

《操作系统》lab4

|  |  |
| --- | --- |
| 学 院： | 计算机与信息技术学院 |
| 专 业： | 计算机科学与技术 |
| 班 级： | 计科1602 |
| 姓 名： | 麻锦涛 |
| 学 号： | 16281262 |

**日期：2019.5.25**

1. **实验目的**

该项目的目标是为连续的内存分配实践不同的算法

1. **实验要求**

在我们的课本中，我们为连续的内存分配提出了不同的算法。该项目将涉及管理大小为MAX的连续内存区域，其中地址范围为0 ... MAX - 1.您的程序必须响应四个不同的请求：

1.请求连续的内存块（分配内存）

2.释放连续的内存块（回收内存）

3.将未使用的未使用的存储器孔压缩成一个块（紧缩内存）

4.报告空闲和分配内存的区域（查看内存状态）

您的程序将在启动时传递初始内存量。

例如，以下内容使用1 MB（1,048,576字节）的内存初始化程序：

./allocator 1048576

程序启动后，将向用户显示以下提示：

allocator>

然后它将响应以下命令：RQ（请求），RL（释放），C（压缩），STAT（状态报告）和X（退出）。

对40,000字节的请求将显示如下：

allocator>RQ P0 40000 WRQ

命令的第一个参数是需要内存的新进程，后跟请求的内存量，最后是策略。 （在这种情况下，“W”指的是最差的。）

同样，发布将显示为：

allocator> RL P0

此命令将释放已分配给进程P0的内存。压缩命令输入为：

allocator>Ç

此命令将未使用的内存孔压缩到一个区域中。最后，用于报告内存状态的STAT命令输入为：

allocator>STAT

根据此命令，您的程序将报告已分配的内存区域和未使用的区域。例如，一种可能的内存分配安排如下：

Addresses [0：315000] ProcessP1

Addresses [315001：512500] ProcessP3

Addresses [512501：625575] Unused

Addresses [625575：725100] ProcessP6

Addresses [725001]…

**分配内存**

您的程序将使用我们的教科书中列出的三种方法之一来分配内存，具体取决于传递给RQ命令的标志。标志是：

•F -first fit 首次适应算法

•B-—best fit 最佳适应算法

•W-worst fit 最坏适应算法

这将要求您的程序跟踪代表可用内存的不同孔。当内存请求到达时，它将根据分配策略从一个可用漏洞中分配内存。

如果没有足够的内存分配给请求，它将输出错误消息并拒绝该请求。

您的程序还需要跟踪哪个内存区域已分配给哪个进程。这是支持STAT命令所必需的，当通过RL命令释放内存时也需要它，因为释放内存的进程被传递给此命令。如果释放的分区与现有孔相邻，请务必将两个孔合并为一个孔。

**紧缩内存**

如果用户输入C命令，程序会将这组孔压缩成一个较大的孔。例如，如果您有四个大小为550 KB，375 KB，1,900 KB和4,500 KB的单独孔，则您的程序会将这四个孔组合成一个大小为7,325 KB的大孔。

有几种实施压实的策略，其中一种在我们的教科书中提出。请务必更新受压缩影响的任何进程的起始地址。

1. **实验原理**

这里可变分区存储管理，在系统运行当然开始，假设初始状态下，可用的内存空间为640KB，存储器区被分为操作系统分区（40KB）和可给用户的空间区（600KB）。

（作业1 申请130KB、 作业2 申请60KB、 作业3 申请100KB 、 作业2 释放 60KB 、

作业4 申请 200KB、 作业3释放100KB、 作业1 释放130KB 、 作业5申请140KB 、

作业6申请60KB 、作业7申请50KB）

当作业1进入内存后，分给作业1（130KB），随着作业1、2、3的进入，分别分配60KB、100KB，经过一段时间的运行后，作业2运行完毕，释放所占内存。此时，作业4进入系统，要求分配200KB内存。作业3、1运行完毕，释放所占内存。此时又有作业5申请140KB，作业6申请60KB，作业7申请50KB。为它们进行主存分配和回收。

1、采用可变分区存储管理，使用空闲分区链实现主存分配和回收。

空闲分区链：使用链指针把所有的空闲分区链成一条链，为了实现对空闲分区的分配和链接，在每个分区的起始部分设置状态位、分区的大小和链接各个分区的前向指针，由状态位指示该分区是否分配出去了；同时，在分区尾部还设置有一后向指针，用来链接后面的分区；分区中间部分是用来存放作业的空闲内存空间，当该分区分配出去后，状态位就由“0”置为“1”。

设置一个内存空闲分区链，内存空间分区通过空闲分区链来管理，在进行内存分配时，系统优先使用空闲低端的空间。

设计一个空闲分区说明链，设计一个某时刻主存空间占用情况表，作为主存当前使用基础。初始化空间区和已分配区说明链的值，设计作业申请队列以及作业完成后释放顺序，实现主存的分配和回收。要求每次分配和回收后显示出空闲内存分区链的情况。把空闲区说明链的变化情况以及各作业的申请、释放情况显示打印出来。

2.采用可变分区存储管理，分别采用首次适应算法、最佳适应算法和最坏适应算法实现主存分配和回收。

3、**主存空间分配：这里我把三种方法都实现了一遍**

（1）首次适应算法

在该算法中，把主存中所有空闲区按其起始地址递增的次序排列。在为作业分配存储空间时，从上次找到的空闲分区的下一个空闲分区开始查找，直到找到第一个能满足要求的空闲区，从中划出与请求的大小相等  
的存储空间分配给作业，余下的空闲区仍留在空闲区链中。

