实验一 用循环首次适应法模拟操作系统的存储管理

1. 实验目的

1）熟悉操作系统的可变分区存储管理的思想，掌握首次适应法和循环首次适应法的思想；

2）复习 C 语言编程知识和数据结构的编程知识，用 C 语言程序模拟操作系统的存储管理。

2. 实验原理

主要的实验原理来源于操作系统存储管理的思想。模拟的方法是，向系统申请一些空间用来模拟内存分配，并建立空闲存储区表。随后通过键入 UNIX 命令，来实现模拟任务的空间分配和空间释放。

空间分配和释放的算法使用循环首次适应算法。其基本思想是：把空闲表设计成顺序结构或者链接结构的循环队列，空闲区的地址按地址从低到高的次序登记在空闲区的队列中，同时设置一个起始查找指针，初始时刻指向第一个表项。

每次分配存储区时，从设定好的指针处开始查找，如果有足够大空间的存储区即进行分配并更改指针位置为已分配区的下一块空闲存储区。在释放分配好的空间时，要考虑到各种情况，特别注意需要合并存储区的情况，还要注意根据实际情况调整起始查找指针的位置。

为了描述得更加清晰，采用图片表示存储管理空闲区表和内存的的一般状态与对应关系。某时刻，空闲分区表的状态如下图所示：

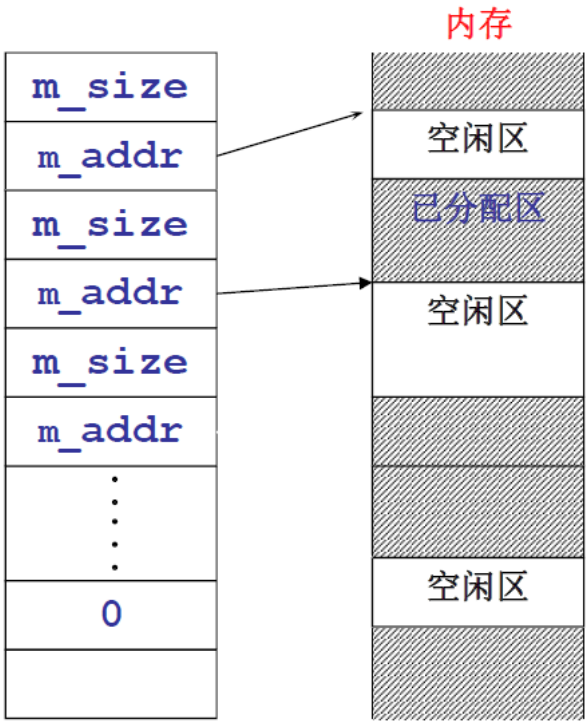

图2-8 某一时刻空闲分区表的状态

3. 程序功能说明

本实验中实现的程序功能如下：

1）使用循环首次适应算法，模拟分配内存空间和释放内存空间。

2）主要的输入输出说明：

（1） 本实验中的程序采用键盘输入（没有采用文件输入）。首先，输入需要分配的内存空间大小（单位：kB）；

（2） 随后，可以输入命令，命令的格式：

a. m 空间大小（单位：kB）：为一个新任务分配空间；

b. f 空间大小（单位：kB）相对地址：释放对应起始相对地址位置、对应已分配空间大小的内存块；

c. p：打印空闲存储区表；

d. h：打印命令说明；

e. q：退出程序。

（3） 输出的内容：对于分配和释放空间的命令和打印命令，输入命令后都会打印一次空闲存储区表，该结果会更新到 output.txt 文件中。

3）注：本实验中，采取相对地址进行空间的释放，其实也不难通过绝对地址进行释放，但由于使用相对地址管理对用户较为友好，因此采取输入相对地址的方式确定需要释放的内存块。

4. 主要数据结构和变量说明

1）空闲存储区表的数据结构：使用自建 map 类型，结构体代码为：

```
    struct map {
            unsigned m_size;// 空闲分区的长度
            char *m_addr; // 空闲区的起始地址
     };
struct map coremap[N];
```

2）重要的变量：

（1） char 指针类型变量 startPoint：顾名思义，作为起始查找指针，是循环首次适应法空间分配的关键；

（2） int 类型变量 cycle：作为算法中“循环”的依据。由于没有采取循环链表的链接结构而采取了数组这样的线性结构，所以需要这样的一个数字，作为循环（转到第一个空闲存储区块）的标志；

