实验__三__题目__**动态分区分配方式的模拟**__第_周星期

一、实验目的

了解动态分区分配方式中的数据结构和分配算法，并进一步加深对动态分区存储管理方式及其实现过程的理解

二、实验内容和要求

（1）用 C 语言分别实现采用首次适应算法和最佳适应算法的动态分区分配过程和回收过程。其中，空闲分区通过空闲分区链（表）来管理；在进行内存分配时，系统优先使用空闲区低端的空间。

（2）假设初始状态下，可用的内存空间为 640KB，并有下列的请求序列：

•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

•作业 8 申请 60KB

请分别采用首次适应算法和最佳适应算法进行内存的分配和回收，要求每次分配和回收后显示出空闲内存分区链的情况。

三、实验主要仪器设备和材料

实验环境

硬件环境：个人台式机 Windows7

软件环境：C 语言编程环境,vs 2010

四、实验设计方案及原理

1、循环首次适应算法

在该算法中，把主存中所有空闲区按其物理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表或链中。

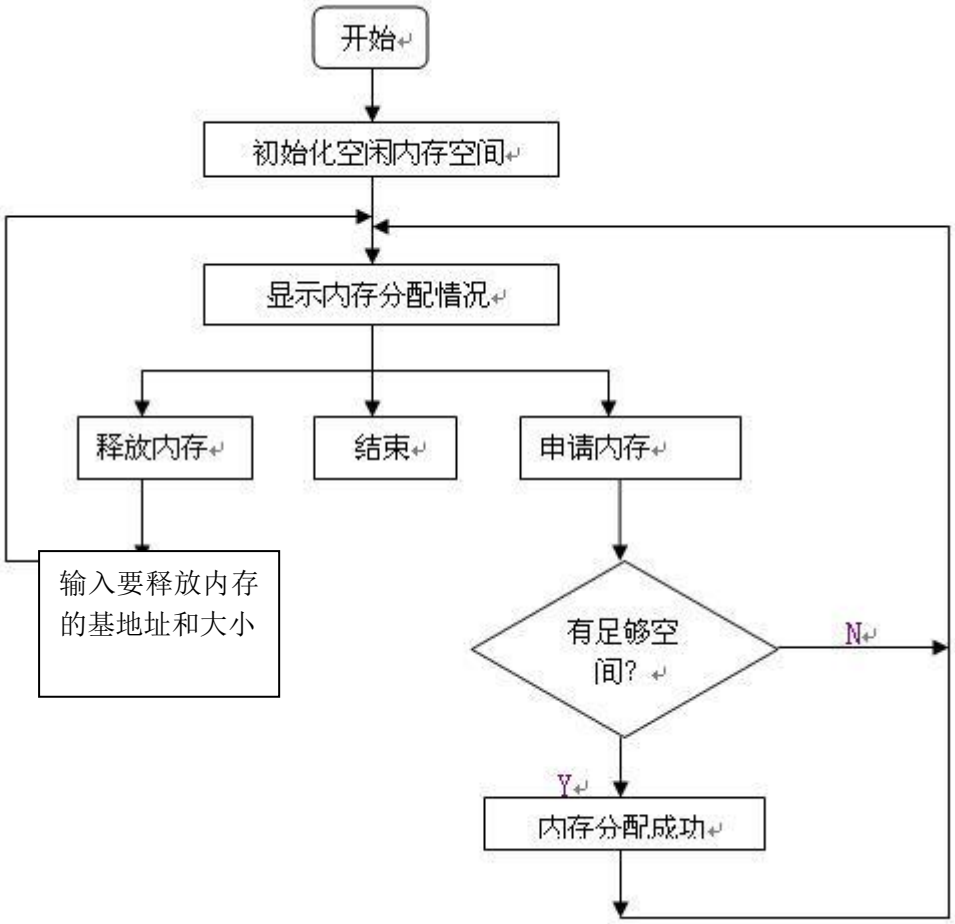
1、 实验步骤

（1）初始化空闲分区；

（2）反复对现有的空闲分区进行进程创建和撤消，即内存分配和回收；

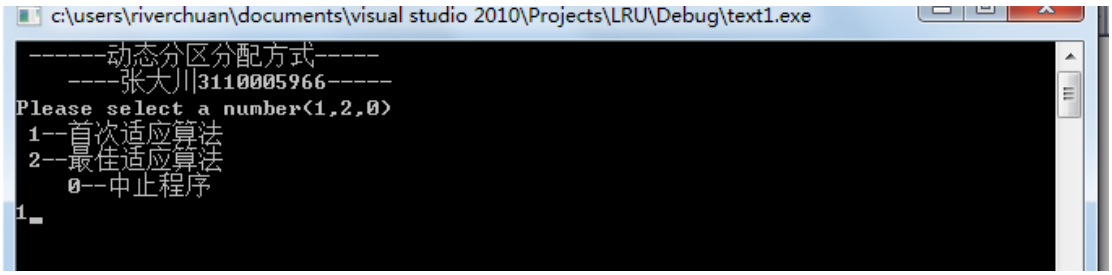
（3）退出。

3、流程图



五、实验结果及分析

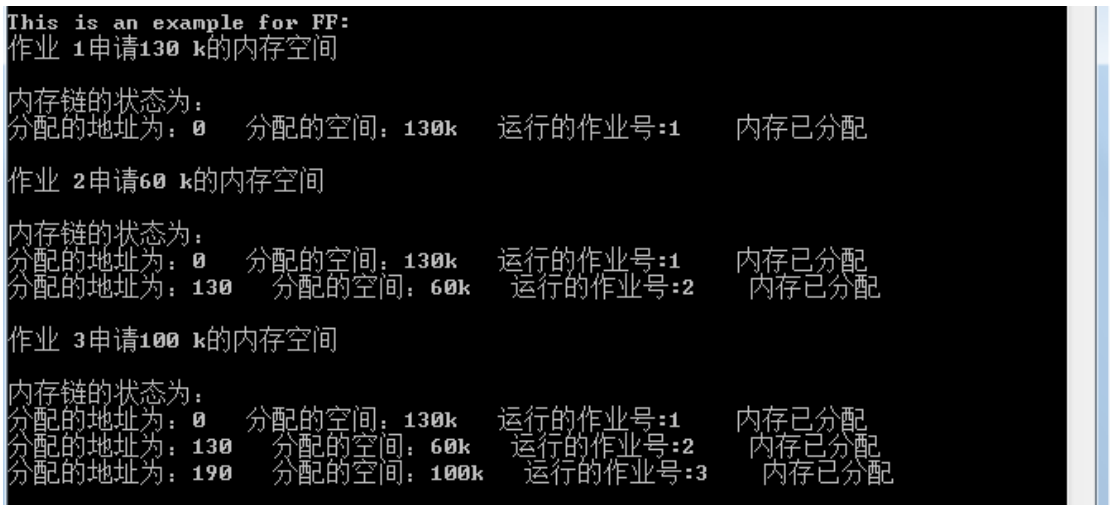
打开程序，运行界面如下：



