内存管理

存储器程序思想：指令和数据以同等地位存于存储器；指令和数据用二进制表示。存储器用来存放数据和程序。
主存储器的编制单元是字节，内存地址也叫物理地址。

内存性能指标：存储容量、单位成本，存储速度。
存储容量 = 存储字长*存储字数
每位价格 = 总成本/总容量
传输速率 = 数据带宽/存储周期

内存层次结构（由快到慢）：寄存储器，缓存，主存，磁盘，光盘，磁带。
其中缓存-主存解决速度问题，主存-辅存解决容量问题。

基本概念

实现内存的分配、回收、保护、扩充。

- **逻辑地址LA**：程序经过汇编或编译后，形成目标代码，每个目标代码都以0为机制顺序进行编址，原来用富豪名访问的单元用具体的数据*单元号*取代。这样生成的目标程序占据一定地址空间，称为作业的逻辑地址空间（逻辑地址）
- **物理地址PA**：内存由若干个存储单元组成，每个存储单元有一个唯一标识一个存储单元的编号，称为内存地址（物理地址） 把内存看成一个从0字节一直到内存最大容量诸子阶编号的存储单元数组（每个存储单元与内存地址的编号相对应）
- **内存映射**：逻辑地址 -> 物理地址
- **内存保护**：内存分配前，保护操作系统不受用户进程影响，同时保护用户进程不受其他用户进程的影响
    - *硬件实现*：在CPU内设置一对下限寄存器和上限寄存器，存放正在执行的程序在主存中的上限和下限地址。每当CPU访存，硬件自动将被访问的主存地址与节点寄存器进行比较，判断是否越界。若无越界，则按此地址访问主存，否则产生程序中断（存储保护中断
- **内存分配和回收**：程序执行时分配空间，不执行时则回收
- **内存扩充**：物理上对内存扩充（插内存条），软件上使用虚拟内存

程序的运行过程

编译->链接->装入->执行
编写程序代码，*编译*代码并产生目标模块，通过*链接*程序到*装入*模块，放入程序，最后到内存中*执行*
过程可分为四个部分：预处理器->编译器->汇编器->连接器

运行过程

1. 预处理：
    1. 处理所有注释以空格代替
    2. 将所有#define删除，并展开所有宏定义
    3. 处理条件编译指令#if, #ifdef, #dlif, #sles, #endif
    4. 处理#include，展开文件包含
    5. 保留编译器需要用#program指令
2. 编译：将经过预处理后的程序转换成特定汇编代码

3. 汇编：将汇编代码转为机器码，产生二进制的目标文件
4. 链接：将有关目标文件相连接，即将在一个文件中引用符号同该符号在另一个文件中定义连接起来，使所有这些目标文件能成为一个能让操作系统装入执行的统一整体

链接方式

静态链接：程序执行前，将若干0地址开始的目标模块及所需库函数链接为完整、从唯一“0”地址开始的装配模块。

装入时动态链接：边装入内存边链接个目标模块。

运行时动态装入：程序执行过程中需要目标模块时，才对其进行链接。

装入方式

绝对装入方式：编译程序产生的绝对地址的目标代码，程序中的逻辑地址与实际内存地址完全相同；在编译时，如果知道程序将放到内存中的位置，编译程序将产生绝对目标地址的代码。
装入程序按照装入模块中的地址，将程序和数据装入内存。只适用于单道程序环境。程序中使用的绝对地址，可在编译或汇编时给出，也可以由程序员直接赋予。通常情况下都是编译或汇编时再转换为绝对地址。

可重定位装入方式：多道程序环境下，装入一次性完成逻辑地址到物理地址的变换，并根据内存的当前情况将装入模块装入到内存的适当位置（又称静态重定位）。
静态重定位必须分配其要求的全部内存空间，如果没有足够内存则不能装入改作业。作业一旦进入内存后，运行期间不能再移动、也不能申请内存空间。

运行时动态装入：装入内存后所有地址仍为相对地址，地址变换在程序执行期间、随着对每条指令的访问自动进行（又称动态重定位）。
编译、链接后的装入模块地址都从0开始。装入程序把装入模块装入内存后并不立即把逻辑地址转为物理地址，装入内存后所有地址依然是逻辑地址（地址转换等到程序真正要执行时才进行）。

