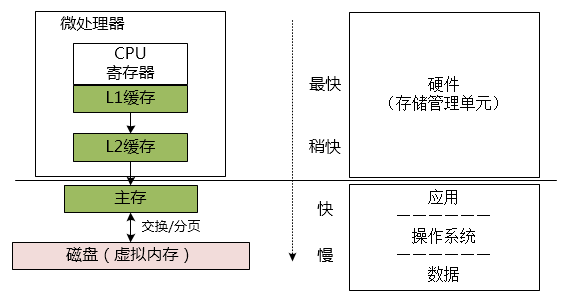
# 内存管理与虚拟内存

## 0. 内存分层体系



## 1. 内存管理的需求

**1. 重定位：**操作系统中可用的内存空间被多个进程共享，当一个进程被换出磁盘再换入内存时，不必将其放在被换出前相同的位置（否则会是很大的限制）。所以操作系统需要把进程重定位到内存的不同区域。处理器硬件和操作系统软件必须能够通过内存管理把程序中代码的内存访问转换成实际的物理内存地址，并反映程序在内存中的当前位置。

**2. 保护：**每个进程都受到保护，一个进程的程序不能访问其他进程的数据区。

**3. 共享：**保护机制必须具有一定灵活性，以允许多个继承访问内存的同一部分。

**4. 逻辑组织：**程序不必考虑底层细节，连续的逻辑地址空间

**5. 物理组织：**存储器分为两级，内存和外存，根据容量和存储、访问需求分别存储。

## 2. 内存分区

### 2.1. 固定分区

在系统生成阶段，内存被划分为许多静态分区。进程可以被装入到大于或等于自身大小的分钟。优点：实现简单。缺点：程序可能太大而不能放到一个分区中，有**内存碎片**（被装入的数据块小于分区的大小），内存利用率低。

### 2.2. 动态分区

分区是动态创建的，因而使得每个进程可以被装入到与自身大小正好相等的分区中。缺点：随着时间推移，内存中会出现许多小的空洞，即**外部碎片**（分配的单间之间的未使用内存），内存的利用率随之下降。

解决外部碎片的办法：

**1. 压缩技术：**操作系统不时移动进程，使进程之间连续占用内存，所有已用内存连成一片，所有空闲内存连成一片。但压缩过程耗时，浪费处理时间，而且需要动态重定位能力。

**2. 放置算法：**操作系统利用放置算法，根据不同策略将进程分配到内存中。

1. 最佳适配：选择与要求的大小最接近的块；
2. 首次适配：从开始扫描内存，选择大小足够的第一块；
3. 下次适配：从上一次放置的位置开始扫描内存，选择下一个足够大小的块。

## 3. 分页与分段

### 3.1. 分页

固定分区和动态分区都会产生内存碎片，内存利用率极低。分页技术是把内存划分为大小固定相等的**页框**（**frame**），进程中称为**页**（**page**）的块可以指定到内存中称为**页框**的可用块。每个进程被划分成许多大小与页框相等的页。要装入一个进程，需要把进程包含的所有页都装入内存内不一定连续的某些页框中。

进程的所有页不一定会加载到内存的连续的页框中。操作系统为每个进程维护了一个**页表（page table）**。页表给出了进程中每一页对应的页框位置。划分物理内存为固定大小的frame，划分逻辑地址空间至相同大小的页。

一个逻辑地址是一个二元组(p,offset)，p代表页号，offset是偏移。若逻辑地址中使用s个bit来表示页号，那么逻辑地址(f,offset)表示的虚拟地址是2^s\*p+offset.

一个物理地址是一个二元组(f,offset)，f代表页框号，offset是偏移。物理地址(f,offset)表示的物理地址是：2^s\*f+offset.

一个逻辑地址addr转换为物理地址的过程为：

1. 根据逻辑地址的前s位得到页号p，根据后几位计算出偏移offset；
2. 查找页表（由OS建立），找到该进程页对应的页框号f；
3. 由f\*2^s+offset得到物理内存地址。

### 3.2. 分段

采用分段技术，可以把程序和其相关的数据划分到几个**段（segment）**中。采用分段技术时，**逻辑地址由段号和偏移量组成**。段有一个最大长度限制，但并不要求所有程序的所有段的长度都相等。

每个进程都有一个段表。在分段方案中，逻辑地址转换为物理地址的过程与分页方案类似。

## 4. 虚拟内存

进程中所有内存访问都是逻辑地址，这些逻辑地址在运行时动态地被换成物理地址。这意味着一个进程可以被换入换出内存，使得进程可以在执行过程中的不同时刻占据内存中的不同区域。同时，一个进程可以被分为很多块（页和段）。在执行过程中，这些块不需要连续地位于内存中。内存中保存着要取的下一条指令所在的块或页，以及将要访问的下一个数据单元的所在块，就可以保持程序暂时继续下去。

