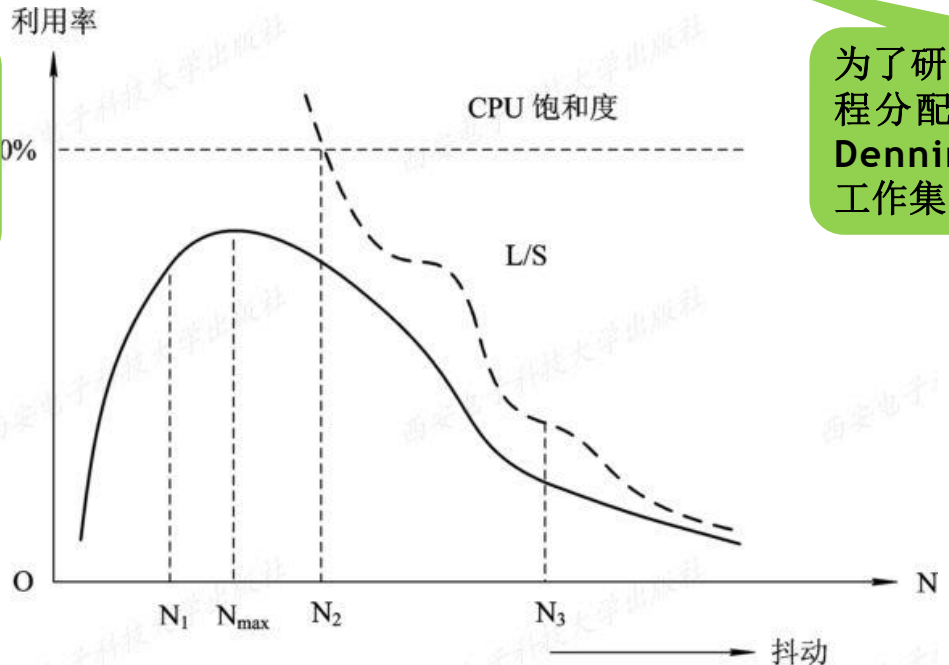
多道程序度与“抖动”

- 多道程序度与处理机的利用率
  - 由于虚拟存储器系统能从逻辑上扩大内存，这时，只需装入一个进程的部分程序和数据便可开始运行，故人们希望在系统中能运行更多的进程，即增加多道程序度，以提高处理机的利用率。但处理机的实际利用率却如图5-9中的实线所示

多道程序度与“抖动”

刚刚换出的页面马上又要换入内存，刚刚换入的页面马上又要换出外存，这种频繁的页面调度行为称为抖动，或颠簸。产生抖动的主要原因是进程频繁访问的页面数目高于可用的物理块数（分配给进程的物理块不够）

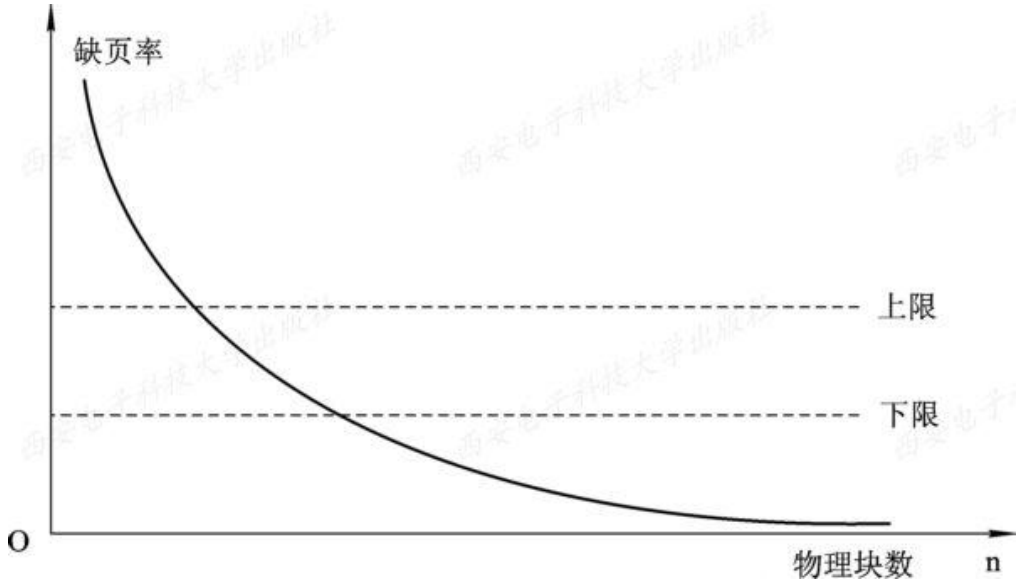
为进程分配的物理块太少，会使进程发生抖动现象。为进程分配的物理块太多，又会降低系统整体的并发度，降低某些资源的利用率



