2. 虚拟内存管理

2.0. Pre

2.0.1. 引入虚存的原因

1. 一次性(全装入后才执行)+驻留性(运行完后作业才脱离内存)，对于大作业难以满足要求
2. 程序执行时有些代码用的较少(错误处理部分)，而程序IO又耗费世家

2.0.1. 局部性原理

1. 时间局部性：同一指令/数据短时间内被高频执行/访问**(这可以归因于大量的循环操作)**
2. 空间局部性：某条指令被访问后，其附近的指令有极大概率也被访问

2.1. 虚存基本概念

2.1.1. 虚存的定义

1. 部分装入：一部分装入内存，一部分放外存
2. 请求调入：执行时访问信息不在内存时，再要求OS将其调入内存
3. 置换功能：OS将内存中暂时不用的内容置换到外存
4. 虚拟内存：逻辑上扩充内存容量的系统

2.1.2. 虚存的特征

1. 离散性(最基本)：程序被打散存储在内存中
2. 多次性(最重要)：一个作业被分成多次调入内存
3. 交换性：作业在运行时可以不断从内存中换入换出
4. 虚拟性：用户使用的内存远大于实际内存

2.1.3. 其他有关虚存

1. 虚存实现手段：请求分页，请求分段
2. 硬件机构：外存外存都要足够大，有中断机构(访问内容不在内存时就中断程序)，地址变换机构，段/页表
3. 好处：较小内存执行较大程序，并发性提高，比覆盖技术编成更简单

2.2. 请求分页存储管理方式

2.2.1. 请求分页原理：局部性原理

1. 请求分页=基本分页+请求调页功能+页面置换功能
2. 页面调入策列
    1. 预调页策略：进程首次调入时，把预计很快要被访问的页，主动调入内存

2. 请求调页策略：进程运行时，将需要的页调入内存(通过**页面置换**把暂时不用的页置换出去)

PS: 内存中每一页都在外存上保留一份副本

3️⃣ 从何处调入页面：硬盘分为文件区+对换区，对换区IO快于文件区

1. 对换区足够大：全从对换区调入所需页面，运行前就把进程必要块放到对换区
2. 对换区不够大：不会被修改的文件都从文件区调入(减少IO)，需要修改的放到调换区
3. UNIX方式：未运行的页放在文件区，运行过又被换出的放对换区

2.2.2. 页表结构

页号	物理块号	状态位	访问字段	修改位	外存地址

1️⃣ 页号与物理块号：与之前相同，虚拟地址页^{映射}⟷物理地址快

2️⃣ 状态位(存在位)：判断页是否在主存，不在时触发**缺页中断**

3️⃣ 访问字段：记录局部性参数——页面一段时间内访问次数/最近多久未被访问

4️⃣ 修改位：记录页面调入内存后是否被修改

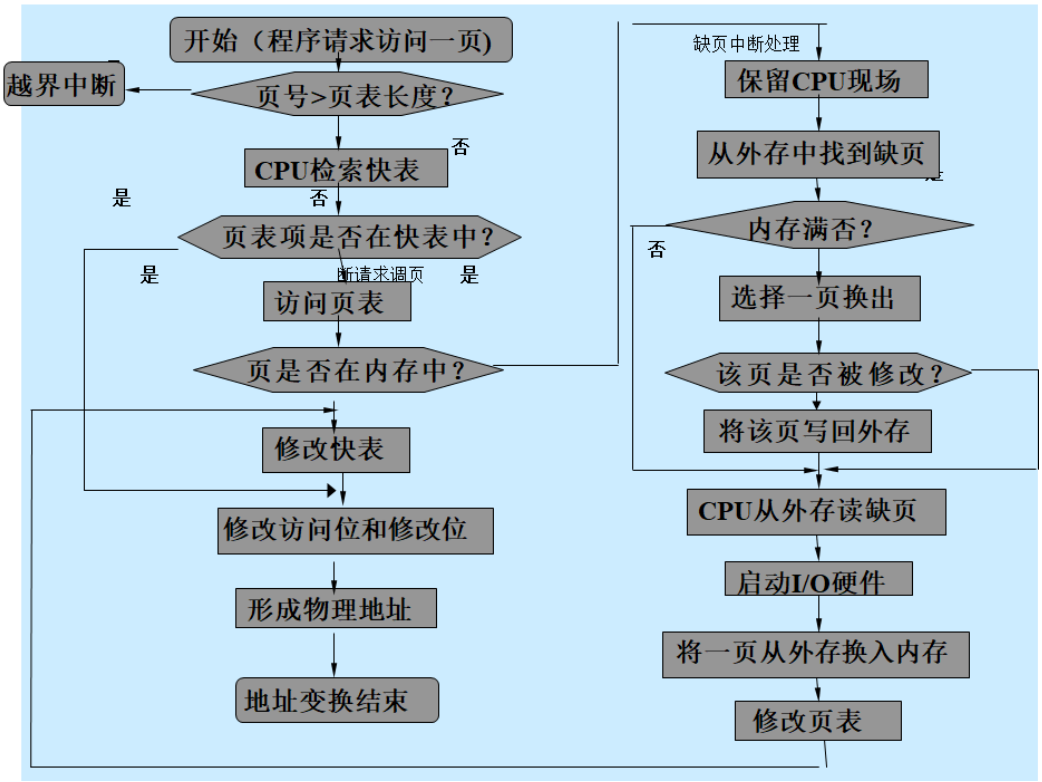
1. CPU以写方式访问页时，修改位被设置
2. 页在内存中没被修改，页换出时就不再写到外存(节省读写次数)

