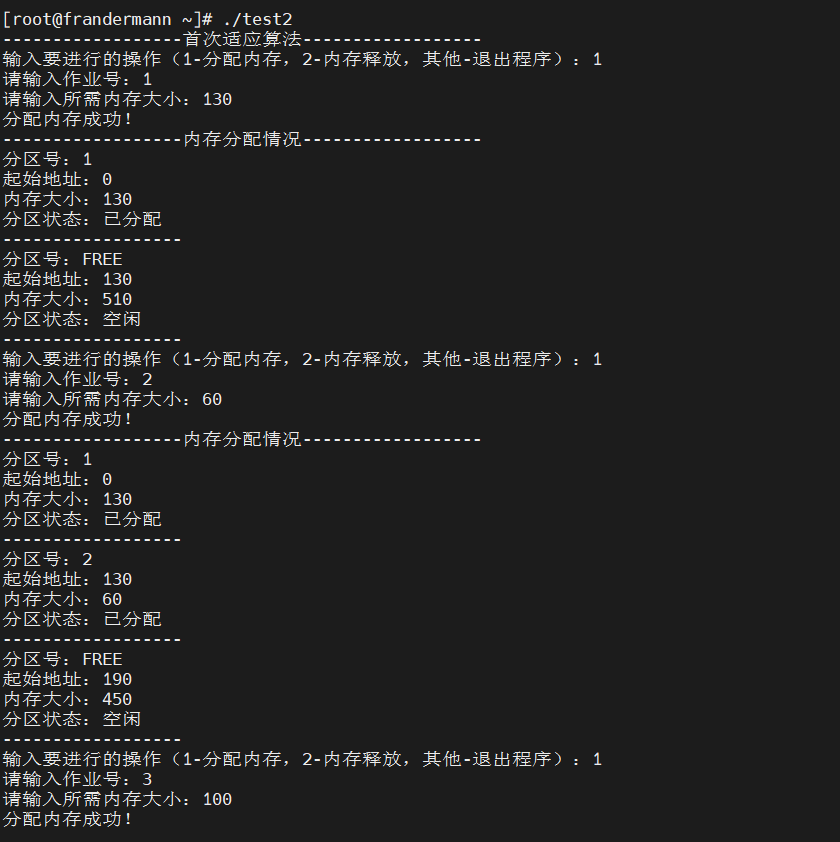
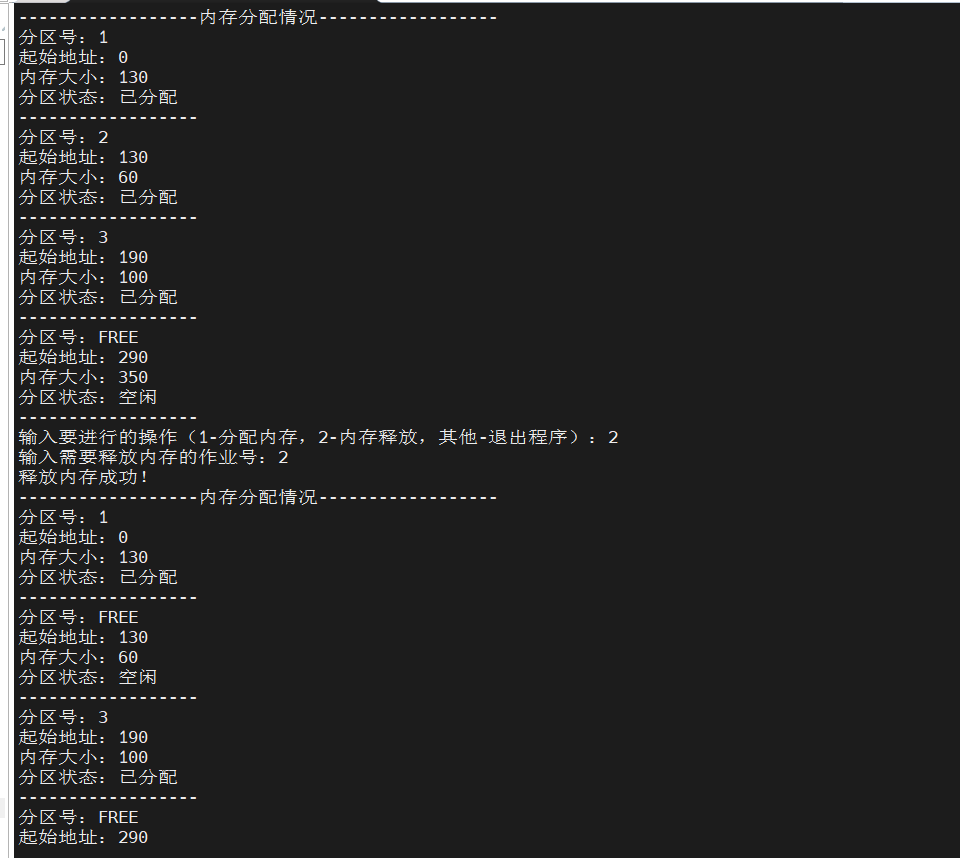
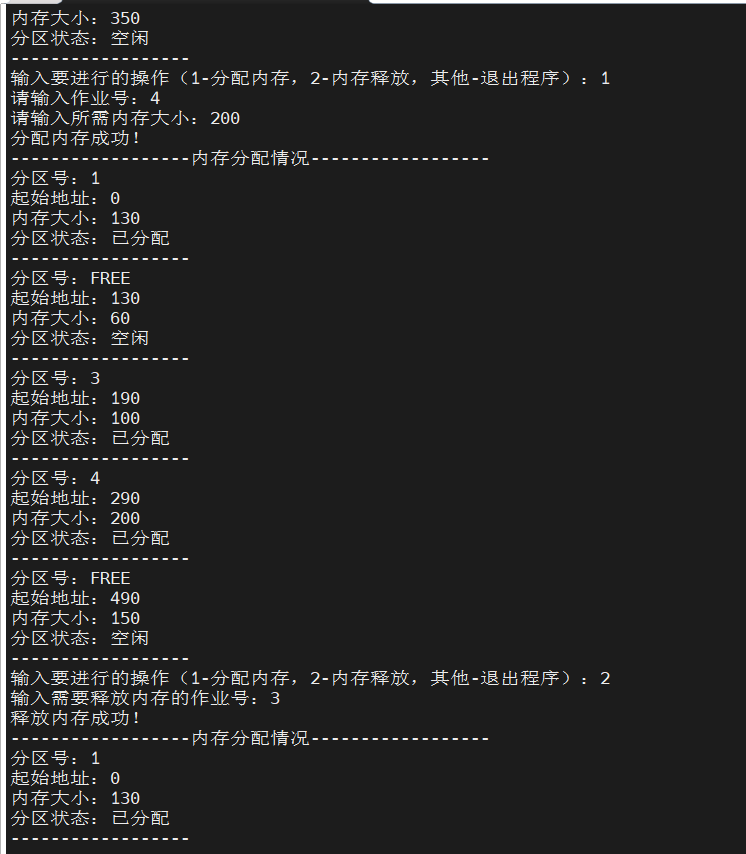
