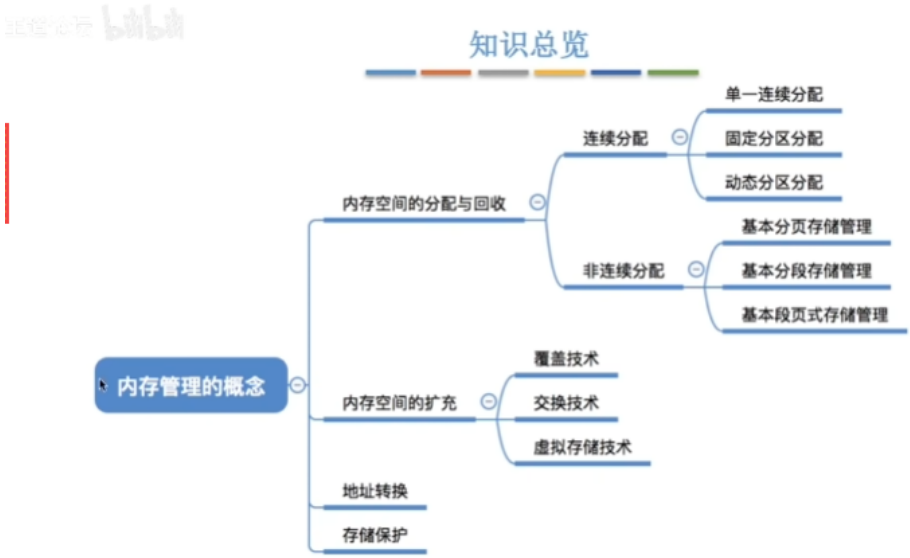
3.2_1_虚拟内存的概念

- gxy总结：
  这一部分，现在脑子不是很清醒，之后再总结一遍吧。
  挺重要的，要明白基本的原理、内容、作用。



- 虚拟内存 也是一种能够进行内存扩充的技术。
  内存扩充：覆盖技术、交换技术、虚拟存储技术。

  虚拟内存是基于 高速缓存 的思想提出来的。

-


  传统存储管理方式的特征、缺点：
  一次性：作业必须一次性全部装入内存之后才可以开始运行。
  驻留性：一旦作业被装入内存，就会一直驻留在内存中，知道作业运行结束。

  缺点：
  一次性：导致，很大的作业如果装不下就没有办法运行。当大量程序运行的时候，只有少部分的作业能装下可以运行，多道程序并发度下降。
  驻留性：举一个例子，一个游戏，在场景A中，不需要场景B的一些数据，但是会把场

景B的数据也放进去。事实上，一个时间段内，往往只需要访问一小部分的数据就可以正常运行，就导致 内存中会驻留大量的、暂时用不到的数据，浪费内存资源。

- 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）

  基于局部性原理，提出来了 高速缓存 技术：
  使用频繁的数据会放到更加高速的存储器中。

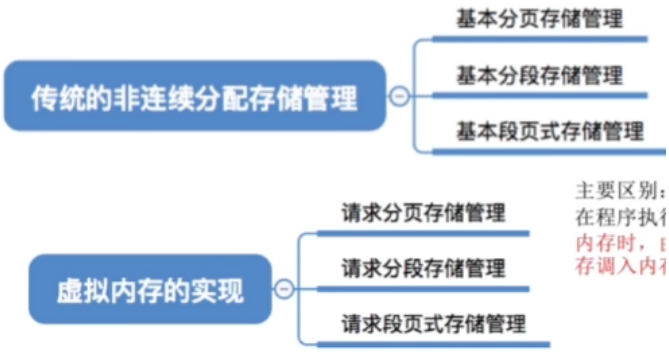
- 使用虚拟内存的特征：
  1.基于局部性原理，在程序装入的时候，可以把程序中很快就会用到的部分装入内存，而把暂时用不到的部分留在外存，就可以让程序开始执行。
  2.在程序运行过程中，如果需要访问的信息不在内存，就由**OS**负责将所需要的信息从外存调入内存，继续执行程序。
  3.如果内存空间不够，由**OS**负责把内存中暂时用不到的信息调到外存。
  4.在OS的上述管理下，在用户的角度，就会得到一个看似比实际内存大得多的内存，就是 虚拟内存 。