为了研究为应该为每个进程分配多 少个物理块，Denning 提出了进程 “工作集”的概念

工作集

- 进程发生缺页的时间间隔与进程所获得的物理块数有关。图5-10示出了缺页率与物理块数之间的关系



工作集

- 所谓工作集，是指在某段时间间隔Δ里，进程实际所要访问页面的集合
- Denning指出，虽然程序只需要少量的几页在内存便可运行，但为了较少地产生缺页，应将程序的全部工作集装入内存中。然而我们无法事先预知程序在不同时刻将访问哪些页面，故仍只有像置换算法那样，用程序的过去某段时间内的行为作为程序在将来某段时间内行为的近似

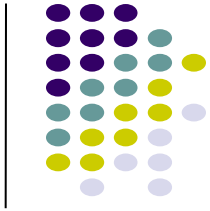
工作集

引用页序列	窗口大小 3	4	5
24	24	24	24
15	15 24	15 24	15 24
18	18 15 24	18 15 24	18 15 24
23	23 18 15	23 18 15 24	23 18 15 24
24	24 23 18	—	—
17	17 24 23	17 24 23 18	17 24 23 18 15
18	18 17 24	—	—
24	—	—	—
18	—	—	—
17	—	—	—
17	—	—	—
15	15 17 18	15 17 18 24	—
24	24 15 17	—	—
17	—	—	—
24	—	—	—
18	18 24 17	—	—

“抖动”的预防方法

- 采取局部置换策略
  - 在页面分配和置换策略中，如果采取的是可变分配方式，则为了预防发生“抖动”，可采取局部置换策略
- 把工作集算法融入到处理机调度中
  - 调度程序在从外存调入新作业之前，先检查每个进程在内存的驻留页面是否足够多。如果足够多则可以从外存调入新的作业，否则应该首先为高缺页率的作业增加新的物理块而不调入新作业

“抖动”的预防方法



- 利用“L=S”准则调节缺页率
  - Denning于1980年提出了“L=S”的准则来调节多道程序度，其中L是缺页之间的平均时间，S是平均缺页服务时间，即用于置换一个页面所需的时间。如果是L远比S大，说明很少发生缺页，磁盘的能力尚未得到充分的利用；反之，如果是L比S小，则说明频繁发生缺页，缺页的速度已超过磁盘的处理能力。只有当L与S接近时，磁盘和处理机都可达到它们的最大利用率。理论和实践都已证明，利用“L=S”准则，对于调节缺页率是十分有效的。
- 选择暂停的进程
  - 当多道程序度偏高时，已影响到处理机的利用率，为了防止发生“抖动”，系统必须减少多道程序的数目

例题1

1、在一个请求分页系统中，采用FIFO页面置换算法时，假如一个作业的页面访问顺序为4，3，2，1，4，3，5，4，3，2， 1，5，当分配给该作业的物理块数M分别为3和4时，试计算访问过程中所发生的缺页次数(包含初始装入次数)__A__和__B__，缺页率分别为A/C和B/C，其中__C__为访问次数。比较所得的结果为__D__。