（2）最佳适应算法

在该算法中，把主存中所有空闲区按其起始地址递增的次序排列。在为作业分配存储空间时，从上次找到的空闲分区的下一个空闲分区开始查找，直到找到一个能满足要求的空闲区且该空闲区的大小比其他满足要求的空闲区都小，从中划出与请求的大小相等的存储空间分配给作业，余下的空闲区仍留在空闲区链中

（3）最坏适应算法

在该算法中，把主存中所有空闲区按其起始地址递增的次序排列。在为作业分配存储空间时，从上次找到的空闲分区的下一个空闲分区开始查找，直到找到一个能满足要求的空闲区且该空闲区的大小比其他满足要求的空闲区都大，从中划出与请求的大小相等的存储空间分配给作业，余下的空闲区仍留在空闲区链中。

4、主存空间回收

当一个作业执行完成撤离时，作业所占的分区应该归还给系统。归还的分区如果与其它空闲区相邻，则应合成一个较大的空闲区，登记在空闲区说明链中，此时，相邻空闲区的合并问题，要求考虑四种情况：

1. 释放区下邻空闲区（低地址邻接）
2. 释放区上邻空闲区（高地址邻接）
3. 释放区上下都与空闲区邻接
4. 释放区上下邻都与空闲区不邻接
5. **项目框架**

**main函数里的流程图**

初始化first和end

整理分区序号

显示空闲分区链

选择算法a

a=1，首次适应算法

a=2，最佳适应算法

a=3，最坏适应算法

选择操作i

i=1，分配空间函数a

i=0，退出程序

i=2，回收空间函数

结束

分配空间里的流程图

p->data.length=request

分配不成功

分配空间函数

a=1

a=2

a=3

输入申请内存大小

按顺序找空闲块

初始化q，使它指向空闲块中长度小的一块

输入申请内存大小

初始化q，使它指向空闲块中长度大的一块

p->data.length>request

p的状态为已分配

剩下

p->data.length-=request

输入申请内存大小

Y

Y

N

N

返回到整理分区序号

回收空间里的流程图

p的状态改为空闲

回收p，p的前一个为first

p的后一个是end

p的后一个状态空

与后面空闲块相连

将p 的状态改为空闲

将p 的状态改为空闲

回收空间函数

p的后一个是end

Y

N

Y

N

Y

N

p的前一个状态空

p的前一个状态空

p的后一个状态空

p的后一个状态空

p的后一个状态空

p的后一个状态空

Y

Y

Y

N

N

N

与前面空闲块相连

p的状态改为空闲

与前面空闲块相连

与后面空闲块相连

Y

N

返回到整理分区序号

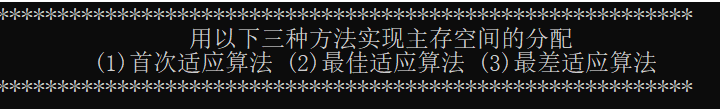
这里我采用了一个struct free\_table数据结构，里面包含分区序号（num）、起始地址（address）、分区长度（length）和分区状态（state）。

同时还用了线性表的双性链表存储结构（struct Node），里面包含前区指针（prior）和后继指针（next）。一开始定义一条（含有first和end）的链，用开始指针和尾指针开创空间链表。然后分别按三种算法进行分配和回收。

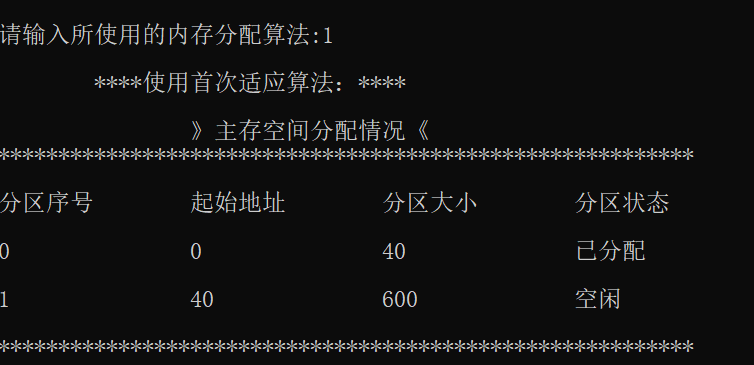
关键函数有以下：sort（）、allocation（）、recovery（）、和First\_fit（）、Best\_fit（）、Worst\_fit（）；其中sort（）函数是用来整理分区序号的，如在删序号3时，她与前面序号2相连在一起了，然后序号2中的长度总满足申请的内存大小，就会在序号2中分配，然后序号在2的基础上加1，一直加，加到与原本序号3的下一个序号也就是4相等，这时sort（）就开始有明显的工作了；allocation（）是分配空间的，也是过渡到三个算法中的，当三个算法中满足或者不满足分配请求，都会又返回值给allocation（）；recovery（）是用来回收内存的，里面包含了四种情况相连结果，即释放区上与空闲区邻接、释放区下与空闲区邻接、释放区上下都与空闲区邻接、释放区上下都与空闲区不邻接这四种情况的结果。

