参考链接

https://www.cnblogs.com/peterYong/p/6556619.html
https://blog.csdn.net/hguisu/article/details/5713164

一、操作系统的内存管理相关概念

- 内存管理：操作系统对内存的划分和动态分配
  包括内存管理和虚拟内存管理。

    - 内存管理包括内存管理概念、交换与覆盖、连续分配管理方式和非连续分配管理方式（分页管理方式、分段管理方式、段页式管理方式）
    - 虚拟内存管理包括虚拟内存概念、请求分页管理方式、页面置换算法、页面分配策略、工作集和抖动。

- 内存管理的功能

  1、内存空间的分配与回收：由操作系统完成主存储器空间的分配和管理，使程序员摆脱存储分配的麻烦，提高编程效率。

  2、地址转换：在多道程序环境下，程序中的逻辑地址与内存中的物理地址不可能一致，因此存储管理必须提供地址变换功能，把逻辑地址转换成相应的物理地址。

  3、内存空间的扩充：利用虚拟存储技术或自动覆盖技术，从逻辑上扩充内存。

  4、存储保护：保证各道作业在各自的存储空间内运行，.互不干扰

- 程序装入和链接

  创建进程首先要将程序和数据装入内存。将用户源程序变为可在内存中执行的程序，通常需要以下几个步骤：

  1、编译：由编译程序将用户源代码编译成若干个目标模块。

  2、链接：由链接程序将编译后形成的一组目标模块，以及所需库函数链接在一起，形成一个完整的装入模块。

  3、装入：由装入程序将装入模块装入内存运行。

- 程序的链接有以下三种方式：

  1、静态链接：在程序运行之前，先将各目标模块及它们所需的库函数链接成一个完整的可执行程序，以后不再拆开。

  2、装入时动态链接：将用户源程序编译后所得到的一组目标模块，在装入内存时，采用**边装入边链接**的链接方式。

  3、运行时动态链接：对某些目标模块的链接，是在程序执行中**需要该目标模块时**，才对它进行的链接。其优点是便于修改和更新，便于实现对目标模块的共享。

- 内存的装入模块在装入内存时，同样有以下三种方式：

    - 绝对装入。在编译时，如果知道程序将驻留在内存的某个位置，编译程序将产生绝对地址的目标代码。绝对装入程序按照装入模块中的地址，将程序和数据装入内存。由于程序中的逻辑地址与实际内存地址完全相同，故不需对程序和数据的地址进行修改。

      > 绝对装入方式**只适用于单道程序环境**。另外，程序中所使用的绝对地址，可在编译或汇编时给出，也可由程序员直接赋予。而通常情况下在程序

> 中采用的是符号地址，编译或汇编时再转换为绝对地址。

- 可重定位装入。在**多道程序环境下**，多个目标模块的起始地址通常都是从0开始，程序中的其他地址都是相对于起始地址的,此时应采用可重定位装入方式。根据内存的当前情况，将装入模块装入到内存的适当位置。**装入时对目标程序中指令和数据的修改过程称为重定位**，地址变换通常是在装入时一次完成的，所以又称为静态重定位。

  > 静态重定位的特点是在一个作业装入内存时，必须分配其要求的全部内存空间，如果没有足够的内存，就不能装入该作业。此外，作业一旦进入内存后，在整个运行期间不能在内存中移动，也不能再申请内存空间。

- 动态运行时装入，也称为动态重定位，程序在内存中如果发生移动，就需要采用动态的装入方式。装入程序在把装入模块装入内存后，并不立即把装入模块中的相对地址转换为绝对地址，而是把这种地址转换推迟到程序真正要执行时才进行。因此，装入内存后的所有地址均为相对地址。这种方式需要一个重定位寄存器的支持。

  > 动态重定位的特点是可以将程序分配到不连续的存储区中；在程序运行之前可以只装入它的部分代码即可投入运行，然后在程序运行期间，根据需要动态申请分配内存；便于程序段的共享，可以向用户提供一个比存储空间大得多的地址空间。

