北 京 林 业 大 学

2019 学年—2020 学年第 2 学期操作系统实验报告书

专业：计算机科学与技术(创新实验班)　班级：计创 18

姓　　名：连月菡　　学　　号：181002222

实验地点：家　　任课教师：李巨虎

实验题目：实验 3　存储管理

实验环境：Visual Studio 2019 Community

一、 实验目的

1. 了解可变分区管理方式下如何进行主存的分配；
2. 了解可变分区管理方式下如何进行主存空间的回收。
3. 了解可变分区的调度算法分配策略。

二、 实验内容

可变分区调度算法有：最先适应分配算法，最优适应分配算法，最坏适应算法。用户提出内存空间的申请；系统根据申请者的要求，按照一定的分配策略分析内存空间的使用情况，找出能满足请求的空闲区，分给申请者；当程序执行完毕或主动归还内存资源时，系统要收回它所占用的内存空间或它归还的部分内存空间。

根据可变分区的主存分配思想，首先设计记录主存使用情况的数据表格，用来记录空闲区和作业占用的区域，即“已分配区表”和“空闲区表”。然后在数据表格上进行主存的分配，其主存分配算法选择一种算法。程序中可选择进行主存分配或主存回收，若是主存分配，输入作业名和所需主存空间大小；若是回收，输入回收作业的作业名，以循环进行主存分配和回收。

当要装入一个作业时，从空闲区表中查找标志为“未分配”的空闲区，从中找一个能容纳该作业的空闲区。如果找到的空闲区正好等于该作业的长度，则把该分区全部分配给该作业。这时应该把该空闲区登记栏中的标志改为“空”，同时在已分配区表中找到一个标志为“空”的栏目登记新装入作业所占用分区的起始地址、长度和作业名。如果找到的空闲区大于作业长度，则把空闲区分成两部分，一部分用来装入作业，另一部分仍然为空闲区，这时只要修改空闲区的长度，且把新装入的作业登记到已分配区表中。本实验中采用首次适应算法、循环首次适应算法、最坏适应算法中的一种为作业分配主存。

动态分区方式下回收主存空间时应该检查是否有与归还区相邻的空闲区域。若有，则应该合并成一个空闲区。一个归还区可能有上邻空闲区，也可能有下邻空闲区，或者既有上邻空闲区又有下邻空闲区，或者既无上邻空闲区也无下邻空闲区。

在实现回收时，首先将作业归还的区域在已分配表中找到，将该栏目的状态变为“空”，然后检查空闲区表中标志为“未分配”栏目，查找是否又相邻空闲区；最后合并空闲区，修改空闲区表。

三、 实验要求

由于动态分区的大小是由作业需求量决定的，故分区的长度是预先不固定的，且分区的个数也随主存分配和回收变动。总之，所有分区情况随时可能发生变化，数据表格的设计必须和这个特点相适应。由于分区长度不同，因此设计的表格应该包括分区在主存中的起始地址和长度。由于分配时，空闲区有时会变成两个分区：空闲区和已分分区，回收主存分区时，可能会合并空闲区，这样如果整个主存采用一张表格记录已分分区和空闲区，就会使表格操作繁琐。主存分配时查找空闲区进行分配，然后填写已分配区表，主要操作在空闲区；某个作业执行完后，将该分区变成空闲区，并将其与相邻的空闲区合并，主要操作也在空闲区。由此可见，主存的分配与回收主要时对空闲区的操作。这样为了便于对主存空间的分配与回收，就建立两张分区表记录主存的使用情况：“已分配区表”记录作业占用分区，“空闲区表”记录空闲区。

这两张表的实现方法一般由两种：链表形式、顺序表形式。在本实验中，可以采用任意一种形式。如果采用顺序表形式，用数组模拟。由于顺序表的长度必须提前固定，所以无论是“已分配区表”还是“空闲区表”都必须事先确定长度。它们的长度必须是系统可能的最大项数，系统运行过程中才不会出错，因此在多数情况下，无论是“已分配表区”还是“空闲区表”都是空闲栏目。已分配区表中除了分区起始地址、长度外，也至少还有一项“标志”，如果是空闲栏目，内容为“空”，如果为某个作业占用分区的登记项，内容为该作业的作业名；空闲区表除了分区起始地址、长度外，也要有一项“标志”，如果是空闲栏目，内容为“空”，如果为某个空闲区的登记项，内容为“未分配”。在实际系统中，这两个表格的内容可能还要多，实验中仅仅使用上述必须的数据。为此，“已分配区表”和“空闲区表”在实验中有如下的结构定义。