1. **实现过程**

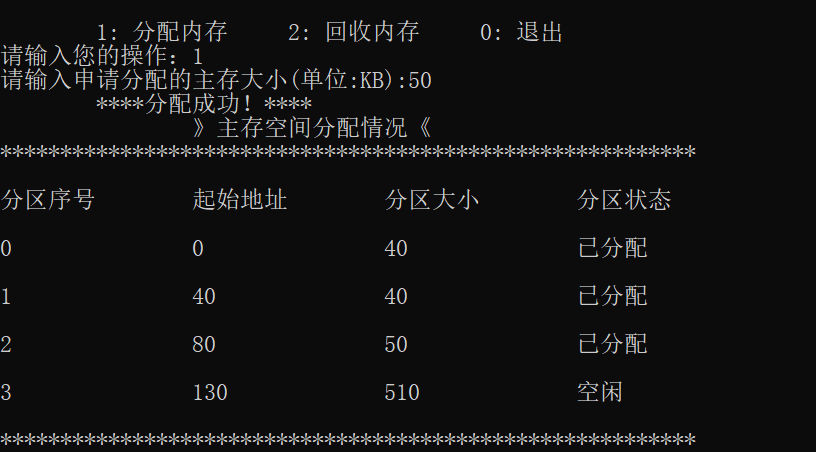
运行得到三种方法实现内存分配：



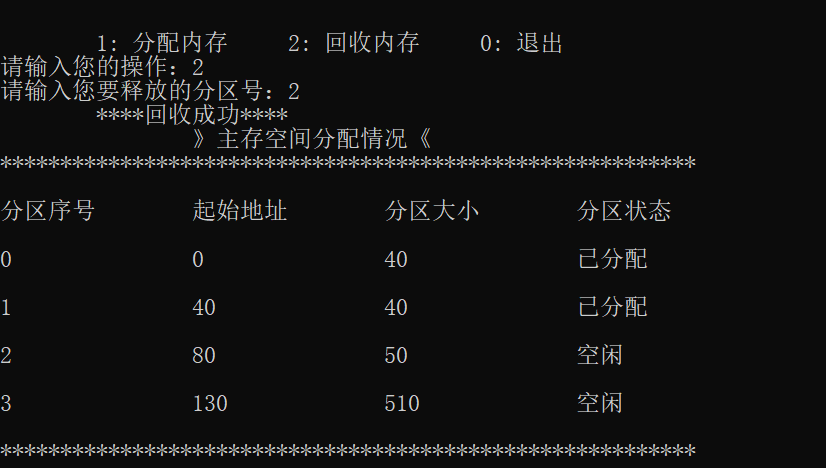
**（1）初始化首次适应算法：**



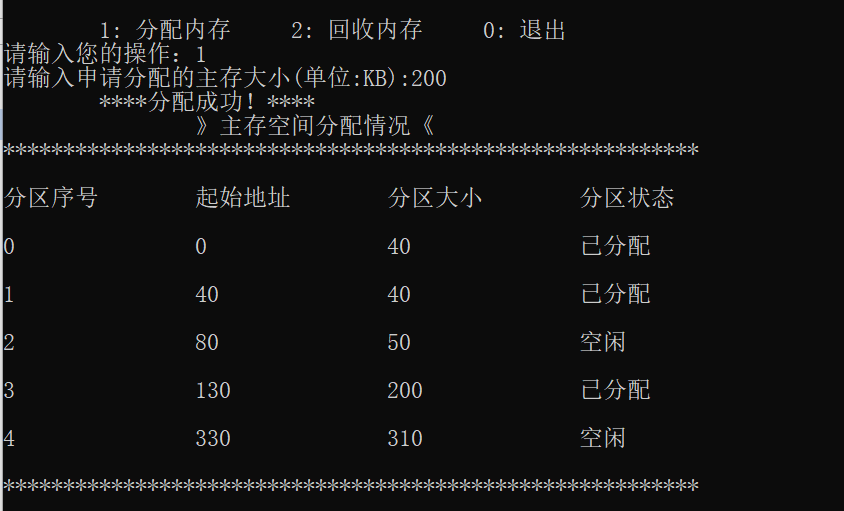
当作业1、2、3顺利分配内存空间后：



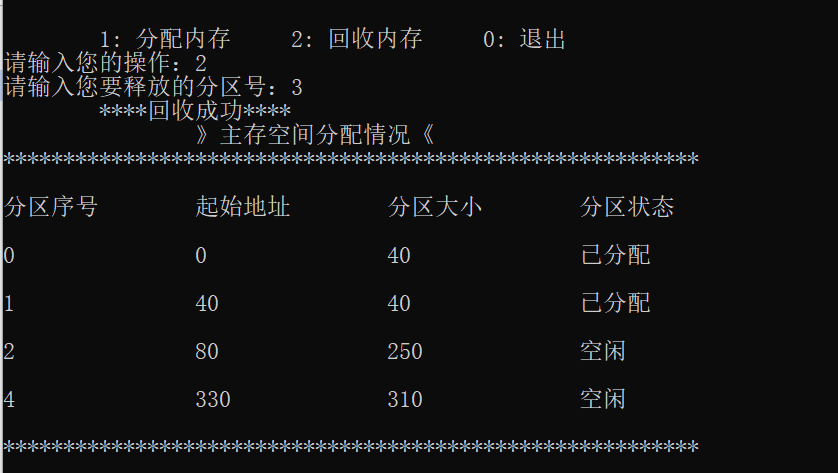
回收序号2里面的内存：