- 逻辑地址空间与物理地址空间
- 逻辑地址
  编译后，每个目标模块都是从0号单元开始编址，称为该目标模块的相对地址（或逻辑地址)。
  当链接程序将各个模块链接成一个完整的可执行目标程序时，链接程序顺序依次按各个模块的相对地址构成统一的从0号单元开始编址的逻辑地址空间。用户程序和程序员只需知道逻辑地址，而内存管理的具体机制则是完全透明的，它们只有系统编程人员才会涉及。不同进程可以有相同的逻辑地址，因为这些相同的逻辑地址可以映射到主存的不同位置。
- 物理地址
  物理地址空间是指内存中物理单元的集合，它是地址转换的最终地址，**进程在运行时执行指令和访问数据最后都要通过物理地址从主存中存取。**当装入程序将可执行代码装入内存时，必须通过**地址转换**将**逻辑地址转换成物理地址**，这个过程称为地址**重定位。**

二、连续分配存储管理方式

连续分配是指为一个用户程序分配连续的内存空间。连续分配有单一连续存储管理和分区式储管理两种方式。

2.1 单一连续存储管理

在这种理方式中，内存被分为两个区域：系统区和用户区。

- 应用程序装入到用户区，可使用用户区全部空间。其特点是，最简单，适用于单用户、单任务的操作系统。CP / M和 DOS 2．0以下就是采用此种方式。
- 这种方式的最大优点就是易于管理。但也存在着一些问题和不足之处，例如对要求内存空间少的程序，造成内存浪费；程序全部装入，使得很少使用的程序部分也占用一定数量的内存。

2.2 分区式存储管理

为了支持多道程序系统和分时系统，支持多个程序并发执行，引入了分区式存储管理。分区式存储管理是把内存分为一些大小相等或不等的分区，操作系统占用其中一个分区，其余的分区由应用程序使用，每个应用程序占用一个或几个分区。分区式存储管理虽然可以支持并发，但难以进行内存分区的共享。

- 分区式存储管理引人了两个新的问题：内碎片和外碎片
  内碎片是占用分区内未被利用的空间，外碎片是占用分区之间难以利用的空闲分区(通常是小空闲分区)。
  为实现分区式存储管理，操作系统应维护的数据结构为分区表或分区链表。表中各表项一般包括每个分区的起始地址、大小及状态(是否已分配)。

> 分区式存储管理常采用的一项技术就是内存紧缩(compaction)。

固定分区(nxedpartitioning)

> 固定式分区的特点是把内存划分为若干个固定大小的连续分区。分区大小可以相等：这种作法只适合于多个相同程序的并发执行(处理多个类型相同的对象)。分区大小也可以不等：有多个小分区、适量的中等分区以及少量的大分区。根据程序的大小，分配当前空闲的、适当大小的分区。

- 优点：易于实现，开销小。
- 缺点主要有两个：内碎片造成浪费；分区总数固定，限制了并发执行的程序数目。

动态分区(dynamic partitioning)

动态分区的特点是动态创建分区：在装入程序时按其初始要求分配，或在其执行过程中通过系统调用进行分配或改变分区大小。
1、与固定分区相比较其优点是：没有内碎片。但它却引入了另一种碎片——外碎片。
2、动态分区的分区分配就是寻找某个空闲分区，其大小需大于或等于程序的要求。若是大于要求，则将该分区分割成两个分区，其中一个分区为要求的大小并标记为“占用”，而另一个分区为余下部分并标记为“空闲”。
3、分区分配的先后次序通常是从内存低端到高端。动态分区的分区释放过程中有一个要注意的问题是，将相邻的空闲分区合并成一个大的空闲分区。
几种常用的分区分配算法：
最先适配法、下次适配法、最佳适配法、最坏适配法

2.3 伙伴系统（解决外部碎片）

固定分区和动态分区方式都有不足之处。固定分区方式限制了活动进程的数目，当进程大小与空闲分区大小不匹配时，内存空间利用率很低。动态分区方式算法复杂，回收空闲分区时需要进行分区合并等，系统开销较大。伙伴系统方式是对以上两种内存方式的一种折衷方案。

- 伙伴系统规定，无论已分配分区或空闲分区，其大小均为 2 的 k 次幂，k 为整数， l≤k≤m，其中：
  2^1 表示分配的最小分区的大小， 2^m 表示分配的最大分区的大小（通常 2^m是整个可分配内存的大小）

