连续内存管理

1. 存储管理方式
2. 连续分配方式（——连续，就会产生外碎片）
   1. 单一分区分配
      1. 适用于单道批处理系统——只有一个作业进内存
      2. 主存被分为两部分——内存低端放操作系统，内存高端放用户进程
      3. 只需考虑用户程序不要影响操作系统 -> 采用基地址寄存器策略
      4. 基址寄存器（重定位寄存器）：存着最小物理地址的值，用于逻辑地址到物理地址的映射；
      5. 限长寄存器：存着逻辑地址的范围（每个逻辑地址必须比限长寄存器的值小）
   2. 多分区：可用于多道程序的较简单的存储管理方式
      1. 固定分区分配
         1. 定义：①作业在装入前，内存就被划分成若干个固定大小的连续分区；②划分工作可以由系统管理员完成，也可由操作系统完成；③划分完成后，系统运行期间不再重新划分，分区个数、分区大小都不变，所以又称“静态分区”。
         2. 划分分区方法：
            1. 分区大小相等：适用于多个相同程序的并发执行、处理多个类型相同的对象。
            2. 分区大小不等：多个小分区、适量的中等分区、少量的大分区。根据程序的大小，分配当前空闲的、适当大小的分区。
         3. 实现：系统中有一张分区说明表，每个表目说明一个分区的大小、起始地址、是否已分配。
         4. 优点：易于实现、开销小
         5. 缺点：分区大小固定 -> 内碎片

分区总数固定 -> 限制并发执行的进程数目

* + 1. 可变分区（动态分区分配）
       1. 定义：分区的划分是动态的，不是预先确定的。
       2. 划分分区方法：当一个进程即将装入内存时，它将从一个足够容纳它分区中分配内存。
          1. 分区分配算法；寻找某个空闲分区，其大小需≥程序的要求。若＞所要求的，则该分区分割成两个分区，其中一个为要求的大小，并在空闲分区表中将该区状态标记为“占用”，并一个分区标记为“空闲”。分区的先后次序通常是从内存低端到高端。

首先适应算法：

定义：分配最先找到的合适的分区。

目的：减少查找时间 。

实现：空闲分区表（链）中的空闲分区按照地址由低到高排序。

特点：①分配和释放的时间性能较好，较大的空闲分区可以被保留在内存高端；②随着低端分区不断划分而产生较多小分区，每次分配时查找时间开销会增大；③在系统不断分配和回收中，必定会出现一些不连续的小的空闲区，称为外碎片。虽然可能所有碎片的总和超过某一个作业的要求，但是由于不连续而无法分配。

最佳适应算法：

搜索整个序列，找到适合条件的最小的分区进行分配。

目的：能使碎片尽量小。

实现：空闲分区表（链）中的空闲分区按照分区大小由小到大排序。

特点：①从个别来看，外碎片较小，但从整体来看，会形成较多无法利用的碎片；②较大的空闲分区可以被保留。

最差适应算法：搜索整个序列，找到适合条件的最大的分区进行分配。

* + - * 1. 分区释放算法：将相邻的空闲分区合并成一个空闲分区。
        2. 紧缩技术：

定义：把一些较小的空闲分区结合成一个大的空闲分区。

目的：减少外碎片。

实现：只有重定位是动态的时候，才有可能进行紧缩，紧缩在执行时期进行。

i/o问题

含义：某个进程执行i/o时刚好做了紧缩，导致i/o无法返回到该进程。

解决：当i/o时，把进程锁定在内存中。

只对操作系统的缓冲区进行i/o。

* + - 1. 实现：系统中有一张空闲分区表存储着已分配的分区和空的分区（或者采用空闲分区链存储内存空间的使用情况）。

1. 分页（——一种离散的内存管理方式）
   1. 含义：
      1. 进程的物理地址空间可以是不连续的，如果有可用的物理内存，它将分给进程。
      2. 把物理内存分成大小固定的块，叫做“frame（帧）（页框）”
      3. 把逻辑内存也分成固定大小的块，叫做“page（页）”
   2. 实现：
      1. 保留所有空闲帧的记录
      2. 运行一个有N页大小的进程，需要找到N个空的页框（帧）读入程序
      3. 建立一个页表，建立页到帧的映射，把逻辑地址转换为物理地址
      4. 可能会产生内碎片
   3. 地址转换
      1. 实现原理：
         1. CPU产生的地址（逻辑地址）被分为页号和偏移，（页号：包含每页在物理内存中的基址，用来作为页表的索引）（偏移：与基址相结合，用来确定送入内存设备的具体物理内存地址）
         2. 若逻辑地址空间为2^m，物理地址空间为2^n，则一个逻辑地址的前（m-n）位表示页号，后n位表示偏移。
      2. 所需硬件：逻辑地址寄存器、页表寄存器、页表、物理地址寄存器、PCB中增加存放页表基址和页表长度的项（让进程完成逻辑地址到物理地址的转换）。
      3. 过程：
         1. 按页的大小分离出页号和偏移量，放入有效地址寄存器（逻辑地址寄存器）中
         2. 将页号与页表长度进行比较，若页号＞页表长度，越界中断
         3. 基于页表寄存器中页表的地址、以页号为索引，查找页表，得到该页的物理帧号。（页表始址+页号×页表项长度=该表项在页表中的位置）
         4. 将该物理帧号装入物理地址寄存器的高址部分
         5. 将有效地址寄存器中的偏移量直接复制到物理地址寄存器的低位部分，从而形成内存地址。
   4. 特点：
      1. 没有外碎片，每个内碎片不超过页大小。
      2. 一个程序不必连续存放。
      3. 程序全部装入内存。
   5. 分页硬件和TLB
      1. 页表的实现：
         1. 页表被保存在主存中，页表基址寄存器指向页表，页表限长寄存器表明页表的长度。
         2. 每次对数据/指令存取需要两次内存存取，一次是查页表，一次是访问存取数据。——通过一个联想寄存器（或TLB），可以解决两次存取的问题。
      2. TLB
         1. 用法：对数据/指令存取时，先查找TLB中有无页号到物理帧号的映射，若有则可以直接把逻辑地址转换为物理地址，若无则去内存查页表完成转换。
         2. 命中率：在TLB中找到页号（TLB命中）的比率，与TLB的大小有关。
2. 分段（——也是一种离散的分配方式）
   1. 含义：
      1. 支持用户观点的内存管理机制
      2. 一个程序是一些段的集合，一个段是一个逻辑单位
   2. 实现：
      1. 一个逻辑地址是两个向量的集合——段号、偏移
      2. 段表，映射二维物理地址，每个表项包括基址和限长。（基址base：内存中段物理地址的起始地址）（限长limit：指定段的长度）
   3. 地址转换：
      1. 所需硬件：段表基址寄存器（指向段表在内存中的地址）、段表限长寄存器（表明被一个程序所使用的段的数目）
      2. 过程：
         1. 系统将逻辑地址中的段号S与段表长度TL进行比较，若S≥TL，访问越界。
         2. 若未越界，根据段表的始址和该段的段号，计算出该段所对应段表项的地址，从中读出该段在内存中的起始地址。（段表始址＋段号×段表项长度=对应段表项地址）
         3. 检查段内地址（偏移量）d与该段段长SL，若d≥SL，越界中断。
         4. 若未越界，将该段的基址与段内地址相加，得到要访问的内存物理地址。