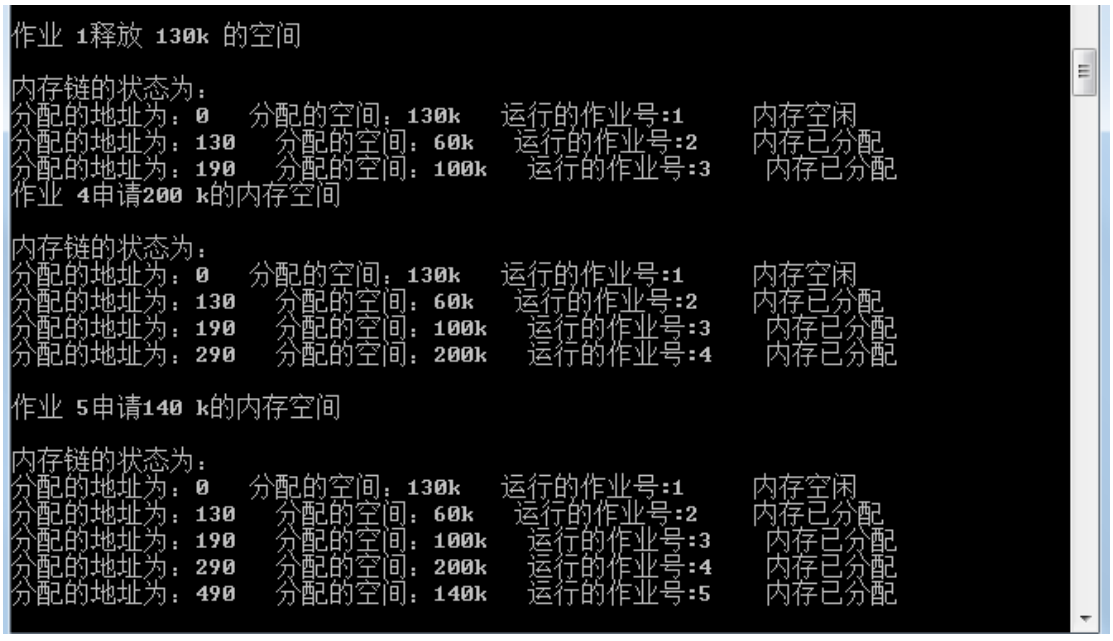
按照题目所给的作业请求序列，对程序进行运行，以下为运行结果：

首次适应算法部分：

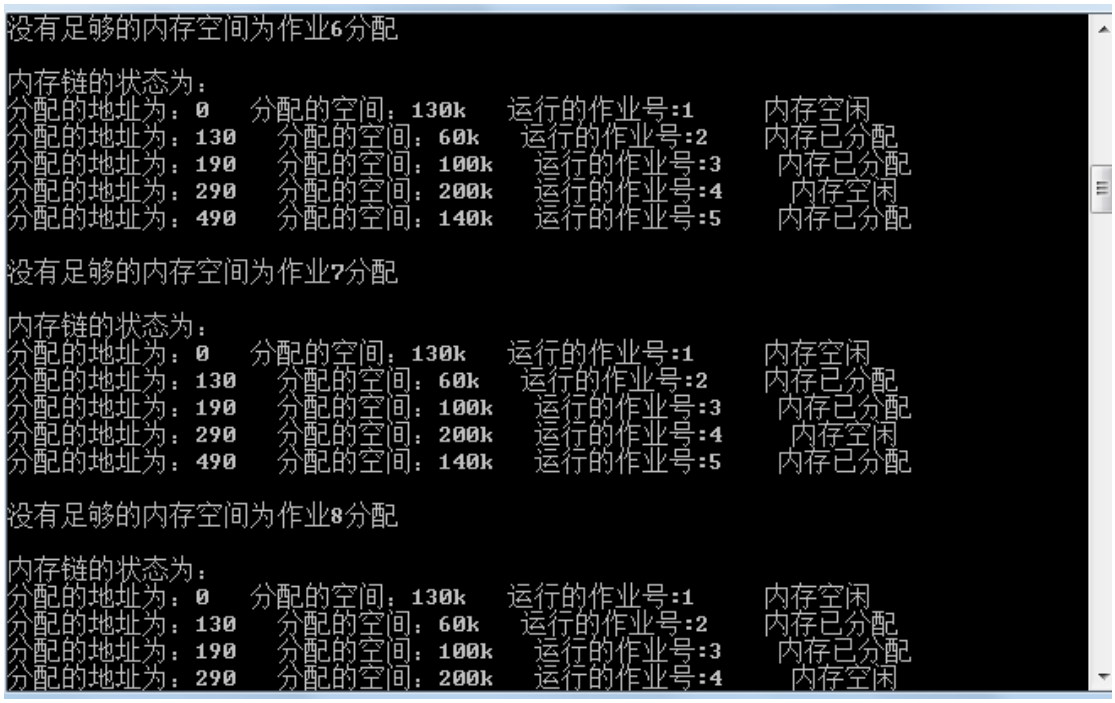
1. 作业 1、2、3 分别申请内存空间



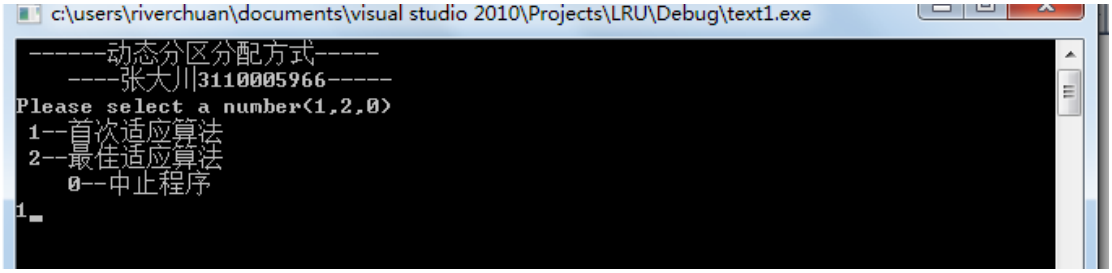
2. 作业 1 释放空间，作业 4、5 申请空间



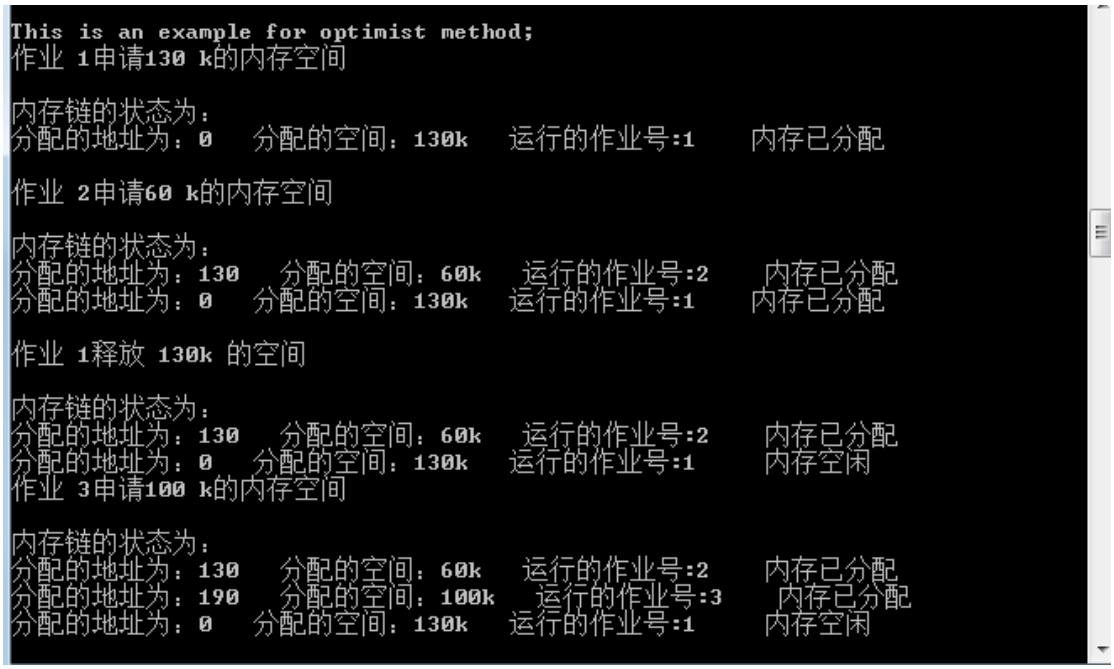
3. 作业 4 释放空间。此时内存不足，不能分配为作业 6、7、8 分配空间。



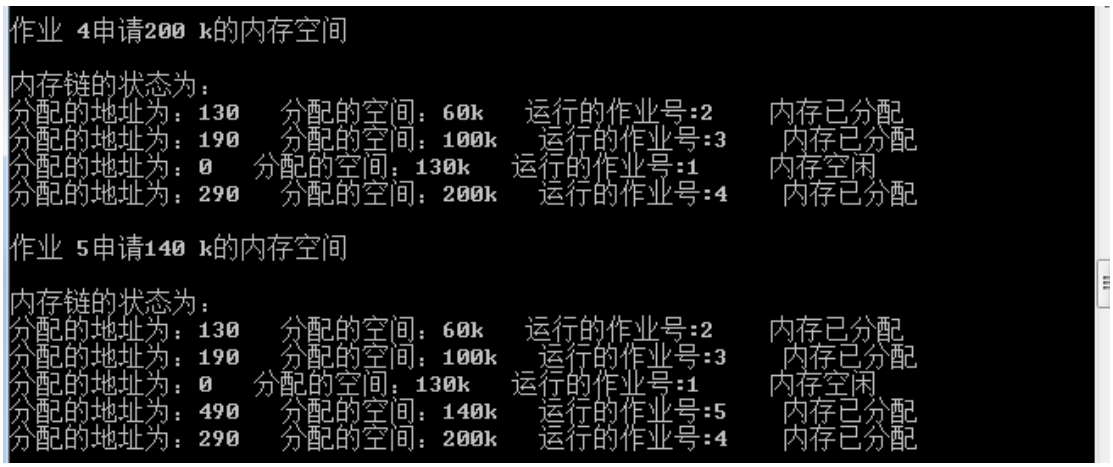
最佳适应算法部分：



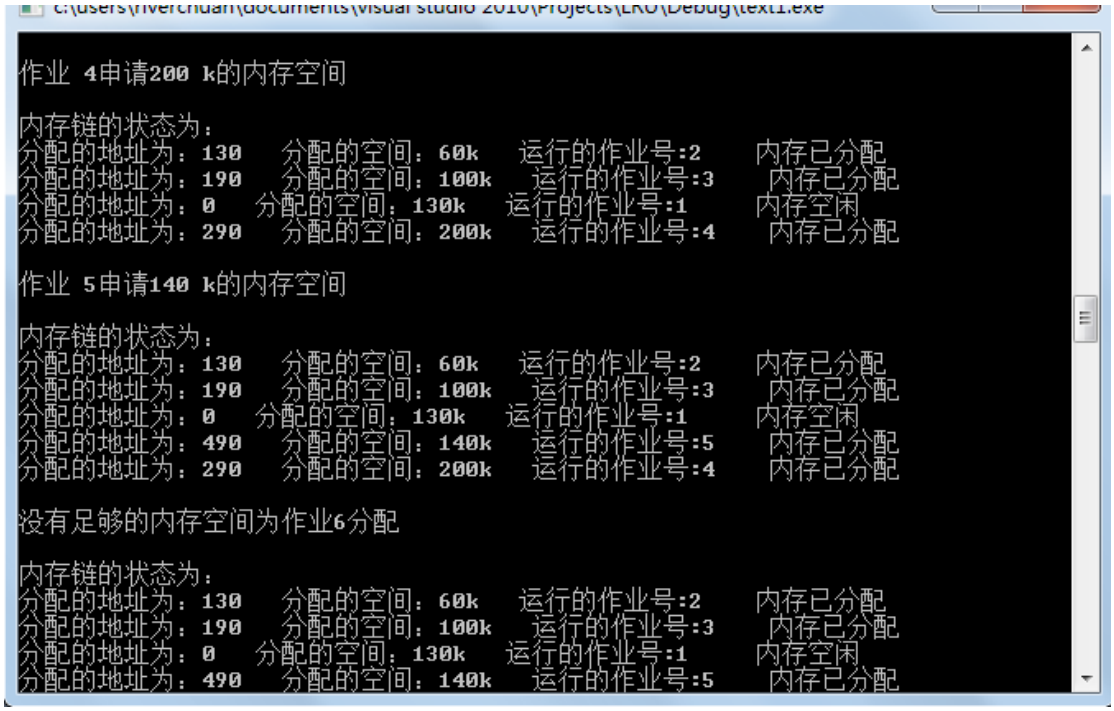
1. 从首先适应算法退出，按照最佳适应算法，为作业 2 申请内存空间，释放作业 1 的空间，为作业 3 申请空间。



2. 为作业 4、5 申请空间。



3. 作业 5、6、7 分别申请空间



六、调试总结及心得体会

通过这次的实验，我进一步加深了对内存管理的理解与应用，也掌握了固定式分区分配的存储管理。刚开始不知怎么做，最后经过本身的努力，终于把这个程序搞定了。知识层面亦得到了提升。

七、比较两种分配特点：

首次适应算法：该算法倾向于优先利用内存中低址部分的空闲分区，从而保留了高址部分的大空闲区，这给以后到达的大作业分配大的内存空间创造了条件。其缺点是低址部分不断被划分，会留下许多难以利用的很小的空闲空间，而每次查找又都是从低址部分开始的，这无疑会增加查找可用的空闲分区时的开销。