（3） int 类型变量 numOfMalloc：和 cycle 变量类似，也是循环的重要标志，和 cycle 变量共同工作，起到循环算法的作用。

5. 程序结构和算法说明

注：此处对程序的含义作简单的讲解。源程序中包含了一些已经注释掉的测试代码，请忽略其实际功能。源代码参见附录。

（1） 主函数 main：主要实现对输入内容的识别、对输出结果的转向、向系统申请和释放一块空间和简单的错误处理。在其中需要调用分配函数、释放函数、打印存储区表函数、帮助函数等有各自具体功能的函数；

（2） 帮助函数 showHelp：在程序运行的界面上打印帮助，主要包括各种本程序支持的命令的格式；

（3） 空闲存储区表打印函数 printCoremap：输出空闲存储区表并通过 I/O 转向的方式输出到 output.txt 文件中（注：每次输出都是对 output 中的表格的更新，对整个操作过程的表格状态都有记录）；

（4） 初始化空闲存储区表函数 initCoremap：向系统申请空间后，需要生成空闲存储区的初始状态。这个函数的任务就是这样，将表项中大小为 0 的项的所有输出参数置零，并生成和打印其他空间非零表项的信息；

（5） 空间分配函数 lmalloc：这个函数实现了该程序的核心算法——循环首次适应法。算法的基本思想前文已经有所讲述，此处只对程序的部分实现思路作讲解：

a. “循环”：主要依靠前文变量说明中的 cycle 和 numOfMalloc 实现循环（转到第一个空闲区表项），使用递归调用函数并更改第二次调用时的“起始查找指针”为第一个表项，从而实现转向；

b. “首次”：主要通过 for 循环从低地址（前面的表项）到高地址（后面的表项）依次询问空间实现，难度不大，需要注意的是需要删除表项时要调整其后表项的序号；

（6） 空间释放函数 lfree：这个函数与首次适应算法的函数内容基本相同，只是需要在增删表项时修改其后面的表项在表中的位置即可。主要分为四种情况，分别为释放区独立、释放区与前空闲区相连、释放区与后空闲区相连、释放区与前后空闲区都相连四种情况，对应地做出表项调整即可。

6. 测试数据

为了完整地测试程序的功能，我采用了两组命令进行测试：

1）测试“首次”功能和出错处理功能：

申请空间大小：1000kB

命令组：

```
m 100
m 100
m 100
f 100 0//测试情况 4：释放区独立
m 200
m 300
m 300//测试空间不足的处理
m 200
f 100 100//测试情况 1：释放区与前空闲区相邻
f 300 500//测试情况 4：释放区独立
f 100 1000//测试地址溢出的情况
m -100//测试空间大小非法的情况
f 200 300//测试情况 3：释放区与后空闲区相邻
f 100 200//测试情况 2：释放区与前后空闲区均相邻
```

2）测试“循环”功能：

申请空间大小：1000kB

命令组：

```
m 100
m 200
m 300
m 200
f 300 300//测试情况 4：释放区独立
m 300//测试循环的转向
```

7. 实验结果：截图与结论

1）实验结果 1

```
Please input the malloc size:1000
Please input command,input h for help:m 100
Please input command,input h for help:m 100
Please input command,input h for help:m 100
Please input command,input h for help:f 100 0
Please input command,input h for help:m 200
Please input command,input h for help:m 300
Please input command,input h for help:m 300
空间不足!
Please input command,input h for help:m 200
Please input command,input h for help:f 100 100
Please input command,input h for help:f 300 500
Please input command,input h for help:f 100 1000
超出范围，无法释放空间!
Please input command,input h for help:m -100
参数size错误!
Please input command,input h for help:f 200 300
Please input command,input h for help:f 100 200
Please input command,input h for help:h
m size : 分配内存
f size addr : 释放内存
p : 打印coremap表
h : 显示帮助
q : 退出执行
Please input command,input h for help:q

Process returned 0 (0x0)   execution time : 64.531 s
Press any key to continue.
```

2）实验结果 2

```
Please input the malloc size:1000
Please input command,input h for help.m 100
Please input command,input h for help.m 200
Please input command,input h for help.m 300
Please input command,input h for help.m 200
Please input command,input h for help.f 300 300
Please input command,input h for help.m 300
Please input command,input h for help.q

Process returned 0 (0x0)   execution time : 33.610 s
Press any key to continue.
```