假设系统的可利用空间容量为2^m个字， 则系统开始运行时， 整个内存区是一个大小为2^m的空闲分区。在系统运行过中， 由于**不断的划分**，可能会形成若干个不连续的空闲分区，将这些空闲分区根据分区的大小进行分类，对于每一类具有**相同大小的所有空闲分区**，单独设立一个空闲分区双向链表。这样，不同大小的空闲分区形成了k(0≤k≤m)个空闲分区链表。

- 在伙伴系统中，其分配和回收的时间性能 取决于 查找空闲分区的位置和分割、合并空闲分区所花费的时间。
  1、与前面所述的多种方法相比较，由于该算法在回收空闲分区时，需要对空闲分区进行合并，所以其时间性能比前面所述的分类搜索算法差，但比顺序搜索算法好，而其空间性能则远优于前面所述的分类搜索法，比顺序搜索法略差。
  2、需要指出的是，在当前的操作系统中，普遍采用的是下面将要讲述的基于分页和分段机制的虚拟内存机制，该机制较伙伴算法更为合理和高效，但在多处理机系统中，伙伴系统仍不失为一种有效的内存分配和释放的方法，得到了大量的

应用。

具体过程

- 假设要请求一个256 个页框的块（即1MB）。
  算法先在256 个页框的链表中检查是否有一个空闲块。如果没有这样的块，算法会查找下一个更大的页块，也就是，在512 个页框的链表中找一个空闲块。如果存在这样的块，内核就把256 的页框分成两等份，一半用作满足请求，另一半插入到256 个页框的链表中。如果在512 个页框的块链表中也没找到空闲块，就继续找更大的块 —— 1024个页框的块。如果这样的块存在，内核把1024个页框块的256 个页框用作请求，然后从剩余的768 个页框中拿512个插入到512个页框的链表中，再把最后的256个插入到256个页框的链表中。如果1024个页框的链表还是空的，算法就放弃并发出错信号。
- 内核试图把大小为b的一对空闲伙伴块合并为一个大小为2b的单独块。
  满足以下条件的两个块称为伙伴：
  两个块具有相同的大小，记作b。
  它们的物理地址是连续的。
  第一块的第一个页框的物理地址是2×b×212的倍数。
  该算法是迭代的，如果它成功合并所释放的块，它会试图合并2b 的块，以再次试图形成更大的块。

2.4 内存紧缩

内存紧缩：将各个占用分区向内存一端移动，然后将各个空闲分区合并成为一个空闲分区。

- 这种技术在提供了某种程度上的灵活性的同时，也存在着一些弊端
  例如：对占用分区进行内存数据搬移占用CPU时间；如果对占用分区中的程序进行“浮动”，则其重定位需要硬件支持。
- 紧缩时机：
  每个分区释放后，或内存分配找不到满足条件的空闲分区时。
- 堆结构的存储管理的分配算法：
  在动态存储过程中，不管哪个时刻，可利用空间都是-一个地址连续的存储区，在编译程序中称之为"堆"，每次分配都是从这个可利用空间中划出一块。
  其实现办法是：设立一个指針，称之为堆指针，始终指向堆的最低（或锻联）地址。当用户申请N个单位的存储块时，堆指针向高地址（或 低地址）称动N个存储单位，而移动之前的堆指针的值就是分配给用户的占用块的初始地址。

- 释放内存空间执行内存紧缩：
  回收用户释放的空闲块就比较麻烦.由于系统的可利用空间始终是一个绝址连续的存储块，因此回收时必须将所释放的空间块合并到整个堆上去才能重新使用，这就是"存储策缩"的任务。
  通常有两种做法：一种是一旦有用户释放存储块即进行回收紧缩； 另一种是在程序执行过程中不回收用户随时释放的存储块，直到可利用空同不够分配或堆指针指向最高地址时才进行存储紧缩。