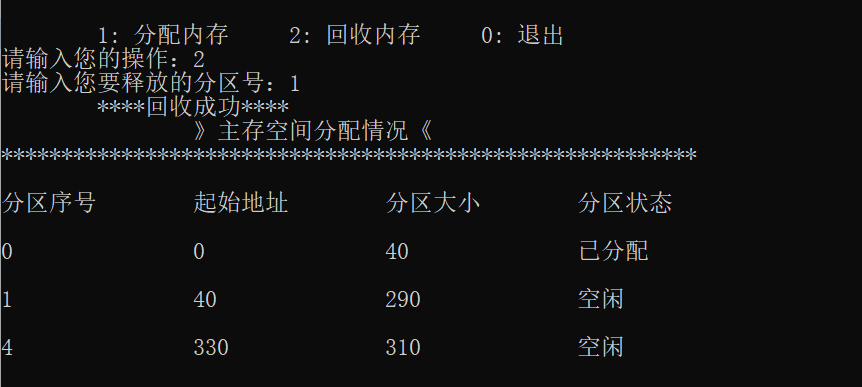
分配作业4：



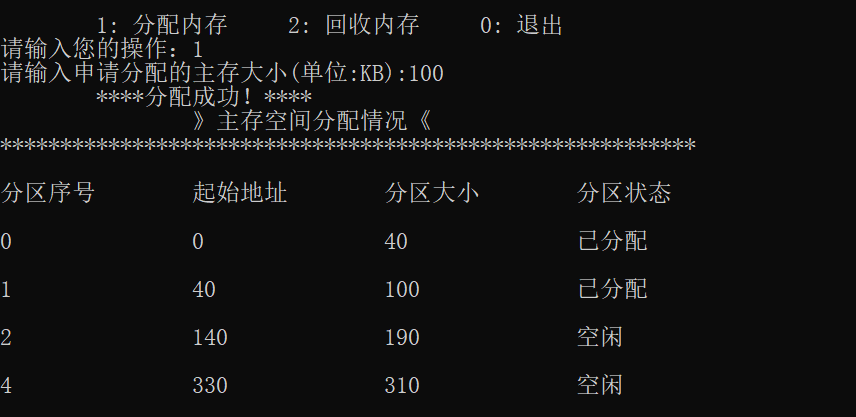
回收序号3里面的内存（与上邻序号2相连了）

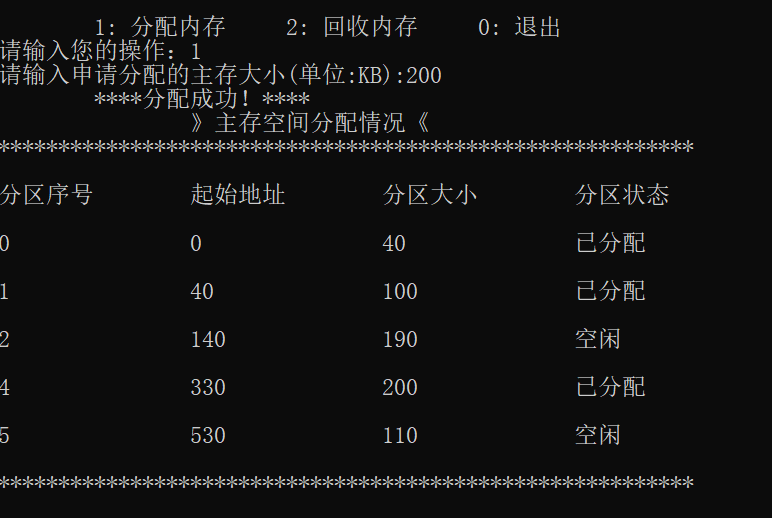


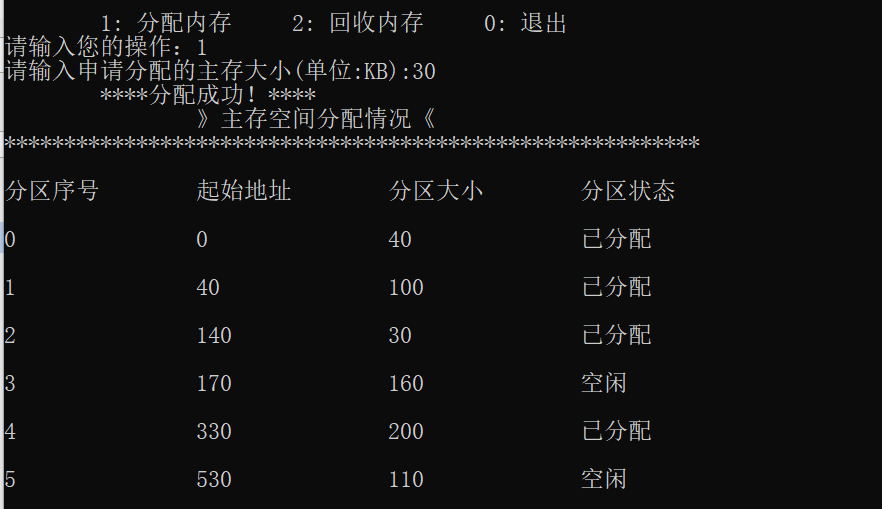
回收序号1里的内存（与下邻序号2相连了）



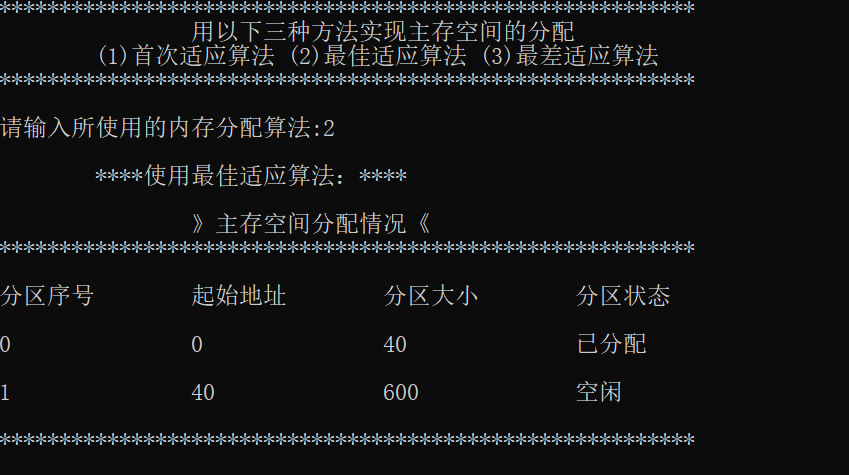
