



经典教材《计算机操作系统》**最新版**

第6章 虚拟存储器

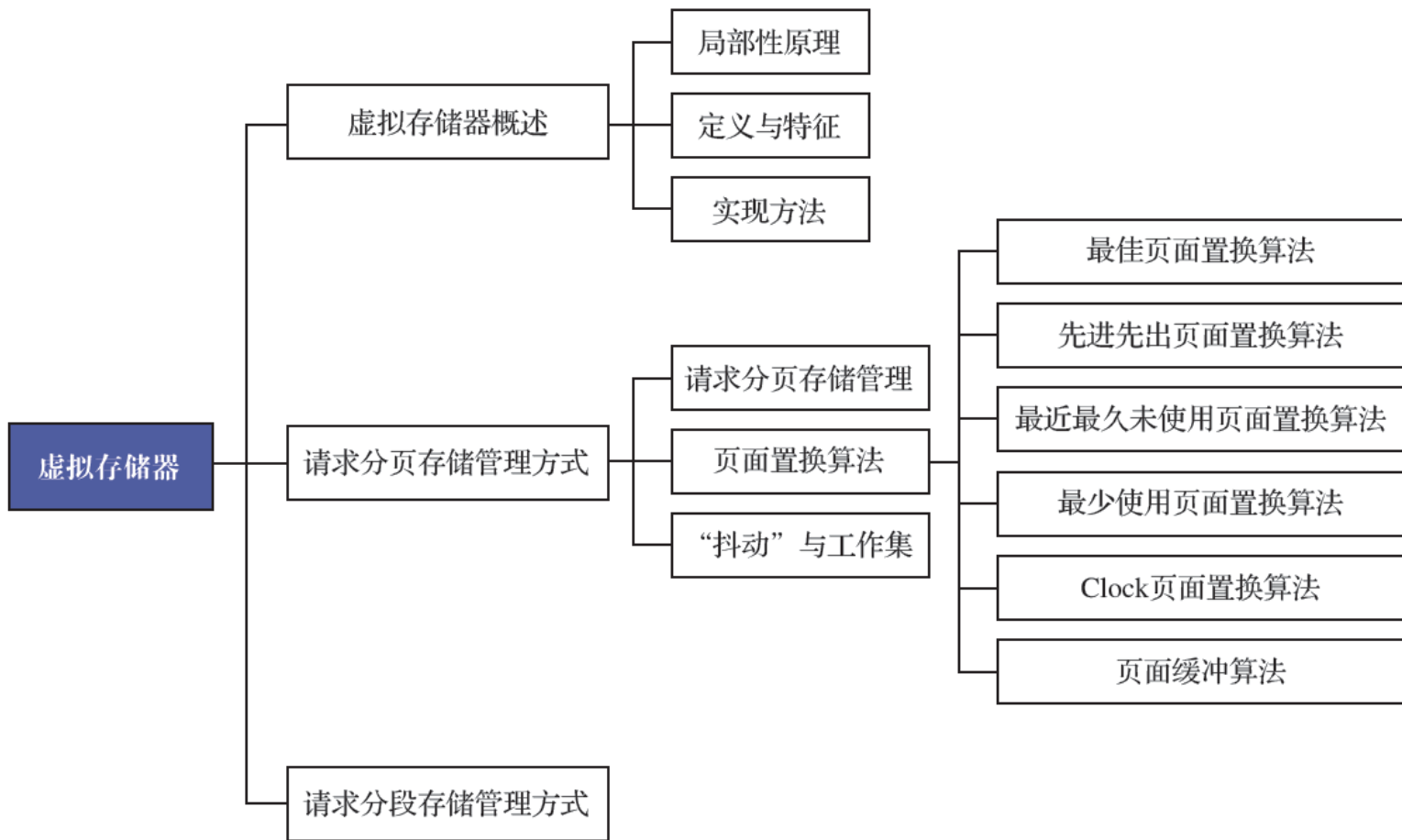
主讲教师：王申











第6章知识导图

第1章	操作系统引论
第2章	进程的描述与控制
第3章	处理机调度与死锁
第4章	进程同步
第5章	存储器管理
第6章	虚拟存储器
第7章	输入/输出系统
第8章	文件管理
第9章	磁盘存储器管理
第10章	多处理机操作系统
第11章	虚拟化和云计算
第12章	保护和安全





-  6.1 虚拟存储器概述
-  6.2 请求分页存储管理方式
-  6.3 页面置换算法
-  6.4 抖动与工作集
-  6.5 请求分段存储管理方式
-  6.6 虚拟存储器实现实例

第6章 虚拟存储器

前面所介绍的各种存储器管理方式，有一个共同特点：作业全部装入内存后方能运行。



问题：

- 大作业装不下
- 少量作业得以运行

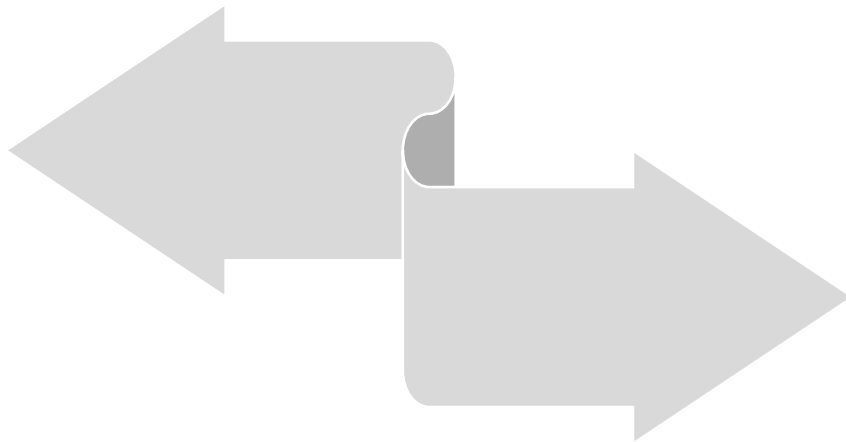


解决办法：

- 方法一：从物理上增加内存容量，**成本高**
- 方法二：从**逻辑**上扩充内存容量->虚拟存储技术（第5章已经提及该思想）



一次性：作业被一次性全部装入内存



驻留性：作业一直驻留在内存

一次性和驻留性使许多在程序运行中不用或暂不用的程序（数据）占据了大量的内存空间，使得一些需要运行的作业无法装入运行。



1968年, Peter Denning 提出:


- 程序执行时, 除了少部分的转移和过程调用外, 在**大多数情况下**仍然是**顺序执行**的。
- 过程调用将会使程序的执行轨迹由一部分区域转至另一部分区域, 过程调用的深度一般**小于5**。程序将会在一段时间内都**局限在这些过程的范围内运行**。
- 程序中存在许多**循环结构, 多次执行**。
- 对**数据结构** (例如数组) 的**处理局限于很小的范围**。




局部性表现在:

- **时间局部性**: 一条指令被执行了, 则在不久的将来它可能再被执行。
- **空间局部性**: 若某一存储单元被使用, 则在一定时间内, 与该存储单元相邻的单元可能被使用。

1. **部分装入**：在程序装入时，不必将其全部读入到内存，而只需将当前需要执行的部分页或段读入到内存，就可让程序开始执行。
2. **请求调页（段）**：在程序执行过程中，如果需执行的指令或访问的数据尚未在内存（称为缺页或缺段），利用OS提供的请求调页（段）功能，将相应的页或段调入到内存，然后继续执行程序。
3. **置换**：如果此时内存已满，无法装入新的页（段），则还必须调用页（段）置换功能。
4. **大作业+多并发**：这样，便可是一个大的用户程序，在较小的内存空间运行；也可在内存中同时装入更多进程并发运行。

IEEE  computer society

IEEE-CS Home | IEEE Computer Society History Committee


 IEEE

Computer Pioneers by J. A. N. Lee

« Dennard, Robert H. index Dessauer, John H. »

Peter J. Denning

Born January 6, 1942, New York City; computer scientist whose work on virtual memory systems helped make virtual memory a Permanent Part of modern operating systems.



Education: BE in electrical engineering, Princeton University, 1964.

Professional life: Princeton University, professor, 1964-1970; head, Computer Institute for Advanced Research Center, 1970-1990, research chair of the Center for Technology and Society, present.

DOI: 10.1145/357980.357997 • Corpus ID: 38344265

The working set model for program behavior

P. Denning • Published 1983 • Computer Science • Commun. ACM

Probably the most basic reason behind the absence of a general treatment of resource allocation in modern computer systems is an inadequate model for program behavior. In this paper a new model, the “working set model,” is developed. The working set of pages associated with a process, defined to be the collection of its most recently used pages, provides knowledge vital to the dynamic management of paged memories. “Process” and “working set” are shown to be manifestations of the same ongoing computational activity; then “processor demand” and “memory demand” are defined; and resource allocation is formulated as the problem of balancing demands against available equipment. [LESS](#)



虚拟存储器定义

基于局部性原理，应用程序在运行之前，没有必要全部装入内存，仅须将那些当前要运行的部分页面或段先装入内存便可运行，其余部分暂留在盘上。

虚拟存储器：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，而成本接近于外存。**



多次性：作业中的程序和数据允许被分成**多次**调入内存允许（相对于**一次性**而言）

对换性：作业运行时无须常驻内存，允许作业在运行过程中**换进**、**换出**（相对于**驻留性**而言）

虚拟性：从逻辑上扩充了内存容量，使用户看到的内存容量远**大于**实际内存容量（相对于**实际内存容量**而言）

Note: 虚拟性的基础是多次性和对换性，而多次性和对换性的实现基础是采用**离散的内存**分配方式

1. **实现原理**: 进程运行**只装入**部分程序和数据, 在外存保留完整副本, 运行中动态调整进程在内存中的部署
2. **技术难点**: 如何确定和记录当前**哪些**部分在内存? 执行中访问**不在**内存的指令和数据时如何处理? 从外存中调入某页时, 内存中空间**不够**如何处理?
3. **优点**: 利用率高, 方便用户, 对多道程序运行有较强的支持
4. **实现方式**: 有**两种**典型虚拟存储器实现方式



请求分页系统（以分页系统为基础）

- 硬件支持：页表、缺页中断、地址变换机构。
- 实现请求分页的软件：请求调页软件、页面置换软件。

较易（页大小固定）



请求分段系统（以分段系统为基础）

- 硬件支持：段表、缺段中断、地址变换机构。
- 实现请求分段的软件：请求调段软件、段置换软件。

较难（段大小不定）



段页式虚拟存储器







- 增加请求调页和页面置换。
- Intel 80386 及以后。

1. **页式虚拟存储**: 在分页系统的基础上, 增加了**请求调页**功能、**页面置换**功能所形成的**页式虚拟存储系统**
2. **基本思想**: 分页管理, 装入少量页运行, 缺页故障后调整
3. **页表结构**: 页号+ 标志位+ 块号+ 外存地址
4. **地址转换**: 正常地址转换, 缺页时产生缺页中断

1. **段式虚拟存储**：在分段系统的基础上，增加请求调段及分段置换功能后，所形成的**段式虚拟存储**
2. **基本思想**：装入部分段，动态**装入**或**调出**段
3. **段表结构**：段号+ 主存起址+ 长度+ 辅存起址+标志位+扩充位...
4. **缺段中断**：缺段时产生**缺段中断**
5. **地址变换机构**：逻辑地址->物理地址，增加**缺段中断**



内容导航:

-  6.1 虚拟存储器概述
-  6.2 **请求分页存储管理方式**
-  6.3 页面置换算法
-  6.4 抖动与工作集
-  6.5 请求分段存储管理方式
-  6.6 虚拟存储器实现实例

第6章 虚拟存储器

请求分页中的硬件支持



请求页表机制



缺页中断机构

- 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



地址变换机构

- 与分页内存管理方式类似



为了实现请求分页，系统要提供一定的硬件支持。除了一定容量的内存和外存，还需要有：**页表机制、缺页中断机构、地址变换机构**

问题一：怎样发现页不在内存？

扩充表——以前页表结构只包含页号和块号两个信息，这是进行地址变换机构所必须的，为了判断页面在不在主存，可在原页表上扩充！

页表机制

缺页中断机构

地址变换机构

用于将用户逻辑地址空间变换为物理地址空间。在页表中增加若干项，以便于标志程序或数据的状态。页表项：

页号	物理块号	状态位P	访问字段A	修改位M	外存地址
----	------	------	-------	------	------

状态位（存在位）P：表示该页是否调入内存（程序访问）

访问字段A：用于记录该页在某段时间内被访问的次数（置换算法换出）

修改位M（dirty位）：表示该页在调入内存后是否被修改过（置换页面）

外存地址：该页在外存上的地址，通常是**外存的**物理块号（调入页面）

页表机制

缺页中断机构

地址变换机构

- 1. 缺页中断：**在地址映射过程中，在页表中发现所要访问的页不在内存，则产生**缺页中断**。操作系统接到此中断信号后，就调出**缺页中断处理程序**，根据页表中给出的外存地址，将该页调入内存，使进程继续运行下去
- 2. 分配：**如果内存中有空闲块，则分配一页，将新调入页装入内存，并修改页表中相应页表项目的状态位及相应的内存块号
- 3. 淘汰：**若此时内存中**没有**空闲块，则要淘汰某页，若该页在内存期间**被修改**过，则要将其写回外存

