

Lecture 7: 知识 - 虚拟内存管理

考试大纲

(二)虚拟内存管理

- 虚拟内存基本概念
- 请求分页管理方式
- 页框分配
- 页面置换算法:
 - 最佳置换算法 (OPT);
 - 先进先出置换算法 (FIFO);
 - 最近最少使用置换算法 (LRU);
 - 时钟置换算法 (CLOCK);
- 内存映射文件 (Memory-Mapped Files)
- 虚拟存储器性能的影响因素及改进方法

虚拟内存的基本概念

引入：传统存储管理方式的缺点

- 一次性：
 - 作业必须一次全部装入内存后才能开始运行;
 - 作业很大时，不能全部装入内存，导致大作业无法运行
 - 当大量作业要求运行时，由于内存无法容纳全部作业，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**;
- 解决 -> 虚拟存储技术;

概念：虚拟内存

- 定义
 - 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，这就是虚拟内存技术;
- 概念
 - 原理：
 - 基于局部性原理，在程序装入时，可以将**程序中很快用到的部分装入内存**，**暂时用不到的部分留在外存**，然后继续执行程序;
 - 请求调页：
 - 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;

- 主要包括三种方式：
 - 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
- 页面置换：
 - 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外存；
- 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，远大于实际容量；
- 注意：
 - 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 = $\min(\text{内存和外存容量之和}, \text{CPU 寻址范围})$
 - 虚拟内存需要页表机制、中断机制以及地址变换的支持；
- 真题：2012年
 - 虚拟内存只能基于非连续分配技术；
 - 当采用连续分配方式时，会使相当一部分内存空间都处于暂时或“永久”的空闲状态，造成内存资源的严重浪费，也无法从逻辑上扩大内存容量；
 - 在虚拟存储技术中，如分页和分段，内存不需要为程序分配一个连续的物理空间。这允许操作系统更灵活地管理内存碎片，因为新程序可以在任何可用的非连续空间加载，并按照规定需要将程序进行部分调入调出，所以虚拟存储技术只能基于非连续分配技术；
- 真题：2023年
 - 虚拟地址和 malloc 函数：
 - malloc 函数在 C 语言中用于分配内存，它返回的是虚拟地址（Virtual Address）；
 - 当调用 malloc 时，它会向操作系统或其内存分配器请求一块指定大小的内存区域，操作系统随后会从进程的虚拟地址空间中分配内存，并返回一个指向该区域的指针。这个指针是一个虚拟地址，操作系统将负责将这个虚拟地址转换为相应的物理地址；
 - 虚拟地址大小和内存、外存容量：
 - 物理内存以及外存的大小限制了操作系统可以映射到实际物理地址的虚拟地址的最大数量，即物理内存的大小决定了操作系统能够支持的虚拟地址空间的上限；
 - 但虚拟地址大小通常不由内存和外存容量限制，而是受到操作系统设计；
 - 比如 32 位系统通常限制虚拟地址空间为 4GB，而 64 位系统可以支持更大的地址空间（例如，16EB 或更多，具体取决于体系结构），这意味着它们可以支持更多的物理内存；

请求分页管理方式

页表机制

概念：页表机制

- 状态位：
 - 表示页面是否已经在内存中
- 访问字段：
 - 记录**最近被访问过几次**，或记录上次访问的时间，供置换算法换出页面时参考；
 - 可以把访问次数最少的换出内存；
- 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；
 - 只有“写指令”才需要修改“修改位”；
 - 一般来说只需修改快表中的数据，只有要将快表项删除时才需要写回内存中的慢表。这样可以减少访存次数；
- 外存地址：
 - 页面在**外存中存放的位置**；
- 图示：
 - 构成： 页号 + 内存块号 + 状态位 + 访问字段 + 修改位 + 外存地址

页号	内存块号	状态位	访问字段	修改位	外存地址
0	无	0	0	0	x
1	b	1	10	0	y
2	c	1	6	1	z

缺页中断以及处理

概念：缺页中断

- 引入：
 - 在基本分页系统地址变换机构的基础上，为实现虚拟内存，需要增加产生和处理缺页中断，以使 OS 可以去外存中访问；
- 概念：
 - **真题**：2019年
 - 缺页是在地址转换时、CPU 检测到的一种异常；
 - 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，请求操作系统的缺页中断处理程序处理，缺页处理程序根据**页故障地址**、从外存读入所缺失的页；
 - 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；
 - 若内存中有空闲页框，则为进程分配一个页框，将所缺页面从外存装入该页框，并修改页表中的相应表项；
 - 若内存中没有空闲页框，则由**页面置换**算法选择一个页面淘汰，若该页在内存期间被**修改过**，则还要将其**写回外存**。未被修改过的页面不用写回外存；
- 实现（无快表时）：
 - 1. 访问页面时，发现页表中对应页号的状态为 0，产生缺页中断；
 - 2. 调用缺页中断处理程序；
 - 3. 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；

- 4.1: 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**;
- 4.2: 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;
- 注意:
 - 缺页中断发生在执行当中, 因此属于内部异常, 不属于外部中断;
 - 一条指令在执行期间, 可能产生多次缺页中断;
 - 页面置换时, 未修改过的页面不用写回外存;
- 中断:
 - 属于 **内中断** -> **故障** 这一类型的中断;