继续分配（会发现总是按顺序查找满足要求的第一个空闲块，一旦发现就会分配）：

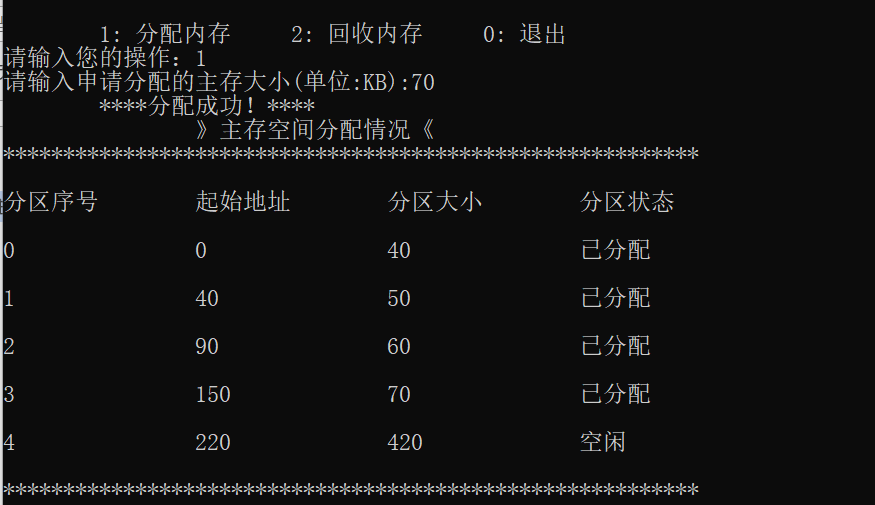


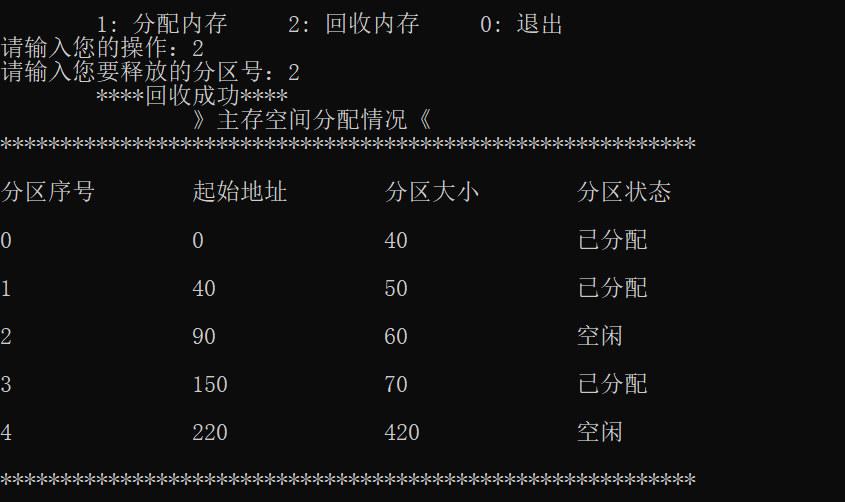


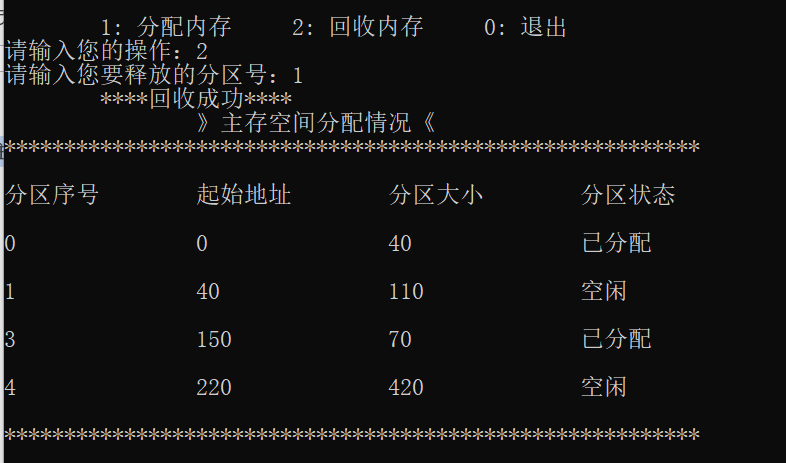


**（2）初始化最佳适应算法：**

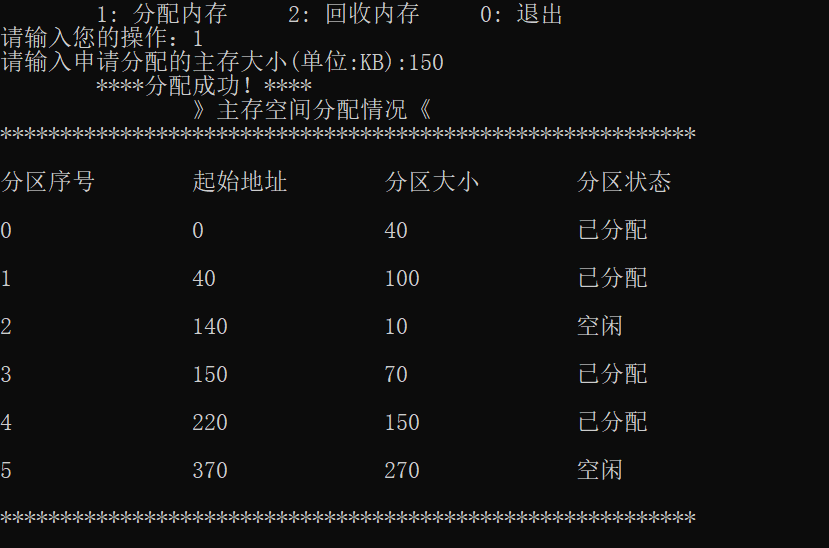