5️⃣ 外存地址：页面在外存的地址，调入该页时使用

2.2.3. 缺页中断与地址变换

1️⃣ 如果访问的页在内存中，地址变换与分页存储管理相同

2️⃣ 访问页不在内存，进程就请求OS从外存调入所需页



➕ 缺页中断与一般中断的区别

1. 缺页/一般中断，发生在指令执行期间/结束后
2. 一指令可触发多次缺页中断，如指令的两个操作数都在外存
3. 缺页/一般中断，返回到**本指令开始/下条指令**重新执行

2.2.4. 请求分页的性能

1. 优点：离散存储碎片少，提供虚拟存储主存利用率高
2. 缺点：硬件复杂，会抖动，最后一页仍有内部碎片
3. 主动动作：处理缺页中断，从磁盘读页(开销最大)，重新开始执行被中断程序
4. 缺页率=访问访问失败次数/进程访问内存总次数

2.3. 页面置换(淘汰)算法

2.3.1. 最佳置换(OPT)

1. 页面号引用串：进程执行时，按时间顺序引用的页面序列
2. OPT：已知页面号引用串的情况下(其实不可能的)，淘汰以后不再使用/最迟被使用的页
3. 特点：缺页率最低，但无法实现(仅作为理论最低缺页率的参考)

2.3.2. 先进先出(FIFO): 最简单

1. 队列结构：指针指向最先进入的页，每次淘汰指针指向的页
2. 缺点：会产生Belady异常，源于最早进来(也是最早淘汰)的页使用最频繁

2.3.3. 最近最少使用(LRU)

1. 原理：
    1. 最近最少使用的页$\xrightarrow[\text{性能接近}]{\text{近似的等同于}}$往后最迟被使用的页(OPT)
    2. 基于假设：刚访问的页倾向于马上又被访问
2. 含义：淘汰最久不被使用的页
3. 硬件支持：每个页表项有一个计数器，记录访问情况，作为被淘汰依据

2.3.4. 最不常用/最常用置换(LFU/MFU)

1. 含义：选择当前为止访问最少次/最多次的页面淘汰
2. 思想：淘汰最少使用的→访问多的倾向于再被访问；淘汰最多使用的→访问少的页往往最新调入
3. 实现方式：设置一个计数器，被访问一次该页的计数器就+1

2.3.5. LRU近似算法

2.3.5.1. 时钟置换(CLOCK)/最近未使用(NRU)/二次机会

LRU+FIFO的折中

1. 数据结构：每页设一个访问位(被访问后置1)，内存中所有页构成一个**循环链表**
2. 访问页在链表中：访问位改为1
3. 访问页不在链表中：
   1. 指针指向上次被淘汰页的下一页
   2. 指针顺序&循环遍历**循环链表**
      - 2.1. 指针指向页的访问位=1时，就把该访问位清零，然后继续遍历
      - 2.2. 指针指向页的访问位=0时，就淘汰该位然后访问位变1
4. 示例：