  虚拟性：内存其实并没有在物理层面扩大，但是在逻辑层面扩大了，在逻辑上进行了扩充，就是虚拟性。

  虚拟内存的几个主要特征：
  1.多次性：无需作业运行的时候一次性全部装入内存，允许被分成多次调入内存。
  2.对换性：在作业运行的时候不需要一直常驻内存，允许作业在运行的过程中，将作业进行换入、换出。
  3.虚拟性：在逻辑上扩充了内存的容量，使得用户看到的内存容量远大于实际的容量。

  多次性 对应上面的 一次性，对换性对应上面的 驻留性 。

- 如何实现虚拟内存技术：
  如果使用 连续性内存分配技术 ，一个作业多次调入内存，但是多次调入之后还是连续的，明显不合理、很难做到，所以 虚拟内存技术 基于 离散分配的 内存管理技术。



实现虚拟内存技术的三种方式和非连续性内存管理的三种方法是对应的。

主要区别：
1.在程序执行的过程中，当访问的信息不在内存，需要os将需要的信息调入内存，之后继续执行。
2.如果内存空间不够，由OS负责将内存中暂时用不到的信息换出外存。

所以OS要在原来的基础上新增一些功能：请求调页，**OS**要提供页面置换功能。
请求调页：将信息从外存调入内存。
页面置换：将内存中暂时用不到的信息换出到外存。通过置换把暂时用不到的分页、分段换到外存中。

如果是分段存储管理方式就需要新增请求调段、提供段置换功能。

3.2_2_请求分页管理方式

- 首先，因为请求分页管理和基本分页管理不一样，所以会多很多 需要存储的属性，明白属性的作用，然后明白 请求分页和基本分页的区别。

  掌握请求分页管理方式的 地址变换过程。
  掌握其实就是掌握和基本分页管理方式的不同。
  能够分清楚，缺页中断是内中断。

  这一节课的内容，重点在于理解，但是需要能说出来，感觉和会背也没什么区别。

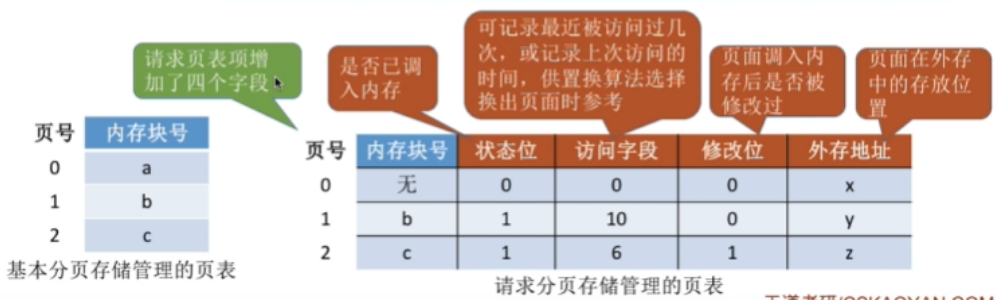
- 请求分页管理方式在基本分页存储管理方式上添加了请求调页(如果访问的数据不在内存中，要从外存调进来)和页面置换(如果内存满了，就要在内存中找一个页面进行置换)两种功能。

- 页表机制：

  需求：在`请求分页管理方式中`，OS需要知道这个页面是不是在内存里面，还需要知道在外存中的位置，在`页面置换`中，可能是根据一些指标来暗指优先级，而这样的指标也需要存储下来，需要存储一个页面有没有被修改过，因为一个没有被修改过的页面在最后调入外存的时候不需要修改，但是一个已经修改过的调入外存的时候，是需要覆盖原先的外村的旧数据的。

  所以在`请求分页存储管理方式`中需要添加：

  状态位(是否调入内存)，访问字段(记录一些指标)，修改位(数据是否修改过)，外存地址(在外存的存放位置)。



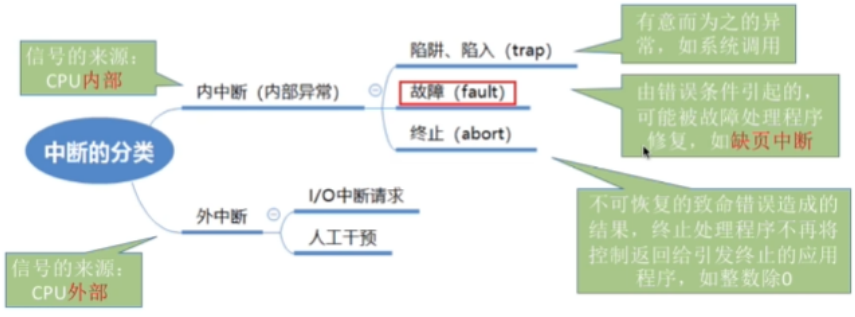
- 缺页中断机构：

  在地址转换中，假设当前想要访问页面0，先检查`0号页面`是否在内存中，如果不在需要产生`缺页中断`，然后让OS的`缺页中断处理程序`来处理这个中断。中断发生后，当前进程阻塞，放入阻塞队列，调页完成之后再被唤醒，放回就绪队列。