连续内存的管理方式

连续分配：单一连续分配，固定分区分配（内部碎片），动态分区分配。

- 动态分区分配
  - 外部碎片、紧凑
  - 分配策略：
    - 首次首映算法：地址递增
    - 最佳适应算法：容量递增
    - 最坏适应算法：容量递减
    - 邻近适应算法
  - 回收：回收块与相邻空闲块合并；空闲块再排序

内碎片：占用分区之内未被利用的空间；
外碎片：占用分区之间难以利用的空闲分区（通常是小空闲分区）

单一分配方式

把内存分为系统区（仅供OS使用，内存的低址部分）、用户区（系统去外全部内存空间）。最简单的存储管理。

实现简单；无外部碎片；可采用覆盖技术扩充内存；不一定需要采取内存保护。
但只用于单用户、单任务操作四天；有内部碎片；存储器利用率低；用户区在小作业时浪费。

固定分配方式

将用户控件华为若干个固定大小的分区，每个分区中只装入一道作业，形成一种最简单的可运行多道程序的内存管理方式。（有分区大小相等和不等两种形式）。

分配流程：检索分区使用表：

- 存在满足用户程序大小且未被分配的分区：
  - 用户程序装入满足条件的分区
  - 分区使用表：占用一个空白表项（用户程序大小，起始地址，分区状态：为已分配）
- 不存在满足条件的分区：拒绝分配内存

操作系统需要建立一个数据结构：分区说明表，来实现每个分区的分配与回收。每个表项对应一个分区，通常按分区大小排列。每个表项包括对应分区大小、起始地址、状态（是否已分配）。由内核态管理。

实现简单，无外部碎片。
但当用户程序太大时，可能所有分区都不能满足需求，此时不得不采用覆盖技术解决，又会降低性能；会产生内部碎片，内存利用率低。

动态区分分配方式

在进程装入内存时根据进程大小动态地创建分区，并使分区大小正好适合进程的需要。因此系统分区大小和数目是可变的，又称可变分区分配。

动态分配算法有：

- 首次适应法FF：顺序找到一满足的就分配，但可能存在浪费
  - 空闲分区一地址递增的次序链接，分配内存是顺序查找，找到大小能满足要求的第一个空闲分区。
  - 目的在于减少时间。
  - 空闲分区表（空闲区链）中的空闲分区要按地址由高到低排序。
- 循环首次适应发NF：类似哈希算法中左右交叉排序，满足就分配
  - 有首次适应算法演变。分配内存从时从上次查找结束的位置开始继续查找。
  - 空闲分区分布更均匀，查找开销小。
  - 从上次找到的空闲区的下一个空闲区开始查找，直到找到第一个能满足要求的空闲区位置，并从中划出一块与请求大小相等的内存空间分配给作业。
- 最佳适应算法BF：找到最合适的，但大区域访问次数少
  - 空闲分区按容量递增形成分区链，找到第一个能满足要求的空闲分区。
  - 能使外碎片尽量小。外部碎片较多。
  - 空闲分区表（空闲区链）中的空闲分区要按大小从小到大进行排序，自表头开始查找到第一个满足要求的自由分区分配。
- 最坏适应算法WF：相对于最好而言，找到最大的区域下手，导致最大区域可能很少，也造成许多碎片
  - 空闲分区以容量递减的次序链接，找到第一个能满足要求的空闲分区，也就是挑选出最大的分区。
  - 外部碎片较少，但可能没有足够多的空间留给未来的较大程序。
  - 空闲分区按由大到小排序。

连续分配算法总结：

动态分区	算法思想	性能分析	内部碎片	外部碎片
首次适应法	地址递增，满足大小的第一个分区	效果最好	无	有
最佳适应算法	空闲分区按容量递增，满足大小的第一个分区	效果最差、外部碎片最多	无	有
最坏适应算法	空闲分区一容量递减的次序链接，满足大小的第一个分区	缺少大分区	无	有
临近适应算法	地址递增，分配内存时从上次查找结束的位置开始继续查找	碎片分布整个区间	无	有

静态分区	算法思想	性能分析	内部碎片	外部碎片
单一连续分配	内存分为系统区和用户区	只能单道运行	有	无
固定分区分配	内存事先分为不同的区域	可能无法存放较大程序	有	无