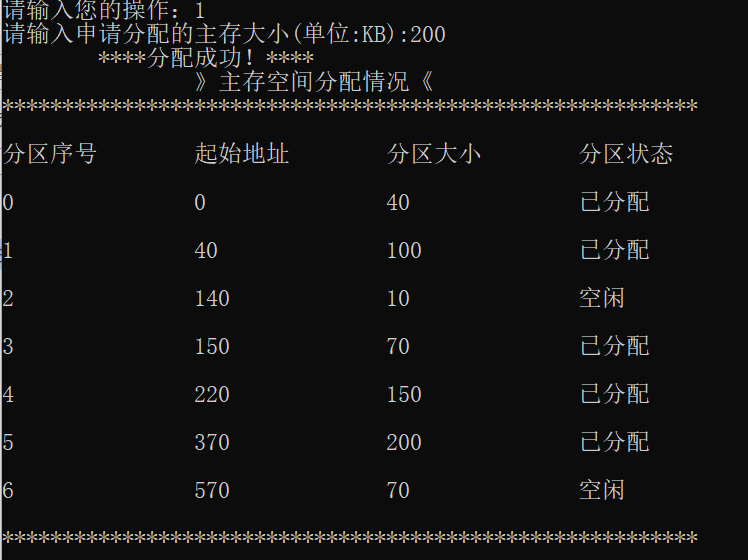


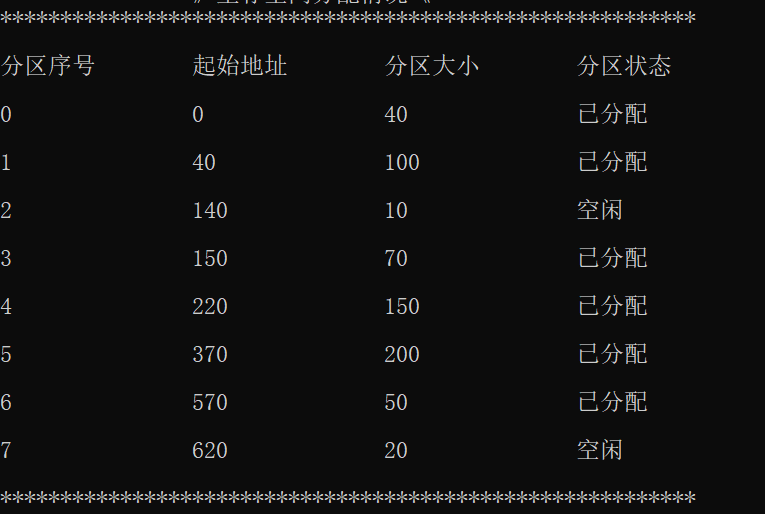




继续分配（会发现总是查找满足要求的空闲块且其比其他空闲块长度小，一旦发现就会分配）：

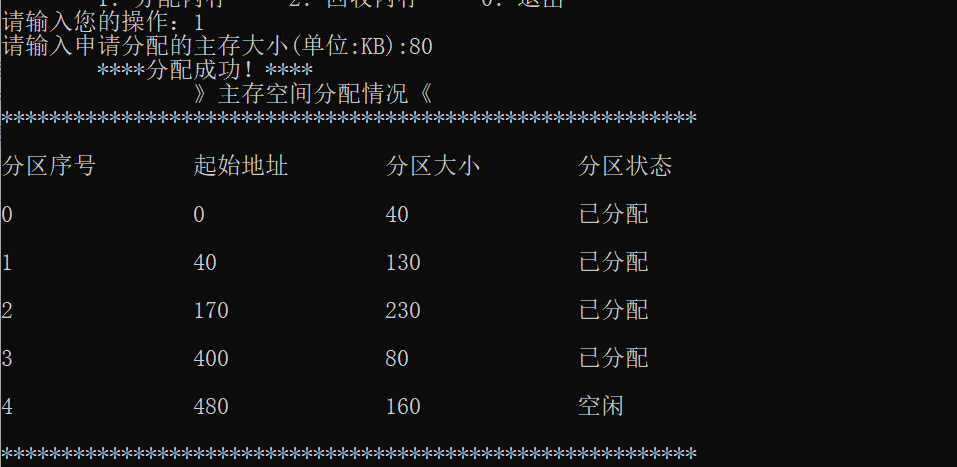


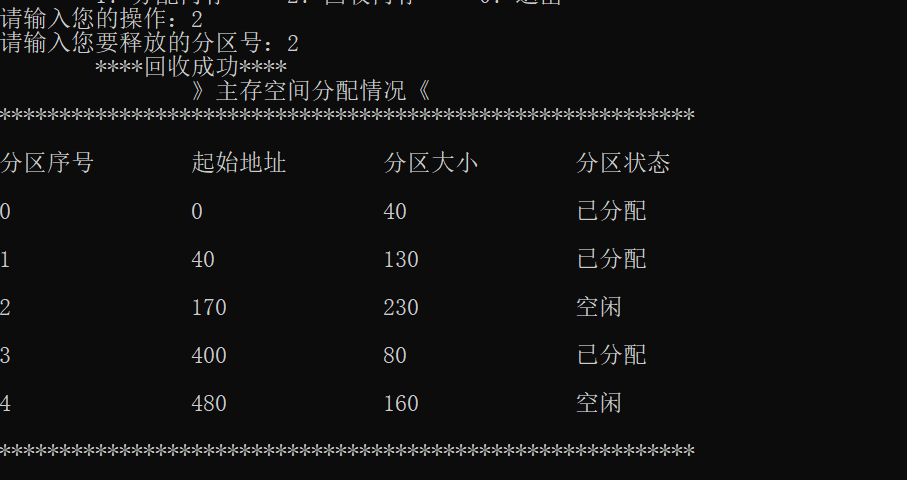




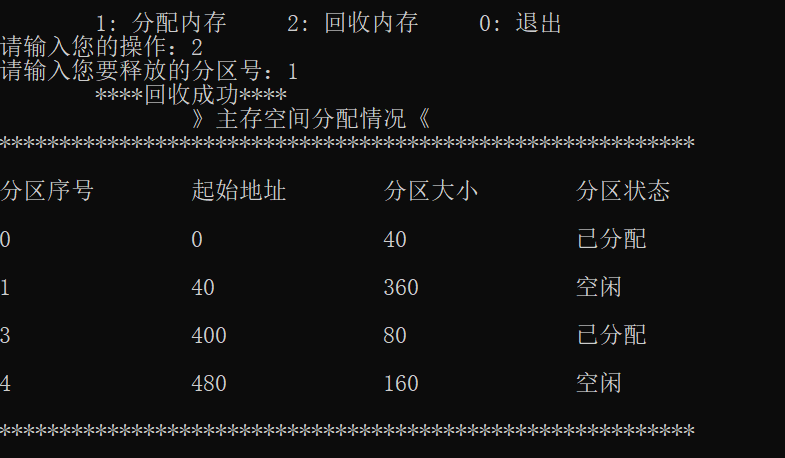