八、源程序代码：

```
#include <iostream>
#include <stdlib.h>
#include <time.h>

using namespace std;
struct memory
{
    struct memory *former;
    int address;//地址
    int num;//作业号
    int size;//分配内存大小
    int state;//状态0表示空闲1表示已分配
```

```
    struct memory *next;
};
typedef struct memory MEMORY;
MEMORY *mem;
const int size_min=10;//内存允许的最小空闲块的大小
bool is_optimist=false;//判断是否是最佳适应算法

void init();
void FF();
void alloc(MEMORY *,MEMORY *);//首次适应算法分配内存
void free(MEMORY *);//首次适应算法回收内存
void sort(MEMORY *);//对内存链进行排序
void insert(MEMORY *,MEMORY *);
void free_optimist(MEMORY *);
void print(MEMORY *);//打印内存链
void main()
{
    int i=0;
    while(1)
    {
        cout<<" ------动态分区分配方式-----"<<endl;
        cout<<"    ----张大川3110005966-----"<<endl;
        cout<<("Please select a number(1,2,0)");
        cout<<("\n 1--首次适应算法");
        cout<<"\n 2--最佳适应算法"<<endl;
        cout<<"    0--中止程序"<<endl;
        cin>>i;

        if(i==1)
        {
            cout<<("\nThis is an example for FF:\n");
            is_optimist=false;
            init();
            FF();
        }
        else if(i==2)
        {
            cout<<"\nThis is an example for optimist method;\n";
            is_optimist=true;
            init();
            FF();

        }
        else if(i==0)
```

```
        {
            exit(1);
        }
    }
}
void init()
{
    mem=new MEMORY;
    mem->size=640;
    //mem->state=0;
    mem->former=0;
    mem->next=0;
}
void FF()//首次适应算法
{
    int i;
    int work[]={130,60,100,200,140,60,50,60};//作业序列
    //int assignment;
    MEMORY *running;
    for(i=0;i<sizeof(work)/sizeof(int);i++)
    {
        running=(MEMORY *)malloc(sizeof(MEMORY));//初始化作业
        if(running!=NULL)
        {
            running->former=NULL;
            running->address=0;
            running->num=i+1;
            running->size=work[i];
            running->state=0;
            running->next=NULL;
            //cout<<"作业初始化成功"<<running->num<<endl;
            if(is_optimist==true)//最佳适应算法
            {
                //cout<<"xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"<<endl;
                alloc(mem,running);
            }
            else//首次适应算法
            {
                alloc(mem,running);
            }
            print(mem);
            cout<<endl;
        }
        else
```

```
                cout<<"没有足够的内存空间"<<endl;


            if(rand()%3==1)
            {
                if(is_optimist==false)//首次适应算法
                {
                    free(mem);
                }
                else//最佳适应算法
                {
                    ::free_optimist(mem);
                }
            }
        }
}
void free(MEMORY *ptr)//作业处理完后释放内存空间
{
    MEMORY *previous,*current;
    previous=ptr;
    current=previous->next;
    while(current!=NULL)
    {
        if(current->state==1&&rand()%3==1)
        {
            break;
        }
        previous=current;
        current=current->next;
    }
    if(current==NULL)
    {
        //cout<<"内存中没有任何作业!!! "<<endl;
        return;
    }
    else if(current->next==NULL)
    {
        if(previous->state==0)
        {
            MEMORY *temp;
            temp=current;
            previous->size=previous->size+current->size;
            previous->next=NULL;
            cout<<"作业 "<<(current->num)<<"释放 "<<(current->size)<<"k 的空间"<<endl;
```

```
            delete temp;
            print(mem);
        }
        else
        {
            current->state=0;
            cout<<"作业 "<<(current->num)<<"释放 "<<(current->size)<<"k 的空间"<<endl;
            print(mem);
        }
    }
    else if((current->next)->next==NULL)
    {
        if(previous->state==0&&(current->next)->state==0)
        {
            MEMORY *temp1,*temp2;
            temp1=current;
            temp2=current->next;
            previous->size=previous->size+current->size+(current->next)->size;
            previous->next=NULL;
            cout<<"作业 "<<(current->num)<<"释放 "<<(current->size)<<"k 的空间"<<endl;