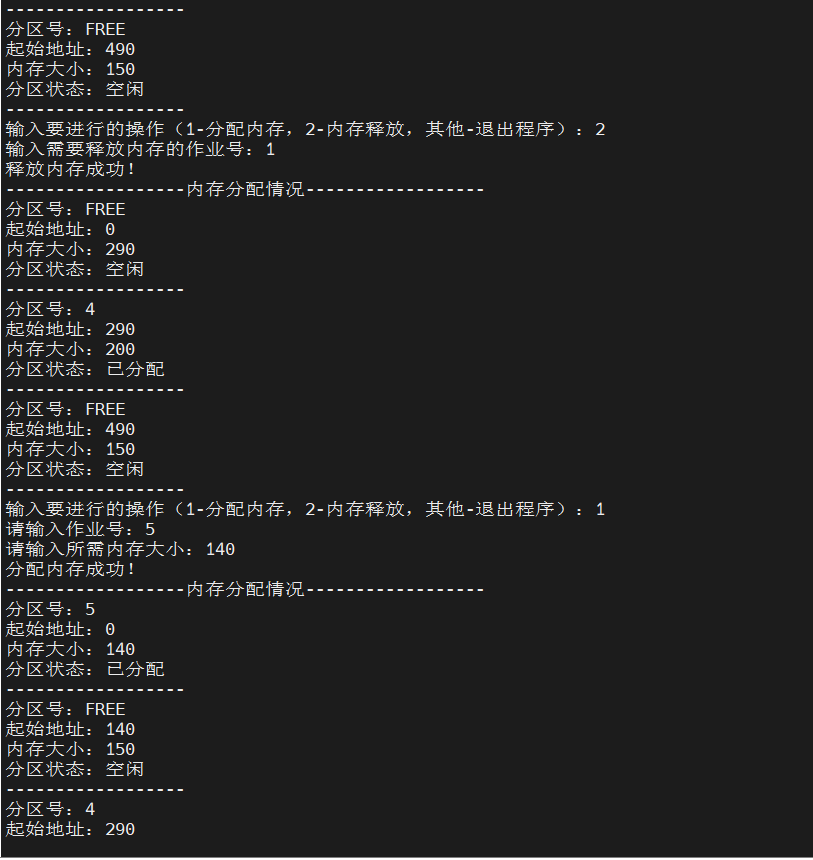
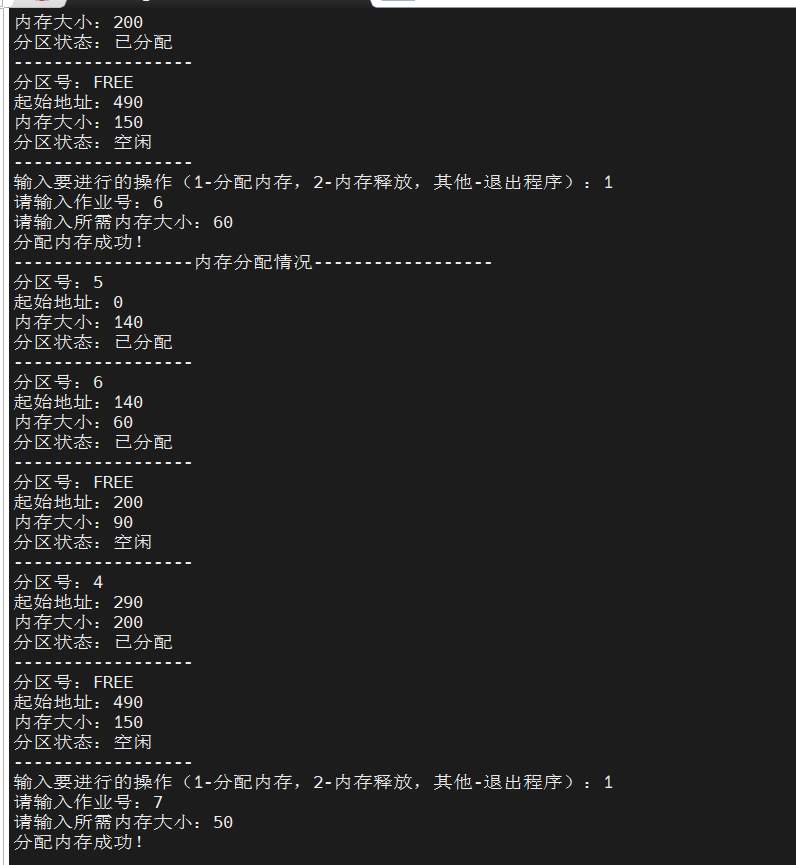
首次适应算法