### 4.1. 局部性原理

局部性原理表现在以下两个方面：

* **时间局部性：**如果程序中的某条指令一旦执行，不久以后该指令可能再次执行；如果某数据被访问过，不久以后该数据可能再次被访问。产生时间局部性的典型原因，是由于在程序中存在着大量的循环操作。
* **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将被访问，即程序在一段时间内所访问的地址，可能集中在一定的范围之内，这是因为指令通常是顺序存放、顺序执行的，数据也一般是以向量、数组、表等形式簇聚存储的。

时间局部性是通过将近来使用的指令和数据保存到高速缓存存储器中，并使用高速缓存的层次结构实现。空间局部性通常是使用较大的高速缓存，并将预取机制集成到高速缓存控制逻辑中实现。虚拟内存技术实际上就是建立了“内存一外存”的两级存储器的结构，利用局部性原理实现髙速缓存。

### 4.2. 虚拟内存技术的实现

虚拟内存中，允许将一个作业分多次调入内存，虚拟内存的实现需要建立在离散分配的内存管理方式的基础上。虚拟内存的实现有以下三种方式：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。  
不管哪种方式，都需要有一定的硬件支持。一般需要的支持有以下几个方面：

* 一定容量的内存和外存。
* 页表机制（或段表机制），作为主要的数据结构。
* 中断机构，当用户程序要访问的部分尚未调入内存，则产生中断。
* 地址变换机构，逻辑地址到物理地址的变换。

### 4.3. 请求分页管理方式实现虚拟内存

请求分页系统建立在基本分页系统基础之上，为了支持虚拟存储器功能而增加了请求调页功能和页面置换功能。请求分页是目前最常用的一种实现虚拟存储器的方法。

在请求分页系统中，只要求将当前需要的一部分页面装入内存，便可以启动作业运行。在作业执行过程中，当所要访问的页面不在内存时，再通过调页功能将其调入，同时还可以通过置换功能将暂时不用的页面换出到外存上，以便腾出内存空间。

为了实现请求分页，系统必须提供一定的硬件支持。除了需要一定容量的内存及外存的计算机系统，还需要有**页表机制**、**缺页中断机构**和**地址变换机构**。

#### 4.3.1. 页表机制

请求分页系统的页表机制不同于基本分页系统，请求分页系统在一个作业运行之前不要求全部一次性调入内存，因此在作业的运行过程中，必然会出现要访问的页面不在内存的情况，如何发现和处理这种情况是请求分页系统必须解决的两个基本问题。为此，在请求页表项中增加了四个字段：

1. **驻留位P**：用于指示该页是否已调入内存，1-内存，0-外存。
2. **访问位A**：用于记录本页在一段时间内被访问的次数，或记录本页最近己有多长时间未被访问，供置换算法换出页面时参考。
3. **修改位M**：标识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，通常是物理块号，供调入该页时参考。

#### 4.3.2. 缺页中断机构

在请求分页系统中，每当所要访问的页面不在内存时，便产生一个缺页中断，请求操作系统将所缺的页调入内存。此时应将缺页的进程阻塞（调页完成唤醒)，如果内存中有空闲块，则分配一个块，将要调入的页装入该块，并修改页表中相应页表项，若此时内存中没有空闲块，则要淘汰某页（若被淘汰页在内存期间被修改过，则要将其写回外存)。

缺页中断作为中断同样要经历，诸如保护CPU环境、分析中断原因、转入缺页中断处理程序、恢复CPU环境等几个步骤。但与一般的中断相比，它有以下两个明显的区别：

1. 在指令执行期间产生和处理中断信号，而非一条指令执行完后，属于内部中断。
2. 一条指令在执行期间，可能产生多次缺页中断。

#### 4.3.3. 地址变换机构

在进行地址变换时，先检索快表：

1. 若找到要访问的页，便修改页表项中的访问位（写指令则还须重置修改位)，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。
2. 若未找到该页的页表项，应到内存中去查找页表，再对比页表项中的状态位P，看该页是否已调入内存，未调入则产生缺页中断，请求从外存把该页调入内存。