注：文件输出结果详见报告压缩包附带的文档

3）实验结论

上述的程序较为完整地实现了循环首次适应法的思想，并成功地模拟了 UNIX 系统的存储管理机制，同时完成了大部分的错误处理。

8. 实验结果讨论

1）程序可能存在的不足

在程序中，我选用了顺序结构的表。这样的结果是，在核心算法部分（分配空间函数）处理“循环”部分就比较困难，比如使用了递归这种时间复杂度很高的方式；而且处理表项前移、后移的程序段设计起来也相对复杂。如果采用老师给出的循环链表的方式，或许能够简化这些过程，减小时间复杂度。但是，这会

带来的问题是初始化链表的相对困难和涉及较容易犯错误的指针。所以，这两种数据结构是各自有利有弊的，但是采用链表可能会稍微简单一些。

2）在实验中遇到的问题

在实验中我遇到的最大的问题就是“循环”算法的实现，因为稍不慎就会导致内存溢出和地址溢出。在尝试多次的条件语句之后，在教材上给出的程序的基础上，如果需要循环（回到第一个表项），增加一步递归，同时设置标志，避免死循环的出现。

另外一个问题是文件的输出。通过参考网上的一些资料，选择 fprintf 来输出必要的内容到 txt 文件上，并在每次程序运行之前清空文件之前存留的内容。这或许是较为完整的输出解决办法。

9. 收获与感想

在这次的实验中，我对存储管理这一章的内容，尤其是可变分区存储管理中的算法内容有了更加深刻的理解。同时，我对程序设计和数据结构相关的内容进行了必要的复习。但是，程序仍然存在最坏时间复杂度较高的问题。可以说在实验中我既积累了知识，也获得了编程的经验。在此，对薛老师和助教老师表示感谢！