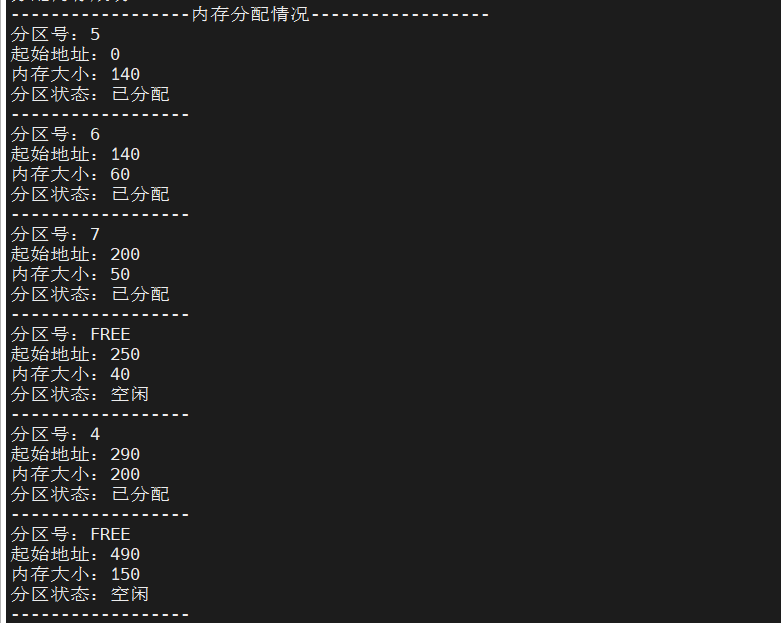


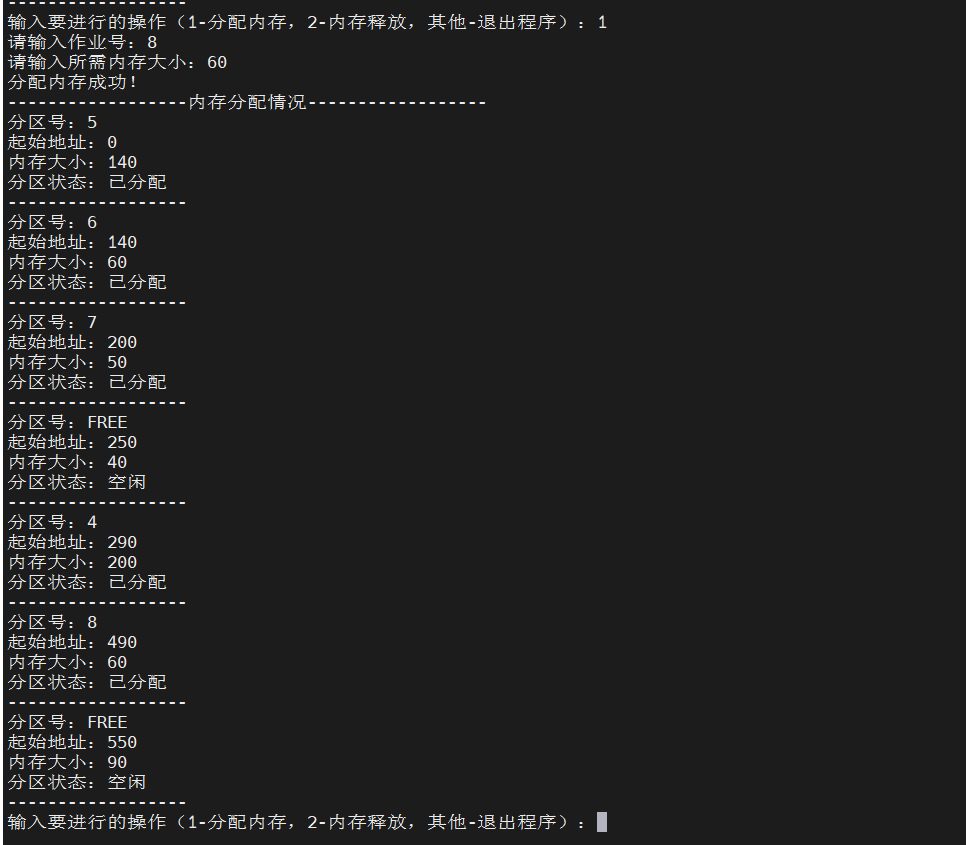












//代码

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define FREE 0

#define BUSY 1

#define Max\_length 640

typedef struct freearea//空闲区的结构体

{

int ID;//分区号

int size;//分区大小

int address;//分区地址

bool isUsed;//使用状态，0为未占用，1为已占用

} freearea;

typedef struct DuNode//首尾不互连的双向链表结点

{

freearea data;//数据域

struct DuNode \*prior;//指针域

struct DuNode \*next;

