实验四 模拟内存分配与回收

一、实验目的

用高级语言编写和调试一个简单的内存分配与回收程序，模拟内存分配与回收的工作过程。从而对内存分配与回收的实质内容和执行过程有比较深入的了解。

二、实验内容

任务：设计并实现一个简单的内存分配与回收程序

在动态分区存储管理方式中，主要的操作是分配内存和回收内存。

1） 分配内存

程序采用某种分配算法，从空闲分区表中找到所需大小的分区。设请求分区的大小为u.size，表中每个空闲分区的大小可表示为m.size。若 m.size- u.size≤size（size 是事先规定的不再切割的剩余分区的大小），说明多余部分太小，可不再切割，将整个分区分配给请求者；否则，从该分区中按请求的大小划分出一块内存空间分配出去，余下的部分仍然留在空闲分区表中。然后，将分配区的起始地址返回给调用者。

2） 回收内存

当进程运行完毕释放内存时，程序根据回收区的首地址，从空闲区表中找到相应的插入点，此时，可能出现以下四种情况之一：

①回收区与插入点的前一个空闲分区 F1 相邻接，见图 4(a)。此时应将回收区与插入点的前一个分区合并，不必为回收区分配新表项，而只需修改前一分区 F1 的大小。

②回收区与插入点的后一个空闲分区 F2 相邻接，见图 4(b)。此时也可将两分区合并，形成新的空闲分区，但用回收区的首地址作为

新空闲区的首地址，大小为两者之和。

③回收区同时与插入点的前、后两个分区邻接，见图 4(c)。此时将三个分区合并，使用 F1 的表项和 F1 的首地址，取消 F2 的表项，大小为三者之和。

④回收区既不与 F1 相邻，又不与 F2 邻接。这时应为回收区单独建立一新表项，填写回收区的首地址和大小。

```
#include<iostream>
#include<vector>
using namespace std;
class Memory {
    struct Date {
        int size;//分区大小
        int head;//分区始址
        bool Free = true;//空闲状态
        Date(int head, int size) {
            this->head = head;
            this->size = size;
        }
    };
    int size;//内存大小
    int MIN_SIZE = 3;//最小剩余分区大小
    vector<Date> mPartition;//内存分区
    int location;//上次分配的空闲区位置
public:
    Memory() {
        this->size = 100;//默认内存大小为 100kb
        this->location = 0;//默认上次分配的空闲区位置为 0
```

```
        mPartition.push_back(Date(0, size));
    }
    Memory(int size) {
        this->size = size;
        this->location = 0;
        mPartition.push_back(Date(0, size));
    }

void allocation(int size) {
        for (location = 0; location < mPartition.size(); location++) {
            Date temp = mPartition[location];
            if (temp.Free && (temp.size > size)) {
                Distribute(size, location, temp);
                return;
            }
        }
        cout << "无可用内存空间!\n";
    }

void Distribute(int size, int location, Date temp) {
        //如果分隔后分区剩余大小过小，则将分区全部分配，否则分隔为两个分区
        if (temp.size - size <= MIN_SIZE) {
            temp.Free = false;
        }
        else {
            Date split = Date(temp.head + size, temp.size - size);
            mPartition.insert(mPartition.begin() + location, split);
```

```
            temp.size = size;
            temp.Free = false;
        }
        cout << "成功分配" << size << "KB 的内存！ \n";
    }
    void showmpartition() {
    cout<<"*********************************************\n";
        cout << "分区编号\t 分区始址\t 分区大小\t 空闲状态\t\n";
        for (int i = 0; i < mPartition.size(); i++) {
            Date temp = mPartition[i];
            cout << i << "\t\t" << temp.head << "\t\t" << temp.size <<
"\t\t" << temp.Free << endl;
        }
    cout<<"*********************************************\n";
    }

    void collection(int id) {
        if (id >= mPartition.size()) {
            cout << "没有此分区号！ \n";
            return;
        }
        Date temp = mPartition[id];
        int size = temp.size;
        if (temp.Free) {
            cout << "分区未被分配，无需回收";
            return;
        }
        //如果回收分区不是尾分区且后一个分区为空闲，则与后一个
```