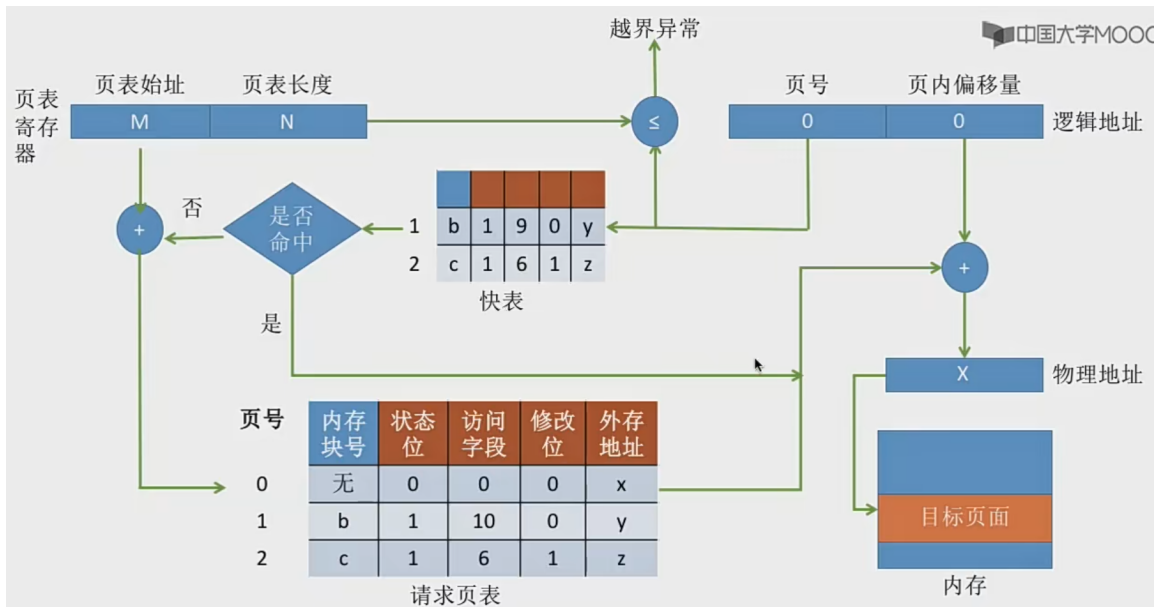
分析: 产生缺页中断时, 操作系统可能的操作

- 可能的操作:
 - **真题:** 2011年
 - 在缺页处理当中, **OS** 可能执行的操作包括 **修改页表**、**磁盘 IO**、以及 **分配页框**;
 - 因为需要将所需页面从外存调入页框中, 因此可能会涉及**磁盘 IO**;
- 不可能的操作:
 - **真题:** 2013年
 - 产生缺页时不可能产生 **越界错误**, 因为越界检查发生在查询页表之前, 而此处产生了缺页中断, 说明已经正常进行到查询页表;

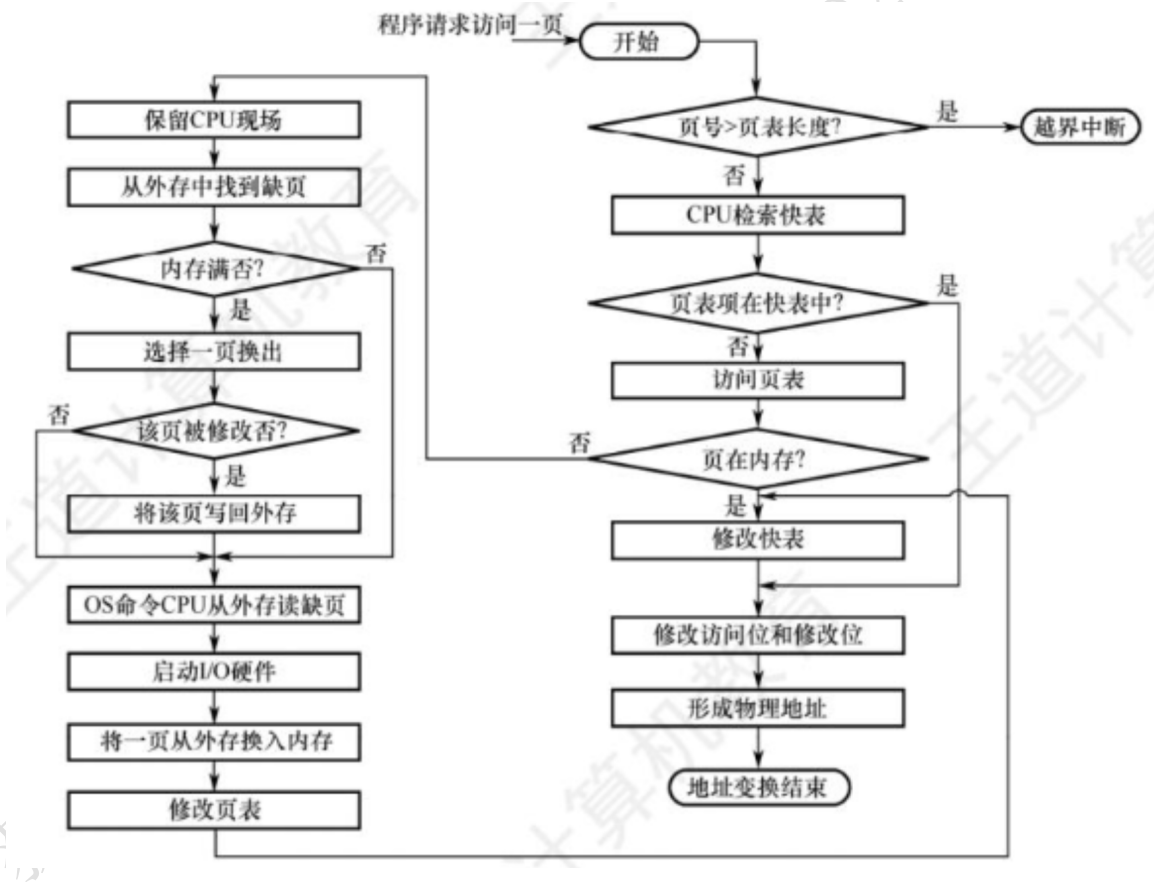
地址变换机构

概念: 地址变换机构

- 第一步 - 1: 查快表, 快表命中
 - 先检索快表, 若命中, 则从相应表项中取出该页的物理块号, 并修改页表项中的访问位, 以供置换算法换出页面时参考;
 - 对于写指令, 还需要将修改位置为 1;
- 第一步 - 2: 查快表, 快表未命中
 - 快表未命中, 则要到页表中查找;
 - 若找到, 则从相应表项中取出物理块号, 并将该页表项**更新入快表**, 若快表已满, 则需采用某种算法替换;
 - 若在页表中未找到, 则需要进行缺页中断处理, 请求系统将该页从外存换入内存, 页面被调入内存后, 由操作系统负责**更新页表和快表**, 并获得物理块号;
- 第二步: 形成物理地址
 - 利用得到的物理块号和页内地址拼接形成物理地址, 用该地址去访存
- 图示:
 - 过程:
 - 查快表 (未命中) -> 查慢表 (发现未调入内存) -> 调页 (调入的页面对应的表项会直接加入快表) -> 查快表 (命中) -> 访问目标内存单元;
 - 结构图



- 流程图：



页缓冲队列

概念： Linux 中的页缓存

- 概念：页缓存
 - 页高速缓存比缓冲区高速缓存简单得多，它是页 IO 操作访问数据所使用的磁盘高速缓存；
 - 在 Linux 中，`read()`、`write()`和 `mmap()` 系统调用对文件的访问都是通过页高速缓存来完成的；
- 概念：页缓存与页
 - 高速缓存中所保留的信息单元是一个整页，因为页 IO 操作每次都要传输一整页的数据；

- 一个页未必是包含物理上相邻的磁盘块，因此就不能使用设备号和块号来标识页。相反，页高速缓存中一个页的标识是通过文件的索引节点和文件中的偏移量达到的；
- 概念：页缓存相关操作
 - (1) 当访问的文件部分不在高速缓存中时，增加页；
 - (2) 当高速缓存变得太大时，删除一页；
 - (3) 查找一个给定文件偏移量所在的页。