已分配区表的定义：

```
#define n 10                //假定系统允许的最大作业数量为 n
struct{
 float address;       //已分分区起始地址
 float length;        //已分分区长度，单位为字节
 int flag;           //已分配表区登记栏标志，用 0 表示空栏目，
}used_table[n];          //已分配区表
```

空闲区表的定义：

```
#define m 10                //假定系统允许的空闲区表最大为 m
struct{
 float address;       //空闲区起始地址
 float length;        //空闲区长度，单位为字节
 int flag;           //空闲区表登记栏目用 0 表示空栏目，1 表示未分配
}free_table[m];          //空闲区表
```

四、 实验结果

学号:181002222%3=0,以下内容按照最先适应分配算法实现。

宏定义,结构体,全局变量

```cpp
#include<iostream>
#include<stdlib.h>
using namespace std;
#define Free 0 //空闲状态
#define Busy 1 //已用状态
#define OK 1    //完成
#define ERROR 0 //出错
#define MAX_length 640  //定义最大主存信息640KB
typedef int Status;
int flag;//标志位  0为空闲区      1为已分配的工作区
typedef struct FreAarea//定义一个空闲区说明表结构
{
    long size;    //分区大小
    long address; //分区地址
    int state;    //状态
}ElemType;


typedef struct DuLNode// 线性表的双向链表存储结构
{
    ElemType data;
    struct DuLNode* prior; //前趋指针
    struct DuLNode* next;  //后继指针
}

DuLNode, * DuLinkList;
DuLinkList block_first; //头结点
DuLinkList block_last;  //尾结点
ElemType work[10];//作业队列
Status Alloc(int);//内存分配
Status free(int); //内存回收
Status First_fit(int);//首次适应算法
void show_free();//查看分配
Status Initblock();//开创空间表
Status Initwork();//初始化作业队列
```

初始化内存表

```
Status Initblock()//开创带头结点的内存空间链表
{
    block_first = (DuLinkList)malloc(sizeof(DuLNode));
    block_last = (DuLinkList)malloc(sizeof(DuLNode));
    block_first->prior = NULL;
    block_first->next = block_last;
    block_last->prior = block_first;
    block_last->next = NULL;
    block_last->data.address = 0;
    block_last->data.size =rand()%(MAX_length/5)+1;
    block_last->data.state = Free;
    DuLNode* pre = block_last;
    DuLNode* p;
    for (int i = 0; i < 9; ++i)
    {
        p = (DuLinkList)malloc(sizeof(DuLNode));
        pre->next=p;
        p->prior = pre;
        p->next = NULL;
        p->data.address = p->prior->data.address + p->prior->data.size;
        if(i==8)
            p->data.size =MAX_length-p->data.address;
        else
            p->data.size = rand() % (MAX_length / 6) + 1;
        p->data.state = Free;
        pre = p;
    }
    return OK;
}
```

初始化内存表

```cpp
Status Initwork()//初始化作业队列
{
    cout << "此次作业队列:" << endl;
    cout << "作业名" << "\t内存大小" << " 申请的操作" << endl;
    cout << "+++++++++++++++++++++++++++++++" << endl;
    for (int i = 0; i < 10; ++i)
    {
        work[i].address = i;
        work[i].state = rand()%2;//要求申请或者释放
        work[i].size = rand()%(MAX_length/5)+1;
    }
    int pos = 0;
    for (int i = 0; i < 10; ++i)//为了保持释放和分配的大小一致
    {
        if (work[i].state == 0)
        {
            int boo = 0;
            for (int j = pos + 1; j < 10; j++)
            {
                if (work[j].state == 1)
                {
                    work[i].size = work[j].size;
                    work[i].address = work[j].address;
                    pos = j;
                    boo = 1;
                    break;
                }
            }
            if (!boo) work[i].state = 1;
        }
    }
    for (int i = 0; i < 10; ++i)
    {
        cout << work[i].address << "\t" << work[i].size << "\t";
        if (work[i].state == 1) cout << "待分配" << endl;
        else  cout << "待释放" << endl;
    }
    return OK;
}
```