页表机制

缺页中断机构

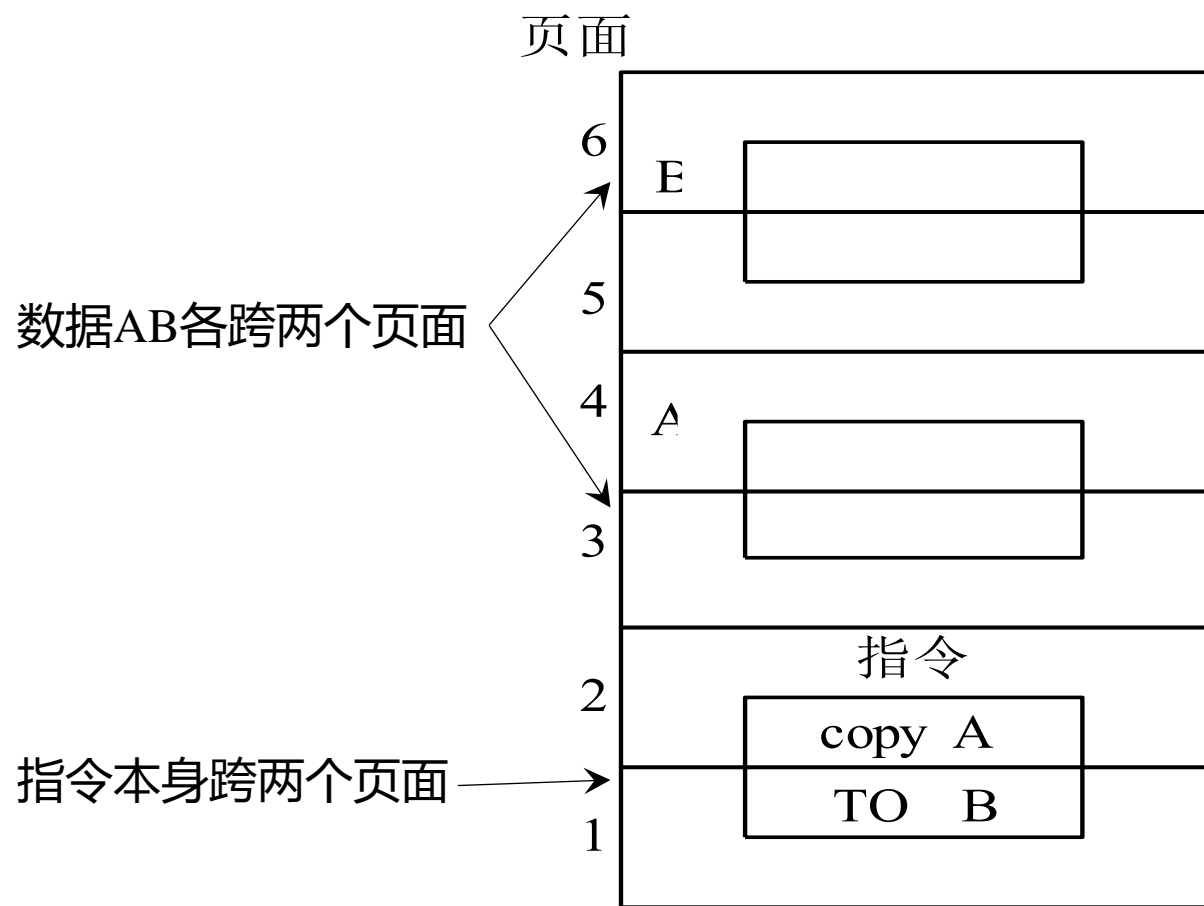
地址变换机构

1. **处理过程：**保护CPU现场、分析中断原因、转入缺页中断处理程序、恢复CPU环境
2. **特点：**缺页中断**发生在指令执行期间**，而通常情况下，CPU是在一条指令执行完后，才检查是否有中断请求到达；一条指令在执行期间，可能产生多次缺页中断。

页表机制

缺页中断机构

地址变换机构



涉及6次缺页中断的指令

硬件机构应能保存**多次中断**时的状态，并保证最后能返回到中断前产生缺页中断的指令处，继续执行



页表机制

缺页中断机构

地址变换机构

问题二：缺页中断，调入所缺页，如果内存不足，选一页淘汰，选择哪一页呢？

页表机制

缺页中断机构

地址变换机构

页号	物理块号	状态位P	访问位A	修改位M	外存地址
----	------	------	------	------	------

访问位是用来指示某页最近被访问过没有（置换算法换出）

修改位是用来指示某页的数据修改过没有（置换页面）

访问位 { “0”表示没有访问过
“1”表示已被访问过

修改位 { 1修改过 —— 写回辅存（外存）
0未修改过 —— 不必写回辅存



页表机制

缺页中断机构

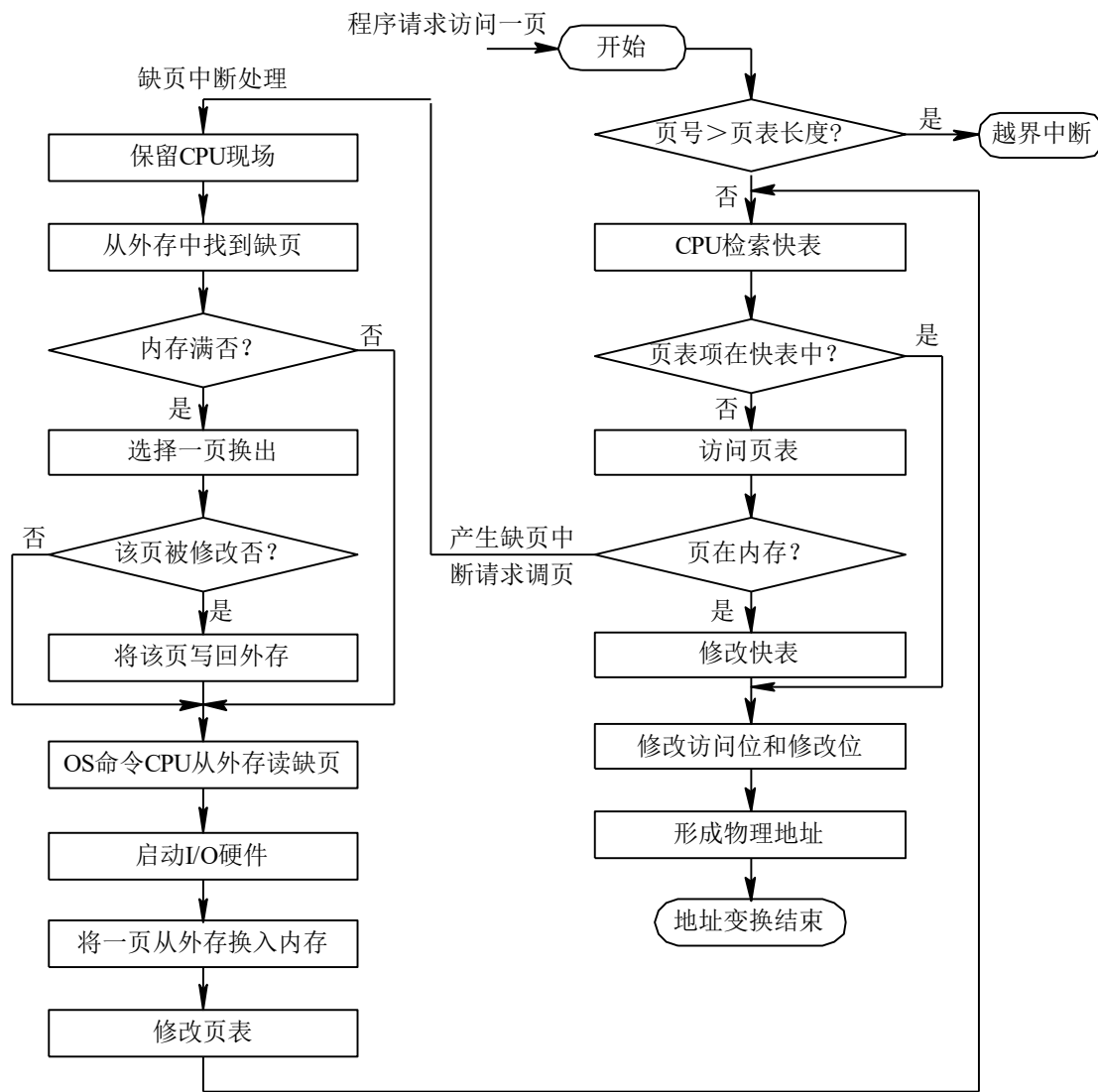
地址变换机构

问题三：如何实现地址变换？

页表机制

缺页中断机构

地址变换机构



请求分页中的地址变换过程

页表机制

缺页中断机构

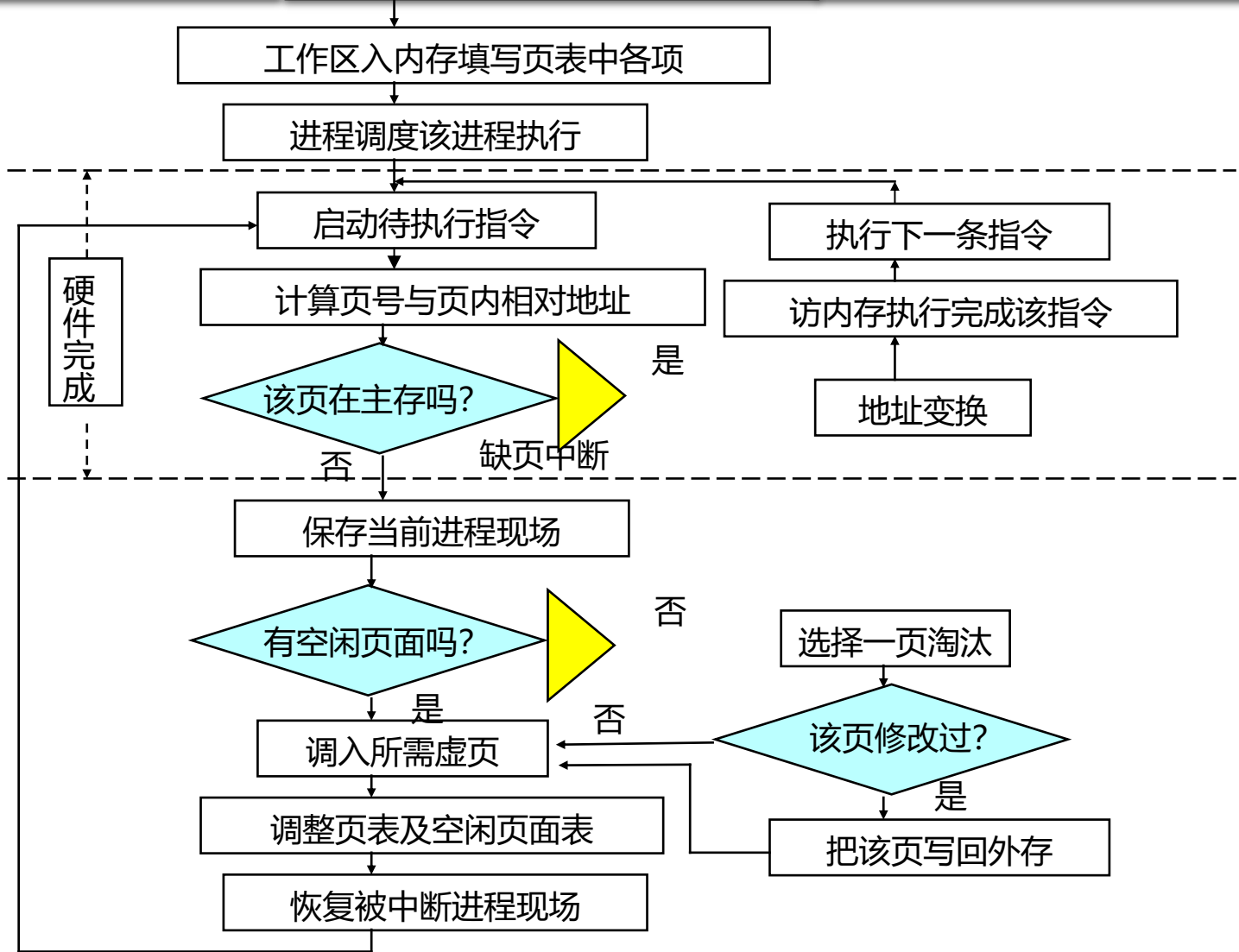
地址变换机构

1. 在地址变换时，**首先检索快表**，试图从中找到要访问的页。如找到，修改其访问位。对于“写”指令，还要设置修改位的值。如未找到，则转3。
2. 利用页表项中的物理块号和页内地址，形成物理地址。
3. 查找页表，找到页表项后，判断其状态位P，查看该页是否在内存中。如果在，则将该页写入快表（若快表已满，则应该先调出某个或某些页表项）。如果不在，则产生缺页中断，由OS从外存将该页调入内存。