页框分配

驻留集、抖动以及工作集

概念：驻留集

- 概念
 - 对于分页式的虚拟内存，在进程准备执时，不需要也不可能将一个进程的所有页都读入主存；
 - 因此，操作系统必须决定读取多少页，并决定给特定的进程分配几个页框；
 - 给一个进程 **分配的页框的集合**，就是这个进程的驻留集；
- 驻留集大小：
 - 驻留集大小需要虑以下两点：
 - (1) 驻留集越小，驻留在内存中的进程就越多，可以提高多道程序的并发度，但分配给每个进程的页框太少，会导致**缺页率较高**，CPU 需耗费大量时间来处理缺页；
 - (2) 驻留集越大，当分配给进程的页框超过某个数目时，再为进程增加页框对缺页率的改善是不明显的，反而只能是浪费内存空间，还会导致多道程序**并发度的下降**；

概念：抖动现象

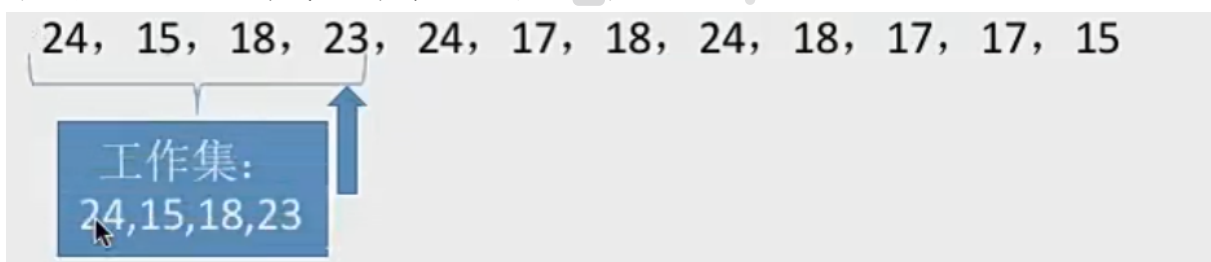
- 别名：
 - 颠簸现象；
- 概念：
 - 刚刚换出的页面马上就要换出内存，刚刚换入的页面马上又要换出外存；
 - 这种频繁的页面调度行为称为**抖动**或**颠簸**；
- 原因：
 - 进程频繁访问的页面数目高于可用的物理块数；
 - 即：**分配给进程的物理块不够**；
- 真题：2011年
 - 问题：
 - 为了减少抖动，可以采取的有效措施有哪些？
 - 解答：
 - 正确解答：撤销部分进程；
 - 错误解答：增加磁盘交换区容量、提高用户进程的优先级；
 - 分析：
 - 产生抖动的根本原因，是同时在系统中运行的进程太多，导致**分配给每个进程的物理块太少**，不能满足进程正常运行的基本要求，致使每个进程在运行时会频繁地出现缺

页；

- 在上述三个方法中，只有撤销部分进程可以腾出内存空间，为留下进程多分配一些物理块以解决抖动，因为撤销部分进程可以使得**当前进程的驻留集变大，进而减少抖动**；
- 增加交换区并没有直接增大进程的驻留集，交换区的主要作用是提升了页面读取的速度，而不是扩大进程的驻留集；

概念：工作集

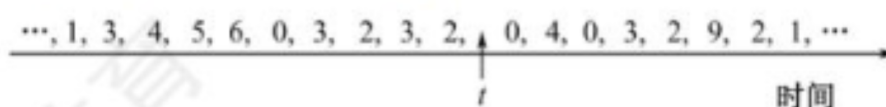
- 定义
 - 指在某段时间间隔里，进程**实际访问**页面的集合；
 - 因为是实际访问的页面集合，所以重复访问的部分需要去除掉；
- 内容
 - 工作集大小可能小于窗口尺寸，实际应用中，操作系统可以统计进程的工作集大小，根据工作集大小给进程分配若干内存块。
 - 一般来说，**驻留集大小应当大于工作集大小**，否则进程运行过程中将频繁缺页；
- 举例
 - 某进程的页面访问序列如下，窗口尺寸为 4；



- 真题：2016年

- 问题：

【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（ ）。

- 解答：

- 当前 t 时刻，最近访问的六个页面为 6, 0, 3, 2, 3, 2，去掉重复的页面后，剩下的是工作集；
- 因此答案为：6, 0, 3, 2

内存分配策略

概念：内存分配策略概述

- (1) 固定分配 + 局部置换
 - 为每个进程分配**固定数目的物理块**，在进程运行期间都不改变；
 - 所谓局部置换，是指若进程在运行中发生缺页，则**只能从分配给该进程在内存的页面中选出一页换出**，然后再调入一页，以保证分配给该进程的内存空间不变；

- 实现这种策略时，难以确定应为每个进程分配的物理块数目：太少会频繁出现缺页中断，太多又会降低 CPU 和其他资源的利用率；
- (2) 可变分配 + 全局置换
 - 先为每个进程分配一定数目的物理块，在进程运行期间可根据情况适当地增加或减少；
 - 所谓全局置换，是指若进程在运行中发生缺页，则系统从空闲物理块队列中取出一块分配给该进程，并将所缺页调入；
 - 这种方法比固定分配局部置换更加灵活，可以动态增加进程的物理块；
 - 但也存在弊端，如它会盲目地给进程增加物理块，从而导致系统多道程序的并发能力下降；
- (3) 可变分配 + 局部置换
 - 为每个进程分配一定数目的物理块，当某进程发生缺页时，只允许从该进程在内存的页面中选出一页换出，因此不会影响其他进程的运行；
 - 若进程在运行中频繁地发生缺页中断，则系统再为该进程分配若干物理块，直至该进程的缺页率趋于适当程度；
 - 反之，若进程在运行中的缺页率特别低，则可适当减少分配给该进程的物理块，但不能引起其缺页率的明显增加；
 - 这种方法在保证进程不会过多地调页的同时，也保持了系统的多道程序并发能力。当然它需要更复杂的实现，也需要更大的开销，但对比频繁地换入/换出所浪费的计算机资源，这种牺牲是值得的；
- 真题：2015年
 - 问题：
 - 分配策略和置换策略哪一种不同组合使用？
 - 解答：
 - 固定分配 + 全局置换 不能组合使用；
 - 固定分配时只有局部置换；
 - 分析：
 - 全局置换是进程在运行中发生缺页时，系统自动从空闲物理块队列中取出一块分配给该进程，并将所缺页调入，因此该过程会破坏固定分配的特性；

分析：请求分页系统的局部页面置换策略

- 题目：

【2012 统考真题】某请求分页系统的页面置换策略如下：从 0 时刻开始扫描，每隔 5 个时间单位扫描一轮驻留集（扫描时间忽略不计）且本轮未被访问过的页框将被系统回收，并放入空闲页框链尾，其中内容在下一次分配之前不清空。当发生缺页时，若该页曾被使用过且还在空闲页链表中，则重新放回进程的驻留集中；否则，从空闲页框链表头部取出一个页框。

忽略其他进程的影响和系统开销。初始时进程驻留集为空。目前系统空闲页的页框号依次为 32, 15, 21, 41。进程 P 依次访问的<虚拟页号, 访问时刻>为<1, 1>, <3, 2>, <0, 4>, <0, 6>, <1, 11>, <0, 13>, <2, 14>。请回答下列问题：