} DuNode, \*DuLinkList;

DuLinkList m\_rid;

DuLinkList m\_last;

void init()//空闲区队列初始化

{

m\_rid = (DuLinkList)malloc(sizeof(DuNode));

m\_last = (DuLinkList)malloc(sizeof(DuNode));

m\_rid->prior = NULL;

m\_rid->next = m\_last;

m\_last->prior = m\_rid;

m\_last->next = NULL;

m\_rid->data.size = 0;

m\_rid->data.isUsed = BUSY; //首结点不会被使用，定义为占用状态防止分区合并失败

m\_last->data.address = 0;

m\_last->data.size = Max\_length;

m\_last->data.ID = 0;

m\_last->data.isUsed = 0;

}

int first\_fit(int ID,int size)//首次适应算法

{

DuLinkList temp = (DuLinkList)malloc(sizeof(DuNode));

DuNode \*p = m\_rid->next;

temp->data.ID=ID;

temp->data.size=size;

temp->data.isUsed=BUSY;

while(p)

{

if(p->data.ID == ID)//不允许存在同名作业

{

printf("该作业号对应的作业已经在内存中！");

return 0;

}

if (p->data.isUsed==FREE && p->data.size==size)//请求大小刚好满足

{

p->data.isUsed=BUSY;

p->data.ID=ID;

return 1;

}

if (p->data.isUsed==FREE && p->data.size>size)//空闲区比所需内存大，则需要将多的内存作回收处理

{

temp->next=p;

temp->prior=p->prior;

temp->data.address=p->data.address;

p->prior->next=temp;

p->prior=temp;

p->data.address=temp->data.address+temp->data.size;

p->data.size-=size;

return 1;

}

p=p->next;

}

return 0;

}

void alloc()//分配内存

{

int ID,size1;

printf("请输入作业号：");

scanf("%d", &ID);

printf("请输入所需内存大小：");

scanf("%d", &size1);

if (ID<=0 || size1<=0)

printf("错误！请输入正确的作业号和请求的内存大小");

if(first\_fit(ID,size1))

printf("分配内存成功！\n");

else

printf("分配内存失败！\n");

}

void freeNode()//释放内存

{

int ID;

DuNode \*p = m\_rid->next;

printf("输入需要释放内存的作业号：");

scanf("%d", &ID);

while (p)

{

if (p->data.ID == ID)

{

p->data.ID = 0;

p->data.isUsed = FREE;

if (!p->prior->data.isUsed && p->next->data.isUsed)//与前一个空闲区相邻，则合并

{

p->prior->data.size += p->data.size;

p->prior->next = p->next;

p->next->prior = p->prior;

}

if (!p->next->data.isUsed && p->prior->data.isUsed) //与后一个空闲区相邻，则合并

{

p->data.size += p->next->data.size;

if(p->next->next)

{

p->next->next->prior=p;

p->next = p->next->next;

}

else

p->next = p->next->next;

}

if(!p->prior->data.isUsed && !p->next->data.isUsed) //前后的空闲区均为空

{

p->prior->data.size += p->data.size + p->next->data.size;

if(p->next->next)

{

p->next->next->prior = p->prior;

p->prior->next = p->next->next;

}

else

p->prior->next = p->next->next;

}

printf("释放内存成功！\n");

break;

}

p = p->next;

if(!p)

printf("内存中没有该需要释放内存的作业！\n");

}

}

void show()

{

printf("------------------");

printf("内存分配情况");

printf("------------------\n");

DuNode \*p = m\_rid->next;

while(p)

{

printf("分区号：");

if (p->data.ID==FREE)

printf("FREE\n");

else

printf("%d \n", p->data.ID);

printf("起始地址：%d\n", p->data.address);

printf("内存大小：%d\n", p->data.size);

printf("分区状态：");

if (p->data.isUsed==FREE)

printf("空闲\n");

else

printf("已分配\n");

printf("------------------\n");

p=p->next;

}

}

int main()

{

printf("------------------");

printf("首次适应算法");

printf("------------------\n");

init();

int tag = 1;

while(tag < 3 && tag > 0)

{

printf("输入要进行的操作");

printf("（1-分配内存，2-内存释放，其他-退出程序）：");

scanf("%d", &tag);

switch(tag)

{

case 1:

alloc();

show();

break;

case 2:

freeNode();

show();

break;

default:

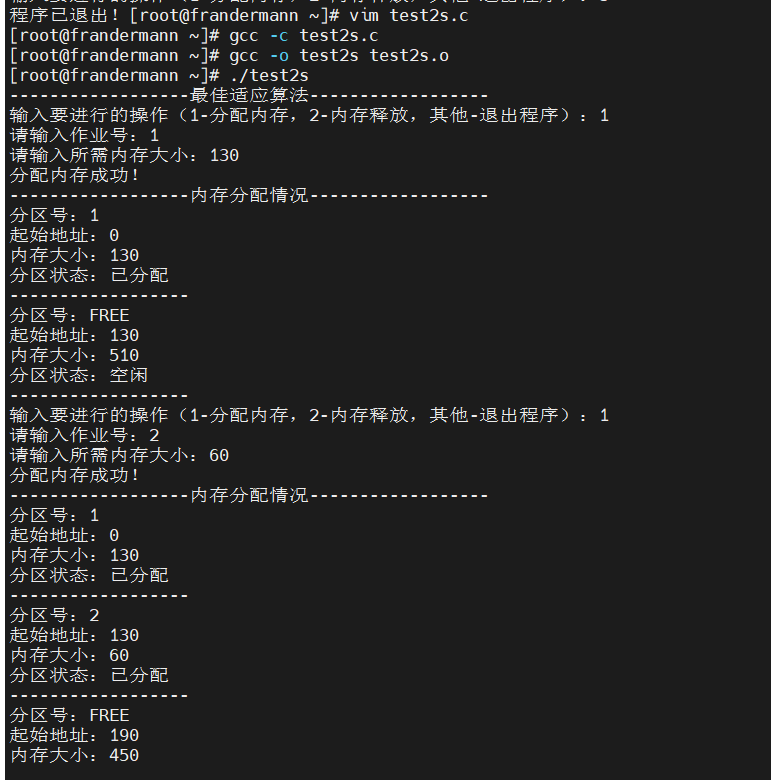
printf("程序已退出！");