页表机制

缺页中断机构

地址变换机构



请求分页的处理流程

请求分页中的内存分配

在为进程分配物理块时，要解决下列的三个问题：

1. 保证进程可正常运行所需要的**最少物理块数**
2. 每个进程的物理块数，是固定值还是可变值 **(分配策略)**
3. 不同进程所分配的物理块数，是采用平均分配算法还是根据进程的大小按照比例予以分配 **(分配算法)**

一、最小物理块数的确定

1. 最小物理块数只保证进程正常运行所需的最小物理块数。
2. 进程应获得的最小物理块数与计算机的硬件机构有关，取决于指令的格式、功能和寻址方式。

简单机器：

单地址指令直接寻址→最小物理块数：2

单地址指令间接寻址→最小物理块数：3

复杂机器（指令长度大于等于2字节）：

指令本身和两个操作数各占两个页面→最小物理块数：6



二、内存分配策略

在请求分页中，可采取两种分配策略，即**固定和可变**分配策略。

在进行置换时，也可采取两种策略，即**全局**置换和**局部**置换

（置换范围不同）。于是组合出三种适用的策略：

1. 固定分配局部置换
2. 可变分配全局置换
3. 可变分配局部置换



1、固定分配局部置换

➤ 思路

- 分配**固定数目**的内存空间，在整个运行期间都不改变（**固定**）

➤ 策略

- 如果缺页，则先**从该进程**在内存的页面中选中一页，进行换出操作，然后再调入一页（**局部**）

➤ 特点

- 为每个进程分配多少页是合适的值难以确定

2、可变分配全局置换

➤ 思路

- 每个进程预先分配**一定数目**的物理块（**可变**），同时OS也保持一个空闲物理块队列

➤ 策略

- 当缺页时，首先将对OS所占有的空闲块进行分配，从而增加了各进程的物理块数。当OS的空闲块全部用完，将**所有进程**的全部物理块为标的进行换出操作（**全局**）



3、可变分配局部置换

➤ 思路

- 每个进程预先分配**一定数目**的物理块（**可变**），如果缺页，则先**从该进程**在内存的页面中选中一页，进行换出操作（**局部**）。

➤ 特点

- 系统根据缺页率动态调整各进程占有的物理块数目，使其保持**在一个比较低的缺页率状态下**，使大部分进程可以达到比较近似的性能

三、物理块分配算法

在采用**固定分配策略**时，可使用下列方法来分配：

1. **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。
2. **按比例分配算法**：按照进程的大小比例分配物理块。
3. **考虑优先权的分配算法**：为了对于紧迫的作业，能够尽快完成。可以将内存的物理块分成两部分，一部分按照比例分配给各进程，另一部分根据进程优先级，适当增加其相应的份额，分配给各进程。

1. 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。

例如，当系统中有100个物理块，有5个进程在运行时，每个进程可分得20个物理块。这种方式貌似公平，但实际上是不公平的，因为它未考虑到各进程本身的大小。如有一个进程其大小为200页，只分配给它20个块，这样，它必然会有很高的缺页率；而另一个进程只有10页，却有10个物理块闲置未用。

2. 按比例分配算法

根据进程的大小按比例分配物理块的算法。

如果系统中共有 n 个进程，每个进程的页面数为 S_i ，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 b_i ，将有：

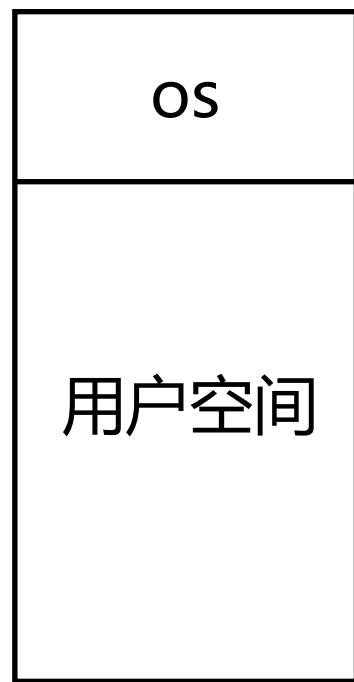
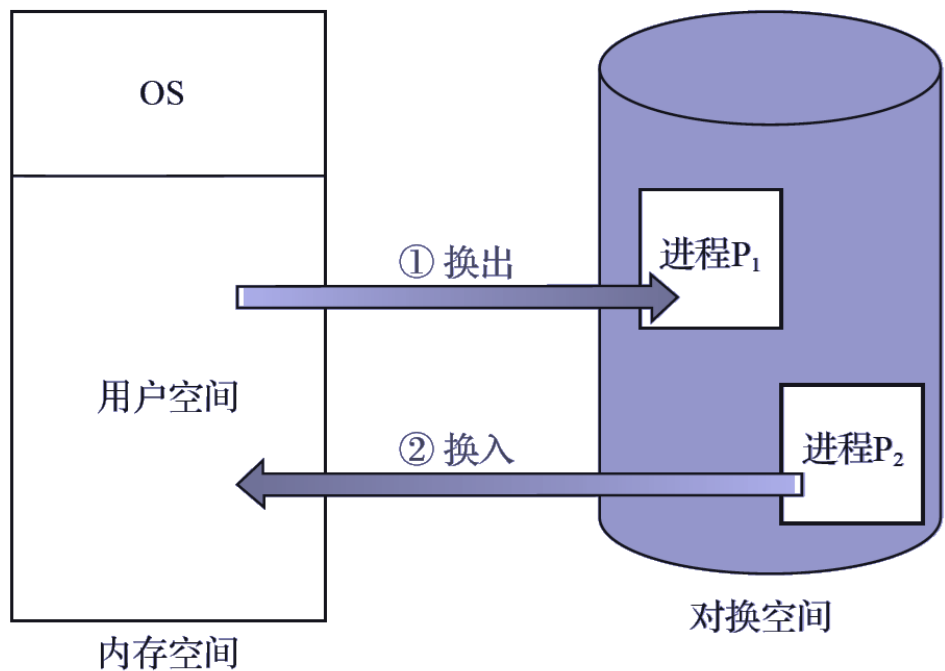
$$b_i = S_i / S \times m$$

b_i 应该取整，它必须大于最小物理块数

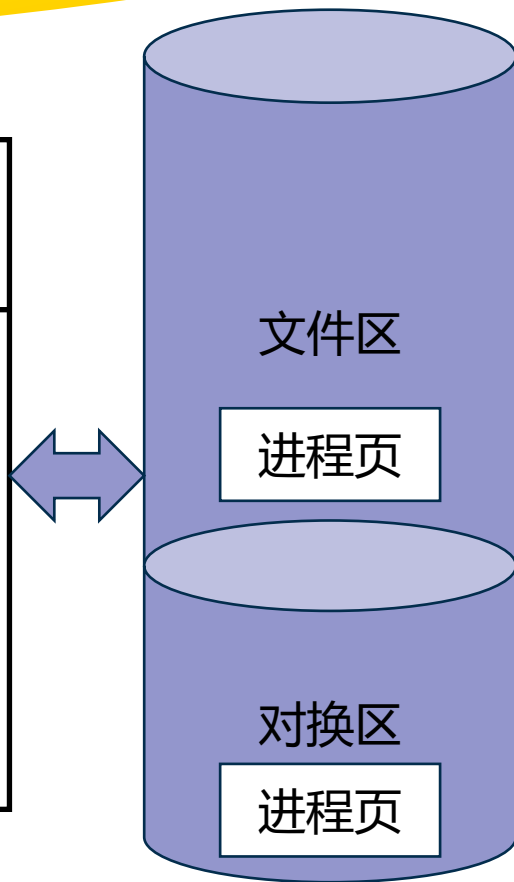
3. 考虑优先权的分配算法

在实际应用中，为了照顾到重要的、紧迫的作业能尽快地完成，应为其分配较多的内存空间。

通常采取的方法是把内存中可供分配的所有物理块分成两部分：
一部分按比例地分配给各进程；另一部分则根据各进程的优先权，适当地增加其相应份额后，分配给各进程。在有的系统中，如重要的实时控制系统，则可能是完全按优先权来为各进程分配其物理块的。



内存



外存

为提高空间利用率，采用**离散**分配方式（第9章）

为提高交换速度，采用**连续**分配方式（第9章）

第5章进程整体对换

本章进程页面对换



何时调入页面? When

- 预调页策略：预先调入一些页面到内存（预测，多页，50%的成功率）
- 请求调页策略：发现需要访问的页面不在内存时，调入内存（易，一页，开销大）



从何处调入页面? Where

- 如系统拥有足够的对换区空间，全部从对换区调入所需页面
- 如系统缺少足够的对换区空间，凡是不会被修改的文件，都直接从文件区调入；当换出这些页面时，由于未被修改而不必再将它们重写磁盘，以后再调入时，仍从文件区直接调入
- UNIX方式：未运行过的页面，从文件区调入；曾经运行过但又被换出的页面，从对换区调入



如何调入页面? How

1. 进程需要的页面不在内存, 引起缺页中断
2. 中断处理程序保留现场环境, 转入缺页中断处理程序
3. 中断处理程序查找页表, 得到对应的外存物理块号。如果内存有空闲, 则启动磁盘操作, 将所缺的页面读入, 并修改页表。否则, 到步骤4。
4. 执行置换算法, 选出要换出的页面, 如果该页修改过, 应将其写入磁盘, 然后将所缺页调入内存, 修改相应表项, 将其存在位置为 '1', 并放入快表。
5. 利用修改后的页表, 形成物理地址, 访问内存数据。

缺页率



访问页面成功(在内存)的次数为S，访问页面失败(不在内存)的次数为F

总访问次数为 $A=S+F$



缺页率为 $f= F/A$



影响因素：页面大小、分配内存块的数目、页面置换算法、程序固有属性
(局部化程度)



被置换的页面被修改的概率为 β ，其缺页中断处理时间为 t_a ；被置换的页面未被修改的缺页中断处理时间为 t_b ，则缺页中断处理时间为 t ：

$$t = \beta \times t_a + (1 - \beta) \times t_b$$

OS 无缺页，存取内存的时间 200 nano-seconds (ns)

OS 平均缺页处理时间 t 为 8 milli-seconds (ms)

OS

$$\begin{aligned} T &= (1 - f) \times 200\text{ns} + f \times 8\text{ms} \\ &= (1 - f) \times 200\text{ns} + f \times 8,000,000\text{ns} \\ &= 200\text{ns} + f \times 7,999,800\text{ns} \end{aligned}$$

OS 如果每1,000次访问中有一个缺页中断 ($f=0.001$) , 那么:

$$T = 8200 \text{ ns}$$

这是导致计算机速度放慢40倍的影响因子!



请求分页存储管理方式小结

➤ 优点:







- 可提供**多个大容量的**虚拟存储器：作业的地址空间不再受主存大小的限制
- **主存利用率**大大提高：作业中不常用的页不会长期驻留在主存，当前运行用不到的信息也不必调入主存
- 能实现**多道作业同时运行**
- **方便用户**：大作业也无须考虑覆盖问题

➤ 缺点:

- 缺页中断处理增加系统开销
- 页面的调入调出增加I/O系统的负担
- 此外，页表等占用空间且需要管理，存在页内零头



内容导航:

-  6.1 虚拟存储器概述
-  6.2 请求分页存储管理方式
-  6.3 **页面置换算法**
-  6.4 抖动与工作集
-  6.5 请求分段存储管理方式
-  6.6 虚拟存储器实现实例

第6章 虚拟存储器

- 功能:

- 需要置换页面时，选择内存中哪个物理页面被置换。

- 目标:

- 把未来不再使用的或短期内较少使用的页面调出，通常只能在局部性原理指导下依据过去的统计数据预测。

- 抖动:

- 不适当的算法会导致进程发生“抖动”，即刚换出的页很快就要被访问，又需重新调入

几种页面置换算法:

- 最佳置换算法(OPT, optimal)*
- 先进先出算法(FIFO)*
- 最近最久未使用算法(LRU, Least Recently Used)*
- 轮转算法(clock)
- 最不常用置换算法(LFU, Least Frequently Used)*
- 页面缓冲算法(page buffering)

最佳置换算法

先进先出算法

最近最久未使用算法

轮转算法

最不常用置换算法

页面缓冲算法

1. 最佳置换算法是由Belady于1966年提出的一种**理论上的**算法。其所选择的被淘汰或置换的页面，将是**以后永不使用的**，或许是在**最长(未来)时间内不再被访问**的页面。采用最佳置换算法，通常可保证获得最低的缺页率。
2. 这是一种**理想情况**，是实际执行中无法预知的（未来不可知），因而不能实现。可用作**性能评价的依据（Benchmark）**。