  在`请求调页`之后，先检查内存中有没有空余页框、内存块，如果有直接放进去；如果没有就通过`页面置换算法`先将内存中的一个页面淘汰，如果这个页面被修改过，就需要写回外存。把请求调入的页面覆盖这个页面。

  缺页中断的特性：

  因为当前执行的指令先要访问的目标不在内存中，而引起中断，所以是`内中断`。

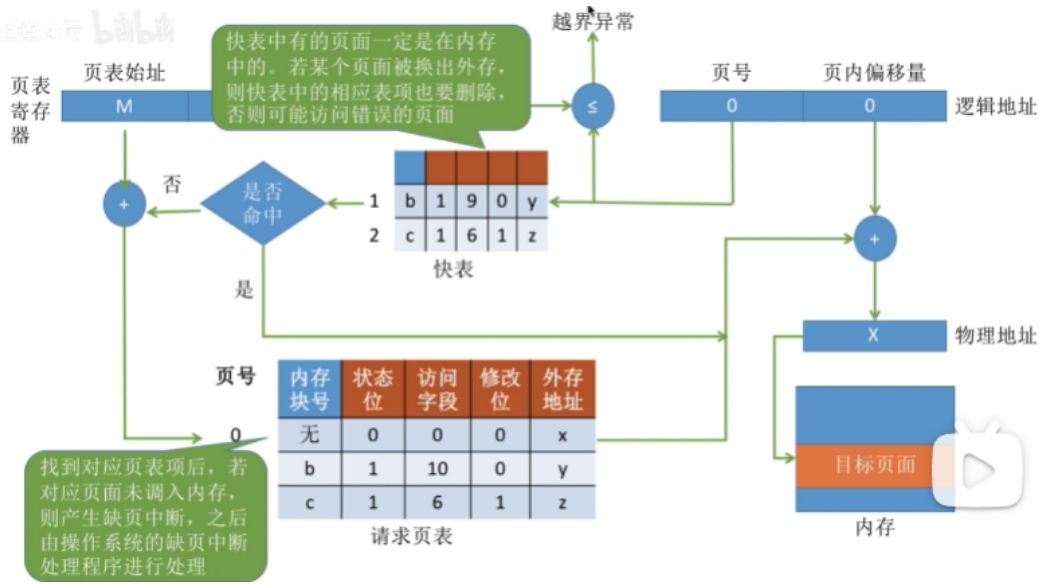


  一个指令如果要访问不止一次地址，就可能会引发不止一次`缺页中断`。

- 请求分页和基本分页的主要区别：
  1.请求分页管理方式中需要请求调页，需要检查当前的内存中有无目标页面。
  2.需要进行页面置换，（如果需要调入，并且没有空闲的内存块的时候）。
  3.需要修改 请求页表 中新增的表项，在页面调入、调出、访问的时候，有关的页表项需要进行修改。

- 在访问地址中的主要区别：
  得到访问的目标页面的页号之后，会首先访问快表，如果找到就直接找到。如果没有找到就去慢表里面寻找，如果慢表里面没有，就 缺页中断 ，请求调入，可能页面置换。（如果被置换的页面在快表中存储了，就需要把快表中它的相关信息删除掉，因此就说明了：快表中的页面号对应的页面一定在内存中）。