            delete temp1;
            delete temp2;
            print(mem);
        }
        else if(previous->state==0)//释放的地址空间前面有空闲块则把它和前面的合并
        {
            MEMORY *temp;
            temp=current;
            previous->size=previous->size+current->size;
            (current->next)->former=previous;
            previous->next=current->next;
            cout<<"作业 "<<(current->num)<<"释放 "<<(current->size)<<"k 的空间"<<endl;
            delete temp;
            print(mem);
        }
        else if((current->next)->state==0)//释放的地址空间后面有空闲块则把它和后面的空
闲块合并
        {
            MEMORY *temp;
            temp=current->next;
            current->size=current->size+(current->next)->size;
            current->state=0;
            current->next=NULL;
            cout<<"作业 "<<(current->num)<<"释放 "<<(current->size)<<"k 的空间"<<endl;
```

```
            delete temp;
            print(mem);

        }
        else//处理完的作业前后都没有空闲块时直接把它的状态改为没分配
        {
            current->state=0;
            cout<<"作业 "<<(current->num)<<"释放 "<<(current->size)<<"k 的空间"<<endl;
            print(mem);
        }

    }
    else
    {
        if(previous->state==0&&(current->next)->state==0)
        {
            MEMORY *temp1,*temp2;
            temp1=current;
            temp2=current->next;
            previous->size=previous->size+current->size+(current->next)->size;
            ((current->next)->next)->former=previous;
            previous->next=(current->next)->next;

            cout<<"作业 "<<(current->num)<<"释放 "<<(current->size)<<"k 的空间"<<endl;
            delete temp1;
            delete temp2;
            print(mem);
        }
        else if(previous->state==0)//释放的地址空间前面有空闲块则把它和前面的合并
        {
            MEMORY *temp;
            temp=current;
            previous->size=previous->size+current->size;
            (current->next)->former=previous;
            previous->next=current->next;
            cout<<"作业 "<<(current->num)<<"释放 "<<(current->size)<<"k 的空间"<<endl;
            delete temp;
            print(mem);
        }
        else if((current->next)->state==0)//释放的地址空间后面有空闲块则把它和后面的空
闲块合并
        {
            MEMORY *temp;
```

```
            temp=current->next;
            current->size=current->size+(current->next)->size;
            current->state=0;
            ((current->next)->next)->former=current;
            current->next=(current->next)->next;

            cout<<"作业 "<<(current->num)<<"释放 "<<(current->size)<<"k 的空间"<<endl;

            delete temp;
            print(mem);

        }
        else//处理完的作业前后都没有空闲块时直接把它的状态改为没分配
        {
            current->state=0;
            cout<<"作业 "<<(current->num)<<"释放 "<<(current->size)<<"k 的空间"<<endl;
            print(mem);
        }
    }
}

void alloc(MEMORY *ptr,MEMORY *assign)//内存分配
{
    if(ptr->next==NULL)//内存没有作业运行
    {
        if(ptr->size>=assign->size)//内存空间大于作业所需空间
        {
            ptr->size=ptr->size-assign->size;//为内存分配空间
            assign->state=1;
            ptr->next=assign;
            assign->former=ptr;

            cout<<"作业 "<<(assign->num)<<"申请"<<(assign->size)<<" "<<"k的内存空间
"<<endl;
        }
        else
        {
            cout<<"没有足够的内存空间为作业"<<(assign->num)<<"分配"<<endl;
            delete assign;
        }
    }
    else//内存中如果已经分配了空间
    {
        MEMORY *previous,*current;
```

```
        previous=ptr;
        current=previous->next;
        while(current!=NULL)
        {
            if(current->size>assign->size&&ptr->state==0)//如果当前内存空间大于作业所需空间并且内存没有被分配
            {
                break;
            }
            previous=current;
            current=current->next;
        }

        if(current==NULL)//空闲链中没有为作业分配所需的空间
        {

            if(ptr->size>=assign->size)//内存中还有足够没分配的空间为此作业分配
            {
                assign->address =640-(ptr->size);//max+size_offset;//作业在内存中的首地址
                ptr->size=ptr->size-assign->size;
                assign->state=1;
                assign->former=previous;
