内存管理

综合考虑容量+速度+价格，采用多级存储系统：（CPU）寄存器-高速缓冲存储器-内存-外存，变化趋势

内存管理：1、内存的使用情况 2、内存的分配和回收 3、内存的共享和保护 4、内存不足的措施

程序和数据需要装载到内存能够参与计算，三种装入方式：A完全静态装入、B静态重定位装入、C动态重定位装入

A、完全静态装入：相当于将外存中的可执行文件整体平移到内存，并且要求连续，可能会出现内存不够无法装入。也就是说**装入模块中的地址都是物理地址，直接装入** B、静态重定位装入：装入模块中的地址是逻辑地址，**在装入时需**要重定位转化为物理地址，并且也要求是连续的。优点是不需要特定硬件作为支持，缺点是要求内存连续的空间、且并没有真正实现虚拟地址；C、动态重定位装入：**将要执行指令时**，才进行地址转化，即虚拟地址->物理地址。因此需要一个重定位寄存器，相当于基址寄存器，基址+偏移=物理地址。优点是真正实现了虚拟地址，并且可以不连续装入，缺点是需要地址变换机构？//我也不太确定，臆想

分区管理

将存储空间分为若干大小不等的区，操作系统独占一个区，其余用于放置各个并发进程，需要一个分区说明表来描述。分为（固定分区+可变分区），也就是静态和动态。

固定分区：每个区域大小固定，进程需要装入时依此查找。有一个分区使用表，记录分区号、大小、起始地址、状态（是否已经使用）。优点：实现简单；缺点：1、分区大小固定，可能会出现程序无法装入的情况 2、存在内部内存碎片问题 3、最大进程数量受到限制

可变分区：起始只有两个分区：一个操作系统占用、一个空闲分区，程序装入后分配一块区域，剩余的是一个新的空闲区。需要用到已分配的分区表+空闲分区表（用于记录空闲分区的的大小和起始位置）+空闲分区链表+资源请求表。空闲分区表会占用额外的内存，空闲分区链表不会，它使用的是该分区起始的几个单元。

程序在装入内存时的分区选择算法：A、最先适配 B、下次适配 C、最优适配 D、最差适配

A、从起始地址开始查，找到的第一个合适的分区，用来存放程序 B、与A类似，不过起始位置为上次选中的分区 C、将空闲分区从小到大排序，依此比较，选取第一个适配的 D、与C相反

分区的回收：最重要的是相邻空闲区的合并，基于靠上侧的空闲区的地址，sum大小，进行空闲区合并，更新相应记录表格。另外，对于无法处理的外部碎片，可以通过移动程序来使得内存紧凑，进而可以利用起之前的空闲区碎片。但是相应的，会带来巨大的系统开销。

产生碎片的原因是：连续分配->离散分配来解决这个问题->页式、段式、段页式

页式管理

进程的虚拟地址空间分为若干大小相等的页，实际的物理内存也分为大小相同的页帧，只会在最后一个页帧内部产生碎片，分页管理实现了离散装入。有效减少了碎片，提高了CPU内存的利用率。这时虚拟地址可以分为两部分:页号+页内地址，并且引入一个页表来记录进程的页号与页帧的对应关系。（静态分页管理+动态分页管理）

静态分页管理：执行前将进程全部调入内存中，这样可能出现内存页帧不够用的情况，这时候只能等待。

连续的页号对应不连续的页帧，页表中的表项是页号和页帧号，页表也是存放在内存中的，因此PTR页表寄存器中记录着对应页表的地址和大小，该信息最开始保存在PCB中，发生进程调度之后进入PTR中，用于将程序调入内存，这就说明每个进程都会有一个页表。

地址转换：拼接。为了提高查找速度，引入快表TLB，其中内容是页表的一个子集，保存着最近访问的页号与页框号，对其进行访问并不是访存。快表的表项：页号、页框号、访问位、状态位,当进行进程切换时，需要刷新快表。

为了解决页表大，查找慢的情况，可以引入多级页表，将页表也实现离散分配。以二级页表为例：有一个一级页表，其中存放着多个二级页表的物理块号，通过该物理块号得到对应二级页表，二级页表中放置着多个页号与物理块号的对应关系，那么相应的，这时的PTR中存放的应该是一级页表的地址和长度；逻辑地址也被分为：一级页号+二级页号+页内地址