### 4.4. TLB

TLB - translation lookaside buffer，快表，直译为旁路快表缓冲，也可以理解为页表缓冲，**地址变换高速缓存**，**缓存近期访问的页帧转换表项**。由于页表存放在主存中，因此程序每次访存至少需要两次：一次访存获取物理地址，第二次访存才获得数据。提高访存性能的关键在于依靠页表的访问局部性。当一个转换的虚拟页号被使用时，它可能在不久的将来再次被使用到。内存管理硬件使用TLB来改善虚拟地址到物理地址的转换速度。使用TLB内核可以快速的找到虚拟地址指向物理地址，而不需要请求RAM内存获取虚拟地址到物理地址的映射关系。

当CPU要访问一个虚拟地址/线性地址时，首先根据虚拟地址的高s位(页号)在TLB中查找。如果是表中没有相应的表项，称为TLB miss，需要通过访问慢速RAM中的页表计算出相应的物理地址。同时，物理地址被存加入到TLB表项中，以后对同一线性地址的访问，直接从TLB表项中获取物理地址即可，称为TLB hit（TLB命中）。

### 4.5. 局部页面置换算法

局部页面置换算法的目的：尽可能减少缺页中断次数

#### 4.5.0. Belady现象

**描述：**在使用FIFO算法时，有时会出现分配的物理页面数增加，缺页率反而上升的异常现象。

**原因：**FIFO算法的置换特征与进程的访问内存的动态特征是矛盾的，与置换算法的目标并不一致（即替换使用较少的页面）。

#### 4.5.1. 最优页面置换算法

思路：当缺页中断发生，对于保存在内存中的每一个逻辑页面，计算在它下一次访问之前，还需要多长时间（实际上OS无从知道），从中选择等待时间最长的那个作为被置换的页面。

**这种算法是种理想情况，作为评价其他算法性能的依据。**

#### 4.5.2. 先进先出算法FIFO

思路：选择在内存中驻留时间最长的页面并淘汰。系统维护了一个链表，记录了所有位于内存当中的逻辑页面。从链表的排列顺序，链首页面驻留时间最长，链尾页面驻留时间最短。却也中断发生时，链首页面被交换，新页面被添加至链尾。

特点：性能差，调出的页面可能是经常访问的页面，且有belady现象。

#### 4.5.3. 最近最久未使用算法LRU (Least Recently Used)

思路：当一个缺页中断发生时，选择最久未使用的页面并淘汰。该算法是对最优页面置换算法的一个近似（依据历史预测未来），根据程序的局部性原理。

实现：需要记录各个页面的使用时间的先后顺序，开销较大。

* **链表实现**：最近刚使用的页面作为表头，最久未被访问的页面作为表尾。每次访问内存时，找到相应页面，从表中取出并放到表头。每次缺页中断时淘汰表尾。
* **栈实现**：访问某页时，页号压入栈。若栈内有与此页号相同的页，则抽出。缺页中断时，淘汰栈顶。

#### 4.5.4. 最不常用算法LFU (Least Frequently Used)

思路：缺页中断时，选择访问次数最少的页面并淘汰。需要为每个页面设置一个访问计数器。

#### 4.5.5. 时钟页面置换算法 (Clock算法)

1. 要用到页表项中的访问位（读写均为访问），该位初始化为0，被访问时由硬件置1；
2. 各个页面组成一个环形链表，类似时钟表面，把指针指向最老的页面（最先进来）；
3. 缺页中断时，考察指针指向的最老页面，若访问位为0，立即淘汰。若访问位为1，置为0，指针往后移一格，如此下去，直到找到被淘汰的页面。

#### 4.5.6. 二次机会算法 (Enhanced Clock Algorithm)

基于Clock算法，通过增加使用的位数目，可以使得CLOCK算法更加高效。在使用位的基础上再增加一个修改位，则得到改进型的CLOCK置换算法。这样，每一帧都处于以下四种情况之一：

1. 最近未被访问，也未被修改(u=0, m=0)。
2. 最近被访问，但未被修改(u=1, m=0)。
3. 最近未被访问，但被修改(u=0, m=1)。
4. 最近被访问，被修改(u=1, m=1)。

算法执行如下操作步骤：

1. 从指针的当前位置开始，扫描帧缓冲区。在这次扫描过程中，对使用位不做任何修改。选择遇到的第一个帧(u=0, m=0)用于替换。
2. 如果第1)步失败，则重新扫描，查找(u=0, m=1)的帧。选择遇到的第一个这样的帧用于替换。在这个扫描过程中，对每个跳过的帧，把它的使用位设置成0。
3. 如果第2)步失败，指针将回到它的最初位置，并且集合中所有帧的使用位均为0。重复第1步，并且如果有必要，重复第2步。这样将可以找到供替换的帧。