分配主存以及最初适应算法

```cpp
Status Alloc(int request)//分配主存
{
    if (request < 0 || request == 0)
    {
        cout << "分配大小不合适，请重试！" << endl;
        return ERROR;
    }
    if (First_fit(request) == OK) cout << "分配成功！" << endl;
    else cout << "内存不足，分配失败！" << endl;
    return OK;
}

Status First_fit(int request)//最初适应算法
{
    //为申请作业开辟新空间且初始化
    DuLinkList temp = (DuLinkList)malloc(sizeof(DuLNode));
    temp->data.size = request;
    temp->data.state = Busy;

    DuLNode* p = block_first->next;
    while (p)
    {
        if (p->data.state == Free && p->data.size == request)
        {//有大小恰好合适的空闲块
            p->data.state = Busy;
            return OK;
            break;
        }
        if (p->data.state == Free && p->data.size > request)
        {//有空闲块能满足需求且有剩余
            temp->prior = p->prior;
            temp->next = p;
            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 -= request;
            return OK;
            break;
        }
        p = p->next;
    }
    return ERROR;
}
```

主存回收

```
Status free(int flag)//主存回收
{
    DuLNode* p = block_first;
    while (p != NULL) {
        if (p->data.size==flag)//找到一块被占用的内存
            break;
        p = p->next;
    }
    if (p == NULL)
        return ERROR;
    if (p->prior != block_first && p->prior->data.state == Free)//与前面的空闲块相连
    {
        p->prior->data.size += p->data.size;//空间扩充,合并为一个
        p->prior->next = p->next;//去掉原来被合并的p
        p->next->prior = p->prior;
        p = p->prior;
    }
    if (p->next != block_last && p->next->data.state == Free)//与后面的空闲块相连
    {
        p->data.size += p->next->data.size;//空间扩充,合并为一个
        p->next->next->prior = p;
        p->next = p->next->next;
    }
    if (p->next == block_last && p->next->data.state == Free)//与最后的空闲块相连
    {
        p->data.size += p->next->data.size;
        p->next = NULL;
    }

    return OK;
}

```

主存回收

```
1
2 void show_free()//显示空闲情况
3 {
4     int flag = 0;
5     cout << "\n空闲表:\n";
6     cout << "+++++++++++++++++++++++++++++++++++++++++++++++\n\n";
7     DuLNode* p = block_first->next;
8     cout << "分区号\t起始地址\t分区大小\t状态\n\n";
9     while (p)
10    {
11        if (p->data.state != Free)
12        {
13            p = p->next;
14            continue;
15        }
16        cout <<flag++ << "\t";
17        cout << p->data.address << "\t\t";
18        cout << p->data.size << "KB\t\t";
19        if (p->data.state == Free) cout << "空闲\n\n";
20        p = p->next;
21    }
22    cout << "+++++++++++++++++++++++++++++++++++++++++++++++\n\n";
23 }
24
25 void show_allo()//显示分配情况
26 {
27    int flag = 0;
28    cout << "\n分配表:\n";
29    cout << "+++++++++++++++++++++++++++++++++++++++++++++++\n\n";
30    DuLNode* p = block_first->next;
31    cout << "分区号\t起始地址\t分区大小\t状态\n\n";
32    while (p)
33    {
34        if (p->data.state == Free)
35        {
36            p = p->next;
37            continue;
38        }
39        cout << flag++ << "\t";
40        cout << p->data.address << "\t\t";
41        cout << p->data.size << "KB\t\t";
42        cout << "已分配\n\n";
43        p = p->next;
44    }
45    cout << "+++++++++++++++++++++++++++++++++++++++++++++++\n\n";
46 }
47
```

主函数

```cpp
int main()//主函数
{
    Initblock(); //开创空间表
    int choice;  //操作选择标记
    Initwork(); //开创作业队列
    for (int i = 0; i < 10; ++i)
    {
        show_free();
        show_allo();
        cout << endl << "======================" << endl;
        choice = work[i].state;
        if (choice == 1) {

            Alloc(work[i].size); // 分配内存
            cout << "作业" << work[i].address << "分配了" << work[i].size << "KB"
<< endl;
        }
        else  // 内存回收
        {
            int flag;
            flag = work[i].size;
            int s=free(flag);
            if (s == ERROR) {
                cout << "释放失败,当前没有被分配的此容量大小的内存\n"; continue;
            }
            cout << "作业" << work[i].address << "释放了" << work[i].size << "KB"
<< endl;
        }
    }
    cout << endl << "本次程序已结束,以下是最后的内存分配情况！ " << endl;
    show_free();
    show_allo();
}
```

点击运行程序:

1.利用 srand(time(NULL)),rand()生成随机数 0-1 表示作业申请的操作类型

```
此次作业队列:
作业名   内存大小 申请的操作
++++++++++++++++++++++++++++++++
0        16       待分配
1        109      待分配
2        45       待分配
1        109      待释放
4        46       待分配
2        45       待释放
6        56       待分配
7        109      待分配
8        3        待分配
4        46       待释放
```