  在请求分页管理方式的快表中也会比 基本分页管理方式 的快表多一些 表项 。



- 请求分页的地址变换过程和一些补充的细节：

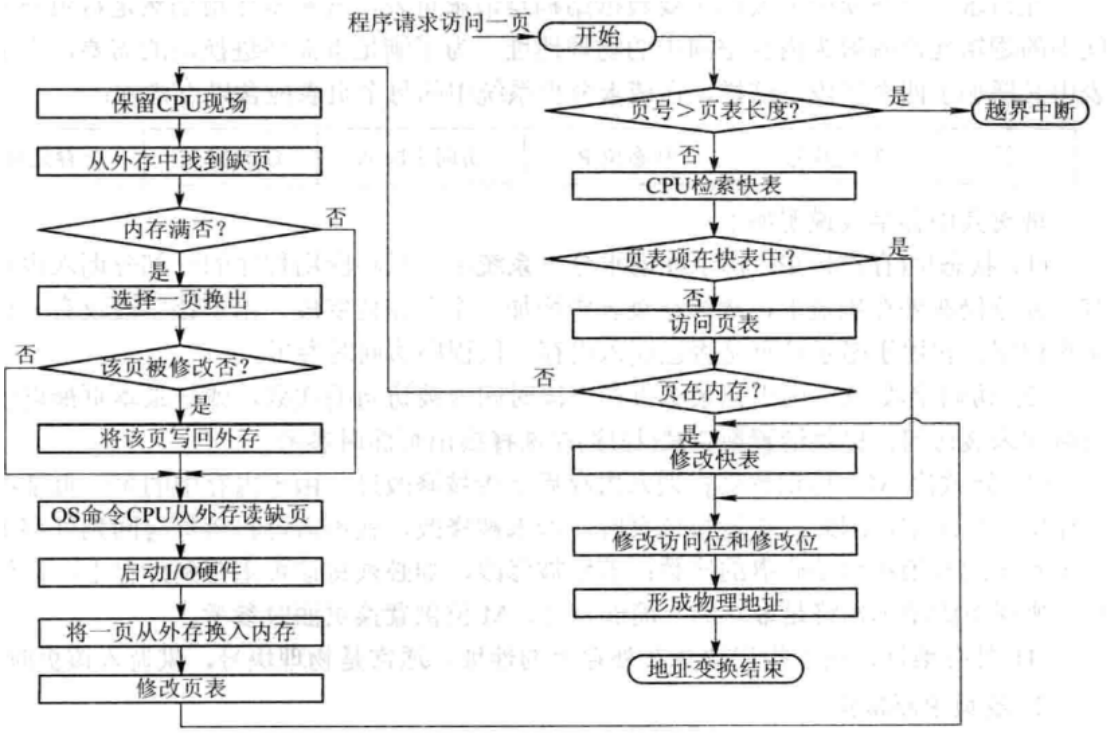


图 5-2　请求分页中的地址变换过程

补充细节：
①只有“写指令”才需要修改“修改位”。并且，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。

②和普通的中断处理一样，缺页中断处理依然需要保留CPU现场。

③需要用某种“页面置换算法”来决定一个换出页面（下节内容）

④换入/换出页面都需要启动慢速的I/O操作，可见，如果换入/换出太频繁，会有很大的开销。

⑤页面调入内存后，需要修改慢表，同时也需要将表项复制到快表中。

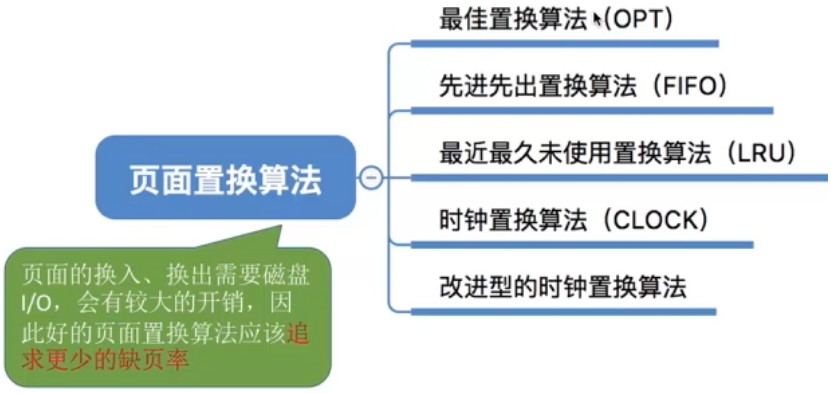
在内存调入之后，也会写到快表里面，之后会访问快表，再往下运行。

在具有快表机构的请求分页系统中，访问一个逻辑地址时，若发生缺页，则地址变换步骤是：
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

3.2_3_页面置换算法

- gxy总结：
  改进型号的时钟置换算法，王道中的视频好像有问题，找了一个博客的笔记。
  理解一下clock,改进的clock。分清楚FIFO,lru,opt。各自对应什么情况。