三、覆盖和交换技术

3.1 覆盖技术

- 引入覆盖 (overlay)技术的目标是
  在较小的可用内存中运行较大的程序。这种技术常用于多道程序系统之中，与分区式存储管理配合使用。
- 覆盖技术的原理：
  一个程序的几个代码段或数据段，按照时间先后来占用公共的内存空间。将程序必要部分(常用功能)的代码和数据常驻内存；可选部分(不常用功能)平时存放在外存(覆盖文件)中，在需要时才装入内存。不存在调用关系的模块不必同时装入到内存，从而可以相互覆盖。
- 覆盖技术的缺点
  编程时必须划分程序模块和确定程序模块之间的覆盖关系，增加编程复杂度；从外存装入覆盖文件，以时间延长换取空间节省。

覆盖的实现方式有两种：以函数库方式实现或操作系统支持。

3.2 交换技术

- 交换 (swapping)技术
  在多个程序并发执行时，可以将暂时不能执行的程序（进程）送到外存中，从而获得空闲内存空间来装入新程序（进程），或读人保存在外存中而处于就绪状态的程序。交换单位为整个进程的地址空间。交换技术常用于多道程序系统或小型分时系统中，因为这些系统大多采用分区存储管理方式。与分区式存储管理配合使用又称作“对换”或“滚进 / 滚出” (roll-in / roll-out)。
- 原理：
  暂停执行内存中的进程，将整个进程的地址空间保存到外存的交换区中（换出 swap out），而将外存中由阻塞变为就绪的进程的地址空间读入到内存中，并将

该进程送到就绪队列（换入swap in）。

- 交换技术优点之一
  增加并发运行的程序数目，并给用户提供适当的响应时间；与覆盖技术相比交换技术另一个显著的优点是不影响程序结构。交换技术本身也存在着不足，例如：对换人和换出的控制增加处理器开销；程序整个地址空间都进行对换，没有考虑执行过程中地址访问的统计特性。
- 覆盖与交换比较
  1）与覆盖技术相比，交换不要求程序员给出程序段之间的覆盖结构。
  2）交换主要是在进程与作业之间进行，而覆盖则主要在同一作业或进程内进行。 另外覆盖只能覆盖那些与覆盖程序段无关的程序段。

四、页式存储管理

4.1 基本原理

将程序的逻辑地址空间划分为**固定大小的页(page)，而物理内存划分为同样大小的页框(page frame)**。程序加载时，可将任意一页放人内存中任意一个页框，这些页框不必连续，从而实现了离散分配。该方法需要CPU的硬件支持，来实现逻辑地址和物理地址之间的映射。在页式存储管理方式中地址结构由两部构成，前一部分是页号，后一部分为页内地址w（位移量）

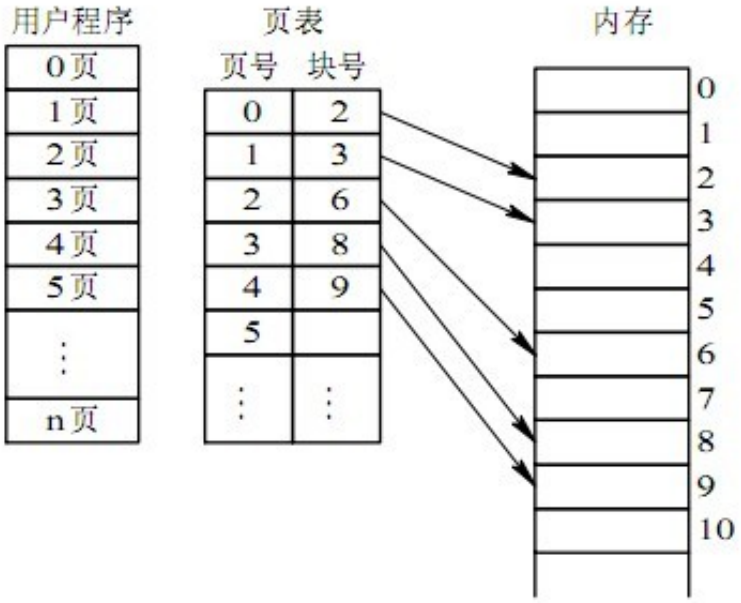
- 页式管理方式的优点是：
  1）没有外碎片，每个内碎片不超过页大。
  2）一个程序不必连续存放。
  3）便于改变程序占用空间的大小(主要指随着程序运行，动态生成的数据增多，所要求的地址空间相应增长)。
- 缺点是：
  要求程序全部装入内存，没有足够的内存，程序就不能执行。