基本分页内存管理方式

在连续存储管理方式中，固定分区会产生内部碎片，动态分区会产生外部碎片——对内存利用率都较低。内存中可能缺少大块连续空间。
分页存储管理将一个进程的逻辑地址空间分成若干个大小相等的片，称为页面或页，并为各页编号，从0开始。
把内存空间分成与页面相同大小的若干个存储块，称为(物理)块或页框frame，甲乙编号，从0#块、1#块等开始。

基本概念：

- 页面：将一个进程的逻辑地址空间分成若干大小相等的片
- 页框frame：内存空间分成与页面相同大小的存储块
- 页内碎片：进程最后一项经常装不满一块而形成的不可利用碎片
- 地址结构：页号P+位移量W(0-31)
- 页表：让进程能够在内存中找到一个页面所对应的物理块，系统为每个进程建立的页面映像表。实现从页面号到物理块号的地址映射

*页框、块、页块、帧、页帧、根、叶根*是一个东西

逻辑地址结构

在为进程分配内存时，一块为单位将进程中的若干个页分别汉族昂入到多个可以不相邻接的物理块中。
分页地址中的地址结构：`|页号P|位移量W|`，
eg.`|31--12|11--0|`：页面大小：`2^W B`，页面大小：`2^12 B`，页数：`2^P 个`，页面个数：`2^20 个页面`。

对于特定机器，其地址结构是一定的。若给定地址为A，页面大小L，页号P和页内地址d公式为：
`P = int(A/L)`，`d = A%L`。

页表

一个进程对应一张页表。进程的每个页面对应一个页表项；每个页表项“块号”和其他信息组成。每个页表项的长度相同。页表项大笑等于也狂所占的比特数。页表纪录进程页面和实际存放内存块之间的映射关系。页表本质上是一个大数组，页号是数组小标，页表项是数组元素，其大小是块号。

地址转化过程

基本的地址变换机构：要访问两次内存；页表大都驻留在内存中；为了实现地址变换功能，在系统中的设置页表寄存器PTR，用来存放页表的始址和页表长度。在进程未执行时，每个进程对应页表的始址和长度都存放在进程的PCB中，当该进程被调度时，就将他们装入页表寄存器。

地址转换的过程如下：

1. 程序执行时，从PCB去除页表的始址和页表常茹，装入页表寄存器PTR
2. 由分页地址变换机构将逻辑地址自动分成页号和页内地址
3. 将页号与页表长度进行比较，若页号≥页表长度，表示本地访问的地址已经超过进程的地址空间，产生越界中断
4. 将页表始址与页号和页表项长度的乘积相加，便得到该页表项中的位置
5. 取出页描述子得到该页物理块号
6. 对该页的存取控制进行检查
7. 将物理块号送入物理地址寄存器中，再将有效地址寄存器中的业内地址直接送入物理地址寄存器的快内地址字段中，拼接得到实际的物理地址

处理器每访问一个内存中的操作数，就要访问两次内存：第一次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。

具有快表的地址转化过程

为了缩短查找时间，将页表从内存装入CPU内部的关联存储器（例如快表），实现按内容查找。
此时的转化过程：CPU给出有效地址后，由地址变换机构自动将页号送入快表，并将此页号与快表中的所有页号进行比较。若有相匹配的页号，表示要访问的页的页表项在快表中。于是可以直接读出该页对应的物理页号，无需访问内存中的页表。（访存次数降低为一次）

二级页表

现代大多数计算机系统都支持非常大的逻辑地址空间（2^32~2^64），页表就变得非常大，并且要求是连续的。于是有两种解决方式：
1，离散分配方式；
2，只将当前需要的部分页表项调入内存，其余页表仍驻留在磁盘上，需要时再调入。

二级页表的逻辑地址结构可描述为：
|外层页号|内层页号|页内地址|
由外部页表去查找内部页表，内部页表的每一项都会对应内存空间中的一块。
（内存的管理单位为块，外部页表与内部页表存储的是块号，任意一级页表大小不得超过一页(块)大小）。
访存三次：页目录表、二级页表、存储单元。

多级页表

对于64位机器，采用二级页表已经不够用了。将外层页再进行分页，将各分页离散地装入不相邻接的物理块，再利用第二级的外层页表来映射它们之间的关系。逻辑地址结构可表述为：

|1级页表|2级页表|3级页表|4级页表|5级页表|页内地址|
访问内存次数=N+1次。

基本分段内存管理方式

分段存储管理方式是为了满足用户和程序员*方便编程*、*信息共享*、*信息保护*、*动态增长*、*动态链接*的需求而引入。

分段的地质结构

|段号|段内地址|
作业的地址空间悲哈氛围若干段，每个段都是一组完整的逻辑信息，每个段都有自己的名字，都是从零开始编址的一段连续地址空间，各段长度不等。
内存悲动态地划分为若干个长度不相同的区域，称为物理段，每个物理段由起始地址和长度确定。