	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



了解每一种算法的思想，还要记住，英文缩写。

- 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最求更好的缺页率。

- OPT：每次选择淘汰的页面是 最长时间内不会被访问到的页面 。

例：假设系统为某进程分配了三个内存块，并考虑到有一下页面号引用串（会依次访问这些页面）：
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

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	√	√	√	√		√		√			√			√				√		

选择从 0，1，7 中淘汰一页。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

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%

缺点：只是理想上的算法，因为实际上无法很好的预判之后所有的页面访问序列，所以是无法实现的。

- FIFO：先进先出算法：每次淘汰的页面是最早进入内存的页面。

  用普通队列，就可以实现。

先进先出置换算法（FIFO）：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。
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：
3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4

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

缺点，会产生Belady异常，内存块明明变多了，但是缺页次数还会增多。
只有**FIFO**会发生Belady异常。
优点缺点：算法实现比较简单，但是与进程规律可能不适应，算法性能差。

- LRU：最近最早未被使用过。每次淘汰的是最近最久未被使用的页面。

  需要借助priority_queue才能实现。

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。

页号	内存块号	状态位	访问字段	修改位	外存地址

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

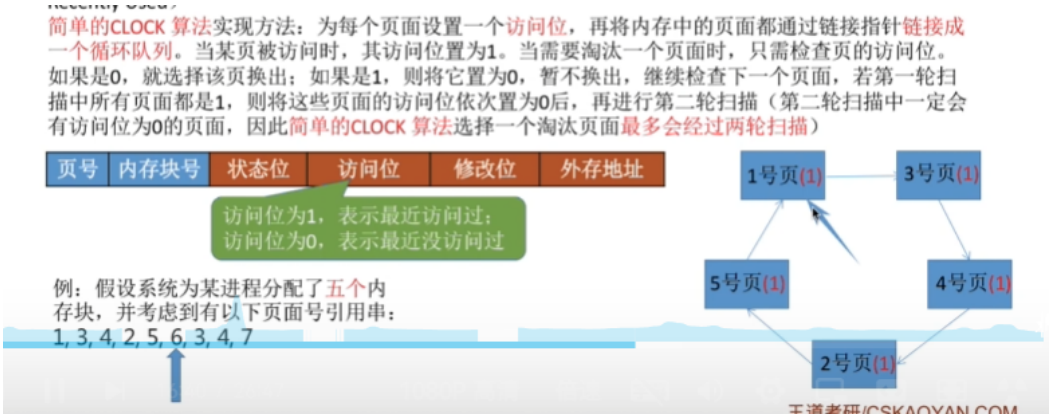
例：假设系统为某进程分配了四个内存块，并考虑到有以下页面号引用串
1, 8, 1, 7, 8, 2, 7, 2, 1, 8, 3, 8, 2, 1, 3, 1, 7, 1, 3, 7

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	√	√		√		√					√						√			

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。

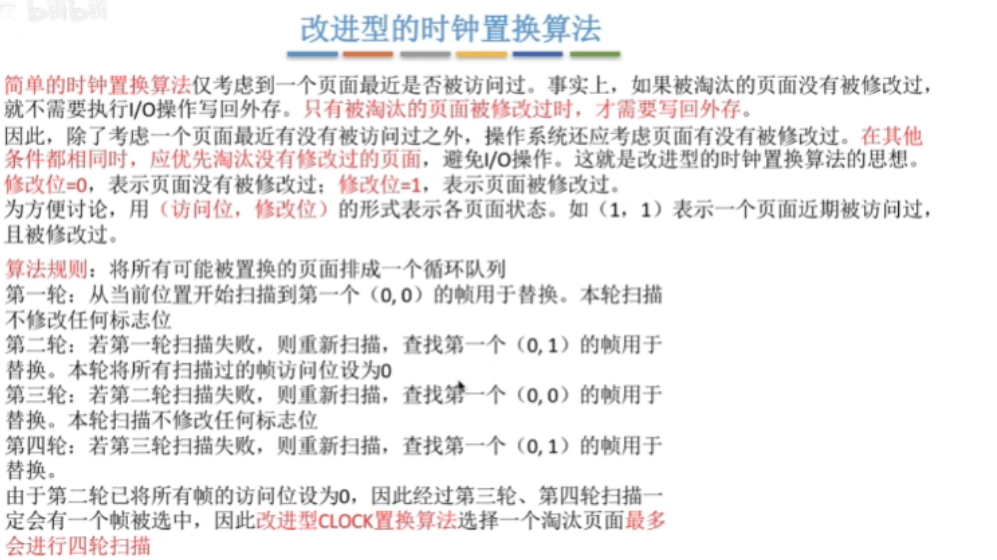
算法性能很好，但是实现困难，开销比较大。

- 时钟置换算法：CLOCK。（最近未用算法）
  实现：为每一个页面设置一个访问位，将内存中的页面通过 链接指针 链接成一个循环队列。当某页被访问的时候，访问位置为1，需要淘汰一个页面的时候，只需要检查页的访问位。如果是0，就该页换出，如果是1，就置为0，继续往下检查。