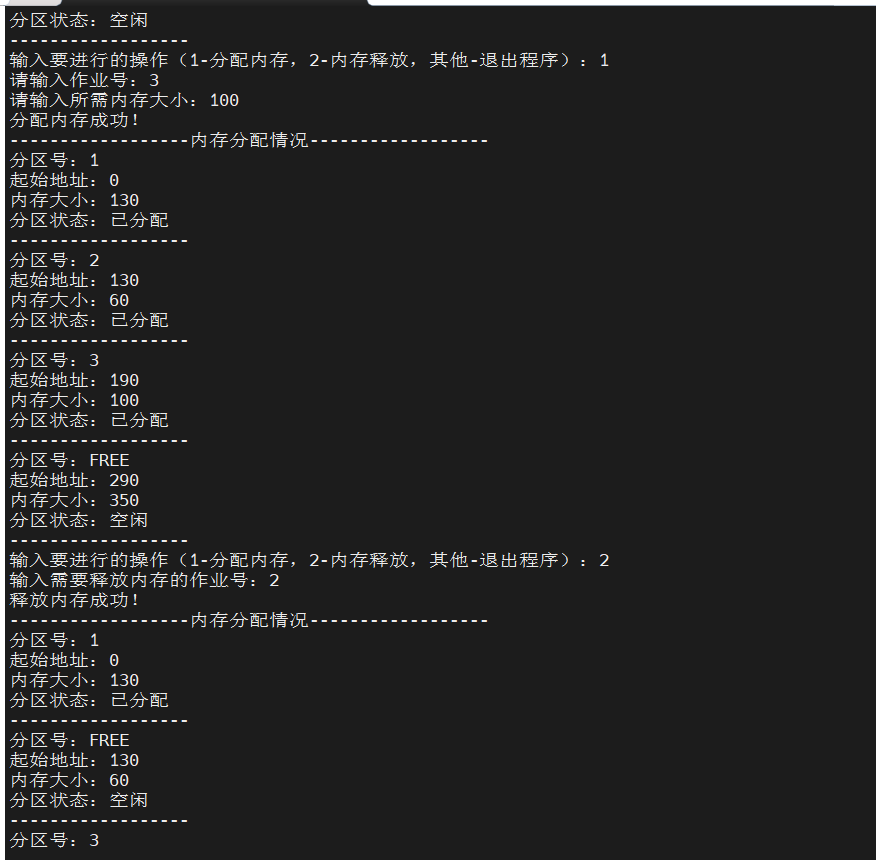
}

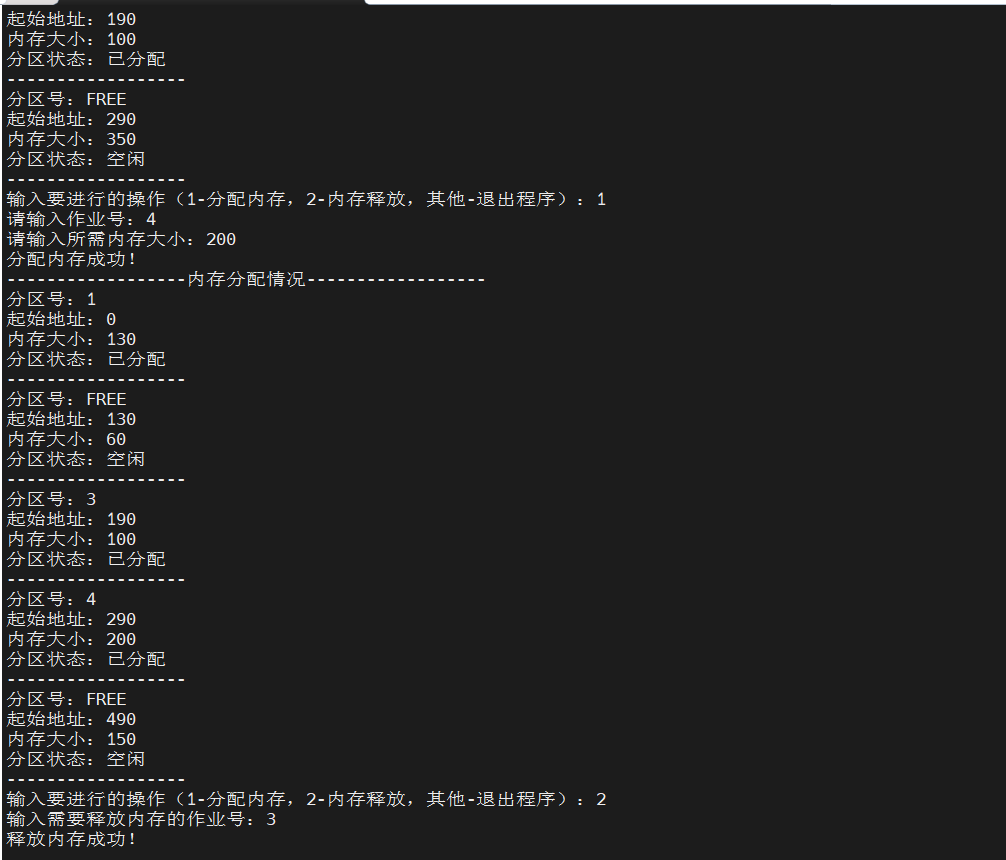
}

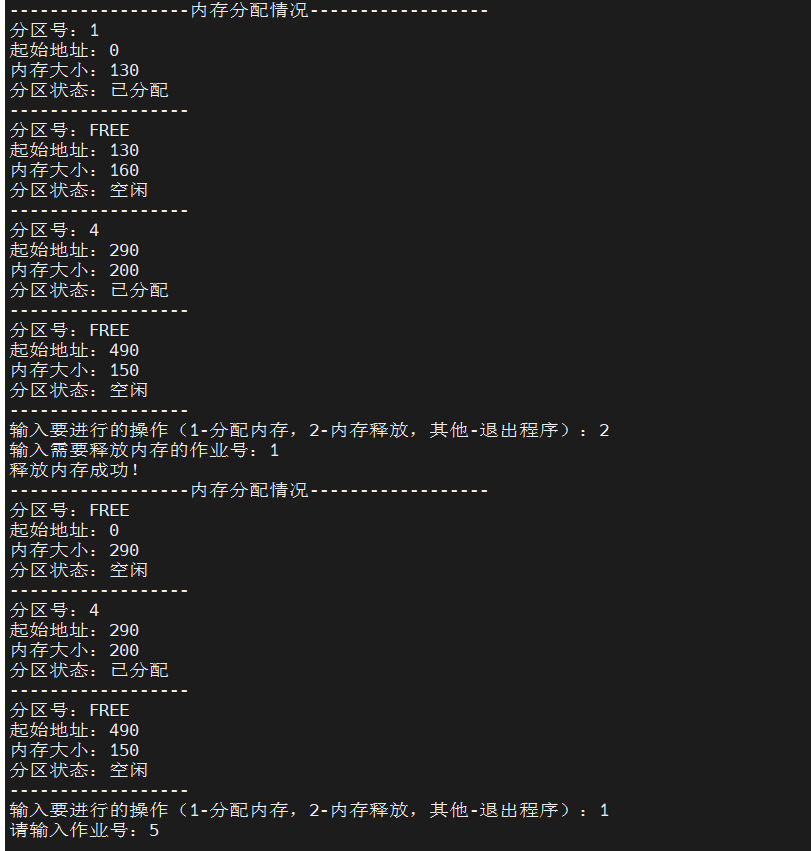
}

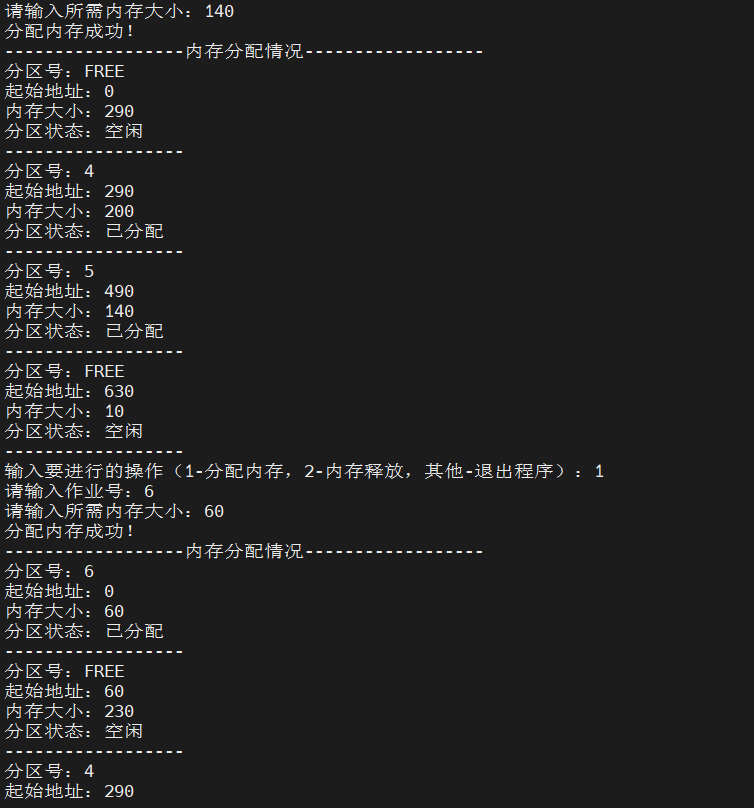
最佳适应算法

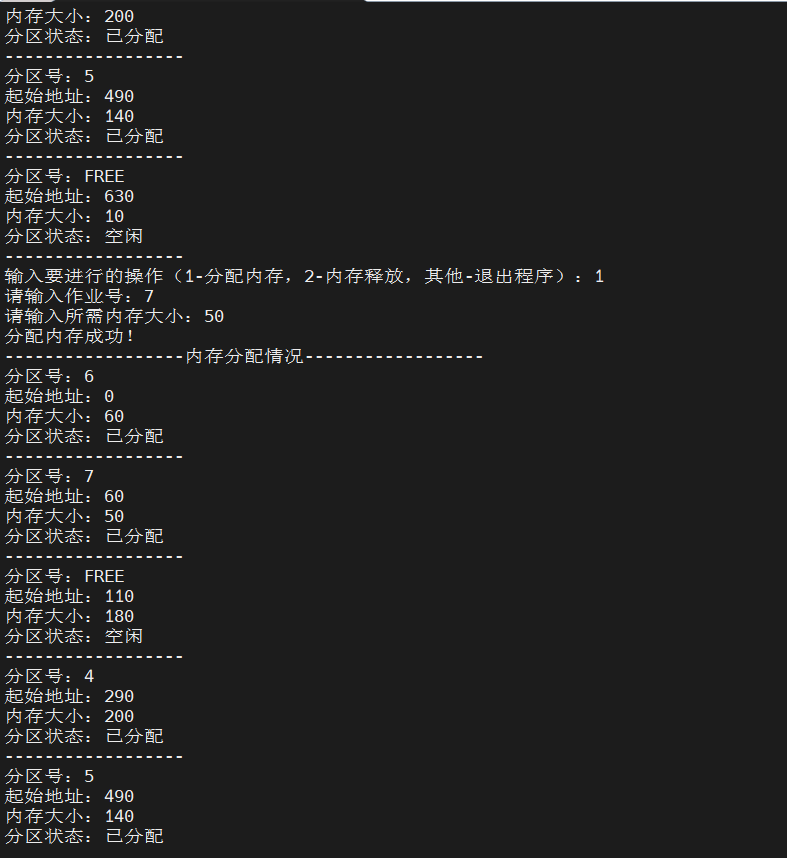


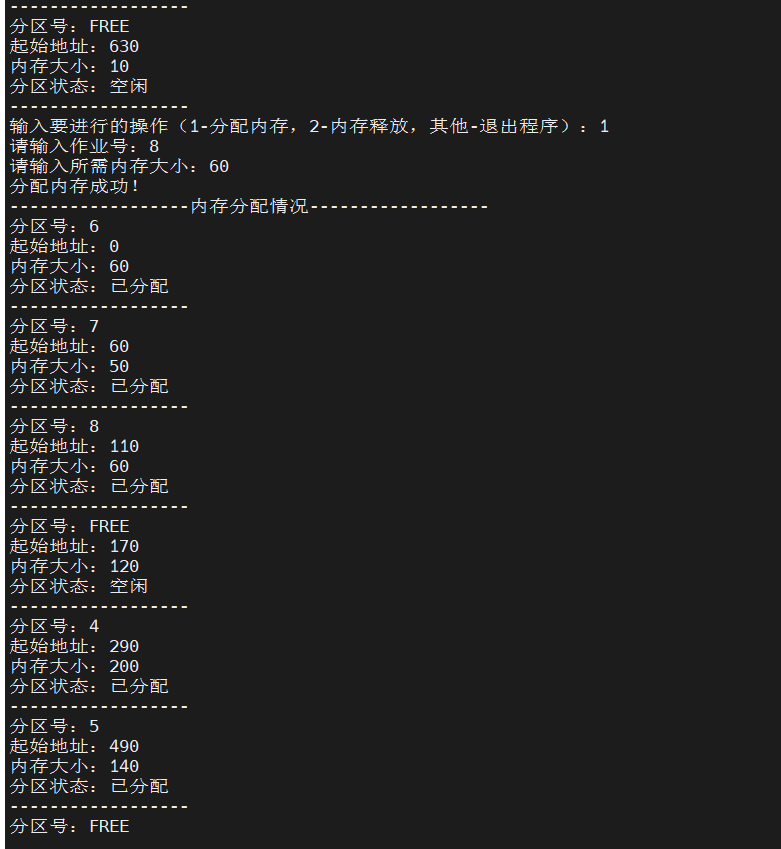














//代码

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define FREE 0

#define BUSY 1

#define Max\_length 640

typedef struct freearea//空闲区的结构体

{

int ID;//分区号

int size;//分区大小

int address;//分区地址

bool isUsed;//使用状态，0为未占用，1为已占用

} freearea;

typedef struct DuNode//首尾不互连的双向链表结点

{

freearea data;//数据域

struct DuNode \*prior;//指针域

struct DuNode \*next;