段表

作业空间指向段表对应内容，段表的每一项都指向一个内存空间对应段。

地址变化过程

1. 程序执行时从PCB取出段表始址和长度并装入段表寄存器
2. 地址变换机构将逻辑地址自动分成段号和段内地址
3. 将段号和段表长度比较，若段号≥段表长度，表示本次访问的地址超越进程的地址空间，产生越界中断
4. 取出段描述子得到该段的段表始址和段长
5. 将段内地址和段长比较
    - 若段内地址≥段长，表示本次访问的地址超越进程的地址空间，产生越界中断
6. 将段表始址和段内地址相加得到物理地址

分段和分页的区别：
页是信息的物理单位。页的大小固定且由系统固定。分页和用户程序地址空间是一维的。段通常比页大，段表比页表短，可以缩短查找时间，提高访问速度；分页是系统管理的需要，分段是用户应用的需要。一条指令或一个操作数可能会跨越两个页的分解诶很粗，而不会跨越两个段的分界处。

虚拟存储器管理方式

具有请求调入功能和置换功能，从逻辑上对内存容量扩充的一种存储器系统。逻辑容量由内存和外存容量之和决定。

- 特点：
    - 离散性，多次性，对换性，虚拟性
- 优点：
    - 大程序：可在较小的可用内存中执行较大程序
    - 大用户空间：提供给用户可用的虚拟内存空间通常大于物理内存
    - 并发：可在内存中容纳更多程序并发执行
    - 易于开发
    - 以CPU时间和外存空间换区昂贵的内存空间
- 实现方式：
    - 硬件，请求分页中的内存分配，物理块分配算法，页面调入策略

常规存储器的管理方式特征：
驻留性：作业被装入内存就会一直驻留在内存中知道运行结束。但实际上只要访问作业的一小部分数据就能正常运行。
时间局部性：程序中存在大量循环操作，使得指令和数据被多次访问。
空间局部性：程序在一段时间内访问的地址可能集中在一定范围内（多见于程序顺序执行和数组）

请求分页管理方式

请求分页的代表机制，重要机构，地址变换

1. 硬件支持：
    1. 页表机制：纯分页的页表机制上增加若干项而形成作为请求分页的数据结构
    2. 缺页中断机构：产生缺页中断时请求OS将所缺页调入内存
    3. 地址变换机构：纯分页地址变换机构的基础上发展形成
2. 请求分页的软件

页表机制

结构如下：
|页号|物理块号|状态位P|访问字段A|修改为M|外存地址|

页号：虚拟地址空间中的页号；
物理块号：该页所占内存的块号；
P：1表示在内存中，0反之且会发生缺页中断信号；
A：系统规定时间内该页是否被引用过；
M：1表示被修改过，0表示未被修改过；
外存地址：该页内容在辅存中的地址，缺页时将所缺页调入内存。

缺页中断机构

1. 处理过程：
    1. 根据当前执行指令中的虚拟地址，形成(页号,页内偏移)，用页号查页表，判断该页是否在内存中
    2. 若缺页中断位为0（页面不在内存），产生缺页中断，让OS中断处理程序进行中断处理
    3. 查询存储分块表，寻找空闲内存块；查询页表得到该页在辅存中的地址，启动磁盘读信息
    4. 把磁盘上读出的信息装入分配的内存块
    5. 根据分配存储块的信息，修改页表、存储分块表汇总相应表目的信息
    6. 完成所需页装入工作后，*返回原指令*重新执行
2. **缺页中断和一般中断的区别【重点】**：
    1. 缺页中断在执行一条指令中产生，并且立即处理；一般中断是在指令执行完毕后响应和处理
    2. 缺页中断处理执行完后，仍返回到原指令处重新执行；一般中断则返回下一条指令执行
    3. 一条指令执行过程中可能会发生多次中断