- 1) 当虚拟页为<0, 4>时，对应的页框号是什么？
- 2) 当虚拟页为<1, 11>时，对应的页框号是什么？说明理由。
- 3) 当虚拟页为<2, 14>时，对应的页框号是什么？说明理由。
- 4) 这种方法是否适合于时间局部性好的程序？说明理由。

• 分析：

- 本题需要根据题目描述的页面管理方式进行模拟；
- 在 0 时刻时，空闲链表状态：32 <- 15 <- 21 <- 41，
 - 32 为队头，第一个出队；
- 在 5 时刻前，内存中状态因为取出了 1 号虚页号、3 号虚页号、0 号虚页号，因此其对应的页框号如下：
 - 1->32、3->15、0->21；
 - 此时的空闲队列：41
- 5 时刻扫描，每一个页面都是新调入的，因此没有页面被调出；
- 在 5~10 时刻，内存中的 0 页面被访问，1、3 页面未被访问；
- 10 时刻扫描，因为 1、3 页面未被访问，被系统回收，1 号和 3 号页面被调出，加入空闲队列；
 - 此时内存状态：0->21；
 - 空闲队列：41<-32<-15 或 41<-15<-32
 - 此处题目没有说明同时回收两个分区时的先后顺序，所以有两种可能，但不影响解题；
- 在 10~15 时刻，1 号页面被访问，此时 1 号页面内容还在 32 号页框中，将 32 号恢复；
 - 内存状态：0->21、1->32；
 - 空闲链表：41<-15
 - 随后 0 号页面被访问，内存和空闲队列无调整；
- 随后 2 号页面被访问，2 号页面不在内存中，所以将空闲链表头部的 41 页框分配给 2 号页面；

• 解答

- (1) 根据以上分析，页框号为 21。
- (2) 根据以上分析，页框号为 32。
- (3) 根据以上分析，页框号为 41。
- (4) 合适，时间局部性好的程序被用过的页面会在不久后又被用到。该方法维护了一个空闲链表，调出的页面内容不会立刻被清空，只是被插入到该空闲队列。当某一页面被调

出后，很快又被访问。此时系统只需要将其从空闲队列中拿掉，页面的内容依旧在内存中，无需 IO；

固定分配策略

概念：物理块调入算法

- 采用固定分配策略时，将系统中的空闲物理块分配给各个进程，可采用下述几种算法。
- 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，根据进程的大小按比例分配物理块。
- 3. 优先权分配算法，为重要和紧迫的进程分配较多的物理块。通常采取的方法是将所有可分配的物理块分成两部分：一部分按比例分配给各个进程；一部分则根据优先权分配。

非固定分配策略

引入：非固定分配策略

- 主要分成三个部分
- 何时调页 -> 两种调用触发的时机选择策略；
- 从哪调页 -> 根据文件区和对换区大小，有不同的调用策略；
- 如何调用 -> 如何将文件从外存调入内存；

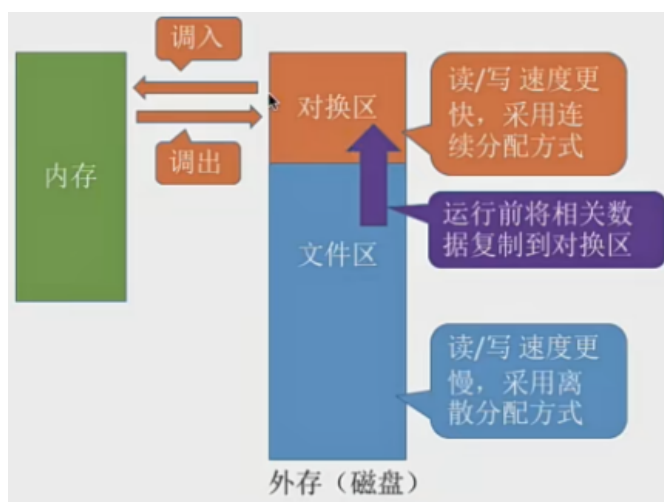
概念：何时调用 - 调入页面的时机

- 为确定系统将进程运行时所缺的页面调入的时机，可采用以下两种调页策略：
- 策略一：预调页策略
 - 根据局部性原理，一次调入若干个相邻的页面会比一次调入一页更高效；
 - 但若提前调入的页面中大多数都未被访问，则又是低效的。因此，可以预测不久之后可能被访问的页面，将它们预先调入内存，但目前预测成功率仅约 50%。因此这种策略主要用于进程的首次调入，由程序员指出应先调入哪些页；
- 策略二：请求调页策略
 - 进程在运行中需要访问的页面不在内存，便提出请求，由系统将其所需页面调入内存；
 - 由这种策略调入的页**一定会被访问**，且比较易于实现，因此目前的虚拟存储器大多采用此策略；
 - 其缺点是每次仅调入一页，增加了磁盘 IO 开销；
- 总结：
 - 预调页实际上就是运行前的调入，请求调页实际上就是运行期间的调入；

概念：从哪调用 - 从外存调入页面到内存

- 文件区和对换区：
 - 请求分页系统中的外存分为两部分：用于**存放文件的文件区**和用于**存放对换页面的对换区**，也称交换区；
 - **对换区采用连续分配方式**，而文件区采用离散分配方式，因此对换区的磁盘 IO 速度比文件区的更快；
 - 这样，当发生缺页请求时，系统调页就出现三种情况；

- 情况一：系统拥有足够的对换区空间；
 - 这时可以全部从对换区调入所需页面，以提高调页速度；
 - 为此，在进程运行前，需将与该进程有关的文件从**文件区复制到对换区**；
 - 图示：



- 情况二：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，由于它们不会被修改而不必再换出；
 - 但对于那些可能被修改的部分，在将它们换出时必须放在对换区，以后需要时再从对换区调入 (因为读比写的速度快)；
- 情况三：UNIX 方式
 - 与进程有关的文件都放在文件区，因此未运行过的页面都应从文件区调入；
 - **曾经运行过但又被换出的页面放在对换区**，因此在下次调入时应从对换区调入；
 - 进程请求的共享页面若被其他进程调入内存，则不需要再从对换区调入；

概念：如何调用 - 将数据从外存调入内存

- 当进程所访问的页面不在内存中时 (存在位为 0)，便向 CPU 发出缺页中断，中断响应后便转入缺页中断处理程序；
- 该程序通过查找页表得到该页的物理块，此时，若内存未满，则启动磁盘 IO，将所缺页调入内存，并修改页表；
- 若内存已满，则先按某种置换算法从内存中选出一页准备换出：
 - 若该页未被修改过 (修改位为 0)，则不需要将该页写回磁盘；
 - 但是，若该页已被修改 (修改位为 1)，则必须将该页写回磁盘，然后将所缺页调入内存，并修改页表中的相应表项，置其存在位为 1；
- 调入完成后，进程就可利用修改后的页表形成所要访问数据的内存地址；

页面置换算法

引入：页面置换算法

- 概念：
 - 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；
- 目标：

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

最佳置换算法

概念：最佳置换算法 (OPT, Optimal)