动态页式管理：有两种实现：请求分页管理/预调入分页管理，重点是请求分页管理，核心是需要时才调入。

修改页表：页号、页帧号、修改位、状态位、有效位、访问位、外存地址。地址转换过程与静态分页管理类似，都是页号->物理块号->物理地址，如果所访问的页不在内存中，则触发缺页中断，将该页调入内存中，若内存有空余位置，则直接吊入；若满，需要相应的页帧置换算法来替换某一页。

页帧置换算法：A、最优置换算法 B、先进先出置换算法 C、最近最久未使用 D、最近未使用

A、选择之后不再访问或者最远访问的某一页换出，实际上不可行，因为不能提前已知访问串。B、按照先后顺序形成队列或者设置计时器。优点：实现简单 ； 缺点：1、与进程访问内存的特性不相符，有可能被淘汰的页是经常被访问的 2、会出现随着也帧数的增加，缺页次数反而增加的blady现象。C、计时器/移位寄存器/栈，来选取距离当前访问最长时间的一页进行淘汰 D、所有页形成循环队列，从上次被淘汰的页开始扫描，不匹配访问位修改=0，若遇到访问位=0的，进行淘汰置换；总之就是淘汰两次置换期间未访问的页。

可以在D的基础上进行修改，增加对修改位的判断，优先淘汰未访问+未修改的页。

页面置换算法可以与页面缓冲算法结合使用，主要用于已修改页的写回。对于淘汰的页，如果被修改放入已修改链表，否则放入空闲也链表，对于已修改页链表，成簇写回。

段式管理

程序按照功能划分为段，每一段都分配相应的内存，段内连续。分段无内部碎片，但是有外部碎片。

逻辑地址：段号+段内地址，为了记录各段的信息，需要数据结构段表，其中记录着段号、段基址、段长度、访问位，修改位、保护位等，该段表在内存的某个位置，具体信息刚开始存放在PCB中，后续同步到段表地址寄存器中，类似于PTR。一个进程对应一个段表。对于**地址转换**：首先查段表得到段号对应的段地址，在结合段内地址计算出物理地址。为了加快查表速度，也引入快表，表项包括：段号、段基址、段长度、访问位、状态位。

分页是程序员不可见的，分段是程序员可见的，页大小有系统决定，段大小用户程序决定；分段有利于共享程序数据，分页有利于提高内存利用率，将两者结合，形成段页式管理。

段页式管理

程序分段，段分页，内存分页，类同页式管理。逻辑地址：段号+页号+页内地址，程序员可见段号+其他，其他通过地址变换机构转化为页号+页内地址。每个进程对应一个段表，每个段表项对应一个页表，每个页表对应若干物理块。最开始的段表的起始地址和长度存在于PCB中，后续同步到段表寄存器中。

也会引入快表的概念，快表项有段号、页号、物理块号、访问位、状态位。这里引入快表十分重要：存放10%的页表内容可以达到90%的命中率。

覆盖和交换（内存扩充技术）

覆盖针对的是进程内部，逻辑上不同时执行的程序段共享内存区，但是其中的分段工作十分复杂，在覆盖过程中也会有系统开销；交换针对的是进程之间，比如将内存中处于等待的进程换出到外存交换区，将外存交换区中的进程换入内存，并执行。

通过全局置换算法来为进程分配可变数目的物理块，需要用到工作集：W(t,Δ)，也就是t时刻之前工作窗口内页面的集合；常驻集：此时在内存中的页面集合。当常驻集包含工作集的时候，缺页率较低，原理是程序的局部性（时间空间）。

还有一个概念是抖动，也就是频繁调入调出，原因：1、页面置换算法不合适 2、物理块<工作集 3、程序结构编写不合适 4、分配的物理块少 5、页面大小

那么就通过工作集来指导分配物理块的数量，算法思路：具体来讲就是求W，因此维护一个链表，也就是W，当访存时，根据访存情况，换出不在工作集中的页面，同时更新W，如果触发缺页，换入页，更新W。