如果上面过程不是很理解，可以看王道视频 `3_2_3 16：00`

- 改进型的时钟置换算法：
  如果一个页面没有被修改过，如果它被淘汰，不需要执行I/O操作，放到外存，所以我们改进上面的CLOCK，让没有被修改过的页面的优先级最高，如果存在没有被修改过的页面，一定会让没有被修改过的优先被淘汰。



视频里面，也会有例子展示。

> 第一轮:从当前位置开始扫描到第一个(A =0, M = 0)的帧用于替换。表示该页面最近既未被访问，又未被修改，是最佳淘汰页
> 第二轮:若第一轮扫描失败，则重新扫描，查找第一个(A =1, M = 0)的帧用于替换。本轮将所有扫描过的帧访问位设为0。表示该页面最近未被访问，但已被修改，并不是很好的淘汰页。
> 第三轮:若第二轮扫描失败，则重新扫描，查找第一个(A =0, M = 1)的帧用于替换。本轮扫描不修改任何标志位。表示该页面最近已被访问，但未被修改，该页有可能再被访问。
> 第四轮:若第三轮扫描失败，则重新扫描，查找第一个A =1, M = 1)的帧用于替换。表示该页最近已被访问且被修改，该页可能再被访问。

总觉得上面的笔记有一些问题，还是用这个吧。

描述：
首先，找0,0.如果没有。就找1,0.第一个的意思是最近有没有用被访问过，找不到的话，就把(1,1) ->(0,1)。前面操作完了之后，第一维度不可能有1，之后再找一下(0,1)。
前面两轮一定把没有修改过的找到了，如果还继续说明剩下的页面都是修改过的，所以第二维度直接目标变为1即可，然后把前两轮的过程再重复一遍。

3.2_4页面分配策略、抖动、工作集

- gxy总结：
  理解一下驻留集，三种可能存在的页面置换和驻留集分配策略的组合。掌握三种的思路。
  对于何时调入页面，读一遍即可，面试应该有不会问。
  不存在固定分配全局置换，因为全局置换的物理块要求是可以增减的，但是固定分配是不能增减的

  工作集的概念，一般不会考察，不过比较简单，还是简单学习一下吧。

- 什么是驻留集
  驻留集：请求分页存储管理中给进程分配的物理块的集合。物理块也叫页框。

  在虚拟存储技术里面，驻留集大小一般小于进程的总大小。
  可能出现的情况：
  如果驻留集太小，会导致缺页频繁，系统需要花大量的时间来处理缺页。
  如果驻留集太大，会导致多道程序并发度下降，资源利用率降低.
  所以应该选择一个合适的驻留集大小。

- 驻留集的分配：
  固定分配：OS为每一个进程分配一组固定数目的物理块，在进程运行期间不再改变。驻留集的大小在分配之后不会改变。
  可变分配：先为每一个进程分配一定数目的物理块，在进程运行期间，可以根据实际情况进行适当的增加或者减少。驻留集大小可变。