A，B，C：（1）3；（2）10；（3）8；（4）9；(5)11；(6)12；(7)13

D：(1) 正常现象，即存储块增加，缺页次数减少；

(2) 存在奇异现象（Bleady），即存储块增加，缺页次数反而增加

(3) 存储块增加，缺页次数不变。

答案：（4）；（2）；（6）；（2）

例题2

2、某内存管理系统中，现假定分配给某进程4个页框，该进程中的4个逻辑页进驻内存及被调用访问的情况如下表：

页号	装入时刻	上次引用时刻	A（访问位）	M（修改位）
0	126	279	1	1
1	230	260	1	1
2	120	272	1	0
3	160	280	1	1

设若接下来要访问的进程页号是5，需要进行页面置换。（设表中时间单位:ns,NRU和改进型NRU算法中，置换指针当前位置均指向第0页所在处）

问：

（1）请问FIFO、LRU、NRU 、以及改进型的NRU各将替换哪一页？

（2）请简要叙述出这几种算法的基本思想。

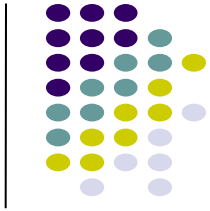
内容

- 4.1 程序的装入和链接
- 4.2 连续分配方式
- 4.3 基本分页存储管理方式
- 4.4 基本分段存储管理方式
- 4.5 虚拟存储器的基本概念
- 4.6 请求分页存储管理方式
- 4.7 页面置换算法
- 4.8 请求分段存储管理方式

请求分段存储管理方式

- 以基本分段内存管理为基础，程序运行前，只需先调入部分分段，便启动执行。其它分段在需要时，通过缺段中断处理程序临时调入
  - 请求分段中的硬件支持
  - 请求分段中的分段共享
  - 请求分段中的分段保护