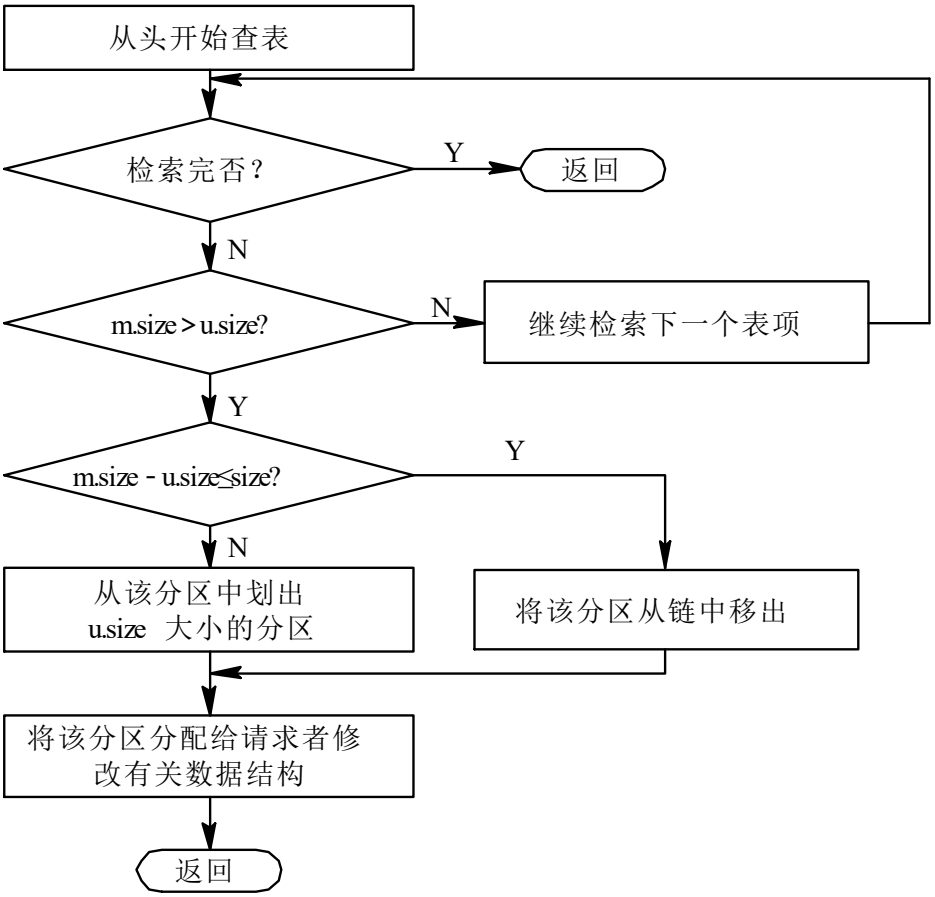
```
分区合并（上不邻下邻或者上下都邻）
        if (id < mPartition.size() - 1 && mPartition[id + 1].Free) {
            Date next = mPartition[id + 1];
            temp.size += next.size;
            mPartition.erase(mPartition.begin() + id + 1);
        }
        //如果回收分区不是首分区且前一个分区为空闲，则与前一个
分区合并（上邻下不邻或者上下都邻）
        if (id > 0 && mPartition[id - 1].Free) {
            Date previous = mPartition[id - 1];
            previous.size += temp.size;
            mPartition.erase(mPartition.begin() + id);

            id--;
        }
        mPartition[id].Free = true;
        cout << "内存回收成功!，本次回收了 " << size << "KB 空
间!"<<endl;
    }

};

int main() {
    cout << "请初始化内存大小：";
    int size;
    cin >> size;
    Memory memory(size);
    memory.showmpartition();
```