内存及控制信息			输入串	指针移动情况及帧替换信息	是否缺页
内存	访问位	指针	7	指针所指的位置恰好有访问位为 0 的	√
6	0			于是就淘汰这个帧，指针下移	
4	0	←			
3	1				
内存	访问位	指针	4	内存中没有 4，需要找到一个帧放入 4	√
6	0			指针所指的位置的访问位为 1	
7	1			将其变成 0，再下移（回到开头）	
3	1	←			
内存	访问位	指针		指针所指的位置恰好有访问位为 0 的	
6	0	←		于是就淘汰这个帧，指针下移	
7	1				
3	0				
内存	访问位	指针	3	内存中有 3，于是 3 所在帧的访问位变为 1	
4	1			指针不变	
7	1	←			
3	0				
内存	访问位	指针	6	内存中没有 6，需要找到一个帧放入 6	√
4	1			指针所指的位置的访问位为 1	
7	1	←		将其变成 0，再下移	
3	1				
内存	访问位	指针		指针所指的位置的访问位为 1	
4	1			将其变成 0，再下移（回到开头）	
7	0				
3	1	←			
内存	访问位	指针		指针所指的位置的访问位为 1	
4	1	←		将其变成 0，再下移	
7	0				
3	0				
内存	访问位	指针		指针所指的位置恰好有访问位为 0 的	
4	0			于是就淘汰这个帧，指针下移	
7	0	←			
3	0				

2.3.5.2. 改进时钟(CLOCK)/增强的二次访问

1️⃣ 改进的核心：增加了修改位(被修改了置1)，访问位同为0时优先淘汰没被修改的位(IO代价更小)

2️⃣ 算法步骤：进程启动将所有(访问位=0, 修改位=0)

1. 从指针位置开始首次循环遍历
2. 先试图找到第一个(**访问位=0**, **修改位=0**)页替换，若没找到⬇️
3. 再试图找到第一个(**访问位=0**, **修改位=1**)页替换，所扫描过之处皆置**访问位=0**
4. 如还没找到，就重复上述过程(重复下去一定能找到)

2.3.7. 页面缓冲(PBA)

1️⃣ 按照FIFO算法选择被置换页

2️⃣ 对那些要被换出的页面，为其在内存中建立两个链表(缓冲)

1. 对于被换出的未修改的未修改的页，丢到空闲页链表尾
2. 对于被换出的已修改的未修改的页，丢到已修改页链表尾

3️⃣ 对于缓冲中的页，在未来一段时间内如果要访问他们，可以快速响应(免去IO)

4️⃣ 直到修改页链表到达一定规模后，再将他们一同IO到磁盘

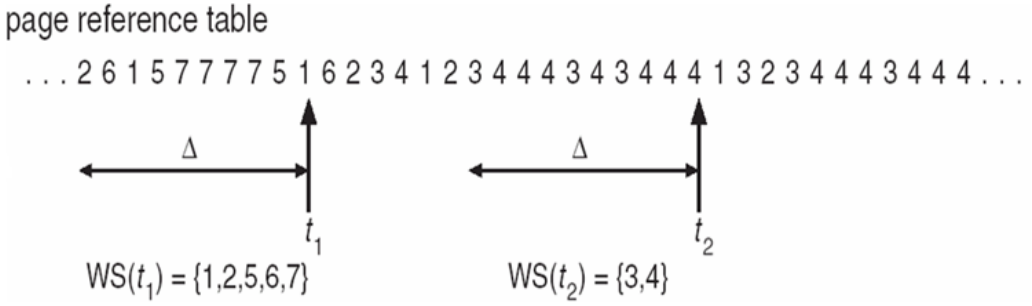
2.4. 工作集理论：基于局部性原理

1️⃣ 目的：解决抖动，提高CPU利用率

2️⃣ 原理：预知程序在特定时间内要访问的页面(活跃页面)并提前加载它们

3️⃣ 基本概念

1. 工作集：最近n次内存访问的页面集合，或者说落入工作集窗口的页面集合
2. 工作集窗口(Δ)：对于给定的访问序列选取定长的区间，其大小选定很重要，Δ过小则不能包含整个局部，Δ过大则可能包含多个局部