出发点：向后看

最佳置换算法

先进先出算法

最近最久未使用算法

轮转算法

最不常用置换算法

页面缓冲算法

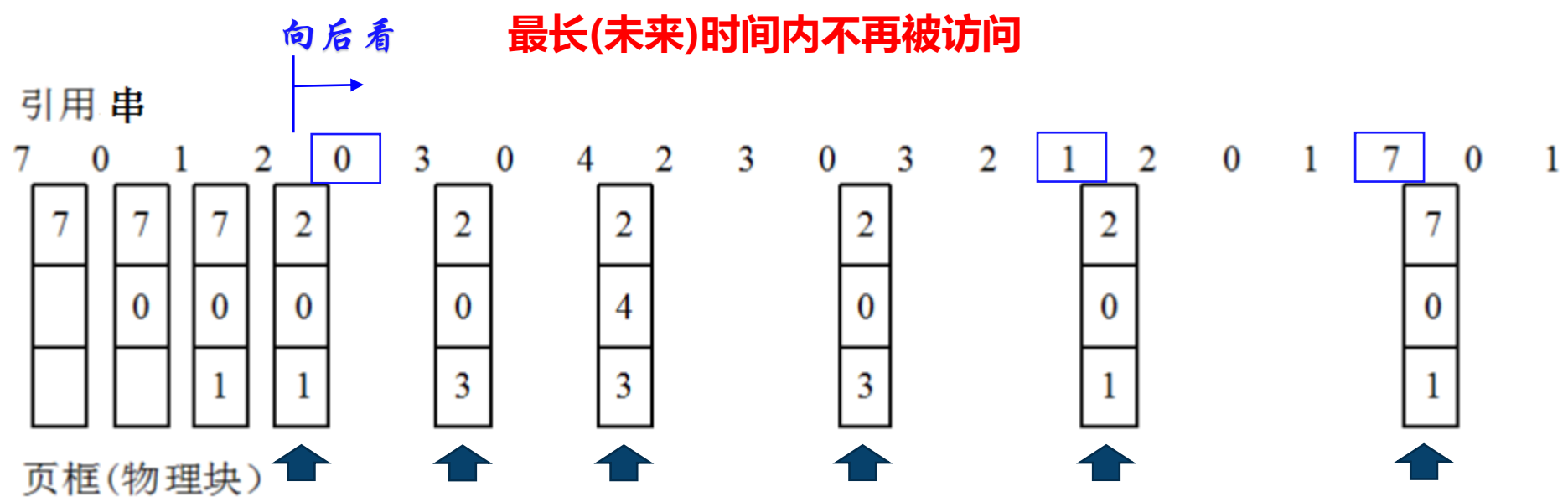
例、假定系统为某进程分配了**三个内存物理块**，并考虑有以下的页面号引用串（5个页）：

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

进程运行时，先将7, 0, 1三个页面装入内存。

当进程要访问页面2时，将会产生缺页中断。此时OS根据最佳置换算法，将选择页面7予以淘汰。

最佳置换算法	先进先出算法	最近最久未使用算法
轮转算法	最不常用置换算法	页面缓冲算法



利用最佳页面置换算法时的置换图(置换6次，缺页9次)

最佳置换算法

先进先出算法

最近最久未使用算法

轮转算法

最不常用置换算法

页面缓冲算法

1. **选择装入最早的页面被置换（内存中驻留最久的页面）**，
可以通过链表来表示各页的建立时间先后。
2. **性能较差**。较早调入的页往往是经常被访问的页，这些
页在FIFO算法下被反复调入和调出。
3. **并且有抖动现象**（刚被替换出去的页，立即又要被访问）。

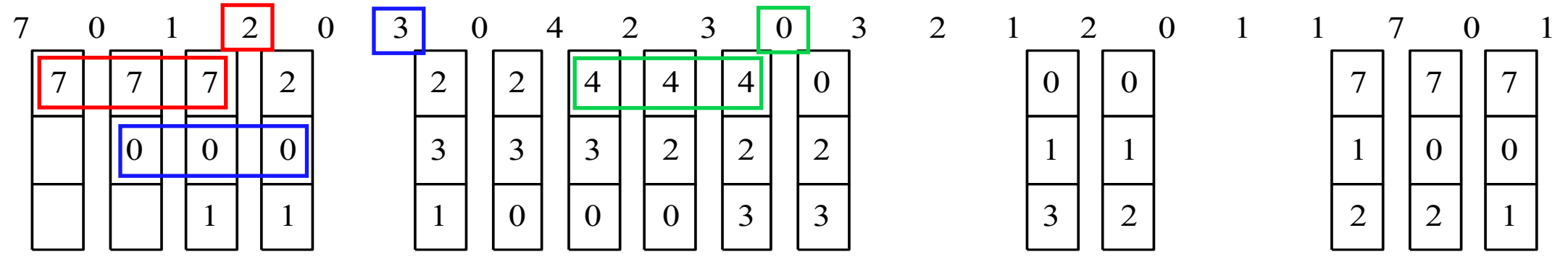


页面置换算法

最佳置换算法	先进先出算法	最近最久未使用算法
轮转算法	最不常用置换算法	页面缓冲算法

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

引用串



页框

利用FIFO置换算法时的置换图（书本上的方式）

最佳置换算法	先进先出算法	最近最久未使用算法
轮转算法	最不常用置换算法	页面缓冲算法

例：某进程运行期间页面访问序列：

A,B,C,D,A,B,E,A,B,C,D,E，分析其按照FIFO算法进行页面置换时的缺页情况（进程的页数分别是3和4）。

页面访问序列	A	B	C	D	A	B	E	A	B	C	D	E
	A	B	C	D	A	B	E	E	E	C	D	D
		A	B	C	D	A	B	B	B	E	C	C
			A	B	C	D	A	A	A	B	E	E
	+	+	+	+	+	+	+			+	+	

(这种方式更简单)

缺页次数=9； 缺页率 $f=9/12=75\%$



页面置换算法

最佳置换算法	先进先出算法	最近最久未使用算法
轮转算法	最不常用置换算法	页面缓冲算法

页面访问序列	A	B	C	D	A	B	E	A	B	C	D	E
	A	B	C	D	D	D	E	A	B	C	D	E
		A	B	C	C	C	D	E	A	B	C	D
			A	B	B	B	C	D	E	A	B	C
				A	A	A	B	C	D	E	A	B
				+	+	+	+	+	+	+	+	+

缺页次数=10; 缺页率 $f=10/12=83\%$

驻留集越大，缺页中断率越小吗？

贝莱迪 (Belady) 异常

最佳置换算法	先进先出算法	最近最久未使用算法
轮转算法	最不常用置换算法	页面缓冲算法

Belady异常现象：一般而言，分配给进程的物理块越多，运行时的缺页次数应该越少。但是Belady在1969年发现了一个反例，使用FIFO算法时，四个物理块时的缺页次数比三个物理块时的多，这种反常的现象称为**Belady异常**（只有FIFO才会出现该现象，因为FIFO并没有考虑程序执行的局部性原理，使得换出的页可能是频繁访问的页）。

最佳置换算法

先进先出算法

最近最久未使用算法

轮转算法

最不常用置换算法

页面缓冲算法

1. 当需要淘汰一个页面时，总是选择在**最近一段时间内最久不用**的页面予以淘汰（**淘汰最久没有用过的**）
2. 利用局部性原理，根据一个作业在执行过程中**过去的页面访问历史来推测未来的行为**，性能接近最佳算法（“最近的过去”对“最近的将来”的近似）
3. 但由于需要记录**页面使用时间的先后关系**，**硬件开销太大**

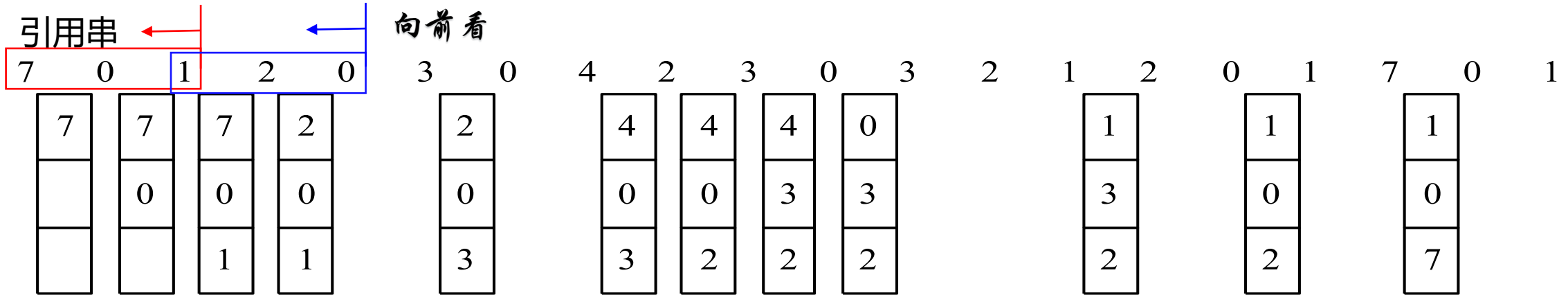
出发点：向前看



页面置换算法

最佳置换算法	先进先出算法	最近最久未使用算法
轮转算法	最不常用置换算法	页面缓冲算法

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



页框

LRU页面置换算法 (书本上的方式)



页面置换算法

最佳置换算法	先进先出算法	最近最久未使用算法
轮转算法	最不常用置换算法	页面缓冲算法

页面访问序列

A	B	C	D	A	B	E	A	B	C	D	E
A	B	C	D	A	B	E	A	B	C	D	E
	A	B	C	D	A	B	E	A	B	C	D
		A	B	C	D	A	B	E	A	B	C
+	+	+	+	+	+	+			+	+	+

缺页次数=10; 缺页率=10/12=83%

(这种方式更简单)

最佳置换算法

先进先出算法

最近最久未使用算法

轮转算法

最不常用置换算法

页面缓冲算法

硬件支持

- 寄存器

- 每个页面设立移位寄存器：被访问时左边最高位置1，定期右移并且最高位补0，于是寄存器数值最小的是最久未使用页面。

- 栈

- 一个特殊的栈：把被访问的页面移到栈顶，于是栈底的是最久未使用页面。

最佳置换算法

先进先出算法

最近最久未使用算法

轮转算法

最不常用置换算法

页面缓冲算法

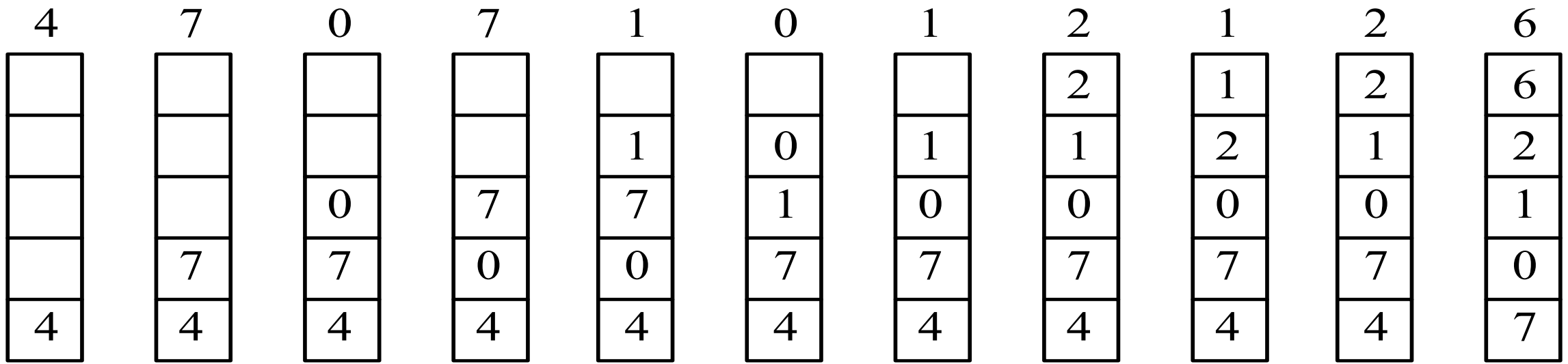
1) 寄存器

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

某进程具有8个页面时的LRU访问情况（8位寄存器R₇-R₀）

最佳置换算法	先进先出算法	最近最久未使用算法
轮转算法	最不常用置换算法	页面缓冲算法

2) 栈



用栈保存当前使用页面时栈的变化情况





页面置换算法

最佳置换算法	先进先出算法	最近最久未使用算法
轮转算法	最不常用置换算法	页面缓冲算法

	时钟周期0	时钟周期1	时钟周期2	时钟周期3	时钟周期4
页面0	10000000	11000000	11100000	11110000	01111000
页面1	00000000	10000000	11000000	01100000	10110000
页面2	10000000	01000000	00100000	00010000	10001000
页面3	00000000	00000000	10000000	01000000	00100000
页面4	10000000	11000000	01100000	10110000	01011000
页面5	10000000	01000000	10100000	01010000	00101000

