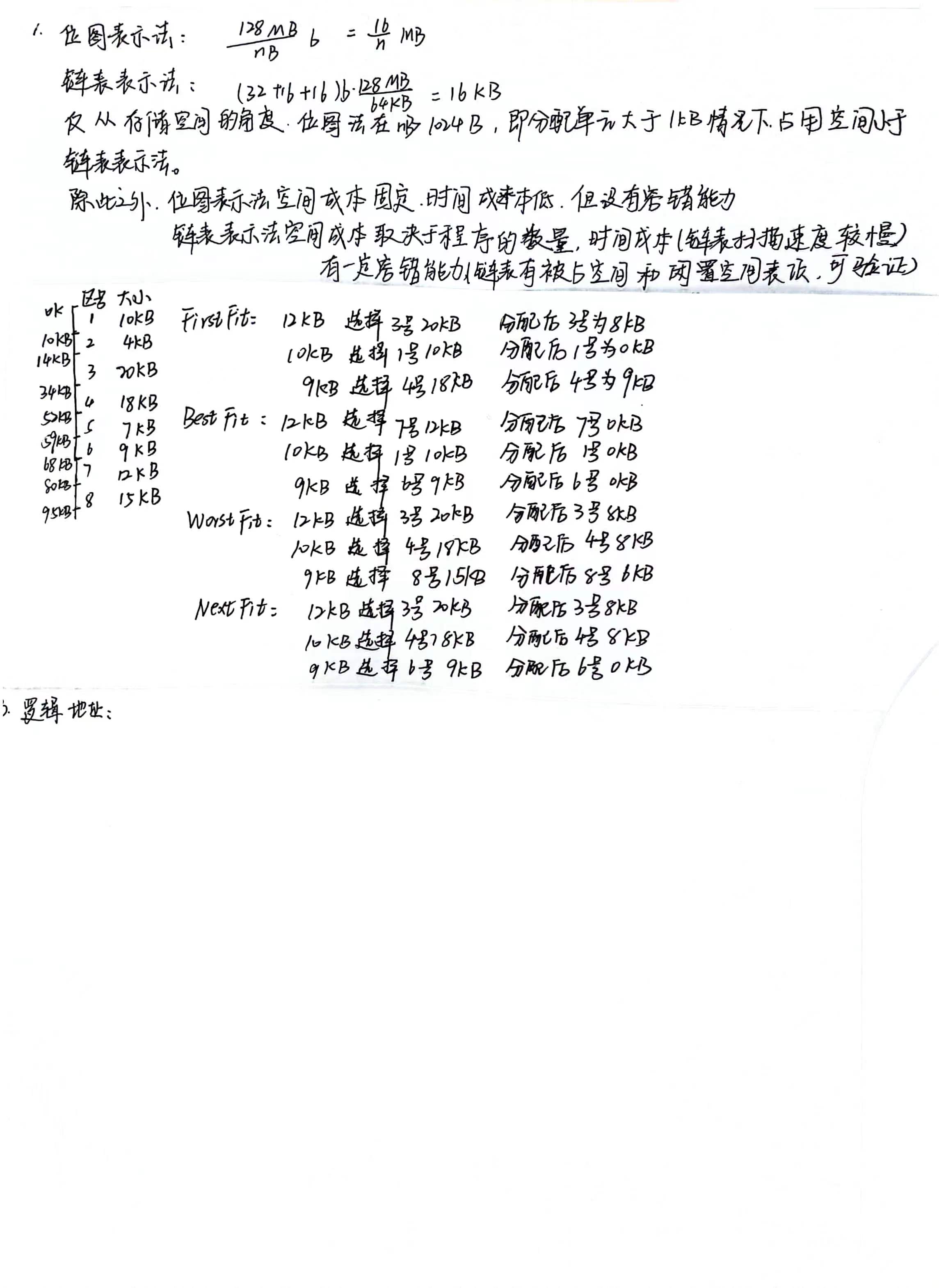
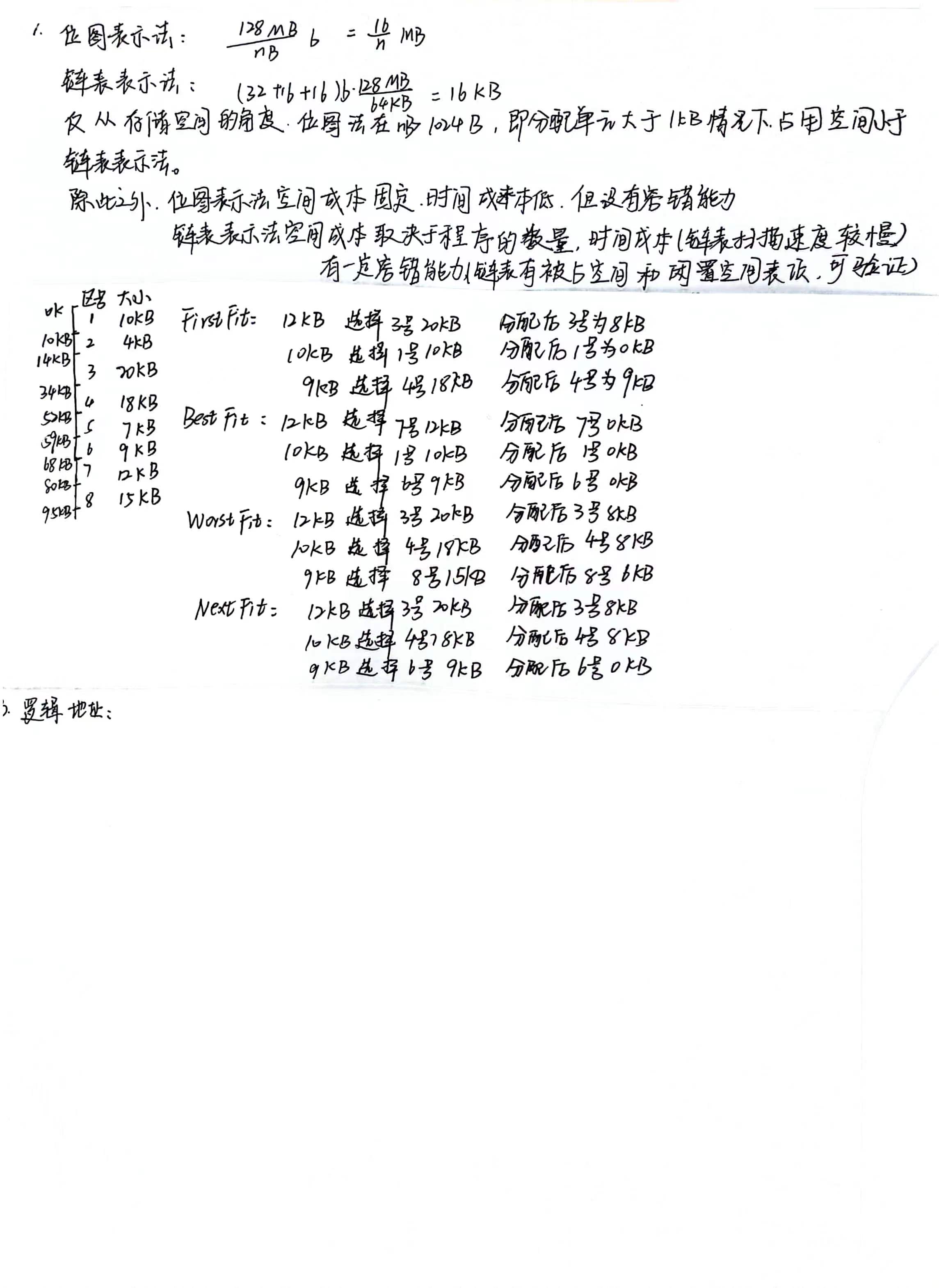
1. 动态内存分配需要对内存分区进行管理，一般使用位图和空闲链表两种方法。128MB的内存以n字节为单元分配，对于链表，假设内存中数据段和空闲区交替排列，长度均为64KB。并假设链表中的每个节点需要记录32位的内存地址信息、 16位长度信息和16位下一节点域信息。这两种方法分别需要多少字节的存储空间？ 哪种方法更好？