### 4.6. 页面分配策略

#### 4.6.1. 驻留集大小

对于分页式的虚拟内存，在准备执行时，不需要也不可能把一个进程的所有页都读取到主存，因此，操作系统必须决定读取多少页。也就是说，给特定的进程分配多大的主存空间，这需要考虑以下几点：

1. 分配给一个进程的存储量越小，在任何时候驻留在主存中的进程数就越多，从而可以提高处理机的时间利用效率。
2. 如果一个进程在主存中的页数过少，尽管有局部性原理，页错误率仍然会相对较高。
3. 如桌页数过多，由于局部性原理，给特定的进程分配更多的主存空间对该进程的错误率没有明显的影响。

基于这些因素，现代操作系统通常釆用三种策略：

1. **固定分配局部置换：**它为每个进程分配一定数目的物理块，在整个运行期间都不改变。若进程在运行中发生缺页，则只能从该进程在内存中的页面中选出一页换出，然后再调入需要的页面。实现这种策略难以确定为每个进程应分配的物理块数目：太少会频繁出现缺页中断，太多又会使CPU和其他资源利用率下降。
2. **可变分配全局置换：**这是最易于实现的物理块分配和置换策略，为系统中的每个进程分配一定数目的物理块，操作系统自身也保持一个空闲物理块队列。当某进程发生缺页时，系统从空闲物理块队列中取出一个物理块分配给该进程，并将欲调入的页装入其中。
3. **可变分配局部置换：**它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，这样就不会影响其他进程的运行。如果进程在运行中频繁地缺页，系统再为该进程分配若干物理块，直至该进程缺页率趋于适当程度； 反之，若进程在运行中缺页率特别低，则可适当减少分配给该进程的物理块。

#### 4.6.2. 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内存的时机，可釆取以下两种调页策略：

1. **预调页策略：**根据局部性原理，一次调入若干个相邻的页可能会比一次调入一页更高效。但如果调入的一批页面中大多数都未被访问，则又是低效的。所以就需要釆用以预测为基础的预调页策略，将预计在不久之后便会被访问的页面预先调入内存。但目前预调页的成功率仅约50%。故这种策略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。
2. **请求调页策略：**进程在运行中需要访问的页面不在内存而提出请求，由系统将所需页面调入内存。由这种策略调入的页一定会被访问，且这种策略比较易于实现，故在目前的虚拟存储器中大多釆用此策略。它的缺点在于每次只调入一页，调入调出页面数多时会花费过多的I/O开销。

#### 4.6.3. 从何处调入页面

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。对换区通常是釆用连续分配方式，而文件区釆用离散分配方式，故对换区的磁盘I/O速度比文件区的更快。这样从何处调入页面有三种情况：

1. **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，以提髙调页速度。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。
2. **系统缺少足够的对换区空间：**凡不会被修改的文件都直接从文件区调入；而当换出这些页面时，由于它们未被修改而不必再将它们换出。但对于那些可能被修改的部分，在将它们换出时须调到对换区，以后需要时再从对换区调入。
3. **UNIX方式：**与进程有关的文件都放在文件区，故未运行过的页面，都应从文件区调入。曾经运行过但又被换出的页面，由于是被放在对换区，因此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，则无需再从对换区调入。

### 4.7. 多级页表

假设虚拟地址空间为32位（4GB），每一页大小为4KB，则进程需要的页表项数目为4GB/4KB=1M。若每个页表项大小为4B，则存储页表项总共需要4M内存。即使用一级页表时，页表占据4M内存空间。每个进程都要预先建立好4M的页表，进程数一多，内存中仅是页表项占据的内存空间已经足够大了。

每个进程都有4GB的虚拟地址空间，而显然对于大多数程序来说，其使用到的空间远未达到4GB，何必去映射不可能用到的空间呢？也就是说，一级页表覆盖了整个4GB虚拟地址空间，但如果某个一级页表的页表项没有被用到，也就不需要创建这个页表项对应的二级页表了，即可以在需要时才创建二级页表。

二级页表由一个4KB的根页表和1024个4K的二级目录表项组成。对于一个32位的虚拟地址，高10位指出二级目录表项的地址，根据该地址定位到某个二级目录表，根据二级目录表中的Frame Number和32位虚拟地址的offset可以转换为物理地址。

## 5.LeetCode置换算法相关题目

### 5.1. LRU题目

面试题 16.25. LRU 缓存（146. LRU 缓存机制）

<https://leetcode-cn.com/problems/lru-cache-lcci/solution/>

### 5.2. LFU题目