4.2 页式管理的数据结构

在页式系统中进程建立时，操作系统为进程中所有的页分配页框。当进程撤销时收回所有分配给它的页框。
在程序的运行期间，如果允许进程动态地申请空间，操作系统还要为进程申请的空间分配物理页框。操作系统为了完成这些功能，必须记录系统内存中实际的页框使用情况。
操作系统还要在进程切换时，正确地切换两个不同的进程地址空间到物理内存空间的

映射。这就要求操作系统要记录每个进程页表的相关信息。为了完成上述的功能，一个页式系统中，一般要采用如下的数据结构。

- 进程页表：完成逻辑页号(本进程的地址空间)到物理页面号(实际内存空间，也叫块号)的映射。
  每个进程有一个页表，描述该进程占用的物理页面及逻辑排列顺序，如图：



- 物理页面表：整个系统有一个物理页面表，描述物理内存空间的分配使用状况，其数据结构可采用位示图和空闲页链表。
  对于位示图法，即如果该页面已被分配，则对应比特位置1，否置0.

19	18	17	16	15	…	4	3	3	2	0
0	1	1	1	1	…	1	1	0	1	1
0	0	0	1	1	…	0	0	1	1	0
0	0	1	1	1	…	0	0	0	0	0

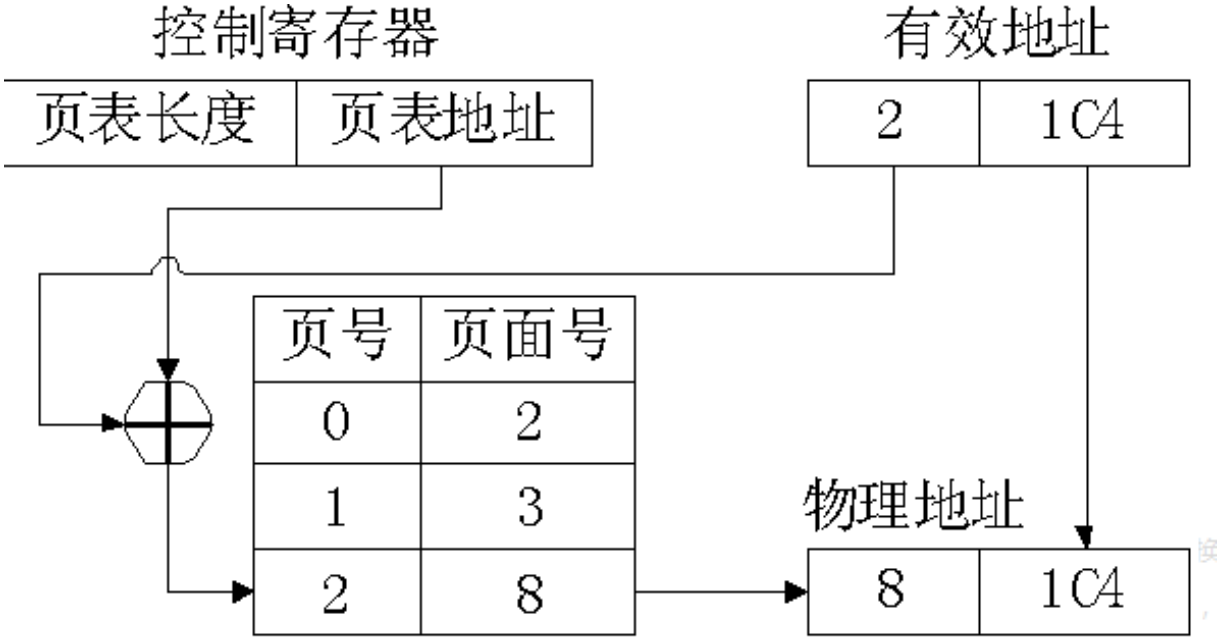
- 请求表：整个系统有一个请求表，描述系统内各个进程页表的位置和大小，用于地址转换也可以结合到各进程的PCB(进程控制块)里。如图：