软件模拟LRU的老化算法



页面置换算法

最佳置换算法

先进先出算法

最近最久未使用算法

轮转算法

最不常用置换算法

页面缓冲算法

国产操作系统openEuler采用的是LRU页面置换算法，openEuler定义了5种链表，分别记录5种不同的页类型，以辅助淘汰页的过程。

最佳置换算法

先进先出算法

最近最久未使用算法

轮转算法

最不常用置换算法

页面缓冲算法

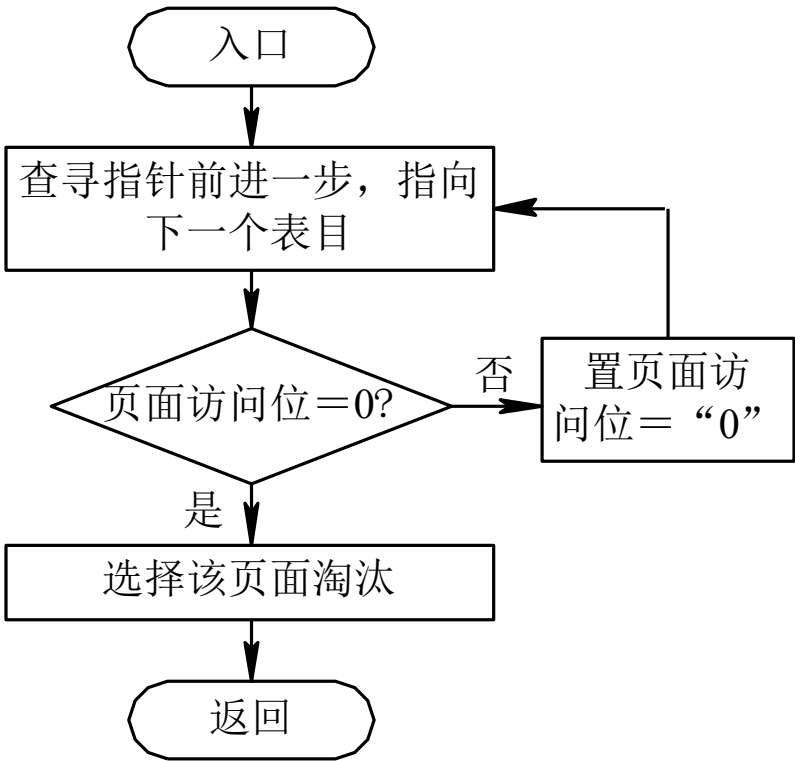
1、简单的Clock置换算法 or 最近未使用算法(NRU, Not Recently Used)

它是最近最久未使用算法 (LRU) 和FIFO的折衷。

- 内存中所有页面通过链接指针形成一个循环队列
- 每页有一个使用访问位(use bit)，若该页被访问则置use bit=1
- 置换时采用一个指针，从当前指针位置开始按地址先后 (FIFO) 检查各页，寻找use bit=0的页面作为被置换页
- 指针经过的user bit=1的页都修改user bit=0，最后指针停留在被置换页的下一个页

最佳置换算法	先进先出算法	最近最久未使用算法
轮转算法	最不常用置换算法	页面缓冲算法

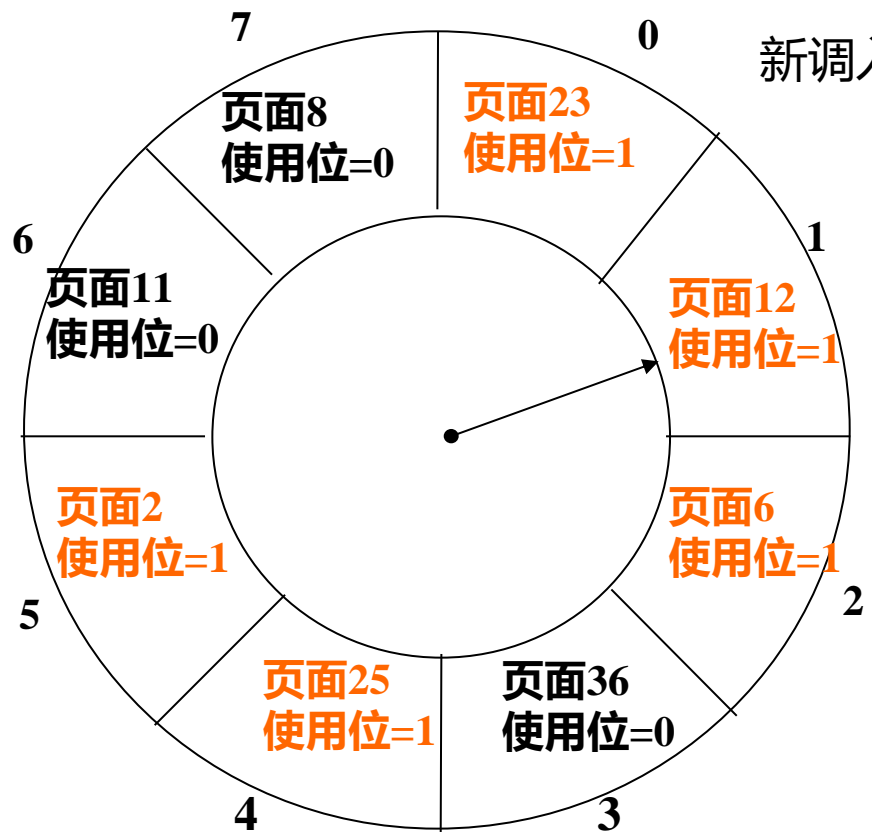
1、简单的Clock置换算法（最近未用算法NRU）



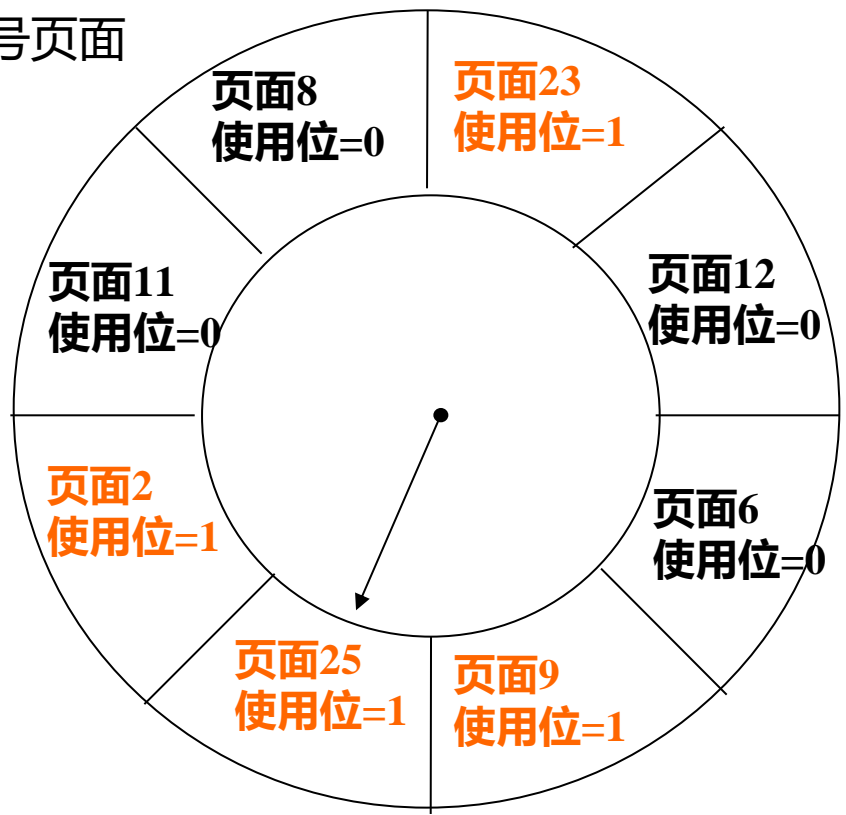
块号	页号	访问位	指针
0			
1			
2	4	0	
3			
4	2	1	
5			
6	5	0	
7	1	1	

替换指针

最佳置换算法	先进先出算法	最近最久未使用算法
轮转算法	最不常用置换算法	页面缓冲算法



(a) 页面置换前状态



(b) 页面置换后状态

最佳置换算法

先进先出算法

最近最久未使用算法

轮转算法

最不常用置换算法

页面缓冲算法

2、改进型Clock置换算法

- 除了考虑页面使用情况外，还需要考虑**置换代价**（换出的页面如果被修改过的话，则必须做拷回磁盘处理，开销比较大）。
- 于是，改进型的Clock算法为每个页又增加了一个**修改位**，选择页面时，**尽量选择既未使用又没有修改的页面**。

最佳置换算法

先进先出算法

最近最久未使用算法

轮转算法

最不常用置换算法

页面缓冲算法

2、改进型Clock置换算法

- 页面：（访问位A，修改位M）有四种不同情形：
 - 1类($A=0, M=0$)：最近既未访问，又没有修改，**最佳淘汰页**
 - 2类($A=0, M=1$)：最近未访问，但是已被修改，**效率低的淘汰页**
 - 3类($A=1, M=0$)：最近被访问，但没有修改，可能再次被访问
 - 4类($A=1, M=1$)：最近既被访问，又有修改，可能再次被访问
- 算法核心思路：循环查找第1类/2类页面，找到为止

最佳置换算法

先进先出算法

最近最久未使用算法

轮转算法

最不常用置换算法

页面缓冲算法

2、改进型Clock置换算法

- 算法：

1. 指针从当前位置开始，开始第一轮扫描循环队列，寻找未使用且没有修改过的页面（第1类页面），找到则可换出
2. 如果找不到，则开始第二轮扫描，寻找未使用但修改过的页面（第2类页面），并且每经过一个页面时，将其访问位A设置为0。如果找到一个第2类页面，则可换出
3. 如果仍旧未找到合适的换出页面，则此时指针回到初始位置，且所有页面其访问位A为0。再转回（1）继续工作

最佳置换算法

先进先出算法

最近最久未使用算法

轮转算法

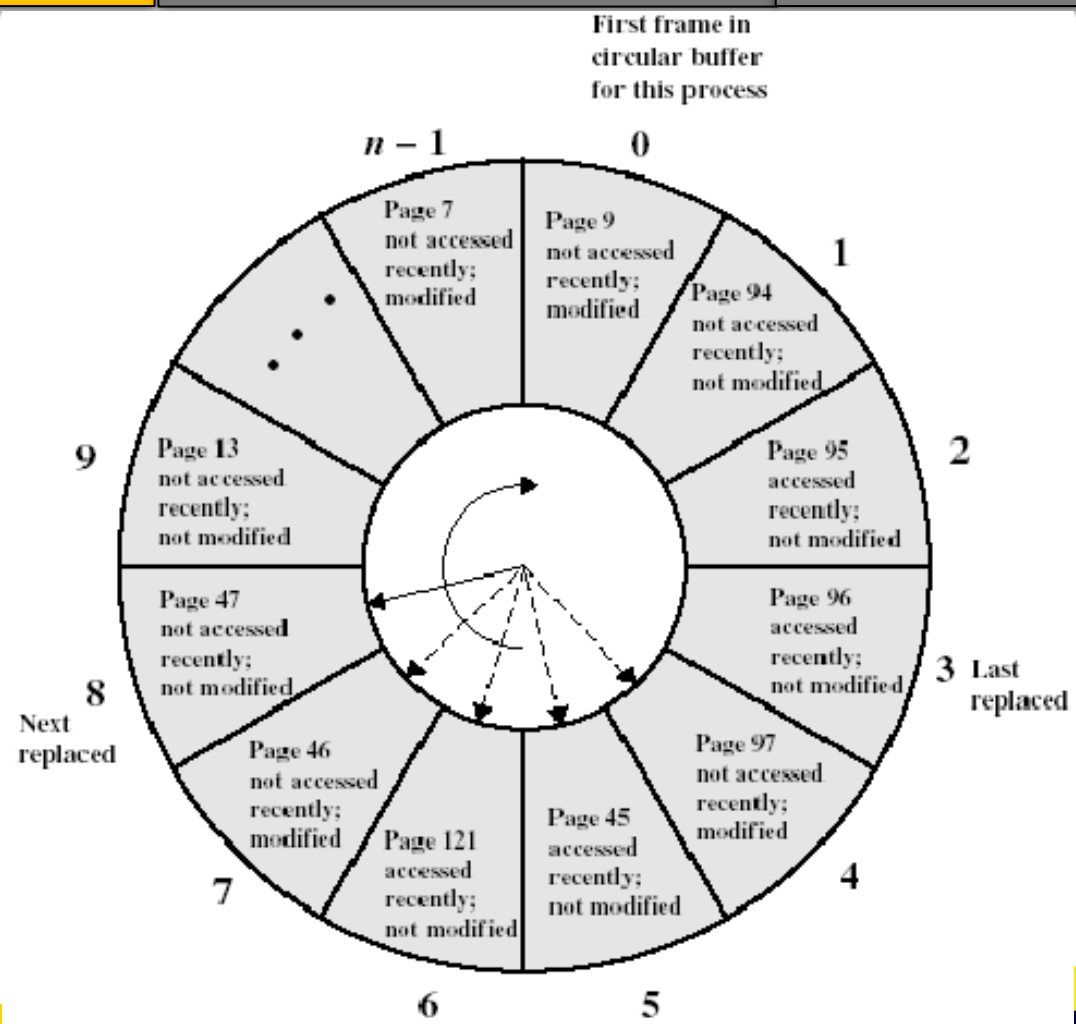
最不常用置换算法

页面缓冲算法

2、改进型Clock置换算法

- 改进型的Clock算法，与简单的Clock相比，可以减少磁盘I/O的次数，但是为了找到一个可置换的页面，可能需要几轮扫描，换言之，实现该算法本身开销有所增加。