地址变换机构

将虚拟地址变换为物理地址。流程如下：

1. 请求调页：查到页表项时进行判断
2. 页面置换：需要调入页面，但没有空闲内存块时进行
3. 需要修改请求页表中新增的表项

逻辑地址LA -> PA 映射情况的可能形式：

1. TLB+内存
2. TLB+页表+内存
3. TLB+页表+中断+TLB+内存

页面置换算法

按照某种机制把一部分页面淘汰出去，把虚拟内存中的页面加载出来。

-	**算法规则**	**优缺点**
OPT	有限淘汰最长时间内不会被访问的页	缺页率最小，性能最好； 无法实现
FIFO	优先淘汰最先进入内存的页	实现简单；性能差，可能 出现Belady异常
LRU	有限淘汰最近最久未访问的页	性能好；但要硬件支持， 算法开销大
简单 CLOCK	循环扫描，淘汰访问位为0的，并将扫描过的页面置为1.第一轮未选 中则进入第二轮	实现简单，算法开销小； 未考虑页面是否被修改过
改进 CLOCK	引入(访问位,修改位)。第一轮淘汰(0,0)；第二轮淘汰(0,1)，置扫描 过页面的访问位为0；第三轮淘汰(0,0)；第四轮淘汰(0,1)	算法开销小，性能较好

最佳置换算法OPT

其所淘汰的页面僵尸以后永不使用、或许是在最长未来时间内不再被访问的页面。理论上可保证获得最低的缺页率，但实际上是无法实现的。
只有在进程执行过程中才能知道未来会访问到那个页面，OS无法提前预判页面访问序列。
不过可以以此为标准评估其它算法。

先进先出算法FIFO

其所淘汰页面是最早进入内存的页面。
把内存页面根据调入先后顺序排成一个队列，选择队头页面换出。队列的最大长度取决于OS为进程分配的内存块数。

Belady奇异现象：采用FIFO时，若对一个进程未分配它所需的全部页面，有时会出现分配页面数增多，缺页率反而提高的异常现象。
因为FIFO未遵循程序局部性原理，完全以进入时间为基准来判断。

最近最久未使用置换算法LRU

其所淘汰页面是最近最久未使用的页面。
赋予每个页面的对应页表项中，用访问字段纪录该页面自上次被访问以来所经历时间t。选择现有页面中t最大的淘汰（此页面即为最近最久未使用的页面）。

为了记录进程在内存中各页的使用情况，需要为每个在内存中的页面配置一个位移寄存器。
效率高，开销大。

简单CLOCK算法

为每个页面设置一个访问位A，再将内存中的页面通过链接指针结成一个循环队列。被访问页的访问位A置为1。
淘汰位时检查访问位A：如果为0，该页换出；如果为1，置为0，暂不换出，继续检查下一页面。第一轮扫描中若都为1，则进行第二轮扫描。淘汰页面时最多经过两轮扫描。

但当缓冲池被填满且访问1时就可以直接命中缓存返回。简单CLOCK无法展现这一点。

改进型CLOCK算法

在简单CLOCK基础上引入修改位M。于是可组成一下四类页面：

1. `A=0, M=0`：未被访问、未被修改，最佳淘汰页
2. `A=0, M=1`：未被访问、已被修改，次佳淘汰页
3. `A=1, M=0`：已被访问、未被修改，可能再被访问
4. `A=1, M=1`：已被访问、已被修改，可能在被访问

算法规则：将所有可能被置换的页面排成循环队列。
第一轮从当前位置扫描到第一个(0,0)帧用于替换，不修改任何标志位；
若第一轮失败则进入第二轮，扫描到第一个(0,1)帧用于替换，将所有扫描过的访问位A置为0；
若第二轮扫描失败则进入第三轮扫描到第一个(0,0)用于替换，不修改任何标志位；
若第三轮失败则进入第四轮扫描到第一个(0,1)帧用于替换。
由于第二轮将所有访问位设为0，则在第三、第四轮必定会有一个帧被选中，因此淘汰页时最多会经过四轮扫描。

两个重要概念

1. **系统抖动/颠簸**：请求分页存储管理中，从驻村刚一走一个页面，根据请求又立刻又调入该页，出现反复调入调出的颠簸现象。
    - 若一个进程在换页上用的时间多余执行时间，则发生颠簸
    - 解决方案：增加工作集大小；选择不同的淘汰算法尽量使得工作集页面在内存中
2. **工作集/驻留集**：某段时间间隔内，进程要访问的页面集合。经常被使用到的页面要在工作集中，而长期不背使用的页面要从工作集中丢弃。为了防止系统抖动，需要选择合适的工作集大小。
    - 工作集会根据窗口尺寸算出
    - 只要划分合理的工作集，就不会发生抖动