- 思想：
 - 最佳置换算法选择淘汰的页面是以后永不使用的页面，或是在最长时间不再被访问的页，以便保证获得最低的缺页率；
 - 然而，由于人们目前无法预测进程在内存的若干页面中哪个是面，未来最长时间不再被访问的，因此该算法无法实现；
 - 但可利用该算法去评价其他算法；
- 实现：
 - 按最佳置换的规则往后寻找，最后一个出现的页号就是要淘汰的页面；
- 举例：

访问页面	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
物理块 1	7	7	7	2		2		2			2			2				7		
物理块 2		0	0	0		0		4			0			0				0		
物理块 3			1	1		3		3			3			1				1		
缺页否	√	√	√	√		√		√			√			√				√		

先进先出置换算法

概念：先进先出置换算法 (FIFO)

- 思想
 - 每次选择淘汰的页面是**最早进入内存的页面**；
- 实现
 - 把调入内存的页面根据调入的先后顺序排除一个队列，需要换出页面时选择对头页面即可；
 - 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；
- 注意
 - 只有 FIFO 算法会产生 Belady 异常，因为 FIFO 策略并没有考虑到当前访问局部性；

分析：Belady 异常异常

- 概念：
 - 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；
- 问题：

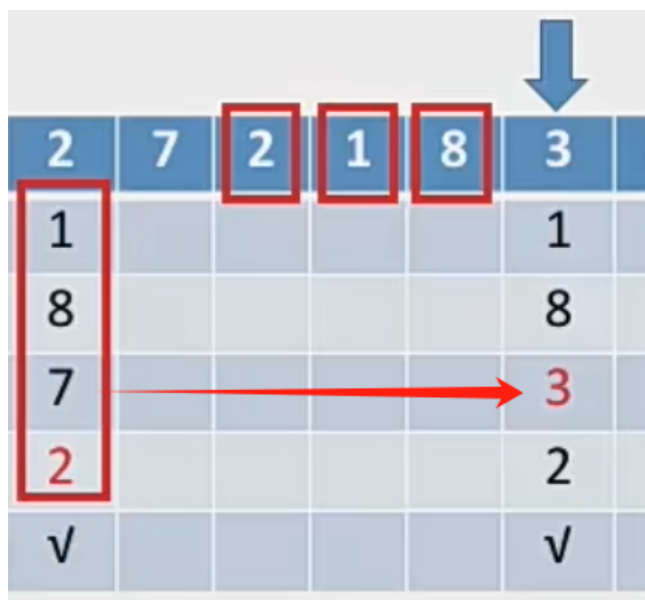
- 假设系统为某进程分配了四个内存块，并考虑到有以下页面号引用串：3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4
- 分析：
 - 分配四个内存块时缺页次数：10 次
 - 分配三个内存块时缺页次数：9 次
 - 当为**进程分配的物理块数增大时，缺页次数不减反增**的异常现象 -> Belady 异常；
- 图示：

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	√	√	√	√			√	√	√	√	√	√

最近最久未使用置换算法

概念：最近最久未使用置换算法 (LRU, least recently used)

- 别名：
 - LRU 算法（一定要记住这个别名，不要搞混）
- 思想
 - 每次淘汰的页面是**最近最久未使用的页面**；
- 实现
 - 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；
- 特点：
 - 优点：
 - 算法性能好，仅次于最佳置换算法；
 - 缺点：
 - 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；
- 方法：
 - 在手动做题时，若需要淘汰页面，可以逆向检查此时在内存中的几个页面号。在**逆向扫描过程中最后一个出现的页号**就是要淘汰的页面；
 - 图示：



• 图示：

页号	内存块号	状态位	访问字段	修改位	外存地址
----	------	-----	------	-----	------

例：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
1, 8, 1, 7, 8, 2, 7, 2, 1, 8, 3, 8, 2, 1, 3, 1, 7, 1, 3, 7

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

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						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	√	√		√	√						√						√			

在手动做题时，若需要淘汰页面，可以逆向检查此时在内存中的几个页面号。在逆向扫描过程中最后一个出现的页号就是要淘汰的页面。

• 真题：2015年

• 问题：

• 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，应淘汰页的页号是什么？

• 解答：

• 页号为 2；

2	0	2	9	3	4	2	8	2	4	8	4	5	7
2	2	2	2	2	2	2	2	2	2	2	2	2	
	0	0	0	0	4	4	4	4	4	4	4	4	X
			9	9	9	9	8	8	8	8	8	8	X
				3	3	3	3	3	7	7	7	5	X

• 第一行是当前存入的页号。通过从右往左看存入的页号，可以知道当前时刻哪个页最近最久未使用；

• 真题：2019年

• 问题：

• 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多少？

• 解答：

• 5 次；

0	1	2	7	0	5	3	5	0	2	7	6
0	0	0	0	0	0	0	0	0	0	0	
	1	1	1	1	5	5	5	5	5	5	
		2	2	2	2	3	3	3	3	7	
			7	7	7	7	7	7	2	2	
					✓	✓			✓	✓	✓

时钟置换算法

概念：时钟置换算法（NRU, Not Recently Used）

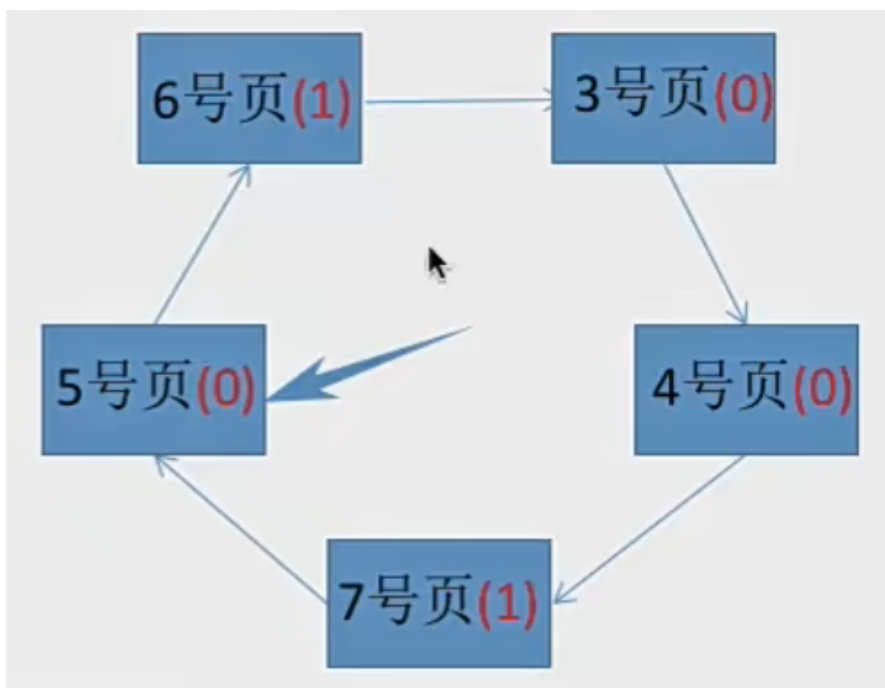
• 引入：

- LRU 算法的性能接近 OPT 算法，但实现起来的开销大。因此，操作系统的设计者尝试了很多算法，试图用比较小的开销接近 LRU 算法的性能，这类算法都是 CLOCK 算法的变体。
- 时钟置换算法又称 CLOCK 算法，或最近未用算法（NRU, Not Recently Used）