4️⃣ 工作集模型：

1. OS跟踪每个进程的工作集(如何跟踪是难点)，为其分配大于其工作集的物理块数
2. 若还有空闲物理块，启动其他进程
3. 若所有进程的工作集之和>可用物理块总数，OS会暂停&换出一个进程，释放的物理块给其他进程

2.5. 页面分配策略：如何给进程一定空闲页

2.5.1. 最少页数

1️⃣含义：保证进程运行的最小物理块数

2️⃣取值依据：指令格式，功能，寻址方式

➕某进程物理块数<最少页数：进程频繁缺页，进而崩溃

2.5.2. 分配&替换策略

1️⃣分配(固定分配)：平均(每个进程块数一样)+按比例+按优先级

2️⃣替换：全局(进程之间可争躲页，块数会增加)+局部(进程只能从自己那获得页，块数不变)

➕全局置换时，无法控制页错误率，系统吞吐率会更高

2.5.3. 组合

	固定分配：每个进程分得相同个物理块	**可变分配：OS给进程动态分配物理块**
局部置换：进程只在自己内存块中置换页面	各进程物理块数相同，进程间不争夺块，块少进程频繁换页/块多进程浪费内存(核心在于进程块数分配)	先给每进程一定物理块，进程互相枪内存。增加频繁换页进程块数，减少缺页率过低进程块数
全局置换：进程需要更多内存时，可以替换掉其他进程的内存页	无	OS维护一空闲物理块队列，进程每次缺页就从队里取一个，取完了就去抢其他进程的块

2.7. 抖动与缺页率

2.7.1. Belady异常

1️⃣含义：FIFO算法中缺页率会随内存中块数增加而增加

2️⃣成因：FIFO算法的策略与内存动态特征违背，总会换掉进程要访问的页

3️⃣PS：LRU算法和最佳置换算法永远不会出现Belady异

2.7.2. 抖动/颠簸现象

1️⃣含义：某一页面刚被换出又被访问(重新调入)，频繁调入调出，CPU利用率低下

2️⃣成因：进程分到的物理块太少

3️⃣解决方案：

1. 全局置换会造成颠簸，局部置换能限制颠簸
2. 根本上要给进程足够的物理块

2.7.3. 缺页率

作业有n页，系统给作业分了m页

如果作业运行时要访问A次页面，当所访问的页不在内存中时，需要将该页调入内存F次

则定义

1 缺页率：f=F/A

2 命中率：1-f

3 控制缺页频率：缺页率太低/高，回收/分给一些进程的页框

2.8. 请求分段存储管理系统

1 将当前需要的若干段装入主存便可运行，访问分段不在主存中时再调入，将不用分段也置换出去

2 段表表项结构

段号	段长	内存始址	访问字段	修改位	状态位	外存地址

PS. 总结

PS.1. 三种离散分配方式

对比及联系	内存管理方式		
	分页存储管理	分段存储管理	段页式存储管理
有无外部碎片	无	有	无
有无内部碎片	有	无	有
优点	内存利用率高，基本解决了内存零头问题	段拥有逻辑意义，便于共享、保护和动态链接	兼有两者的优点
缺点	页缺乏逻辑意义，不能很好地满足用户	内存利用率不高，难以找到连续的空闲区放入整段	多访问一次内存

PS.2. 几种内存管理

比较的方面	单一连续分配	分区		分页		基本分段	基本段页式
		固定分区	可变分区	基本分页	请求分页		
内存块的分配	连续	连续		离散		离散	离散
适用环境	单道	多道		多道		多道	多道
地址维数	一维	一维		一维		二维	二维
是否需要全部程序段在内存	是	是		是	否	是	是
扩展内存	交换	交换		交换	虚拟存储器	交换	交换
内存分配单位	整个内存的用户可用区	分区		页		段	页
地址重定位	静态	静态	动态	动态	动态	动态	动态
重定位机构	装入程序	装入程序	重定位寄存器	页表 页表控制寄存器 加法器		段表 段表控制寄存器 加法器	段表 页表 段表控制寄存器 加法器
信息共享	不能	不能		可以，但限制多		可以	可以