1. 在一个交换系统中，按内存地址排列的空闲区大小是: 10KB、 4KB、 20KB、18KB、 7KB、9KB、 12KB和15KB。对于连续的段请求： 12KB、 10KB、 9KB。使用FirstFit、 BestFit、WorstFit和NextFit将找出哪些空闲区？



3． 解释逻辑地址、物理地址、地址映射，并举例说明。

逻辑地址: 操作系统的用户（程序）编写应用程序时所用的地址。

物理地址：内存中实际的地址。

地址映射： 逻辑地址向物理地址转换的过程。

举例说明：lab2实验中内核虚拟地址与物理地址的地址映射如下：



1. 解释页式（段式）存储管理中为什么要设置页（段）表和快表，简述页式（段式）地址转换过程。
   1. **设置页表：**

把一个逻辑地址连续的的程序分散存放到若干不连续的内存区域内，并保证程序的正确执行，则既可充分利用内存空间，又可减少移动带来的开销。

* 1. **设置段表：**

方便编程：

– 通常一个作业是由多个程序段和数据段组成的，用户一般按逻辑关系对作业分段，并能根据名字来访问程序段和数据段。

信息共享：

– 共享是以信息的逻辑单位为基础的。页是存储信息的物理单位，段却是信息的逻辑单位。页式管理中地址空间是一维的，主程序，子程序都顺序排列，共享公用子程序比较困难，一个共享过程可能需要几十个页面。

信息保护：

– 页式管理中，一个页面中可能装有 2 个不同的子程序段的指令代码，不能通过页面共享实现共享一个逻辑上完整的子程序或数据块。段式管理中，可以以信息的逻辑单位进行保护。

动态增长：

– 实际应用中，某些段（数据段）会不断增长，前面的存储管理方法均难以实现。

动态链接：

– 动态链接在程序运行时才把主程序和要用到的目标程序（程序段）链接起来。

* 1. **设置快表：**

页表机制带来的严重问题就是内存访问效率的严重下降，以二级分页地址机制为例，访存次数由不分页时的 1 次，上升到了 3 次。

为了提高地址转换效率， CPU内部增加了一个硬件单元，称为存储管理单元MMU（Memory Management Unit）。其内部主要部件：页表Cache、TLB控制单元、页表（遍历）查找单元。

快表（TLB）是一种特殊的高速缓冲存储器（Cache），内容是页表中的一部分或全部内容。CPU 产生逻辑地址的页号，首先在快表中寻找，若命中就找出其对应的物理块；若未命中，再到页表中找其对应的物理块，并将之复制到快表。若快表中内容满，则按某种算法淘汰某些页。

* 1. **地址转换过程：**

**（页式）**

当进程要访问某个逻辑地址中的数据时，分页地址变换机构会自动地将逻辑地址分为页号和页内地址两部分。

• 将页号与页表长度进行比较，如果页号大于或等于页表长度，则表示本次所访问的地址已超越进程的地址空间，产生地址越界中断。 （越界保护）

• 将页表始址与页号和页表项长度的乘积相加，得到该表项在页表中的位置，于是可从中得到该页的物理块号，将之装入物理地址寄存器中。 （地址变换）

• 将有效地址寄存器中的页内地址送入物理地址寄存器的块内地址字段中。

**（段式）**

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，表示段号太大，是访问越界，于是产生越界中断信号。

– 若未越界，则根据段表的始址和该段的段号，计算出该段对应段表项的位置，从中读出该段在内存的始址。

2. 再检查段内地址 d，是否超过该段的段长 SL。

– 若超过，即 d >SL，同样发出越界中断信号。

– 若未越界，则将该段的基址与段内地址 d 相加，即可得到要访问的内存物理地址。

1. 叙述缺页中断的处理流程。

当进程执行过程中需访问的页面不在物理存储器中时，会引发发生缺页中断，进行所需页面换入，步骤如下：

1. 陷入内核态，保存必要的信息（OS及用户进程状态相关的信息）。 （现场保护）

2. 查找出来发生页面中断的虚拟页面（进程地址空间中的页面）。这个虚拟页面的信息通常会保存在一个硬件寄存器中，如果没有的话，操作系统必须检索程序计数器，取出这条指令，用软件分析该指令，通过分析找出发生页面中断的虚拟页面。 （页面定位）