                previous->next=assign;


                cout<<"作业 "<<(assign->num)<<"申请"<<(assign->size)<<" "<<"k的内存空间"<<endl;
            }
            else
            {
                cout<<"没有足够的内存空间为作业"<<(assign->num)<<"分配"<<endl;
            }
        }
        else//空闲链中有可为此作业分配的空间
        {

            if((current->size-assign->size)<=size_min)//空闲链所具备的空间与作业所需空间大小差不多时
            {                                          //直接把整个空闲块的空间分配给作业否则从空闲块中
                current->num=assign->num;               //划出与作业等同的空间
                current->state=1;
                delete assign;//free(assign);
```

```
                cout<<"作业 "<<(current->num)<<"申请"<<(current->size)<<" "<<"k的内存
间"<<endl;
            }
            else//从空闲块中划分一块与作业大小等同的空间
            {
                current->size=current->size-assign->size;
                assign->state=1;
                assign->address=current->address+current->size;

                if(current->next==NULL)//此要分配的空间是空闲链的最后一个元素
                {
                    assign->former=current;
                    current->next=assign;
                }
                else
                {
                    assign->next=current->next;
                    (current->next)->former=assign;
                    assign->former=current;
                    current->next=assign;
                }
                cout<<"作业 "<<(assign->num)<<"申请"<<(assign->size)<<" "<<"k的内存空
间"<<endl;
            }
        }

    }

    if((ptr->next)->next!=NULL&&is_optimist==true)
        sort(ptr);//排序由空闲块从小到大
    //print(ptr);

}
void sort(MEMORY *ptr)
{
   MEMORY *temp=new MEMORY;
   temp->next=0;
   temp->former=0;

   while(ptr->next)
   {
   if((ptr->next)->next==NULL)//内存链中只有两个元素
   {
```

```
     MEMORY *p;
     p=ptr->next;
     ptr->next=NULL;
         insert(temp,p);
    }
    else//内存链中有多个元素
    {
       MEMORY *p;
          p=ptr->next;
       p->former=ptr;
       ptr->next=p->next;
      (p->next)->former=ptr;
    insert(temp,p);
    }

    }
    ptr->next=temp->next;
    (temp->next)->former=ptr;
    delete temp;
}
void insert(MEMORY *queue,MEMORY *item)
{
     MEMORY *previous,*current;
     previous=queue;
     current=previous->next;
     while(current!=NULL && item->size>=current->size)
     {
          previous=current;
          current=current->next;
     }

     if(previous==queue)//所要插入的元素最小
     {

          if(queue->next==NULL)//内存链中只有一个元素
          {
               item->next=0;
               queue->next=item;
               item->former=queue;
          }
          else//内存链中有多个元素
          {
               item->next=queue->next;
               (queue->next)->former=item;
```

```
            item->former=queue;
            queue->next=item;
        }
    }
    else//定位到要插入的元素
    {
        item->next=current;
        item->former=previous;
        if(current==NULL)
        {
            previous->next=item;
        }
        else
        {
            current->former=item;
            previous->next=item;
        }

    }
}
void free_optimist(MEMORY *ptr)
{
    MEMORY *previous,*current;
    previous=ptr;
    current=previous->next;
    while(current!=NULL)
    {
        if(current->state==1&&rand()%3==1)
        {
            break;
        }
        previous=current;
        current=current->next;
    }
    if(current==NULL)
    {
        //cout<<"内存中没有任何作业!!! "<<endl;
        return;
    }
    else if(current->next==NULL)
    {
        if(previous->state==0&&((previous->address+previous->size)==current->address))
        {
            MEMORY *temp;
```

```
            temp=current;
            previous->size=previous->size+current->size;
            previous->next=NULL;
            cout<<"作业 "<<(current->num)<<"释放 "<<(current->size)<<"k 的空间"<<endl;
            delete temp;
            print(mem);
        }
        else
        {
            current->state=0;
            cout<<"作业 "<<(current->num)<<"释放 "<<(current->size)<<"k 的空间"<<endl;