• 实现：

- 为每个页面设置一个访问位，再将内存中的页面都通过指针链接成一个循环队列；
- 当某页是目标时，成功访问，其访问位被置为 1；
- 当某页不是目标时，检查页的访问位：
 - 如果是 0，就选择该页换出；
 - 如果是 1，则将它置换为 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依次置换为 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；

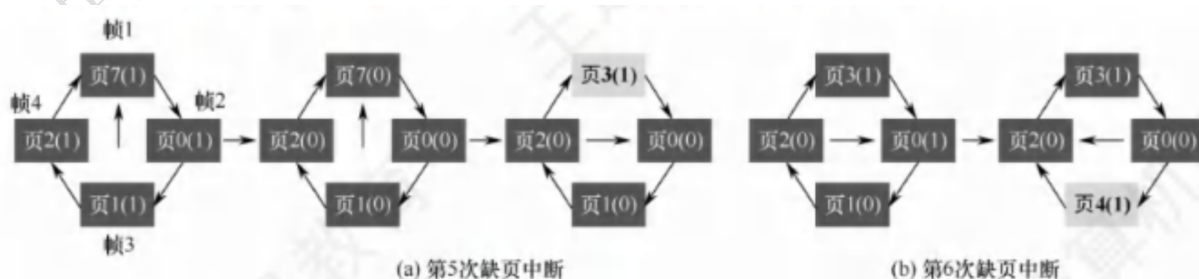
• 图示：



分析: CLOCK 算法举例

- 问题:
 - 假设页面访问顺序为 7,0,1,2,0,3,0,4,2,3,0,3,2,1,3,2, 采用简单 CLOCK 置换算法;
 - 分配 4 个页帧, 每个页对应的结构为 (页面号, 访问位), 求其置换过程;
- 分析:
 - 如果当前有目标, 则指针直接跳到目标位置, 将其访问位置于 1;
 - 如果当前没有目标, 则指针从 上次访问的位置 开始, 顺着链表往后逐个寻找 0;
 - 如果是 1, 置于 0;
 - 如果是 0, 将目标置换于此位置, 此位置置于 1;

访问顺序	7	0	1	2	0	3	0	4	2	3	0	3	2	1	3	2
帧1	7	1	7	1	7	1	7	1	3	1	3	1	3	1	3	1
帧2		0	1	0	1	0	1	0	0	0	0	0	0	0	0	0
帧3			1	1	1	1	1	1	0	1	0	4	1	4	1	4
帧4						2	1	2	1	2	0	2	0	2	1	2
缺页否	√	√	√	√		√		√						√		√



改进型的时钟置换算法

概念: 改进型的时钟置换算法

- 概念:

- 将一个页面换出时，若该页已被修改过，则要将该页写回磁盘，若该页未被修改过，则不必将它写回磁盘；
- 可见，对于修改过的页面，替换代价更大；
- 在改进型 **CLOCK** 算法中，除考虑页面使用情况外，还增加了置换代价：**修改位**；
- 在选择页面换出时，优先考虑**既未使用过又未修改过**的页面；
- 由访问位 A 和修改位 M 可以组合成下面四种类型的页面：；
- 四类页面：
 - 减少访问比修改更重要，因此 2 类的淘汰优先级高于 3 类；
 - 1 类 $A=0, M=0$ ：最近未被访问，且未被修改，是最佳的淘汰页；
 - 2 类 $A=0, M=1$ ：最近未被访问，但已被修改，是次佳的淘汰页；
 - 3 类 $A=1, M=0$ ：最近已被访问，但未被修改，可能再被访问；
 - 4 类 $A=1, M=1$ ：最近已被访问，且已被修改，可能再被访问；
- 过程分析：
 - 第一轮：
 - 从指针的当前位置开始，扫描循环队列，寻找 $A=0$ 且 $M=0$ 的 1 类页面，将遇到的第 1 类页面作为选中的淘汰页；
 - 在第一次扫描期间不改变访问位 A ；
 - 第二轮：
 - 若第一轮失败，则进行第二轮扫描，寻找 $A=0$ 且 $M=1$ 的 2 类页面，将遇到的第一个 2 类页面作为淘汰页；
 - 在第二轮扫描期间，将所有扫描过的页面的访问位都置 0；
 - 第三轮：
 - 若第二轮步也失败，则将指针返回到开始的位置，并将所有的访问位复 0；
 - 重复第一步，并且若有必要，重复第二轮，此时一定能找到被淘汰的页；
- 结论：
 - 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**；

总结：页面置算法

	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

- 注意：改进型 CLOCK 中，第三轮和第四轮实际淘汰的是第一轮开始时的 (1, 0) 和 (1, 1)