进程号	请求页面数	页表始址	页表长度	状态
1	20	1024	20	已分配
2	34	1044	34	已分配
3	18	1078	18	已分配
4	21	⋮	⋮	未分配
⋮	⋮			⋮

4.3 页式管理地址变换

在页式系统中，指令所给出的地址分为两部分：逻辑页号和页内地址。

- 原理：
  CPU中的内存管理单元(MMU)按逻辑页号通过查进程页表得到物理页框号，将物理页框号与页内地址相加形成物理地址。
  逻辑页号，页内偏移地址 - >查进程页表，得物理页号 - >物理地址：



  上述过程通常由处理器的硬件直接完成，不需要软件参与。通常，操作系统只需在进程切换时，把进程页表的首地址装入处理器特定的寄存器中即可。一般来说，页表存储在主存之中。这样处理器每访问一个在内存中的操作数，就要访问两次内存：
  第一次用来查找页表将操作数的 逻辑地址变换为物理地址；
  第二次完成真正的读写操作。

这样做时间上耗费严重。为缩短查找时间，可以将页表从内存装入CPU内部的关联存储器(例如，快表) 中，实现按内容查找。此时的地址变换过程是：在CPU给出有效地址后，由地址变换机构自动将页号送人快表，并将此页号与快表中的所有页号进行比较，而且这 种比较是同时进行的。若其中有与此相匹配的页号，表示要访问的页的页表项在快表中。于是可直接读出该页所对应的物理页号，这样就无需访问内存中的页表。由于关联存储器的访问速度比内存的访问速度快得多。

五、段式存储管理

5.1 基本原理

在段式存储管理中，将程序的地址空间划分为若干个段(segment)，这样每个进程有一个二维的地址空间。

在前面所介绍的动态分区分配方式中，系统为整个进程分配一个连续的内存空间。而在段式存储管理系统中，则为每个段分配一个连续的分区，而进程中的各个段可以不连续地存放在内存的不同分区中。程序加载时，操作系统为所有段分配其所需内存，这些段不必连续，物理内存的管理采用动态分区的管理方法。

- 在为某个段分配物理内存时
  可以采用首先适配法、下次适配法、最佳适配法等方法。
- 在回收某个段所占用的空间时
  要注意将收回的空间与其相邻的空间合并。

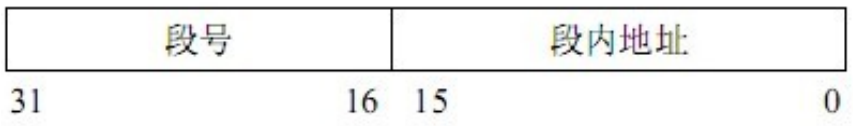
段式存储管理也需要硬件支持，实现逻辑地址到物理地址的映射。

- 程序通过分段划分为多个模块，如代码段、数据段、共享段：
  1、可以分别编写和编译
  2、可以针对不同类型的段采取不同的保护
  3、可以按段为单位来进行共享，包括通过动态链接进行代码共享
  这样做的优点是：可以分别编写和编译源程序的一个文件，并且可以针对不同类型的段采取不同的保护，也可以按段为单位来进行共享。
- 段式存储管理的优点是：
  没有内碎片，外碎片可以通过内存紧缩来消除；便于实现内存共享。缺点与页式存储管理的缺点相同，进程必须全部装入内存。