请求分段中的硬件支持



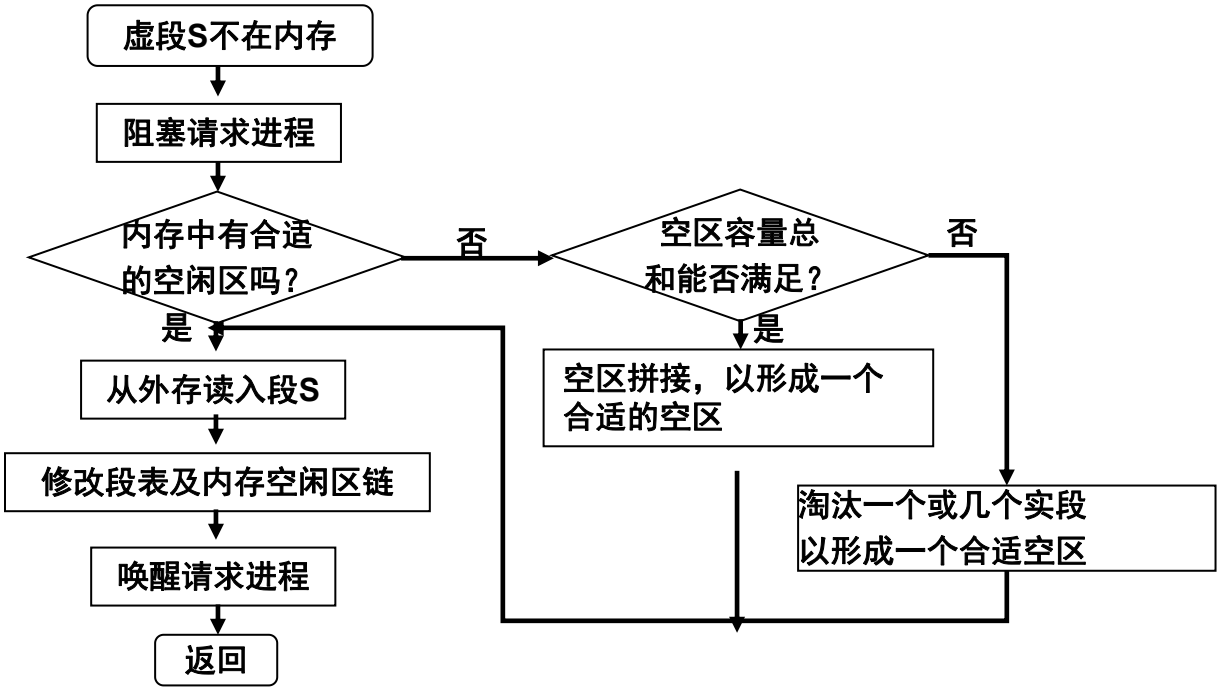
- 段表机制

段名	段长	段的基址	存取方式	访问字段A	修改位M	存在位P	增补位	外存始址

(1) 存取方式：存取属性为只执行、只读或允许读/写
(2) 访问字段A：记录该段被访问的频繁程度
(3) 修改位M：该段在进入内存后是否被修改过
(4) 存在位P：指示本段是否已调入内存
(5)增补位：表示本段在运行过程中，是否做过动态增长
(6)外存始址：本段在外存中的起始地址，即起始盘块号