**（3）初始化最坏适应算法：**

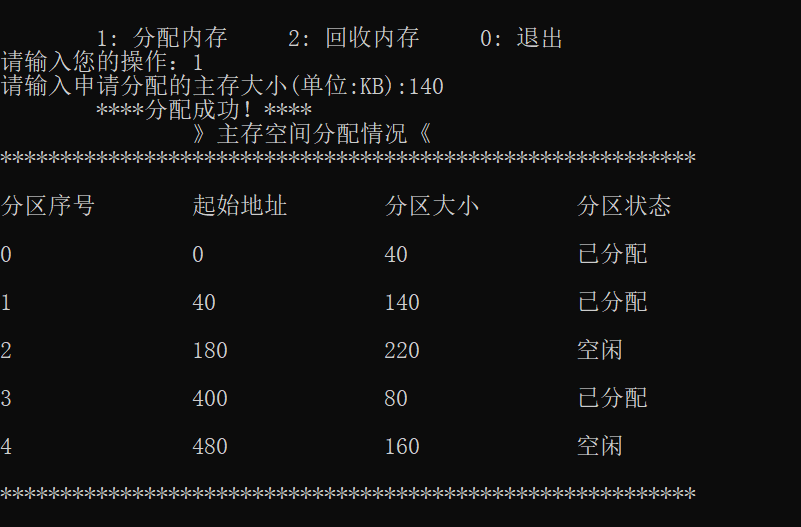


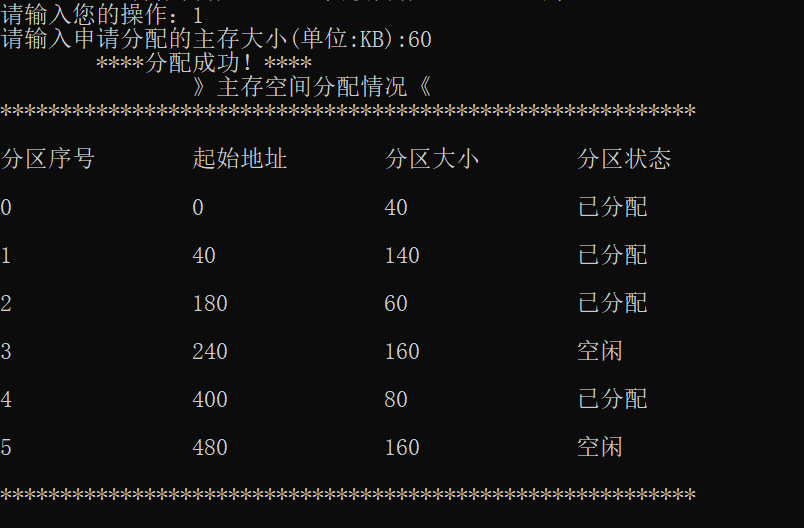


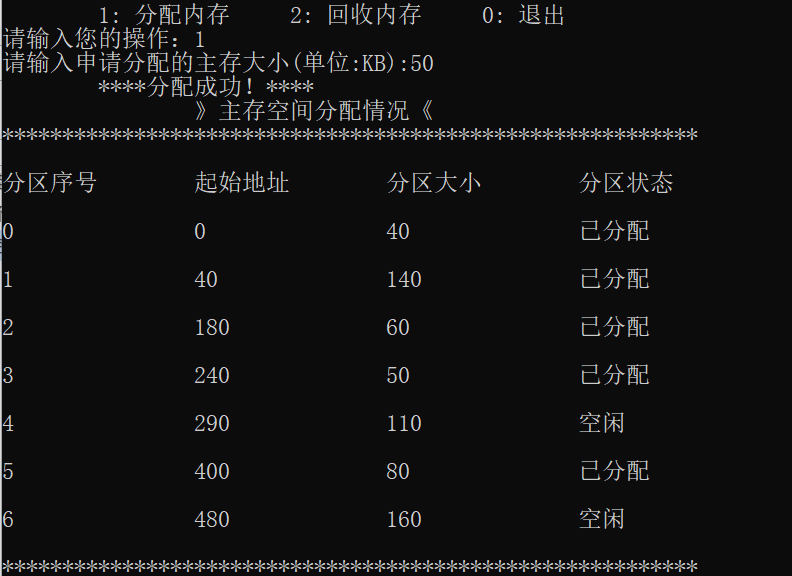


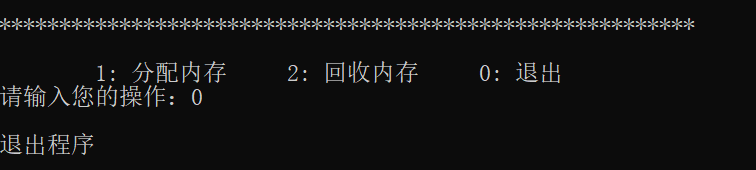
继续分配（**会发现总是查找满足要求的空闲块且其比其他空闲块长度小，一旦发现就会分配**）：