最佳置换算法	先进先出算法	最近最久未使用算法
轮转算法	最不常用置换算法	页面缓冲算法



最佳置换算法

先进先出算法

最近最久未使用算法

轮转算法

最不常用置换算法

页面缓冲算法

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

块号	页号	访问位 (A)	修改位 (M)
5	7	1	1
2	0	1	0
8	1	1	1

最佳置换算法

先进先出算法

最近最久未使用算法

轮转算法

最不常用置换算法

页面缓冲算法

- 目的

- 选择到当前时间为止被访问次数最少的页面被置换（淘汰访问频率最低的）；

- 实现方法1（与LRU完全相同）

- 每个页面设立移位寄存器：被访问时左边最高位置1，定期右移并且最高位补0，这样，在最近一段时间内时用最少的页面将是 $\sum R_i$ 最小的页。

- 实现方法2

- 每页设置访问计数器，每当页面被访问时，该页面的访问计数器加1；发生缺页中断时，淘汰计数值最小的页面，并将所有计数清零。

最佳置换算法

先进先出算法

最近最久未使用算法

轮转算法

最不常用置换算法

页面缓冲算法

- 说明

- 实现方法1：移位寄存器这套硬件既可以实现LFU，又可以实现LRU，而且两个算法的页面访问图完全相同
- LFU并不能真正反映页面使用情况：LFU的计数器是累积值，没有时间衰减机制，导致历史访问权重过高。例如：
 - 页面A在早期被密集访问1000次，但最近未被访问。
 - 页面B在过去被访问10次，但最近被访问了1次。
 - LFU会优先淘汰页面B（因总次数 $10 < 1000$ ），但实际页面A可能已是“冷数据”。

最佳置换算法

先进先出算法

最近最久未使用算法

轮转算法

最不常用置换算法

页面缓冲算法

1、影响页面换进换出效率的若干因素

- ① 页面置换算法（最重要的因素）
- ② 写回磁盘的频率（已被修改的页面，需写回磁盘，如果页面可以累计到一定个数，批量写会磁盘，那么频率会显著降低）
- ③ 读入内存的频率（从磁盘读入页面耗时耗力，怎么能降低从磁盘读入的频率？）

最佳置换算法

先进先出算法

最近最久未使用算法

轮转算法

最不常用置换算法

页面缓冲算法

2、 页面缓冲算法 Page Buffering Algorithm

- 内存分配采用可变分配和局部置换，置换算法为FIFO
- 它是对FIFO算法的发展，通过被置换页面的缓冲，有机会找回刚被置换的页面；
- 被置换页面的选择和处理：由操作系统中专门的页面置换进程，用FIFO算法选择被置换页，把被置换的页面放入两个链表（空闲页面链表和已修改页面链表）之一
 - 如果页面未被修改，就将其归入到空闲页面链表的末尾
 - 否则将其归入到已修改页面链表。

最佳置换算法

先进先出算法

最近最久未使用算法

轮转算法

最不常用置换算法

页面缓冲算法

2、 页面缓冲算法 PBA

- 显著降低了页面换进换出的频率
- 换进换出开销大幅减少，采用简单置换策略如先进先出算法，无需特殊硬件支持，实现简单
- VAX/VMS等系统采用

与基本分页存储不同，请求分页系统，EAT还需考虑缺页中断处理的时间

- 访问页表、访问实际物理地址数据、**缺页中断处理**

λ : 查快表所需时间; t : 访问一次主存所需时间

1. 被访问页在主存，且相应页表项在快表

$$EAT = \lambda + t$$

2. 被访问页在主存，但相应页表项不在快表

$$EAT = \lambda + t + \lambda + t = 2(\lambda + t)$$

查快表 + 查页表 + 修改快表 + 访问主存读取数据

与基本分页存储不同，请求分页系统，EAT还需考虑缺页中断处理的时间

- 访问页表、访问实际物理地址数据、**缺页中断处理**

λ : 查快表所需时间; t : 访问一次主存所需时间; ε : **缺页中断处时间**

3. 被访问页不在主存

$$EAT = \lambda + t + \varepsilon + \lambda + t = \varepsilon + 2(\lambda + t)$$

查快表 + 查页表 + **缺页中断处理** + 修改快表 + 访问主存读取数据

λ : 查快表所需时间; t : 访问一次主存所需时间; ε : 缺页中断处时间

a : 快表命中率; f : 缺页率







考虑快表命中率和缺页率后

$$\begin{aligned} \text{EAT} &= a(\lambda+t) + (1-a)[\lambda + f(\varepsilon+\lambda+t) + (1-f)(\lambda+t) + t] \\ &= \lambda + at + (1-a)[f(\varepsilon+\lambda+t) + (1-f)(\lambda+t) + t] \end{aligned}$$


- ❖ 命中: 在快表找到页号和块号用时 λ , 访问物理地址 t
- ❖ 未命中:
 - 在快表没找到页号, 但还是用时 λ
 - 缺页 f , 内存中找不到对应页号, 但还是耗时 t , 发生缺页中断耗时 ε (该页调入内存), 更新页表耗时 λ
 - 未缺页 $1-f$, 在内存中找到了页表中的页号和对应物理块号耗时 t , 更新页表耗时 λ
 - 访问物理地址 t




内容导航:

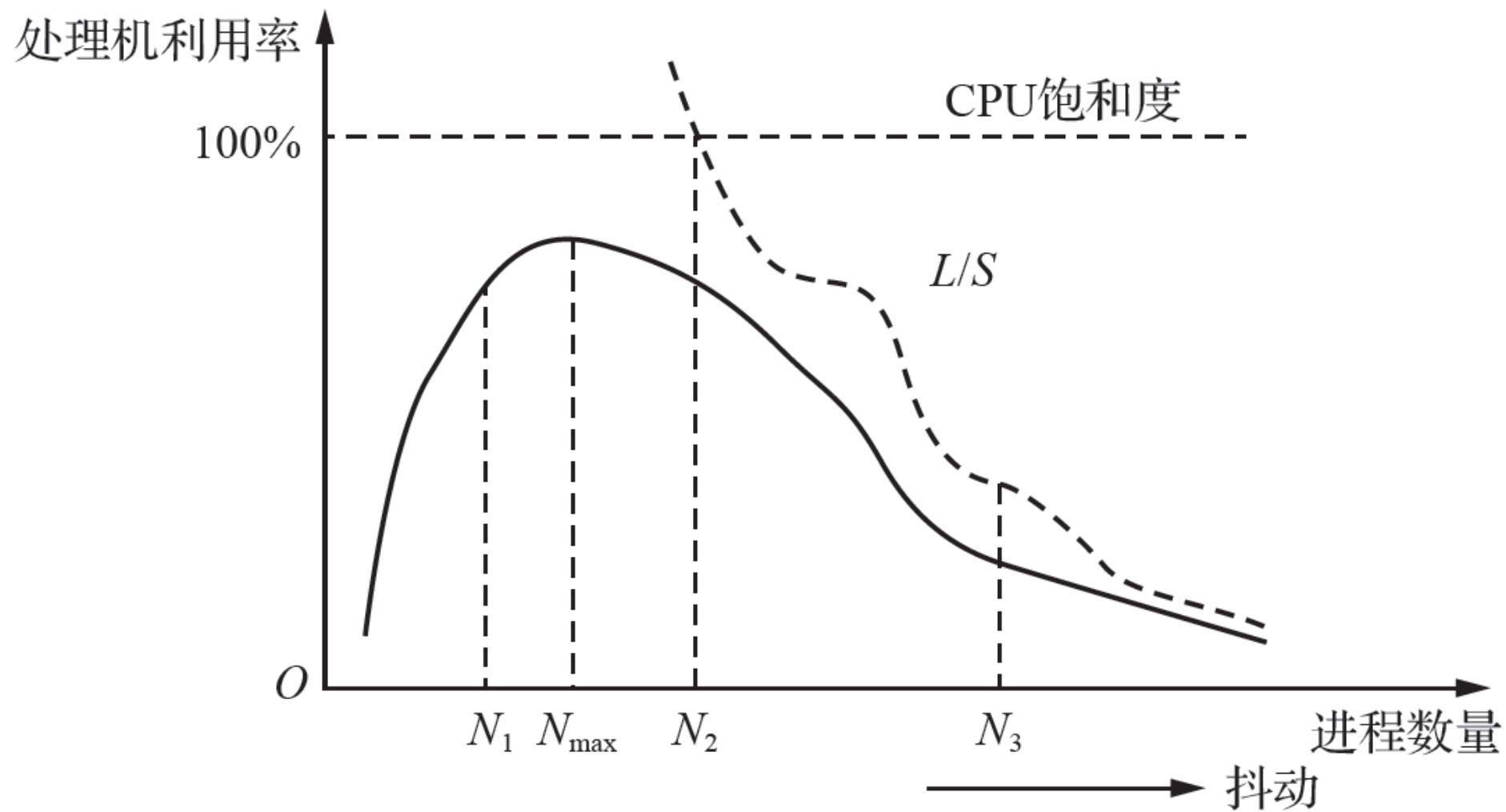
-  6.1 虚拟存储器概述
-  6.2 请求分页存储管理方式
-  6.3 页面置换算法
-  **6.4 抖动与工作集**
-  6.5 请求分段存储管理方式
-  6.6 虚拟存储器实现实例

第6章 虚拟存储器

 如果一个进程没有足够的页，那么缺页率将很高，这将导致：

- CPU利用率低下
- 操作系统认为需要增加多道程序设计的道数
- 系统中将加入一个新的进程

 **抖动**(Thrashing)：一个进程的页面经常换入换出





产生“抖动”的原因

驻留集：请求分页存储管理中给进程分配的物理页面（块）的集合。



根本原因（多道程序度过高，导致平均驻留集过小）

- 同时在系统中运行的进程太多
- 因此分配给每一个进程的物理块太少，不能满足进程运行的基本要求，致使进程在运行时，频繁缺页，必须请求系统将所缺页面调入内存
- 大部分时间都在进行换入换出操作，CPU所做的有效工作减少



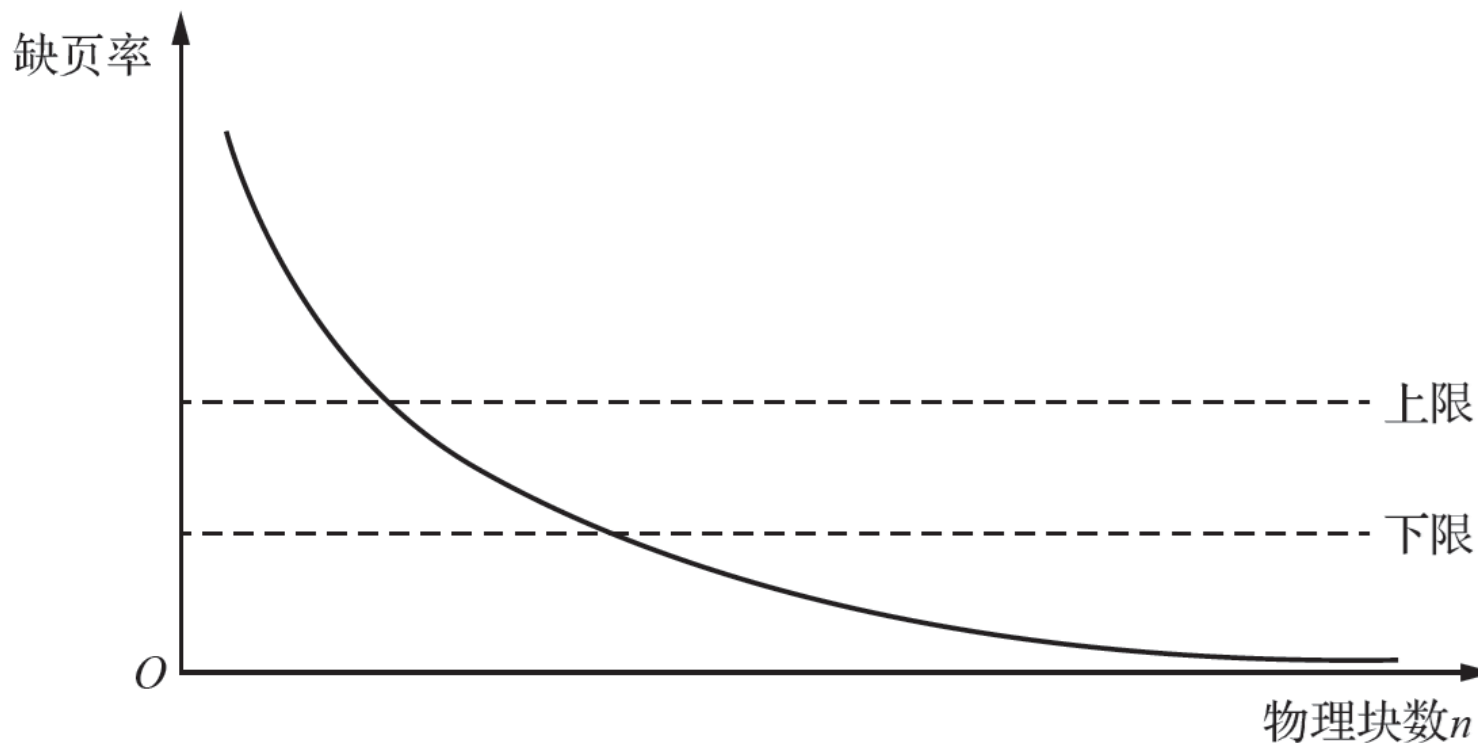
抖动的发生与系统为进程分配物理块的多少（驻留集的大小）有关，因此提出了“工作集”的概念。