缺段中断机构

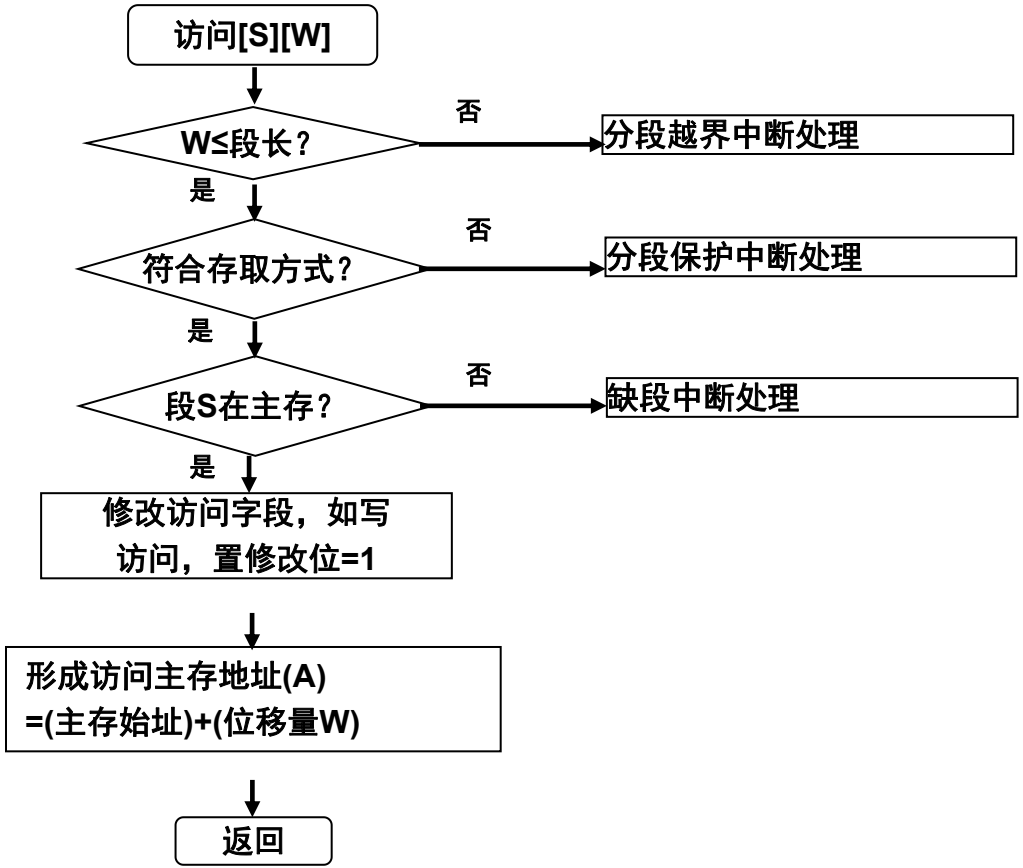
- 以信息分段为单位操作。由于分段不定长，处理时较缺页中断复杂



地址变换机构

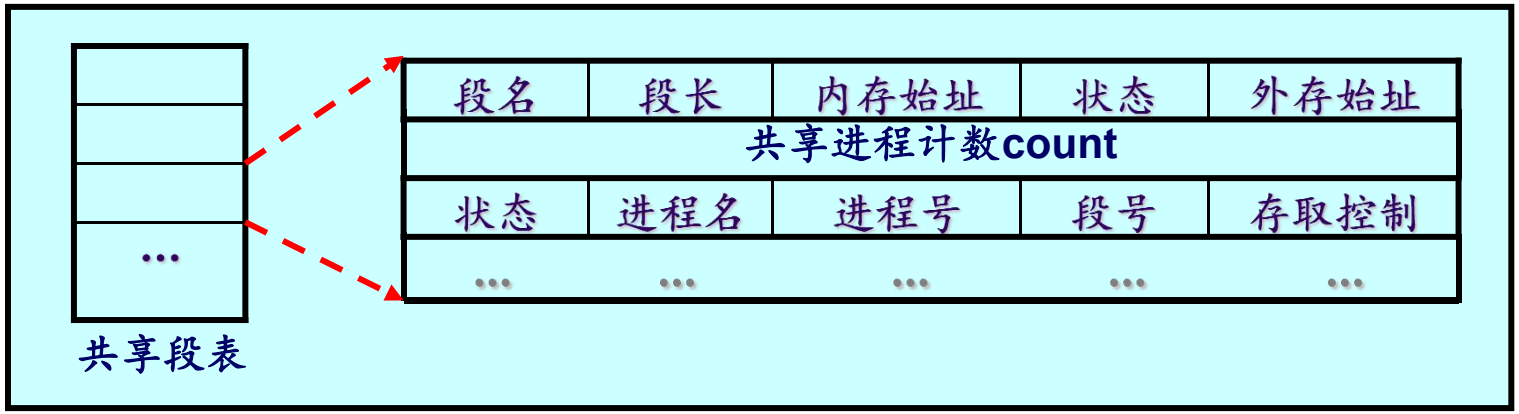
- 请求分段系统中的地址变换机构，是在分段系统地址变换机构的基础上形成的。因为被访问的段并非全在内存，因而在地址变换时，若发现要访问的段不在内存，必须先将所缺的段调入内存，并修改段表，然后才能利用段表进行地址变换。在地址变换机构中需增加缺段中断的处理及请求等功能

地址变换机构


访问[S][W]
W≤段长？
否
分段越界中断处理
是
符合存取方式？
否
分段保护中断处理
是
段S在主存？
否
缺段中断处理
是
修改访问字段，如写访问，置修改位=1
形成访问主存地址(A)=(主存始址)+(位移量W)
返回

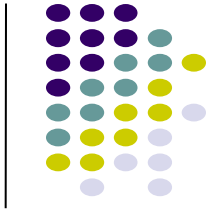
段的共享与保护

- 共享段表



- 共享进程计数count：记录要共享该段的进程数目
- 存取控制字段：对一共享段，给不同进程不同权限
- 段号：不同进程可以各用不同段号去共享某共享段

共享段的分配



- 为共享段分配内存时，对第一个请求使用该共享段的进程，由系统为该共享段分配一物理区，再把共享段调入该区，同时将该区的始址填入请求进程的段表的相应项中，还须在共享段表中增加一表项，填写有关数据，把count置为1；
- 当又有其他进程需要调用该共享段时，无需再为该段分配内存，只需在调用进程的段表中，增加一表项，填写该共享段的物理地址；在共享段的段表中，填上调用进程的进程名、存取控制等，再执行count:=count+1操作。

共享段的回收

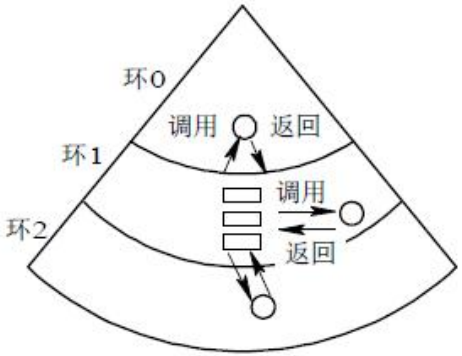
- 当共享该段的进程不再需要该段时，应将该段释放，包括撤消在该进程段表中共享段所对应的表项，以及执行count=count-1操作。如果结果为0，则需由系统回收该共享段的物理内存，以及取消在共享段表中该段所对应的表项，否则只取消调用者进程在共享段表中的有关记录