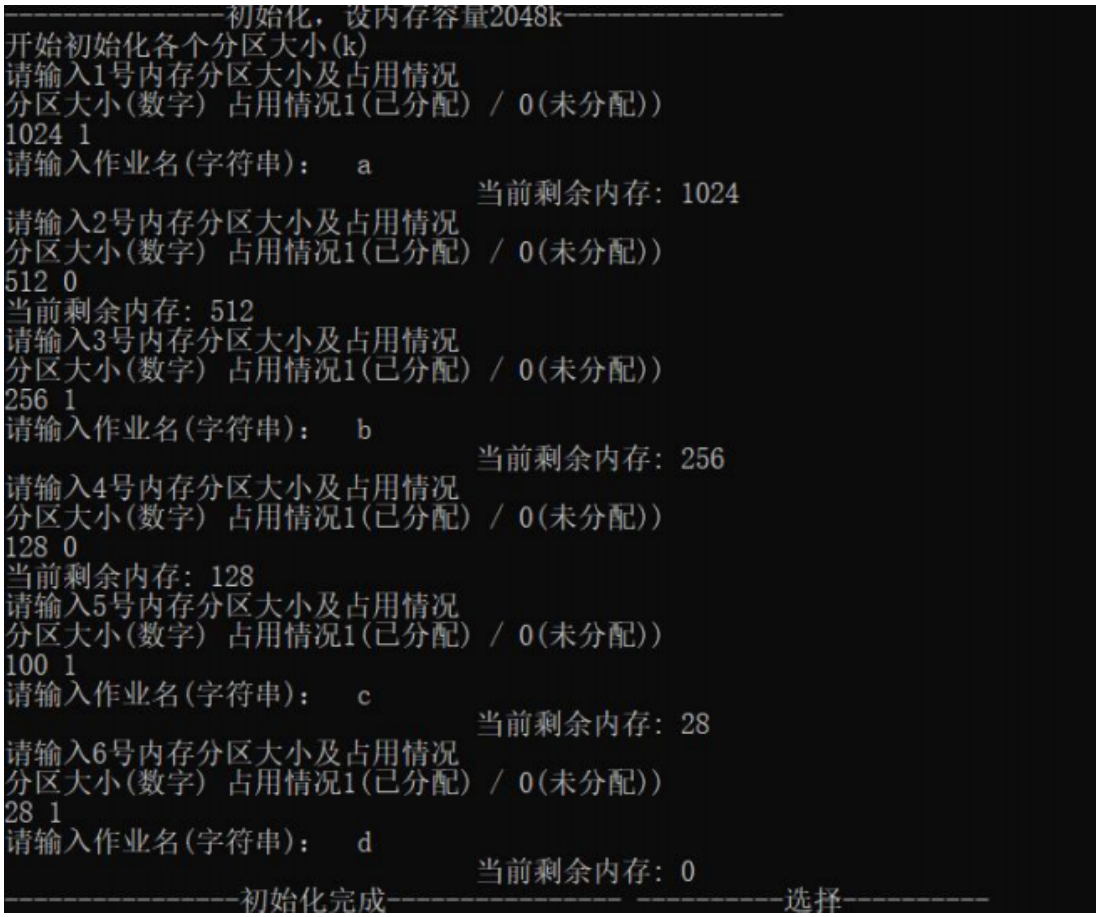
```
    while (true) {
        cout << "1.申请分配空间\n" << "2.回收已分配空间\n" << "3.显
示分区状态\n";
        cout << "请选择指令:";
        cin >> size;
        switch (size)
        {
        case 1:
            cout << "请输入需要申请的空间大小";
            cin >> size;
            memory.allocation(size);
            break;
        case 2:
            cout << "请输入需要回收的分区号：";
            cin >> size;
            memory.collection(size);
            break;
        case 3:
            memory.showmpartition();
            break;
        default:
            cout << "重新选择" << endl;
            break;
        }
    }
}
```