进程发生缺页的时间间隔与所获得的物理块数有关。



根据程序运行的局部性原理，如果能够预知某段时间内程序要访问的页面，并将它们预先调入内存，将会大大降低缺页率。



- OS 所谓**工作集**，指在**某段时间间隔 Δ** 里**进程实际要访问页面的集合**。工作集理论是1968年由Denning提出并推广的。
- OS 把某进程在时间 t 的工作集记为 $w(t, \Delta)$,其中的变量 Δ 称为工作集的“窗口尺寸”。
 - ◆ 工作集 $w(t, \Delta)$ 是二元函数，即在不同时间 t 的工作集大小不同，所含的页面数也不同；工作集与窗口尺寸 Δ 有关，是 Δ 的非降函数，即：

$$w(t, \Delta) \subseteq w(t, \Delta+1)$$

工作集算法：为了较少地产生缺页，应使程序的全部工作集装入内存中（实现方法：以过去预测未来）

窗口大小

访问页面序列	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	—	—

$$w(t, \Delta) \subseteq w(t, \Delta+1)$$

01

采取**局部置换策略**：只能在分配给自己的内存空间内进行置换；

02

把工作集算法融入到处理机调度中（根据窗口大小预先调入页面到内存）；

03

利用 “ **$L=S$** ”**准则**调节缺页率：

➤ L 是缺页之间的平均时间

➤ $L < S$ ，说明频繁缺页

➤ S 是平均缺页服务时间，即用于置换一个页面的时间

➤ $L = S$ ，磁盘和处理机都可达到最大利用率







➤ $L > S$ ，说明很少发生缺页

04

选择暂停进程。



内容导航:

-  6.1 虚拟存储器概述
-  6.2 请求分页存储管理方式
-  6.3 页面置换算法
-  6.4 抖动与工作集
-  **6.5 请求分段存储管理方式**
-  6.6 虚拟存储器实现实例

第6章 虚拟存储器

● OS运行过程中何时需要考虑与分页有关的操作？

- 进程创建：为进程创建并初始化页表，在磁盘交换区中分配空间
- 进程运行：占用CPU时，需更新MMU、刷新快表TLB
- 页面失效：通过PC确定引发缺页中断的页面，然后进行物理页帧分配
- 进程终止：释放页表、页面、磁盘交换空间，共享页面需要保留

● 指令备份与页面锁定

- 页面失效处理流程：硬件陷入 - 状态保存 - 缺页中断 - 页面替换 - 恢复运行
- 指令备份的作用：缺页中断的精确位置难以确定，备份PC内容可保证恢复进程运行时的正确性
- 页面锁定的用途：I/O设备的DMA传输与CPU并行工作，不能被换出

● 后备存储

- 磁盘交换区的作用：页面替换时临时保存物理页面的磁盘空间
- 磁盘交换区空间管理：进程创建时分配（固定）、页面替换时分配（动态）
- 磁盘交换区的实现：OS提供的临时磁盘文件、需要考虑内存增长、信息记录

1. 请求分页系统建立的虚拟存储器，是以**页面为单位**进行换入、换出操作的。
2. 在请求分段系统中实现的虚拟存储器，以**分段为单位**进行换入和换出。
3. 程序在运行之前，只需要装入**必要的若干个分段**即可运行。当访问的分段不在内存时，可由OS将所缺少的段调入内存。

段表机制

缺段中断机构

地址变换机构

- 存取方式：表示段存取属性为只执行、只读或允许读/写
- 访问字段A：记录该段在一段时间内被访问的次数
- 修改位M：标志该段调入内存后是否被修改过
- 存在位P：指示该段是否在内存
- 增补位：表示该段在运行过程中是否做过动态增长, **在请求页式中没有该位**
- 外存始址：指示该段在外存中的起始地址（盘块号）

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

段表机制

缺段中断机构

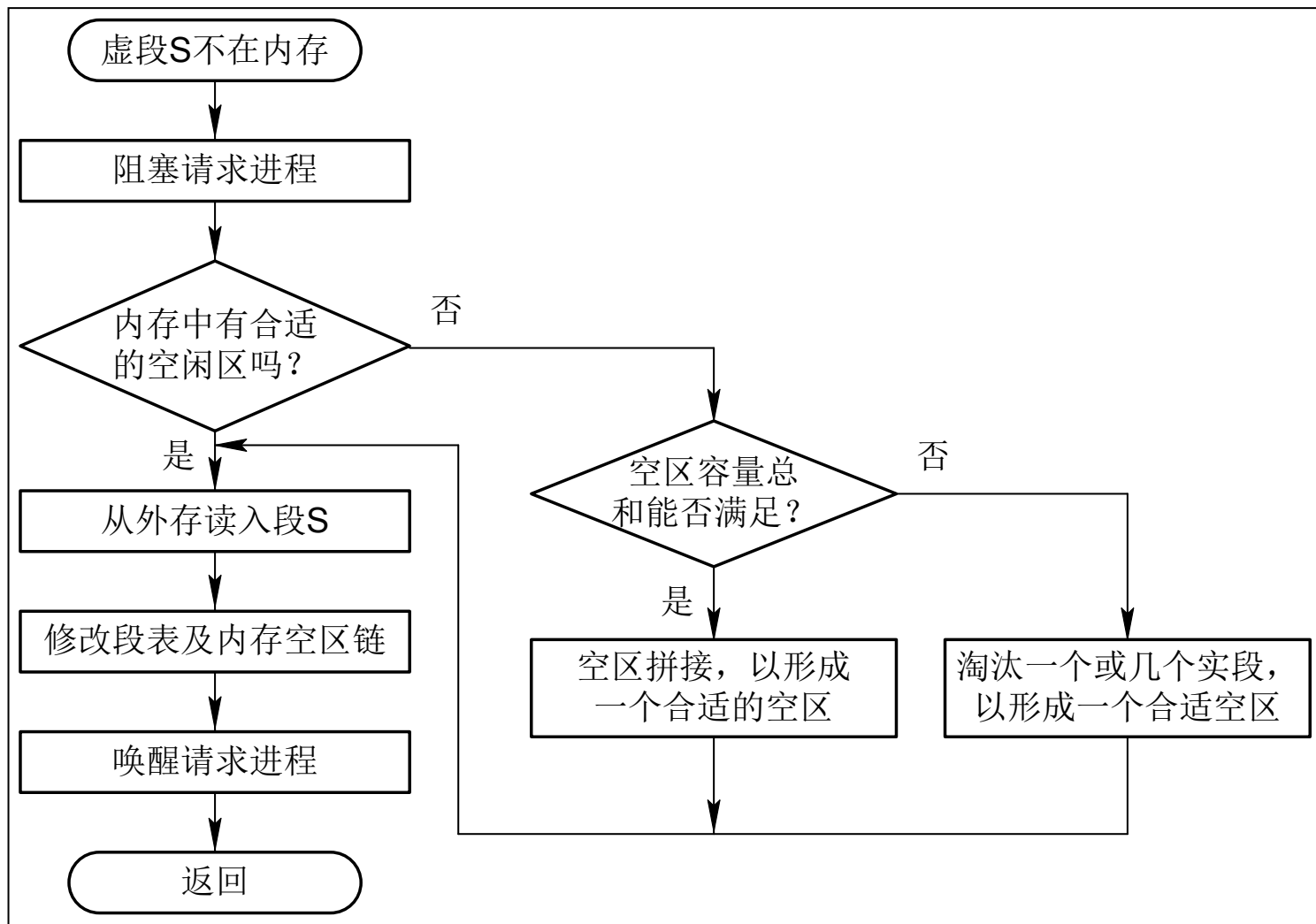
地址变换机构

- 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，可能产生多次缺段中断
- 由于段不是定长的，对缺段中断的处理要比对缺页中断的处理复杂

段表机制

缺段中断机构

地址变换机构



请求分段系统中的中断处理过程

段表机制

缺段中断机构

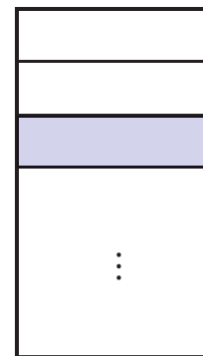
地址变换机构

- 请求分段系统的地址变换机构，是在分段系统的地址变换机构基础上形成的。
- 由于分段可能不在内存，因此会引起缺段中断。先将需要的段调入内存，修改段表，然后再利用段表进行地址变换。



共享段表：保存所有的共享段

- 共享进程计数count
- 存取控制字段
- 段号



共享段表

段名	段长	内存始址	状态位	外存始址
共享进程记数count				
状态	进程名	进程号	段号	存取控制
⋮	⋮	⋮	⋮	⋮



共享段的分配

- 对首次请求使用共享段的用户，分配内存，调入共享段，修改该进程段表相应项，再为共享段表增加一项， $count=1$
- 对其他使用共享段的用户，修改该进程段表相应项，再为共享段表增加一项， $count=count+1$



共享段的回收

- 撤销在该进程段表中共享段所对应的表项，并执行 $count=count-1$ 操作
- 若为0，回收该共享段的内存，并取消共享段表中对应的表项
- 若不为0，只取消调用者进程在共享段表中的有关记录



越界检查：

- 由地址变换机构来完成；
- 比较段号与段表长度；段内地址与段表长度。



存取控制检查：以段为基本单位进行。

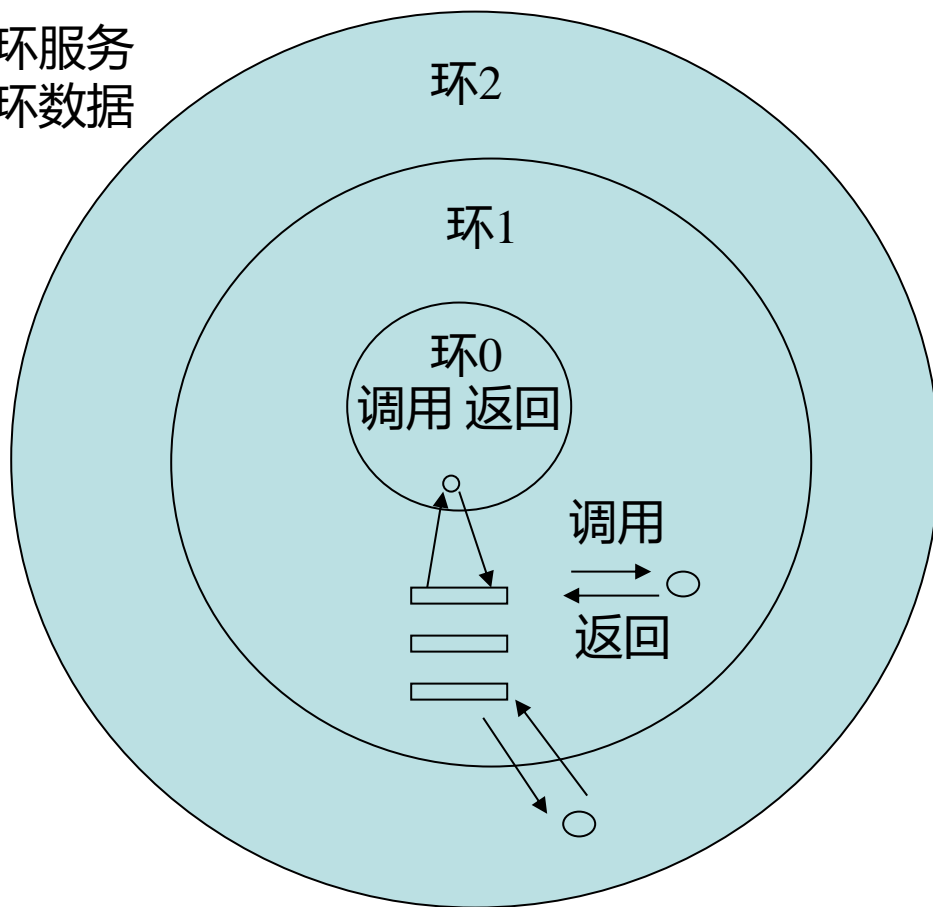
- 通过“存取控制”字段决定段的访问方式（只执行、只读或允许读/写）；
- 基于硬件实现。