} DuNode, \*DuLinkList;

DuLinkList m\_rid;

DuLinkList m\_last;

void init()//空闲区队列初始化

{

m\_rid = (DuLinkList)malloc(sizeof(DuNode));

m\_last = (DuLinkList)malloc(sizeof(DuNode));

m\_rid->prior = NULL;

m\_rid->next = m\_last;

m\_last->prior = m\_rid;

m\_last->next = NULL;

m\_rid->data.size = 0;

m\_rid->data.isUsed = BUSY; //首结点不会被使用，定义为占用状态防止分区合并失败

m\_last->data.address = 0;

m\_last->data.size = Max\_length;

m\_last->data.ID = 0;

m\_last->data.isUsed = 0;

}

int best\_fit(int ID,int size)//最佳适应算法

{

int surplus;//记录可用内存与需求内存的差值

DuLinkList temp = (DuLinkList)malloc(sizeof(DuNode));

DuNode \*p = m\_rid->next;

DuNode \*q = NULL;//记录最佳位置

temp->data.ID = ID;

temp->data.size = size;

temp->data.isUsed = BUSY;

while(p)//遍历链表，找到第一个可用的空闲区间赋给q

{

if (p->data.isUsed==FREE && p->data.size >= size)

{

q = p;

surplus = p->data.size - size;

break;

}

p=p->next;

}

while(p)//继续遍历，找到合适的位置

{

if (p->data.isUsed == FREE && p->data.size == size) //分区大小刚好是作业申请的大小

{

p->data.isUsed = BUSY;

p->data.ID = ID;

return 1;

}

if (p->data.isUsed == FREE && p->data.size > size) //可用内存与需求内存的差值更小

{

if (surplus > p->data.size - size)

{

surplus = p->data.size-size;

q = p;

}

}

p=p->next;

}

if (q == NULL)//没有找到位置

return 0;

else//找到最佳位置

{

//将temp插入到结点q之前

temp->next = q;

temp->prior = q->prior;

temp->data.address = q->data.address;

q->prior->next = temp;

q->prior = temp;

q->data.size = surplus;

q->data.address += size;

return 1;

}

}

void alloc()//分配内存

{

int ID,size1;

printf("请输入作业号：");

scanf("%d", &ID);

printf("请输入所需内存大小：");

scanf("%d", &size1);

if (ID<=0 || size1<=0)

printf("错误！请输入正确的作业号和请求的内存大小");

if(best\_fit(ID,size1))

printf("分配内存成功！\n");

else

printf("分配内存失败！\n");

}

void freeNode()//释放内存

{

int ID;

DuNode \*p = m\_rid->next;

printf("输入需要释放内存的作业号：");

scanf("%d", &ID);

while (p)

{

if (p->data.ID == ID)

{

p->data.ID = 0;

p->data.isUsed = FREE;

if (!p->prior->data.isUsed && p->next->data.isUsed)//与前一个空闲区相邻，则合并

{

p->prior->data.size += p->data.size;

p->prior->next = p->next;

p->next->prior = p->prior;

}

if (!p->next->data.isUsed && p->prior->data.isUsed) //与后一个空闲区相邻，则合并

{

p->data.size += p->next->data.size;

if(p->next->next)

{

p->next->next->prior=p;

p->next = p->next->next;

}

else

p->next = p->next->next;

}

if(!p->prior->data.isUsed && !p->next->data.isUsed) //前后的空闲区均为空

{

p->prior->data.size += p->data.size + p->next->data.size;

if(p->next->next)

{

p->next->next->prior = p->prior;

p->prior->next = p->next->next;

}

else

p->prior->next = p->next->next;

}

printf("释放内存成功！\n");

break;

}

p = p->next;

if(!p)

printf("内存中没有该需要释放内存的作业！\n");

}

}

void show()

{

printf("------------------");

printf("内存分配情况");

printf("------------------\n");

DuNode \*p = m\_rid->next;

while(p)

{

printf("分区号：");

if (p->data.ID==FREE)

printf("FREE\n");

else

printf("%d \n", p->data.ID);

printf("起始地址：%d\n", p->data.address);

printf("内存大小：%d\n", p->data.size);

printf("分区状态：");

if (p->data.isUsed==FREE)

printf("空闲\n");

else

printf("已分配\n");

printf("------------------\n");

p=p->next;

}

}

int main()

{

printf("------------------");

printf("最佳适应算法");

printf("------------------\n");

init();

int tag = 1;

while(tag < 3 && tag > 0)

{

printf("输入要进行的操作");

printf("（1-分配内存，2-内存释放，其他-退出程序）：");

scanf("%d", &tag);

switch(tag)

{

case 1:

alloc();

show();

break;

case 2:

freeNode();

show();

break;

default:

printf("程序已退出！");

}

}

}