页面的分配和调入

最小物理块数确定

- 能保证进程正常运行所需的最小物理块数
- 当OS为进程分配的物理块数少于此数值，进程无法运行
- 进程应获取的最小物理块数与计算机硬件结构有关，取决于指令格式、功能、寻址方式

物理块的分配策略

- 固定分配：OS为每个进程分配一组固定数目的物理块，在进程运行期间不再改变（驻留集不变）
- 可变分配：先给每个进程分配一定数目物理块，运行期间可视情况增减（驻留集大小可变）
- 局部变换：发生缺页时只选进程自己的物理块进行置换

- 全局置换：可将操作系统保留的空闲物理块分配给缺页进程，也可将别的进程持有的物理块置换到外存并在分配给缺页进程

在请求分页OS中，可采取固定和可变分配策略。在进行置换是，可采取全局置换和局部置换

1. 固定分配局部置换
    - OS为进程分配一定数量物理块，运行期间不变，发生缺页从内存中的页面换出
    - 缺点：很难在刚开始确定分配数量是否合理
2. 可变分配全局置换
    - 动态分配，物理块用完时可置换其它进程页
    - 当系统空间被分配完，可在整个空间中随意置换
3. 可变分配局部变换
    - 置换自身进程页，频繁缺页时动态分配

何时调入页面

1. 预调页策略：进程开始时，根据局部性原理，一次调入若干个相邻页面。常用于首次调入，由程序员指出先调入部分
2. 请求调页策略：进程运行期间发现缺页时将所缺部分调入内存。但IO开销较大

从何处调入页面

请求分页系统中的外存分为*存放文件的**文件区***和*用于存放对换页面的**对换区***。文件区离散分配，对换区连续分配。

1. 当系统中有足够对换区空间：将全部从对换区调入所需页面。进程运行前将与该进程有关的文件从文件区拷贝到对换区
2. 缺少足够对换区空间：不会被修改的文件直接从文件区调入；换出页面时，由于未被修改而不必将它们换出，以后调入扔从文件区调入。可能被修改的部分在将其换出时需调到对换区
3. UNIX方式：访问过的都在对换区，未访问过的都在文件区

页面调入过程

- 当程序所访问页未在内存，向CPU发出缺页中断
- 程序查找页表得到物理块后，若此时内存能融安新野，则启动磁盘IO所缺页调入内存，然后修改页表
- 如果内存已满，按某种置换算法从内存中选出一页换出：
    - IF未被修改，不必协会磁盘
    - IF已被修改，协会磁盘
- 缺页调入内存后，利用修改后的页表形成要访问的数据物理地址，再访问内存

补充

- 将逻辑地址转变为内存的物理地址的过程称作：重定位（静态重定位、动态重定位）/装载/装入
- 虚拟内存管理中，地址变换机构将逻辑地址变换为物理地址，形成该逻辑地址的阶段是：编译
    - 编译 形成逻辑地址
    - 装载 形成物理地址
- 起始地址为0的目标模块A、B、C，长度依次为L、M、N，采用静态链接方式链接在一起后，模块C的起始地址为L+M
- 内存保护由硬件机构完成，保证进程空间不被非法访问

- 页式存储管理系统中，若页大小4KB，页号0对应块号2，地址转换机构将逻辑地址0转换为物理为：页号0，DIW 4KB = 0，页内地址 0%4KB=0，`4KB*2=4096*2=8192`
- 若二级页表的虚拟地址结构为：`|10位|10位|12位|`。页大小=2^12B=4KB，页大小=页框大小，页数=2^20。假设页目录项和页表项占4字节，总空间大小为`1024*4B+(1024*4B0*1024)`，页数=`总空间大小/(1024*4B)=1025页`。
- 若逻辑地址为(0,137)，对应段表内容`|段号0|内存始址50K|段长10KB|`，其物理地址为`50K+137`。若逻辑地址为(3,4000)，对应段表内容`|段号3|内存始址80K|段长1KB|`，其会产生越界
- 每页1KB，表示页内地址10位；若2页对应物理块号位4，对于逻辑地址`0A5C`，将其转为二级制`0000 1010 0101 1100`，保留`10 0101 1100`，`0000 10`为2页，对应物理块号5，所以其物理地址为`100 10 0101 1100`