5.2 段式管理的数据结构

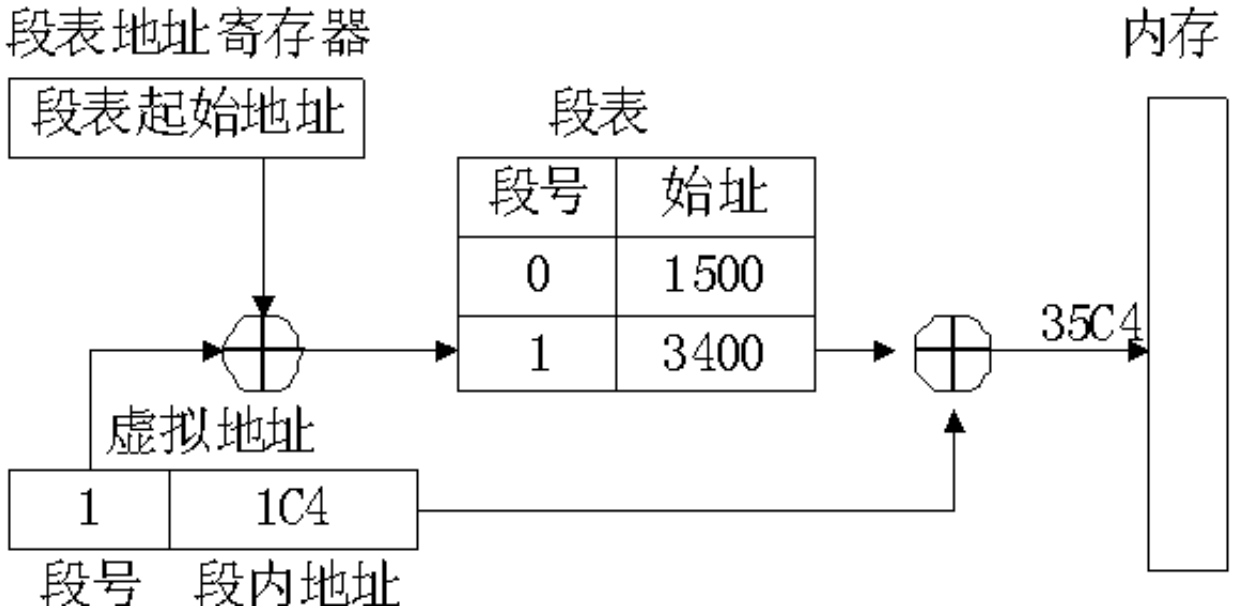
为了实现段式管理，操作系统需要如下的数据结构来实现进程的地址空间到物理内存空间的映射，并跟踪物理内存的使用情况，以便在装入新的段的时候，合理地分配内存空间。

- 进程段表：描述组成进程地址空间的各段，可以是指向系统段表中表项的索引。
  每段有段基址(baseaddress)，即段内地址。
  在系统中为每个进程建立一张段映射表，如图：



- 系统段表：系统所有占用段（已经分配的段）。
- 空闲段表：内存中所有空闲段，可以结合到系统段表中。

5.3 段式管理的地址变换



在段式 管理系统中，整个进程的地址空间是二维的，即其逻辑地址由段号和段内地址两部分组成。

为了完成进程逻辑地址到物理地址的映射，处理器会查找内存中的段表，由段号得到段的首地址，加上段内地址，得到实际的物理地址。这个过程也是由处理器的硬件直接完成的，操作系统只需在进程切换时，将进程段表的首地址装入处理器的特定寄存器当中。这个寄存器一般被称作段表地址寄存器。

六、页式和段式管理的区别

页式和段式系统有许多相似之处。比如，两者都采用离散分配方式，且都通过地址映射机构来实现地址变换。但概念上两者也有很多区别，主要表现在：

- 需求
  分页是信息的物理单位，是为了实现离散分配方式，以减少内存的碎片，提高内存的利用率。或者说，分页仅仅是由于系统管理的需要，而不是用户的需要。
  段是信息的逻辑单位，它含有一组其意义相对完整的信息。分段的目的是为了更好地满足用户的需要。一条指令或一个操作数可能会跨越两个页的分界处，而不会跨越两个段的分界处。
- 大小：
  页大小固定且由系统决定，把逻辑地址划分为页号和页内地址两部分，是由机器硬件实现的。
  段的长度不固定，且决定于用户所编写的程序，通常由编译系统在对源程序进行编译时根据信息的性质来划分。
- 逻辑地址表示：
  页式系统地址空间是一维的，即单一的线性地址空间，程序员只需利用一个标识符，即可表示一个地址。