附录：源代码 cf.h

```
#include <stdio.h>
#include <stdlib.h>
#define N 5
int numOfMalloc = 0;
int tt = 0;
struct map {
        unsigned m_size;//  空闲分区的长度
        char *m_addr;//  空闲区的起始地址
 };
struct map coremap[N];
struct map *startPoint;

char* lmalloc(unsigned size)
{
    int total = 0;
    register char *a;
    register struct map *bp;
    int i = 0;
    int nonZero = 0;
    int cycle = 1;
    for(i=0;i<N;i++)
    {
        total += coremap[i].m_size;
        nonZero++;
    }
    //printf("total:%d\n",total);
    if(total < size||(total == size && nonZero > 1)){return 0;}//明显的空间不足的处理
    if(numOfMalloc == 0){startPoint = coremap;} //循环转到第一个表项的标志，同时，在第一次调用函数时，初始化起始指针

    for (bp = startPoint; bp->m_size; bp++)
    {

        if(bp->m_size >= size) {
            a = bp->m_addr;
            bp->m_addr += size;
            if((bp->m_size -= size) == 0)
                {startPoint = bp;
                  do    {
                    bp++;
                    (bp-1)->m_addr = bp->m_addr;
                } while((bp-1)->m_size = bp->m_size);
                // printf("1 StartPoint:%d",startPoint->m_size);
                //if((bp->m_size -= size) == 0&&startPoint->m_size == 0){startPoint =
```

```
coremap;}
                        //return (a);
                        }
                        startPoint = bp++;
                        numOfMalloc++;//循环标志 1
                        cycle = 2;//循环标志 2
                       // printf("2 StartPoint:%d",startPoint->m_size);
                return(a);
            }
        }

        if(cycle == 1){numOfMalloc = 0;lmalloc(size);tt++;}//递归调用，实现循环



}

void lfree(unsigned size, char *aa)
{
   struct map *bp;
        char *a,*t;
        unsigned tt;

        a = aa;
        for(bp=coremap;bp->m_addr<=a&&bp->m_size!=0;bp++);
        if(bp>coremap&&(bp-1)->m_addr+(bp-1)->m_size ==a){
        (bp-1)->m_size += size; //情况 1
        if(a+size == bp->m_addr)
          {
            (bp-1)->m_size += bp->m_size;
            while(bp->m_size)    //情况 2
            {
                  bp++;
                  (bp-1)->m_addr = bp->m_addr;
                  (bp-1)->m_size = bp->m_size;
            }
           // printf("2.1 StartPoint:%d\n",startPoint->m_size);
            //printf("2\n");
            if(startPoint->m_addr > bp->m_addr)startPoint++;
            //printf("2.2 StartPoint:%d\n",startPoint->m_size);
          }
        }
        else{
        if(a+size==bp->m_addr&&bp->m_size)
        {
```

```
            bp->m_addr -= size; //情况 3
            bp->m_size += size;
            //printf("3.1 StartPoint:%d\n",startPoint->m_size);
              //printf("3\n");
            if(startPoint->m_addr > bp->m_addr)startPoint++;
            else if(startPoint->m_addr-bp->m_size == bp->m_addr)startPoint = bp;
           // printf("3.2 StartPoint:%d\n",startPoint->m_size);
        }
        else if(size)
        {do{                                //情况 4
            t = bp->m_addr;
            bp->m_addr = a;
            a = t;
            tt = bp->m_size;
            bp->m_size = size;
            bp++;
          }while(size = tt);
          //printf("4.1 StartPoint:%d\n",startPoint->m_size);
            //printf("4\n");
        if(startPoint->m_addr > bp->m_addr)startPoint++;
          else if(startPoint->m_addr-bp->m_size == bp->m_addr)startPoint = bp;}}
          //printf("4.2 StartPoint:%d\n",startPoint->m_size);
        }

    void initCoremap(char *base_addr,unsigned size)//初始化空闲区表
    {
        int i;
        coremap[0].m_addr = base_addr;
        coremap[0].m_size = size;
        for(i = 1;i < N;i++)
        {
            coremap[i].m_addr = 0;
            coremap[i].m_size = 0;
        }
        return;
    }

    void printCoremap(char *base_addr)//打印存储区表
    {
        FILE *p;

        p=fopen("output.txt","a");
        int i,zero = 0;
        //printf("%u\n",coremap[0].m_addr);
```

```
        fprintf(p,"                    Size\t\t\tR_Addr\t\tAddr\n");
        for(i = 0;i < N;i++)
        {
            if(coremap[i].m_size != 0)

fprintf(p,"coremap[%d]:%u\t\t%u\t\t%u\n",i,coremap[i].m_size,(coremap[i].m_addr-
base_addr),coremap[i].m_addr);
            else fprintf(p,"coremap[%d]:%u\t\t%d\t\t%d\n",i,coremap[i].m_size,zero,zero);

        }

}

void showHelp()//显示帮助
{
        printf("m size : 分配内存\n");
        printf("f size addr : 释放内存\n");
        printf("p : 打印 coremap 表\n");
        printf("h : 显示帮助\n");
        printf("q : 退出执行\n");
}

int main()
{

        unsigned size,total_size;
        char *base_addr,*addr,*start_addr,cmdchar;
        unsigned r_addr;
        printf("Please input the malloc size:");
        scanf("%u",&total_size);
        base_addr = malloc(total_size);//向系统申请一块空间
        initCoremap(base_addr,total_size);
        FILE *file;
        file = fopen("output.txt","w");//输出 I/O 转向
        do
        {
            printf("Please input command,input h for help.");
            do cmdchar = getchar();
            while(cmdchar == ' '||cmdchar == '\t'||cmdchar == '\n');//吞空格
            switch(cmdchar)//命令识别、错误处理
            {

                case 'm':
                scanf("%u",&size);
```

```
            if(size < 0||size >= total_size)
            {
                printf("参数 size 错误！\n");
                break;
            }
            start_addr = lmalloc(size);
            if(start_addr == 0)
            {
                printf("空间不足！\n");
                break;
            }

            printCoremap(base_addr);
            break;

            case 'f':
            scanf("%u%u",&size,&r_addr);
            if(r_addr<0||r_addr >= total_size)
            {
                printf("超出范围，无法释放空间！\n");
                break;
            }
            addr = base_addr + r_addr;
            lfree(size,addr);
            printCoremap(base_addr);
            break;

            case 'p':
            printCoremap(base_addr);
            break;
            case 'h':
            showHelp();
            break;
            case 'q':
            break;
            default:
            printf("非法命令！请输入 h 查看帮助再行输入！\n");
            continue;

        }
}while(cmdchar != 'q');

free(base_addr);//释放申请的空间
```

```
    return 0;
}
```