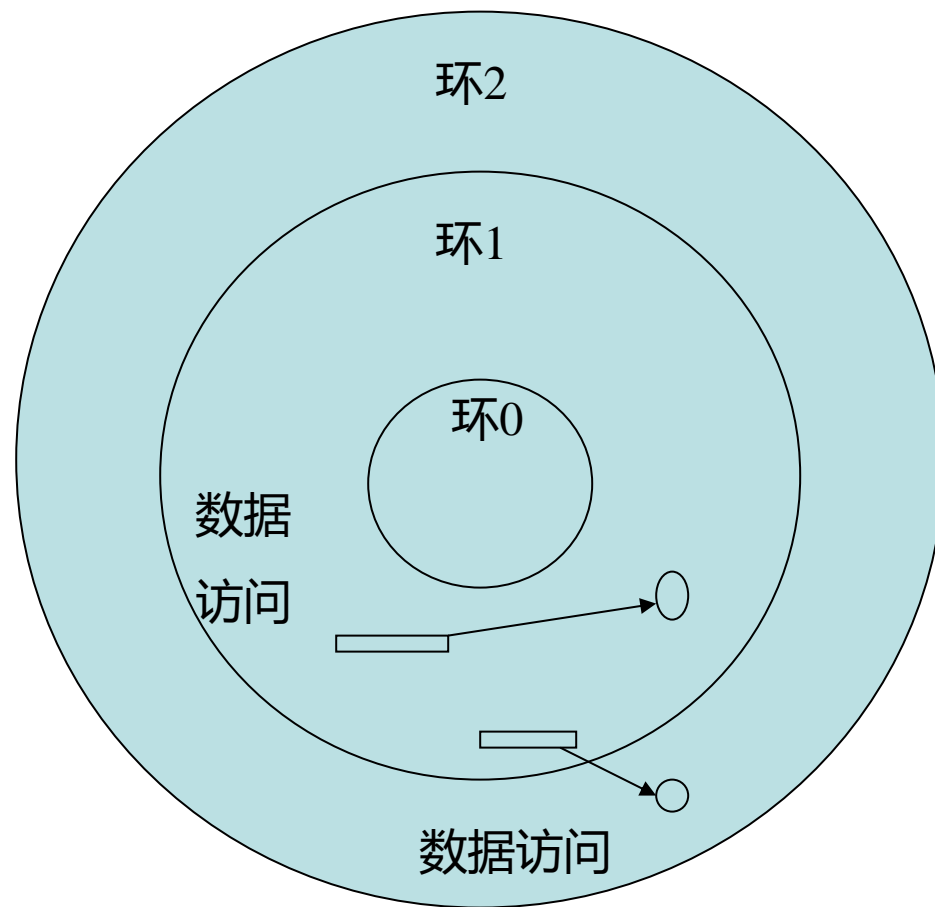
环保护机构：

- 低编号的环具有高优先权；
- 一个程序可以访问驻留在相同环或较低特权环（外环）中的数据；
- 一个程序可以调用驻留在相同环或较高特权环（内环）中的服务。

外环可请求内环服务
内环可访问外环数据









(a)程序间的控制传输



(b)数据访问



内容导航:

-  6.1 虚拟存储器概述
-  6.2 请求分页存储管理方式
-  6.3 页面置换算法
-  6.4 抖动与工作集
-  6.5 请求分段存储管理方式
-  **6.6 虚拟存储器实现实例**

第6章 虚拟存储器



采用请求页面调度以及簇来实现虚拟存储器



使用簇在处理缺页中断时，不但会调入不在内存中的页（出错页），还会调入出错页周围的页



创建进程时，系统会为其分配工作集的最小值和最大值

- 最小值：进程在内存中时所保证页面数的最小值
- 若内存足够，可分配更多的页面，直到达到最大值
- 通过维护空闲块链表（与一个阈值关联）来实现
- 采用局部置换方式



置换算法与处理器类型有关

- 如80x86系统，采用改进型Clock算法

实例2: Linux系统 (以32位为例)

OS 虚拟存储器是大小为4GB的线性虚拟空间。

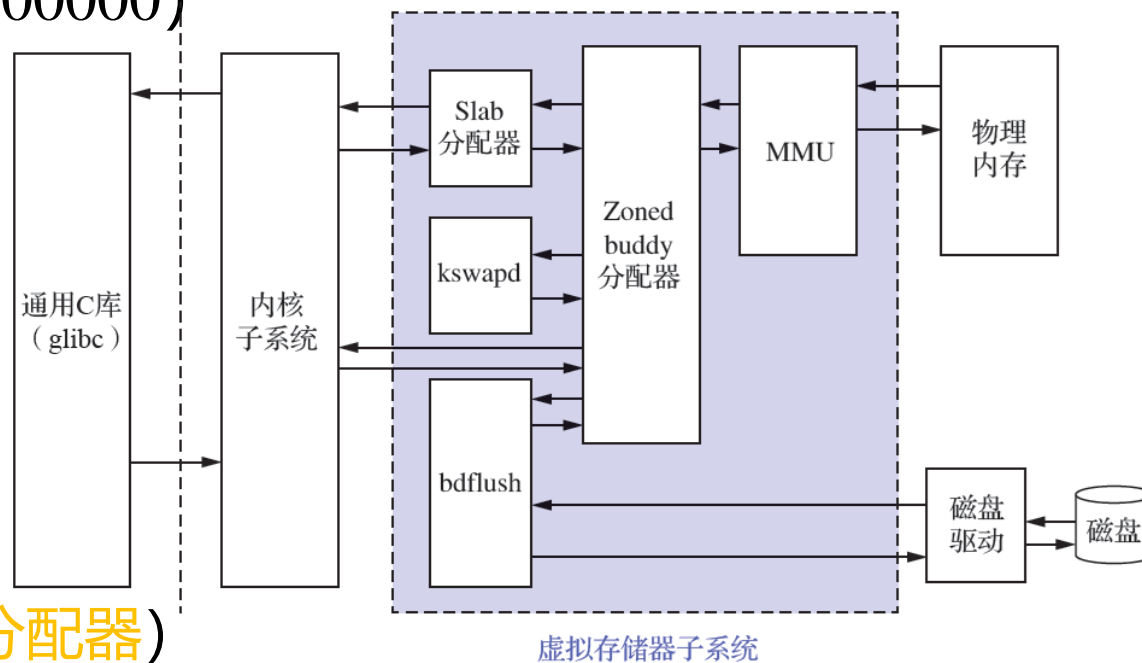
OS 4GB的地址空间分为两个部分:

□ 用户空间占据0~3GB (0xC0000000)

- 由用户进程使用 (MMU)
- 使用请求页式存储管理

□ 内核空间占据3GB~4GB

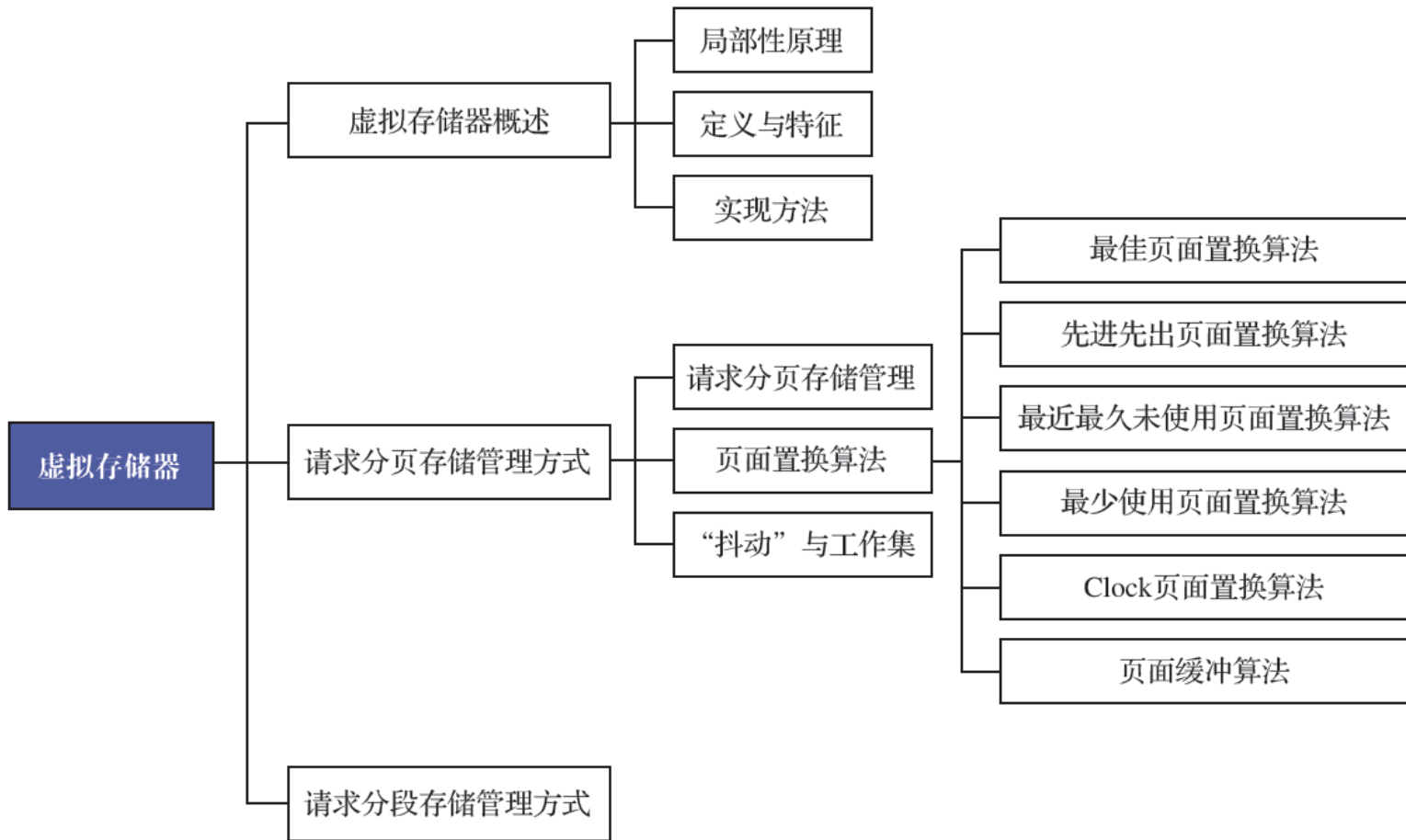
- 由内核负责
- 使用buddy和slab内存管理 (zoned buddy 分配器和slab分配器)





学而时习之（第6章总结）

第1章	操作系统引论
第2章	进程的描述与控制
第3章	处理机调度与死锁
第4章	进程同步
第5章	存储器管理
第6章	虚拟存储器
第7章	输入/输出系统
第8章	文件管理
第9章	磁盘存储器管理
第10章	多处理机操作系统
第11章	虚拟化和云计算
第12章	保护和安全





第九次作业

简答题

1

2

3

4

5

6

7

8

9

10

11

12

计算题

13

14

15

16

17

综合应用题

18

19

20

21

标黄色为本次作业

积极推进操作系统产品化

坚决构建基于国产操作系统的产业氛围

国内Linux系统的发展与国际上Linux系统的市场占有情况密切相关。

随着大数据与云计算等前沿技术的快速发展，越来越多的互联网公司开始构建自主控制与维护的云计算平台。具有开源与跨平台等属性的Linux系统，搭配采用Arm64芯片的计算平台，成为了这些互联网公司的首选技术方案。与此同时，Linux服务器端解决方案通过互联网企业迅速应用到了大数据与云计算的市场环境中。

但是，互联网企业使用Linux服务器端时，并未因采用Linux系统而形成典型的操作系统销售市场，专业的Linux系统厂商在服务器市场中还未形成较大的市场影响力。目前，国内海量的应用软件都是基于Windows系统的，因为该系统用户学习成本低、熟练程度高；而针对Linux系统，存在用户熟练程度低、对专业技术支持团队的依赖程度高、使用和维护成本高等问题。

积极推进操作系统产品化

坚决构建基于国产操作系统的产业氛围

为了更好、更快地解决上述问题，亟须建立顺畅的产品服务情况与用户使用预期的沟通渠道，通过了解并满足用户针对操作系统在使用、维护等方面的多种需求，提升国产（基于Linux系统进行二次开发的）操作系统的整体性能。同时，亟须确定一个兼具稳定性和一致性的开发接口，以使开发Linux系统应用软件的代码可以跨平台落地，进而减少因操作系统不同而导致的应用软件重复开发与测试工作，最终形成基于较为成熟的国产操作系统的产业氛围。

那么，应该如何建立产品服务情况与用户使用预期的沟通渠道？又应该如何确定兼具稳定性和一致性的开发接口呢？