内存映射文件

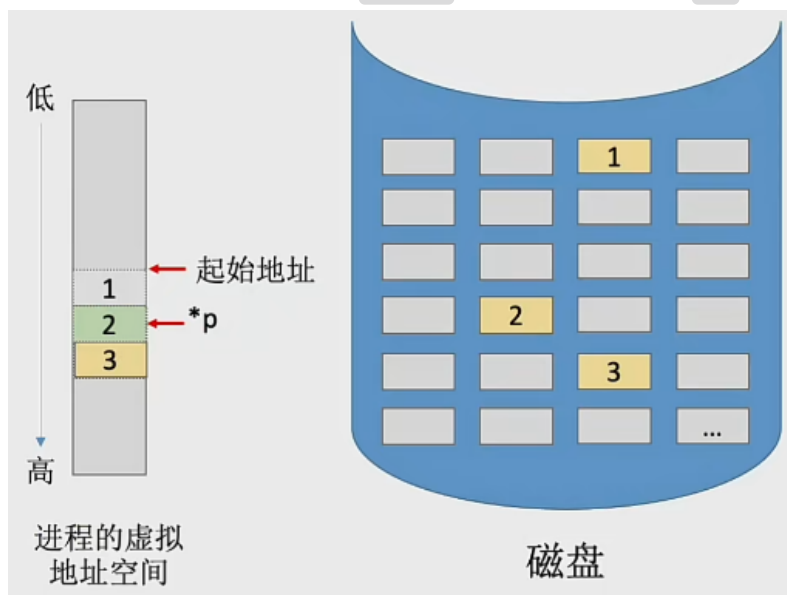
概念：内存映射文件

- 概念：
 - 内存映射文件 (Memory-Mapped Files) 是操作系统向应用程序提供的一个系统调用，在磁盘文件与进程的虚拟地址空间之间建立映射关系；
 - 进程通过该系统调用，将一个文件映射到其虚拟地址空间的某个区域，之后就用访问内存的方式读写文件；
 - 这种功能将一个文件当作内存中的一个大字符数组来访问，而不通过文件 IO 操作来访问，显然这更便利；
 - 磁盘文件的 IO 由操作系统负责完成，对进程而言是透明的。当映射进程的页面时，不会实际读入文件的内容，而只在访问页面时才被每次一页地读入。当进程退出或关闭文件映射时，所有被改动的页面被写回磁盘文件；
 - 进程可通过共享内存来通信，实际上，很多时候，共享内存是通过映射相同文件到通信进程的虚拟地址空间来实现的；
 - 当多个进程映射到同一个文件时，各进程的虚拟地址空间都是相互独立的，但操作系统将对应的这些虚拟地址空间映射到相同的物理内存 (用页表实现)；
 - 一个进程在共享内存上完成了写操作，此刻当另一个进程在映射到这个文件的虚拟地址空间上执行读操作时，就能立刻看到上一个进程写操作的结果。
- 传统方式：打开并修改文件
 - open 系统调用：打开文件
 - seek 系统调用：将读写指针移到某个位置
 - read 系统调用：从读写指针所指位置读入若干数据 (从磁盘读入内存)
 - write 系统调用：将内存中的指定数据，写回磁盘
- 内存映射文件：打开并修改文件

- `open` 系统调用：打开文件
- `mmap` 系统调用：将（外存中的）文件映射到进程的虚拟地址空间，返回起始地址
- 特点：
 - 1. 使程序员的编程更简单，已建立映射的文件，可以以访问内存的方式访问文件数据文件；
 - 2. 数据的读入、写出由操作系统自动完成，进程关闭文件时，操作系统自动将文件被修改的数据写回磁盘；
 - 3. 方便进行多个进程的文件共享；

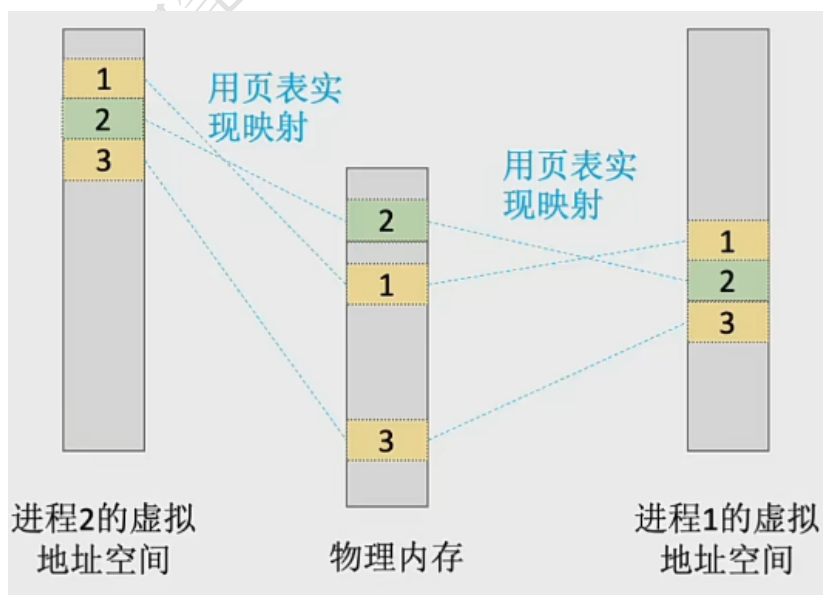
• 图示：自动分配与回收

- 当前访问指针在 2 位置时，OS 将自动将 2、3 位置的内容从磁盘读取进来，而不需要程序员自己调用 `read` 函数；
- 回收内存时，只需要使用 `close` 系统调用，即可 OS 自动将内存回收；



• 图示：实现文件共享

- 多个进程可以实现映射同一个文件，实现共享；
- 在物理内存中，一个文件对应同一份数据，当一个进程修改文件数据时，另一个进程可以立马知道；



补充: Linux 中的内存映射实现

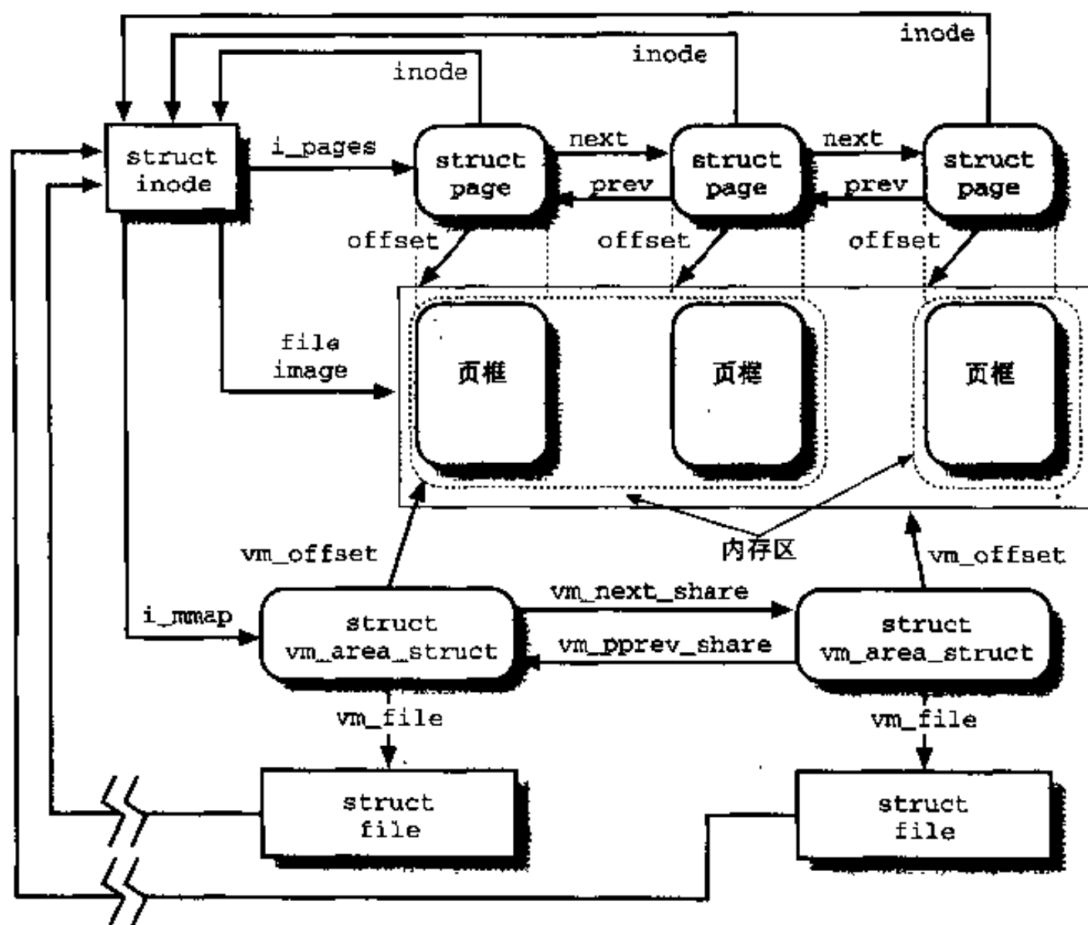
- 概念: 两种内存映射
 - 共享内存映射:
 - 对线性区中页的任何写操作都会修改磁盘上的文件;
 - 而且如果进程对共享内存映射中的一个页进行写, 那么这种修改对于其他映射了相同文件的所有进程来说都是可见的;
 - 如果一个内存映射是共享的, 相应的线性区就设置了 `VM_SHARED` 标志;
 - 私有内存映射:
 - 当进程创建的映射只是为**读文件**, 而不是写文件时才会使用;
 - 出于这种目的, **私有映射的效率要比共享映射的效率更高**。但是对私有映射页的任何写操作都会**使内核不再映射该文件中的页**;
 - 因此, 一个写操作既不会修改磁盘上的文件, 对访问相同文件的其他进程来说这种改变也是不可见的;
 - 如果一个内存映射是私有的, 那么相应的线性区就清除了 `VM_SHARED` 标志
- 概念: 内存映射的使用
 - 进程可以发出一个 `mmap()` 系统调用来创建一个新内存映射, 程序员必须指定一个 `MAP_SHARED` 标志或 `MAP_PRIVATE` 标志作为这个系统调用的参数;
 - 前一种情况下, 映射是共享的, 而后一种情况下, 映射是私有的;
 - 一旦创建了这种映射, 进程就可以从这个新线性区的内存单元读取数据, 也就等价于读取了文件中存放的数据;
 - 如果这个内存映射是共享的, 那么进程可以通过对相同的内存单元进行写而达到修改相应文件的目的。为了撤消或者缩小一个内存映射, 进程可以使用 `munmap()` 系统调用;