            print(mem);
        }
    }
    else if((current->next)->next==NULL)
    {

    if(previous->state==0&&(current->next)->state==0&&((previous->address+previous->siz
e)==current->address)&&((current->size+current->address)==(current->next)->address))
        {
            MEMORY *temp1,*temp2;
            temp1=current;
            temp2=current->next;
            previous->size=previous->size+current->size+(current->next)->size;
            previous->next=NULL;
            cout<<"作业 "<<(current->num)<<"释放 "<<(current->size)<<"k 的空间"<<endl;
            delete temp1;
            delete temp2;
            print(mem);
        }
        else
if(previous->state==0&&((previous->address+previous->size)==current->address))//释放的
地址空间前面有空闲块则把它和前面的合并
        {
            MEMORY *temp;
            temp=current;
            previous->size=previous->size+current->size;
            (current->next)->former=previous;
            previous->next=current->next;
            cout<<"作业 "<<(current->num)<<"释放 "<<(current->size)<<"k 的空间"<<endl;
            delete temp;
            print(mem);
        }
        else
```

```
if((current->next)->state==0&&((current->size+current->address)==(current->next)->address))//释放的地址空间后面有空闲块则把它和后面的空闲块合并
        {
            MEMORY *temp;
            temp=current->next;
            current->size=current->size+(current->next)->size;
            current->state=0;
            current->next=NULL;
            cout<<"作业 "<<(current->num)<<"释放 "<<(current->size)<<"k 的空间"<<endl;

            delete temp;
            print(mem);
        }
    }
    else
    {

    if(previous->state==0&&(current->next)->state==0&&((previous->address+previous->size)==current->address)&&((current->size+current->address)==(current->next)->address))
        {
            MEMORY *temp1,*temp2;
            temp1=current;
            temp2=current->next;
            previous->size=previous->size+current->size+(current->next)->size;
            ((current->next)->next)->former=previous;
            previous->next=(current->next)->next;

            cout<<"作业 "<<(current->num)<<"释放 "<<(current->size)<<"k 的空间"<<endl;
            delete temp1;
            delete temp2;
            print(mem);
        }
        else
if(previous->state==0&&(previous->address+previous->size)==current->address)//释放的地址空间前面有空闲块则把它和前面的合并
        {
            MEMORY *temp;
            temp=current;
            previous->size=previous->size+current->size;
            previous->state=0;
            (current->next)->former=previous;
            previous->next=current->next;
            cout<<"作业 "<<(current->num)<<"释放 "<<(current->size)<<"k 的空间"<<endl;
            delete temp;
```

```
            print(mem);
        }
        else
if((current->next)->state==0&&((current->size+current->address)==(current->next)->addre
ss))//释放的地址空间后面有空闲块则把它和后面的空闲块合并
        {
            MEMORY *temp;
            temp=current->next;
            current->size=current->size+(current->next)->size;
            current->state=0;
            ((current->next)->next)->former=current;
            current->next=(current->next)->next;

            cout<<"作业 "<<(current->num)<<"释放 "<<(current->size)<<"k 的空间"<<endl;

            delete temp;
            print(mem);

        }
        else//处理完的作业前后都没有空闲块时直接把它的状态改为没分配
        {
            current->state=0;
            cout<<"作业 "<<(current->num)<<"释放 "<<(current->size)<<"k 的空间"<<endl;
            print(mem);
        }
    }
    if((ptr->next)->next!=NULL)
        sort(ptr);//排序由空闲块从小到大
}
void print(MEMORY *ptr)
{
    MEMORY *temp;
    temp=ptr->next;
    cout<<"\n内存链的状态为："<<endl;
    while(temp!=NULL)
    {
        cout<<"分配的地址为："<<temp->address<<"   分配的空间："<<temp->size<<"k"
            <<"   运行的作业号:"<<temp->num;
        if(temp->state==0)
        {
            cout<<"     内存空闲";
        }
        else
        {
```

```
            cout<<"    内存已分配";
        }
        cout<<endl;
        temp=temp->next;
    }
}
```