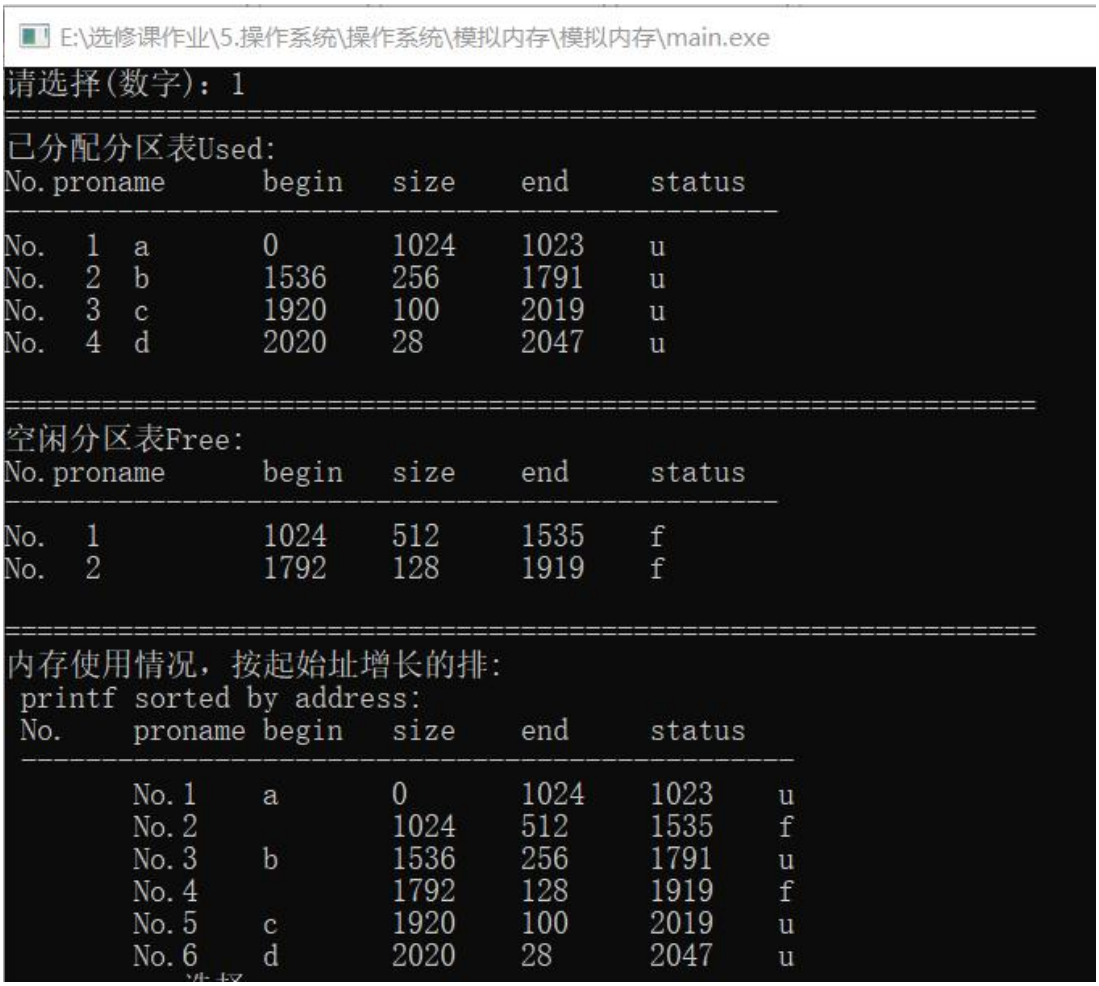


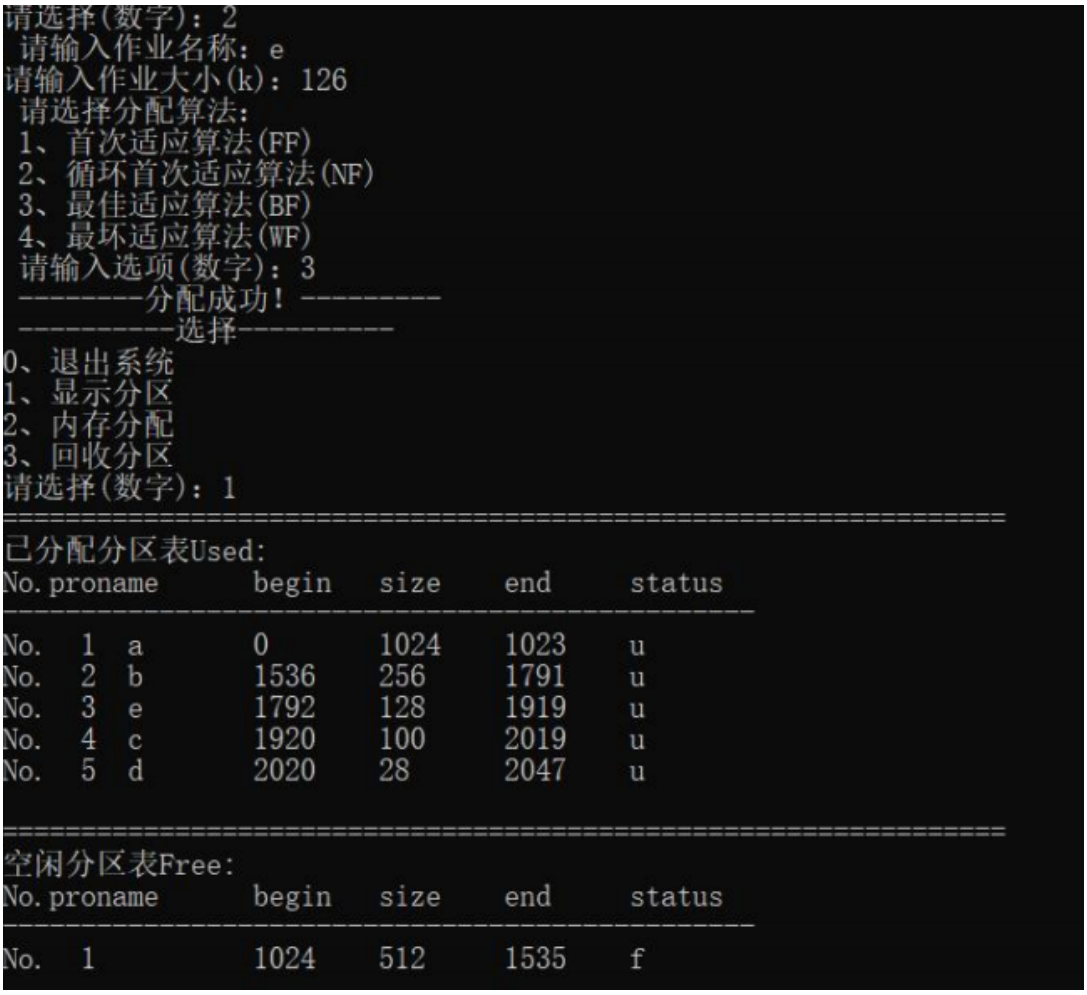
三、实验结果

3） 分配内存

程序采用某种分配算法，从空闲分区表中找到所需大小的分区。设请求分区的大小为u.size，表中每个空闲分区的大小可表示为m.size。若 m.size- u.size≤size（size 是事先规定的不再切割的剩余分区的大小），说明多余部分太小，可不再切割，将整个分区分配给请求者；否则，从该分区中按请求的大小划分出一块内存空间分配出去，余下的部分仍然留在空闲分区表中。然后，将分配区的起始地址返回给调用者。


----------初始化，设内存容量2048k----------
开始初始化各个分区大小(k)
请输入1号内存分区大小及占用情况
分区大小(数字) 占用情况1(已分配) / 0(未分配))
1024 1
请输入作业名(字符串)： a
当前剩余内存: 1024
请输入2号内存分区大小及占用情况
分区大小(数字) 占用情况1(已分配) / 0(未分配))
512 0
当前剩余内存: 512
请输入3号内存分区大小及占用情况
分区大小(数字) 占用情况1(已分配) / 0(未分配))
256 1
请输入作业名(字符串)： b
当前剩余内存: 256
请输入4号内存分区大小及占用情况
分区大小(数字) 占用情况1(已分配) / 0(未分配))
128 0
当前剩余内存: 128
请输入5号内存分区大小及占用情况
分区大小(数字) 占用情况1(已分配) / 0(未分配))
100 1
请输入作业名(字符串)： c
当前剩余内存: 28
请输入6号内存分区大小及占用情况
分区大小(数字) 占用情况1(已分配) / 0(未分配))
28 1
请输入作业名(字符串)： d
当前剩余内存: 0
----------初始化完成---------- ----------选择----------