分段保护

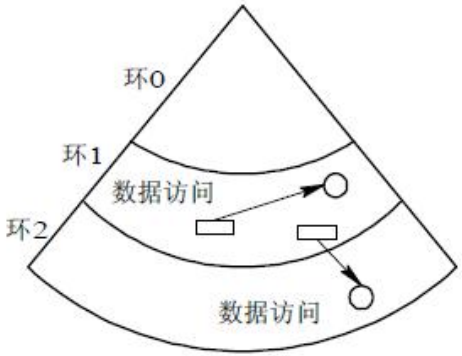
- 越界检查
  - 段表寄存器存放了段表长度；同样，在段表中也存放了每个段的段长。在进行存储访问时，将段号与段表长度比较，段内地址与段长比较
- 存取控制检查
  - 段表中的每个表项都设置了“存取控制”字段，用于规定该段的访问方式: 只读； 只执行； 读/写
- 环保护机构

分段保护

- 环保护机构
  - 规定：低编号的环具有高优先权。OS核心处于0 环内。
  - 遵循的原则：
    - 一个程序可以访问驻留在相同环或较低特权环中的数据。
    - 一个程序可以调用驻留在相同环或较高特权环中的服务



(a) 程序间的控制传输



(b) 数据访问

例题

在某页式管理系统中，假定主存为 64KB，分成 16 块，块号为 0, 1, 2,…, 15。设某进程有 4 页，其页号为 0, 1, 2, 3，被分别装入主存的第 9, 0, 1, 14 块。

1）该进程的总长度是多大？

2）写出该进程每页在主存中的始址。

3）若给出逻辑地址(0, 0), (1, 72), (2, 1023), (3, 99)，请计算出相应的内存地址（括号内的第一个数为十进制页号，第二个数为十进制页内地址）。