分段的作业地址空间是二维的，程序员在标识一个地址时，既需给出段名，又需给出段内地址。

- 页大，因而段表比页表短，可以缩短查找时间，提高访问速度。

其他

用户态和内核态的理解和区别

- 参考链接 https://blog.csdn.net/qq_39823627/article/details/78736650

1、linux进程有4GB地址空间，如图所示：

3G-4G大部分是共享的，是内核态的地址空间。这里存放整个内核的代码和所有的内核模块以及内核所维护的数据。

2、特权级的概念：
对于任何操作系统来说，创建一个进程是核心功能。创建进程要做很多工作，会消耗很多物理资源。比如分配物理内存，父子进程拷贝信息，拷贝设置页目录页表等等，这些工作得由特定的进程去做，所以就有了特权级别的概念。最关键的工作必须交给特权级最高的进程去执行，这样可以做到集中管理，减少有限资源的访问和使用冲突。inter x86架构的cpu一共有四个级别，0-3级，0级特权级最高，3级特权级最低。

3、用户态和内核态的概念：

- 当一个进程在执行用户自己的代码时处于用户运行态（用户态），此时特权级最低，为3级，是普通的用户进程运行的特权级，大部分用户直接面对的程序都是运行在用户态。Ring3状态不能访问Ring0的地址空间，包括代码和数据；
- 当一个进程因为系统调用陷入内核代码中执行时处于内核运行态（内核态），此时特权级最高，为0级。执行的内核代码会使用当前进程的内核栈，每个进程都有自己的内核栈。

用户运行一个程序，该程序创建的进程开始时运行自己的代码，处于用户态。如果要执行文件操作、网络数据发送等操作必须通过write、send等系统调用，这些系统调用

会调用内核的代码。进程会切换到Ring0，然后进入3G-4G中的内核地址空间去执行内核代码来完成相应的操作。内核态的进程执行完后又会切换到Ring3，回到用户态。
这样，用户态的程序就不能随意操作内核地址空间，具有一定的安全保护作用。这说的保护模式是指通过内存页表操作等机制，保证进程间的地址空间不会互相冲突，一个进程的操作不会修改另一个进程地址空间中的数据。

4、用户态和内核态的切换
当在系统中执行一个程序时，大部分时间是运行在用户态下的，在其需要操作系统帮助完成一些用户态自己没有特权和能力完成的操作时就会切换到内核态。

- 用户态切换到内核态的3种方式
  （1）系统调用
  这是用户态进程主动要求切换到内核态的一种方式。用户态进程通过系统调用申请使用操作系统提供的服务程序完成工作。例如fork（）就是执行了一个创建新进程的系统调用。系统调用的机制和新是使用了操作系统为用户特别开放的一个中断来实现，如Linux的int 80h中断。
  （2）异常
  当cpu在执行运行在用户态下的程序时，发生了一些没有预知的异常，这时会触发由当前运行进程切换到处理此异常的内核相关进程中，也就是切换到了内核态，如缺页异常。
  （3）外围设备的中断
  当外围设备完成用户请求的操作后，会向CPU发出相应的中断信号，这时CPU会暂停执行下一条即将要执行的指令而转到与中断信号对应的处理程序去执行，如果前面执行的指令时用户态下的程序，那么转换的过程自然就会是 由用户态到内核态的切换。如硬盘读写操作完成，系统会切换到硬盘读写的中断处理程序中执行后边的操作等。

这三种方式是系统在运行时由用户态切换到内核态的最主要方式，其中系统调用可以认为是用户进程主动发起的，异常和外围设备中断则是被动的。从触发方式上看，切换方式都不一样，但从最终实际完成由用户态到内核态的切换操作来看，步骤一样的，都相当于执行了一个中断响应的过程。系统调用实际上最终是中断机制实现的，而异常和中断的处理机制基本一致。

5、用户态到内核态具体的切换步骤：
（1）从当前进程的描述符中提取其内核栈的ss0及esp0信息。
（2）使用ss0和esp0指向的内核栈将当前进程的cs,eip,eflags,ss,esp信息保存起来，这个过程也完成了由用户栈到内核栈的切换过程，同时保存了被暂停执行的程序的下

一条指令。

（3）将先前由中断向量检索得到的中断处理程序的cs,eip信息装入相应的寄存器，开始执行中断处理程序，这时就转到了内核态的程序执行了。