2. 同样地，随机划分空闲区块

空闲表:

分区号	起始地址	分区大小	状态
0	0	79KB	空闲
1	79	75KB	空闲
2	154	86KB	空闲
3	240	65KB	空闲
4	305	94KB	空闲
5	399	16KB	空闲
6	415	53KB	空闲
7	468	37KB	空闲
8	505	70KB	空闲
9	575	65KB	空闲

分配表:

分区号	起始地址	分区大小	状态

3.依次处理作业

```
==========
分配成功!
作业0分配了16KB
```

每次处理后，显示一遍空闲表和分配表

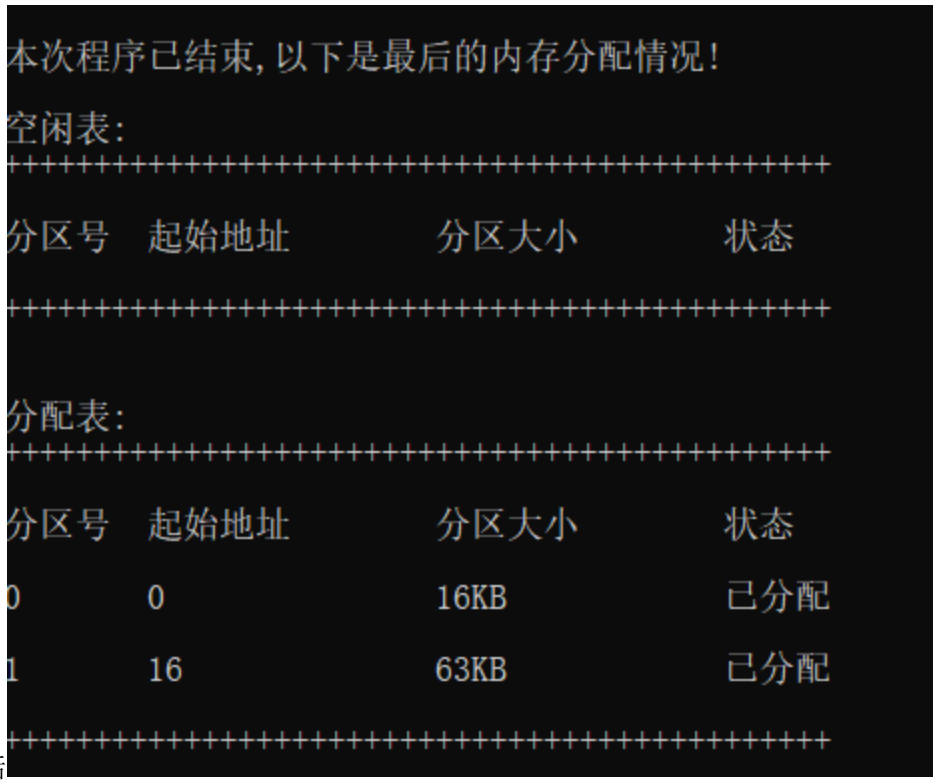
```
======================
分配成功!
作业0分配了16KB

空闲表:
++++++++++++++++++++++++++++++++++++++++++++++++
分区号   起始地址          分区大小          状态
0        16                63KB              空闲
1        79                75KB              空闲
2        154               86KB              空闲
3        240               65KB              空闲
4        305               94KB              空闲
5        399               16KB              空闲
6        415               53KB              空闲
7        468               37KB              空闲
8        505               70KB              空闲
9        575               65KB              空闲
++++++++++++++++++++++++++++++++++++++++++++++++

分配表:
++++++++++++++++++++++++++++++++++++++++++++++++
分区号   起始地址          分区大小          状态
0        0                 16KB              已分配
++++++++++++++++++++++++++++++++++++++++++++++++
```

有时会释放失败，因为当前占用的内存小于释放的内存大小

```
======================
释放失败,当前没有被分配的此容量大小的内存
```



最后

五、 实验心得

在写实验报告的过程中,发现代码仍有不完美之处，例如，使用 rand 函数，每次运行的结果都是一样的，说明没有使用 srand 来生成随机种子。而因为是链表存储的缘故，所以经常遇到指针指向 NULL，这种情况也需要用 if 等语句进行特殊处理。

通过这次实验，让我对首次适应算法有了更深刻的认识，提高了 C++代码的熟练程度，以及对内存的分配方法有了具象的理解。明白了书本和 PPT 中的知识如果转化成自身实践的代码形式，会有不一样的印象，跳跃出平面之外，有了立体生动的程序来重新叙述这个过程。