例题

1）页面的大小为(64/16)KB = 4KB，该进程共有 4 页，所以该进程的总长度为 4×4KB = 16KB。

2）页面大小为 4KB，因此低 12 位为页内偏移地址；主存分为 16 块，因此内存物理地址高 4 位为主存块号。

页号为 0 的页面被装入主存的第 9 块，因此该地址在内存中的始址为 **1001** 0000 0000 0000B，即 9000H。

页号为 1 的页面被装入主存的第 0 块，因此该地址在内存中的始址为 **0000** 0000 0000 0000B，即 0000H。

页号为 2 的页面被装入主存的第 1 块，因此该地址在内存中的始址为 **0001** 0000 0000 0000，即 1000H。

页号为 3 的页面被装入主存的第 14 块，因此该地址在内存中的始址为 **1110** 0000 0000 0000，即 E000H。

3）逻辑地址为(0, 0)，因此内存地址为(9, 0) = **1001** 0000 0000 0000B，即 9000H。

逻辑地址为(1, 72)，因此内存地址为(0, 72) = **0000** 0000 0100 1000B，即 0048H。

逻辑地址为(2, 1023)，因此内存地址为(1, 1023) = **0001** 0011 1111 1111，即 13FFH。

逻辑地址为(3, 99)，因此内存地址为(14, 99) = **1110** 0000 0110 0011，即 0E063H。

例题

【2014 统考真题】下列选项中，属于多级页表优点的是（ ）。 **D**

A. 加快地址变换速度

B. 减少缺页中断次数

C. 减少页表项所占字节数

D. 减少页表所占的连续内存空间

【2016 统考真题】某进程的段表内容如下所示。

段号	段长	内存起始地址	权限	状态
0	100	6000	只读	在内存
1	200	—	读写	不在内存
2	300	4000	读写	在内存

访问段号为 2、段内地址为 400 的逻辑地址时，进行地址转换的结果是（ ）。 **D**

A. 段缺失异常

B. 得到内存地址 4400

C. 越权异常

D. 越界异常

例题

【2019 统考真题】在分段存储管理系统中，用共享段表描述所有被共享的段。若进程 P_1 和 P_2 共享段 S，则下列叙述中，错误的是（ ）。 **B**

A. 在物理内存中仅保存一份段 S 的内容

B. 段 S 在 P_1 和 P_2 中应该具有相同的段号

C. P_1 和 P_2 共享段 S 在共享段表中的段表项

D. P_1 和 P_2 都不再使用段 S 时才回收段 S 所占的内存空间

【2019 统考真题】某计算机主存按字节编址，采用二级分页存储管理，地址结构如下：

页目录号（10 位）	页号（10 位）	页内偏移（12 位）

虚拟地址 2050 1225H 对应的页目录号、页号分别是（ ）。 **A**

A. 081H，101H　　B. 081H，401H　　C. 201H，101H　　D. 201H，401H

例题

【2015 统考真题】某计算机系统按字节编址，采用二级页表的分页存储管理方式，虚拟地址格式如下所示：

10 位	10 位	12 位
页目录号	页表索引	页内偏移量

请回答下列问题：

1）页和页框的大小各为多少字节？进程的虚拟地址空间大小为多少页？

2）若页目录项和页表项均占 4B，则进程的页目录和页表共占多少页？写出计算过程。

3）若某指令周期内访问的虚拟地址为 0100 0000H 和 0111 2048H，则进行地址转换时共访问多少个二级页表？说明理由。

1）页和页框大小均为 4KB。进程的虚拟地址空间大小为 $2^{32}/2^{12}=2^{20}$ 页。

2）$(2^{10}\times4)/2^{12}$（页目录所占页数）+ $(2^{20}\times4)/2^{12}$（页表所占页数）= 1025 页。

3）需要访问一个二级页表。因为虚拟地址 0100 0000H 和 0111 2048H 的最高 10 位的值都是 4，访问的是同一个二级页表。

例题

【2016 统考真题】某系统采用改进型 CLOCK 置换算法，页表项中字段 A 为访问位，M 为修改位。$A=0$ 表示页最近没有被访问，$A=1$ 表示页最近被访问过。$M=0$ 表示页未被修改过，$M=1$ 表示页被修改过。按(A, M)所有可能的取值，将页分为 (0, 0), (1, 0), (0, 1)和(1, 1)四类，则该算法淘汰页的次序为（ ）。 A

A. (0, 0), (0, 1), (1, 0), (1, 1)

B. (0, 0), (1, 0), (0, 1), (1, 1)

C. (0, 0), (0, 1), (1, 1), (1, 0)

D. (0, 0), (1, 1), (0, 1), (1, 0)

【2015 统考真题】系统为某进程分配了 4 个页框，该进程已访问的页号序列为 2, 0, 2, 9, 3, 4, 2, 8, 2, 4, 8, 4, 5。若进程要访问的下一页的页号为 7，依据 LRU 算法，应淘汰页的页号是（ ）。 A

A. 2

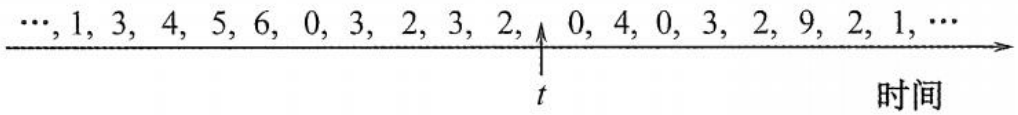
B. 3

C. 4

D. 8

例题

【2016 统考真题】某进程访问页面的序列如下所示。



若工作集的窗口大小为 6，则在 t 时刻的工作集为（ ）。 **A**

A. {6, 0, 3, 2}　　B. {2, 3, 0, 4}

C. {0, 4, 3, 2, 9}　　D. {4, 5, 6, 0, 3, 2}

【2019 统考真题】某系统采用 LRU 页置换算法和局部置换策略，若系统为进程 P 预分配了 4 个页框，进程 P 访问页号的序列为 0, 1, 2, 7, 0, 5, 3, 5, 0, 2, 7, 6，则进程访问上述页的过程中，产生页置换的总次数是（ ）。 **C**

A. 3　　B. 4　　C. 5　　D. 6