3. 检查虚拟地址的有效性及安全保护位。如果发生保护错误，则杀死该进程。 （权限检查）

4. 查找一个空闲的页框(物理内存中的页面)，如果没有空闲页框则需要通过页面置换算法找到一个需要换出的页框。 （新页面调入（1））

5. 如果找的页框中的内容被修改了，则需要将修改的内容保存到磁盘上¹。（注：此时需要将页框置为忙状态，以防页框被其它进程抢占掉） （旧页面写回）

6. 页框“干净”后，操作系统将保存在磁盘上的页面内容复制到该页框中²。（新页面调入（2））

7. 当磁盘中的页面内容全部装入页框后，向操作系统发送一个中断。操作系统更新内存中的页表项，将虚拟页面映射的页框号更新为写入的页框，并将页框标记为正常状态。 （更新页表）

8. 恢复缺页中断发生前的状态，将程序指针重新指向引起缺页中断的指令。 （恢复现场）

9. 程序重新执行引发缺页中断的指令，进行存储访问。 （继续执行）

1. 假设一个机器有38位的虚拟地址和32位的物理地址。

(1) 与一级页表相比，多级页表的主要优点是什么？

一级页表的问题：若逻辑地址空间很大 (2^32 ∼2^64 ) ，则划分的页比较多，页表就很大，占用的存储空间大（要求连续），实现较困难。

(2) 如果使用二级页表，页面大小为16KB，每个页表项有4个字节。应该为虚拟地址中的第一级和第二级页表域各分配多少位？

一页页表的页表项数目：16KB / 4B = 2^12(页内偏移量 2^14)

页表数目：2^38B / 16KB = 2^24

第一级页表域12位，第二级页表域12位

1. 假设页面的访问存在一定的周期性循环，但周期之间会随机出现一些页面的访问。例如： 0,1,2…,511,431,0,1,2…511,332,0,1,2,…,511等。请思考：
2. LRU、 FIFO和Clock算法的效果如何？

每一次循环有512个可能得页面需要装载，因此若物理页框数大于512，那么不管哪种算法都不会产生缺页中断，LUR、FIFO和Clock算法得效果是相同的（事实上他们都没有应用过）。

如果物理页框数不足512个，那么三种算法的效果也是几乎等同的。在首先装满所有物理页框之后，对于FIFO而言，在需要淘汰物理页的时候会从首先装入，即周期性循环中最靠前的物理页开始淘汰，不及该循环中的物理页再次出现，它一定会被淘汰，从而FIFO在应对这种情况的循环时，除了那个随机出现的页面之外，所有页面都会缺页。对于Clock算法而言情况也并不会太好，Clock在FIFO的原则之上给页面“第二次机会”，即如果一个页面装入后被再次访问过，就“再给它一次生存的机会”，但在上述情况中等待被淘汰的物理页大概率均不会有装入内存后的第二次访问机会，随机出现的页面可能会有再次访问从而享受到Clock的“庇护”，但大部分页面大概率都会缺页。对于LRU而言，它的效果也和前两者几乎等同，这是因为LRU淘汰最久没有用到的页面，在题目所述的循环中最久不用的页面和最先到来的页面几乎是等价的，因此LRU也会面临几乎所有页面都缺页的窘境。

综上所述，面临题目所述的周期性页面访问时，LRU、FIFO和Clock的效果几乎一致，在页框足够时不会产生任何缺页，但在页框不足时几乎每次访问都会缺页，缺页率接近100%。

1. 如果有500个页框，能否设计一个优于LRU、 FIFO和Clock的算法？

针对周期性页面访问，在物理页框不足的情况下，使用与FIFO先进先出完全相对的“FILO 先进后出”（或后进先出）算法，其性能会大大优于LRU、FIFO和Clock。在最初的500个物理页被占满之后，到来的第500号虚拟页会替换掉499号，第501号又会替换500，等等，在此第500-511号虚拟页会全部缺页，但在新的循环到来时，前499个虚拟页会全部命中，夹在两次循环之间的随机页也会有500/512的概率命中，缺页率降至约为12/512=2.34%，远远优于LRU、FIFO和Clock。

1. 一个交换系统通过紧缩技术来清理碎片。如果内存碎片和数据区域是随机分配的。而且假设读写32位内存字需要10nsec. 那么如果紧缩128MB的内存需要多久？简单起见，假设第0个字是碎片的一部分而最高位的字包含了有效的数据。

内存字的数量为：128 \* 2^20B / 4B = 32 \* 2^20个内存字。对每个内存字既需要读也需要写，需要2 \* 32 \* 2^20 \* 10 nsec 约等于 0.64s