1. **实验总结与收获**

**不同内存管理方法总结：**

**一 页式管理**1 页式管理的基本原理将各进程的虚拟空间划分成若干个长度相等的页(page)，页式管理把内存空间按页的大小划分成片或者页面（page frame），然后把页式虚拟地址与内存地址建立一一对应页表，并用相应的硬件地址变换机构，来解决离散地址变换问题。页式管理采用请求调页或预调页技术实现了内外存存储器的统一管理。   
  
它分为   
1 静态页式管理。静态分页管理的第一步是为要求内存的作业或进程分配足够的页面。系统通过存储页面表、请求表以及页表来完成内存的分配工作。静态页式管理解决了分区管理时的碎片问题。但是，由于静态页式管理要求进程或作业在执行前全部装入内存，如果可用页面数小于用户要求时，该作业或进程只好等待。而且作业和进程的大小仍受内存可用页面数的限制。   
  
2 动态页式管理。动态页式管理是在静态页式管理的基础上发展起来的。它分为请求页式管理和预调入页式管理。   
  
优点： 没有外碎片，每个内碎片不超过页大小。一个程序不必连续存放。便于改变程序占用空间的大小（主要指随着程序运行而动态生成的数据增多，要求地址空间相应增长，通常由系统调用完成而不是操作系统自动完成）。   
  
缺点：程序全部装入内存。   
要求有相应的硬件支持。例如地址变换机构，缺页中断的产生和选择淘汰页面等都要求有相应的硬件支持。这增加了机器成本。增加了系统开销，例如缺页中断处理机，请求调页的算法如选择不当，有可能产生抖动现象。 虽然消除了碎片，但每个作业或进程的最后一页内总有一部分空间得不到利用果页面较大，则这一部分的损失仍然较大。   
  
**二 段式管理的基本思想**把程序按内容或过程（函数）关系分成段，每段有自己的名字。一个用户作业或进程所包含的段对应一个二维线形虚拟空间，也就是一个二维虚拟存储器。段式管理程序以段为单位分配内存，然后通过地址影射机构把段式虚拟地址转换为实际内存物理地址。   
程序通过分段(segmentation)划分为多个模块，如代码段、数据段、共享段。其优点是： 可以分别编写和编译。 可以针对不同类型的段采取不同的保护。 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。   
三 段页式管理的实现原理   
1 虚地址的构成   
一个进程中所包含的具有独立逻辑功能的程序或数据仍被划分为段，并有各自的段号s。这反映相继承了段式管理的特征。其次，对于段s中的程序或数据，则按照一定的大小将其划分为不同的页。和页式系统一样，最后不足一页的部分仍占一页。这反映了段页式管理中的页式特征。从而，段页式管理时的进程的虚拟地址空间中的虚拟地址由三部分组成：即段号s，页号P和页内相对地址d。虚拟空间的最小单位是页而不是段，从而内存可用区也就被划分成为着干个大小相等的页面，且每段所拥有的程序和数据在内存中可以分开存放。分段的大小也不再受内存可用区的限制。   
 2 段表和页表   
为了实现段页式管理，系统必须为每个作业或进程建立一张段表以管理内存分配与释放、缺段处理、存储保护相地址变换等。另外，由于一个段又被划分成了若干页，每个段又必须建立一张页表以把段中的虚页变换成内存中的实际页面。显然，与页式管理时相同，页表中也要有相应的实现缺页中断处理和页面保护等功能的表项。另外，由于在段页式管理中，页表不再是属于进程而是属于某个段，因此，段表中应有专项指出该段所对应页表的页表始址和页表长度。   
 3 动态地址变换过程   
在一般使用段页式存储管理方式的计算机系统中，都在内存中辟出一块固定的区域存放进程的段表和页表。因此，在段页式管理系统中，要对内存中指令或数据进行一次存取的话，至少需要访问三次以上的内存：   
第一次是由段表地址寄存器得段表始址后访问段表，由此取出对应段的页表在内存中的地址。   
第二次则是访问页表得到所要访问的物理地址。   
第三次才能访问真正需要访问的物理单元。  
显然，这将使CPU的执行指令速度大大降低。为了提高地址转换速度，设置快速联想寄存器就显得比段式管理或页式管理时更加需要。在快速联想寄存器中，存放当前最常用的段号s、页号p和对应的内存页面与其它控制用栏目。当要访问内存空间某一单元时，可在通过段表、页表进行内存地址查找的同时，根据快速联想寄存器查找其段号和页号。如果所要访问的段或页在快速联想寄存器中，则系统不再访问内存中的段表、页表而直接把快速联想寄存器中的值与页内相对地址d拼接起来得到内存地址。   
 总之，因为段页式管理是段式管理的页式管理方案结合而成的，所以具有它们二者的优点。但反过来说，由于管理软件的增加，复杂性和开销也就随之增加了。另外，需要的硬件以及占用的内存也有所增加。更重要的是，如果不采用联想寄存器的方式提高CPU的访内速度，将会使得执行速度大大下降。