分析: Linux 下内存映射的实现

- 概念:
 - `struct inode`: 索引节点;
 - 每个索引节点对象的 `i_mmap` 域指向了双向链表的第一个元素, 比如图示的 `struct page` 和 `struct vm_area_struct`;
 - 索引节点对象的 `i_mmap` 指向双向链表的第一个元素, 该链表中包含了当前对该文件进行映射的所有线性区;
 - 如果 `i_mmap` 为 `NULL`, 那么就没有任何线性区映射该文件;
 - `struct page`
 - 表示一个页表, 其指向了一个页框, 存储 `offset` 表示其偏移量;
 - `struct vm_area_struct`
 - 它是使用 `vm_next_share` 和 `vm_prev_share` 实现的;
 - `vm_file`
 - 每个线性区描述符的 `vm_file` 域都包含被映射文件的文件对象的地址;
 - 如果该域为空, 那么在内存映射中就没有使用这个线性区;
 - 文件对象还包含几个域, 这些域允许内核既标识拥有内存映射的进程, 也标识被映射的文件;
 - `vm_offset`

- 第一个所映射的位置存放在线性区描述符的 `vm_offset` 域中；
- 被映射的文件部分的长度就是这个线性区的大小，该值可以从 `vm_start` 和 `vm_end` 域中计算出来；

• 图示：



虚拟存储器性能影响因素

概念：虚拟存储器性能影响因素

- 引入：
 - **缺页率**是影响虚拟存储器性能的主要因素，而缺页率又受到以下因素的影响：
 - (1) 页面大小； (2) 分配给进程的物理块大小； (3) 页面置换算法； (4) 写回磁盘的策略； (5) 程序的编制方法；
- 页面大小：
 - 根据局部性原理，页面较大则缺页率较低，页面较小则缺页率较高；
 - 页面较小时，一方面减少了内存碎片，有利于提高内存利用率。另一方面，也会使每个进程要求较多的页面，导致页表过长，占用大量内存；
 - 页面较大时，虽然可以减少页表长度，但会使页内碎片增大；
- 物理块大小：
 - 分配给进程的物理块数越多，缺页率就越低；
 - 但是当物理块超过某个数目时，再为进程增加一个物理块对缺页率的改善是不明显的。可见，此时已没有必要再为它分配更多的物理块，否则也只能是浪费内存空间；
 - 只要保证活跃页面在内存中，保持缺页率在一个很低的范围即可；

- 页面置换算法：
 - 好的页面置换算法可使进程在运行过程中具有较低的缺页率；选择 LRU、CLOCK 等置换算法，将未来有可能访问的页面尽量保留在内存中，从而提高页面的访问速度；
- 写回磁盘的策略：
 - 换出已修改过的页面时，应当写回磁盘，如果每当一个页面被换出就将一次磁盘，效率极低；
 - 为此在系统中建立一个已修改换出页面的链表，对每个要被换出的页面（已修改），可以暂不将它们写回磁盘，而将它们挂在该链表上，仅当被换出页面数达到给定值时，才将它们一起写回磁盘，这样就可显著减少磁盘 IO 的次数，即减少已修改页面换出的开销；
 - 此外，若有进程在这批数据还未写回磁盘时需要再次访问这些页面，则不需从外存调入，而直接从已修改换出页面链表上获取，这样也可以减少页面从磁盘读入内存的频率，减少页面换进的开销；
- 程序编址方法：
 - 编写程序的局部化程度越高，执行时的缺页率就越低。若存储采用的是按行存储，则访问时就要尽量采用相同的访问方式，避免按列访问造成缺页率过高的现象；

分析：请求分页系统有效平均访存时间的因素

- **真题：**2020年
- 缺页率；
 - 缺页率高会导致系统访问时花费大量时间在处理缺页中断上，系统访问时可能产生缺页中断，所以执行缺页处理程序的 CPU 时间长，自然会导致整体的访问时间上升；
- 磁盘读写时间；
 - 磁盘处理时间属于是缺页中断的一部分，磁盘处理时间长，会增加缺页中断处理的时间，进而影响分页系统的平均访问时间；
- 内存访问时间；
 - 主要有两个部分设计内存访问：（1）查询页表；（2）查目标物理地址；
 - 这两个过程会影响虚实地址转换的速度，进而影响平均的访存时间；
- 执行缺页处理程序的 CPU 时间；
 - 同理，平均访存时间包括了处理缺页中断的时间，因此会增加平均访存时间；

分析：能够加快虚实转换速度的措施

- **真题：**2014年
- 能够加快的措施：增大快表容量、让页表常驻内存
 - 增大快表容量可以提高快表命中率，命中率高可以节省很多不命中所耗费的时间，因此可以提高虚实转换速度；
 - 让页表常驻内存，使得查询页表时可直接从内存中查询，有效提高了查询页表的速度，因此提高了获得物理基址的速度，因此可以加快虚实转换的速度；
- 不能够加快的措施：增加交换区容量
 - 虚实地址转换的目的是生成读取页面所需的地址，因此此过程发生在读取页面之前；
 - 而增大交换区的主要作用，在于提高了读取页面的速度，而这和虚实地址转换无关；