页面置换的时候 范围 如何规定？

局部置换：发生缺页只能在自己的物理块进行置换。

全局置换：可以将OS保留的空闲物理块分配给缺页进程，也可以将别的进程持有的物理块置换到外存，再分配给缺页进程。

	局部置换	全局置换
固定分配	√	—
可变分配	√	√

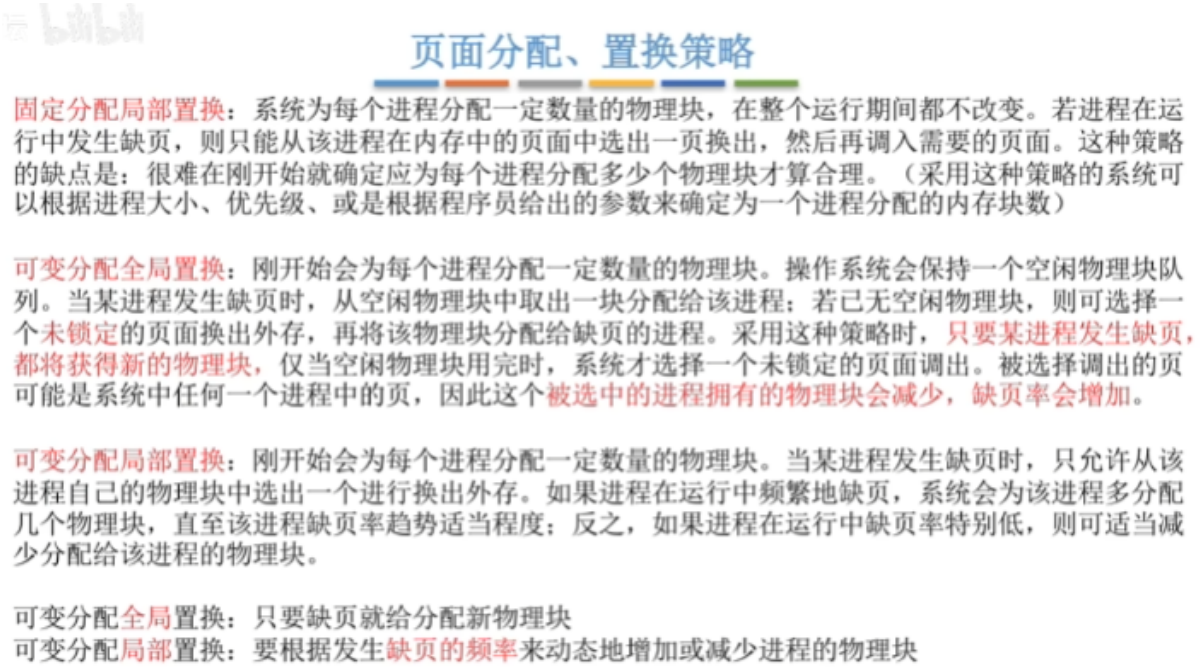
只会存在 固定分配局部置换 ， 可变分配局部置换 ， 可变分配全局置换 。

- 页面分配、置换策略说明：

  固定分配局部置换在一开始就固定分配多少物理块，缺页只能在自己这一部分选择一个页面换出再调入，但是很难在一开始就分配的比较好。

  可变分配全局置换：也是一种不太合理的，一旦发生缺页，就添加一个新的空闲的物理块。

  可变分配 局部置换 ：开始的时候进程有自己的物理块，之后缺页只能在自己已经分配的物理块中进行换出再调入，但是如果缺页率高，系统可以为进程多分配几个物理块，如果缺页率很低，可以适当减少分配给这个进程的物理块。