E:\选修课作业\5.操作系统\操作系统\模拟内存\模拟内存\main.exe
请选择(数字)：1
已分配分区表Used:
No. proname begin size end status
No. 1 a 0 1024 1023 u
No. 2 b 1536 256 1791 u
No. 3 c 1920 100 2019 u
No. 4 d 2020 28 2047 u
空闲分区表Free:
No. proname begin size end status
No. 1 1024 512 1535 f
No. 2 1792 128 1919 f
内存使用情况，按起始址增长的排:
printf sorted by address:
No. proname begin size end status
No. 1 a 0 1024 1023 u
No. 2 1024 512 1535 f
No. 3 b 1536 256 1791 u
No. 4 1792 128 1919 f
No. 5 c 1920 100 2019 u
No. 6 d 2020 28 2047 u


请选择(数字)：2
请输入作业名称：e
请输入作业大小(k)：126
请选择分配算法：
1、首次适应算法(FF)
2、循环首次适应算法(NF)
3、最佳适应算法(BF)
4、最坏适应算法(WF)
请输入选项(数字)：3
--------分配成功！---------
----------选择----------
0、退出系统
1、显示分区
2、内存分配
3、回收分区
请选择(数字)：1
已分配分区表Used:
No. proname begin size end status
No. 1 a 0 1024 1023 u
No. 2 b 1536 256 1791 u
No. 3 e 1792 128 1919 u
No. 4 c 1920 100 2019 u
No. 5 d 2020 28 2047 u
空闲分区表Free:
No. proname begin size end status
No. 1 1024 512 1535 f

4） 回收内存

当进程运行完毕释放内存时，程序根据回收区的首地址，从空闲区表中找到相应的插入点，此时，可能出现以下四种情况之一：

①回收区与插入点的前一个空闲分区 F1 相邻接，见图 4(a)。此时应将回收区与插入点的前一个分区合并，不必为回收区分配新表项，而只需修改前一分区 F1 的大小。

②回收区与插入点的后一个空闲分区 F2 相邻接，见图 4(b)。此时也可将两分区合并，形成新的空闲分区，但用回收区的首地址作为新空闲区的首地址，大小为两者之和。

③回收区同时与插入点的前、后两个分区邻接，见图 4(c)。此时将三个分区合并，使用 F1 的表项和 F1 的首地址，取消 F2 的表项，大小为三者之和。

④回收区既不与 F1 相邻，又不与 F2 邻接。这时应为回收区单独建立一新表项，填写回收区的首地址和大小。

```
3、回收分区
请选择(数字)：3
请输入回收的内存分区号(数字)：1
                        --------回收成功！-------- ----------选择----------
0、退出系统
1、显示分区
2、内存分配
3、回收分区
请选择(数字)：1
============================================================
已分配分区表Used:
No.proname      begin   size    end     status
------------------------------------------------
No.   1  b       1536    256     1791    u
No.   2  e       1792    128     1919    u
No.   3  c       1920    100     2019    u
No.   4  d       2020    28      2047    u

============================================================
空闲分区表Free:
No.proname      begin   size    end     status
------------------------------------------------
No.   1          0       1536    1535    f

============================================================
内存使用情况，按起始址增长的排:
 printf sorted by address:
 No.     proname begin   size    end     status
 ------------------------------------------------
        No.1            0       1536    1535    f
        No.2    b       1536    256     1791    u
        No.3    e       1792    128     1919    u
        No.4    c       1920    100     2019    u
        No.5    d       2020    28      2047    u
```