可变分配局部置换需要根据发生缺页的频率动态地增加、减少进程的物理块。

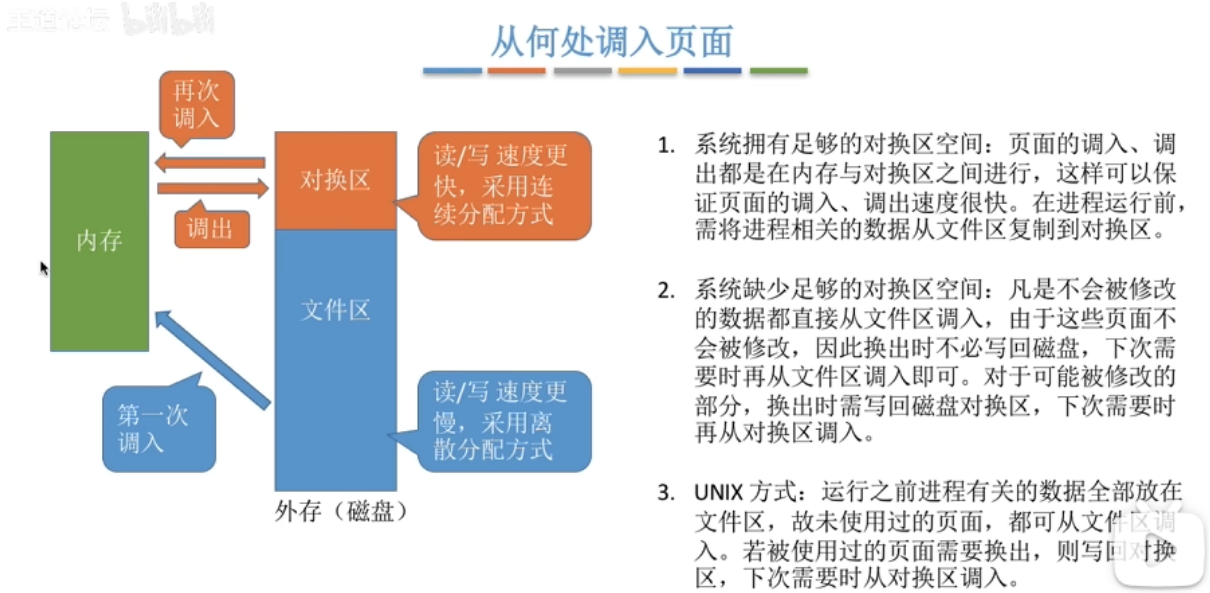
- 何时调入页面问题：

  之前所说的都属于 请求调页策略 ，只有进程运行期间发生却也的时候才会将页面调入内存。（每一次都需要I/O操作，因此I/O开销比较大）。

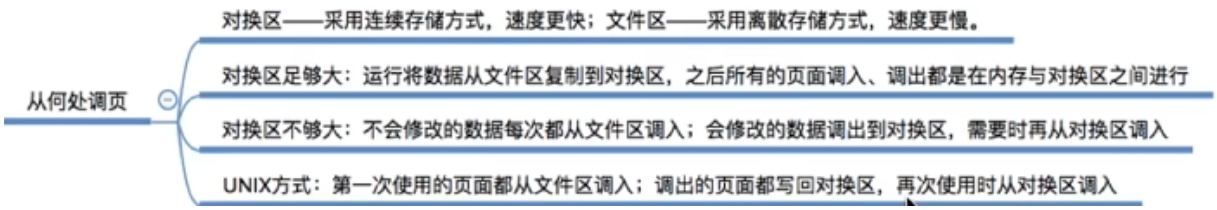
  另外一种 预调页策略 ：根据局部性原理，一次调入若干个页面的相邻页面。但是如果

周围的很少使用，又是很低效的一件事情，所以会提前预测，这种策略目前主要用在进程的首次调入。

- 从哪里调入页面问题：



！！！这个内容先不做记录

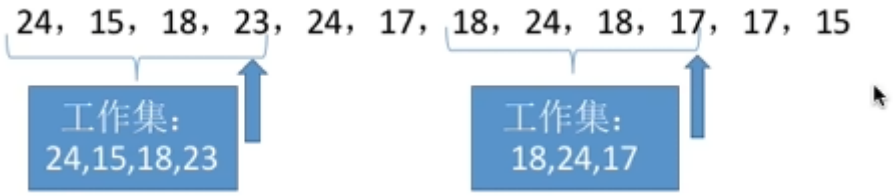


- 经常考察的点：抖动(颠簸)现象：
  抖动是指：刚刚换出到外存的页面马上又要换入内存，刚刚换入到内存的页面马上又要换出外存，这种频繁的页面调度行为就是抖动，也称为颠簸。
  颠簸现象的主要原因是为进程分配的物理块太少。

- 工作集：
  定义：在某段时间内，进程实际访问页面的集合。
  工作集的计算方法：假设当前窗口尺寸(驻留集尺寸为x)对于某一个点，往前看x个数(也包括自己)，工作集就是前x个数里面出现的数，要去重。

工作集的大小比较重要。

举例：第一个箭头工作集合为24,15,18,23 第二个箭头是18,24,17。



工作集合的大小可能会小于窗口尺寸。

操作系统可以首先计算进程中所有可能出现的工作集的大小，之后分配驻留集：
如果工作集最大为3，驻留集可以分配≥3。
一般驻留集不能小于工作集大小，否则会频繁 抖动/颠簸 。

另外：可以根据页面在不在工作集中来设计一个页面置换算法：选择不在工作集合的换出去。

拓展：基于局部性原理可知，进程在一段时间内访问的页面与不久之后会访问的页面是有相关性的。因此，可以根据进程近期访问的页面集合（工作集）来设计一种页面置换算法——选择一个不在工作集中的页面进行淘汰。

3.2_5_内存映射文件

有一个链接
操作系统 | 内存文件映射 —— 文件到内存的映射_内存映射文件-CSDN博客
到时候学完IO操作 再回来整理这个。