

实验三 存储管理

- 课程：操作系统
- 学院：信息工程学院
- 专业班级：计科2002
- 学号：2020010505
- 姓名：薛飞宇

一、实习目的

1. 模拟操作系统对存储器管理
2. 加深了解存储管理的工作原理
3. 理解存储管理的原理及数据结构
4. 理解“最佳分配算法”算法
5. 编写和调试内存管理调度模拟程序，掌握存储管理算法
6. 阅读“伙伴算法”，加深对存储管理的理解

二、实习内容

任务1：代码阅读并调试实验

任务2：设计测试实验

任务3：页结构分析

三、实习任务及完成情况

任务1：代码阅读并调试实验

1. 阅读下面源代码，完善程序空白处内容。

1. 遍历所有可用的空闲区，找到一个可分配内存的存储空间

```
for(i=0; i<m; i++) /*1遍历所有可用的空闲区，找到一个可分配内存的存储空间*/
    if(free_table[i].length>=xk&&free_table[i].flag==1)
        if(k== -1 || free_table[i].length<free_table[k].length)
            k=i;
```

2. 寻找一个可分配的分区

```
while(used_table[i].flag!=0&&i<n) /*2寻找一个可分配的分区*/
    i++;
```

3. 如果上邻表和下邻表空，将上邻下邻表目合并，并把下邻表目设为空

```
if(k!=-1)
    if(j!=-1)
        /* 3如果上邻表和下邻表空，将上邻下邻表目合并，并把下邻表目设为空*/
        {

            free_table[k].length=free_table[j].length+free_table[k].length+L;
```

```

        free_table[j].flag=0;
    }
    else
        /*上邻空闲区，下邻非空闲区，与上邻合并*/
        free_table[k].length=free_table[k].length+L;
else if(j!=-1)
    /*上邻非空闲区，下邻为空闲区，与下邻合并*/
    {
        free_table[j].address=S;
        free_table[j].length=free_table[j].length+L;
    }
else
    /*上下邻均为非空闲区，回收区域直接填入*/
    {
        /*在空闲区表中寻找空栏目*/
        t=0;
        while(free_table[t].flag==1&& t<m)
            t++;
        if(t>=m)/*空闲区表满，回收空间失败，将已分配表复原*/
        {
            printf("主存空闲表没有空间，回收空间失败\n");
            used_table[s].flag=J;
            return;
        }
        free_table[t].address=S;
        free_table[t].length=L;
        free_table[t].flag=1;
    }
}

```

2. 阅读代码，写出源程序采用的调度算法、算法流程图和程序功能

1. 调度算法：

1. 内存分配

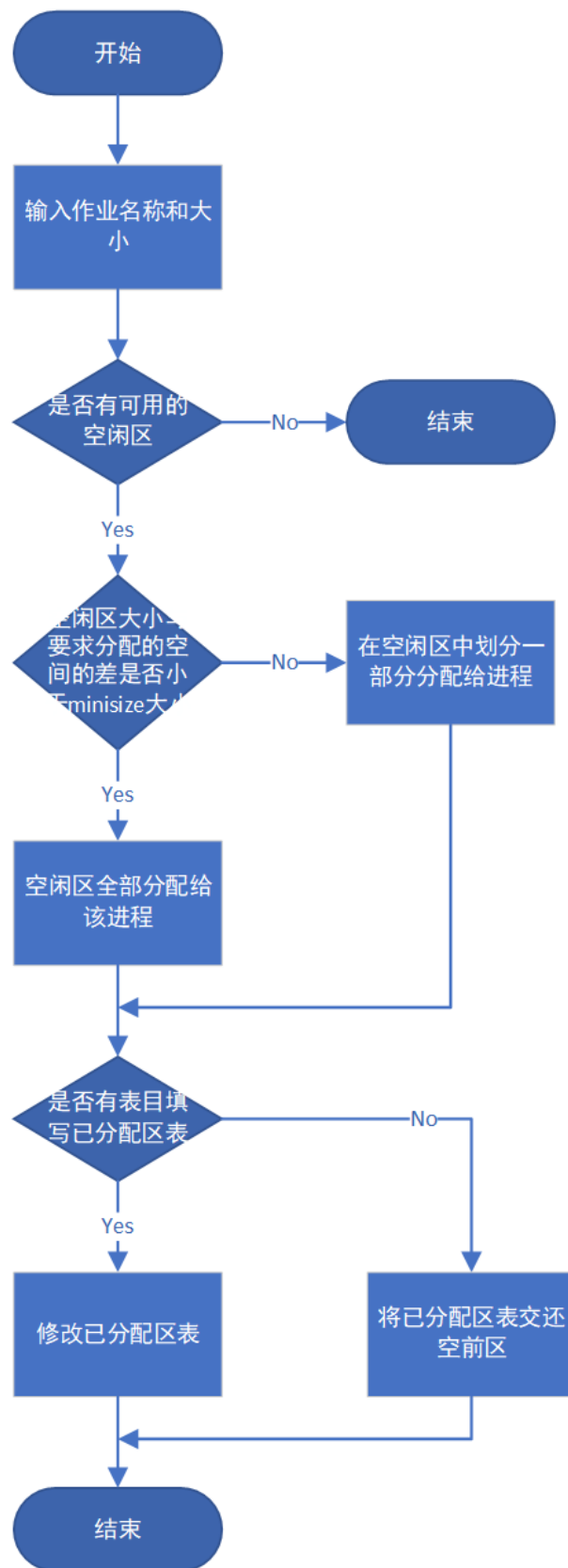
采用**最佳分配算法**分配内存，先在空闲区找一块内存，然后给这块内存分配内存

2. 内存回收

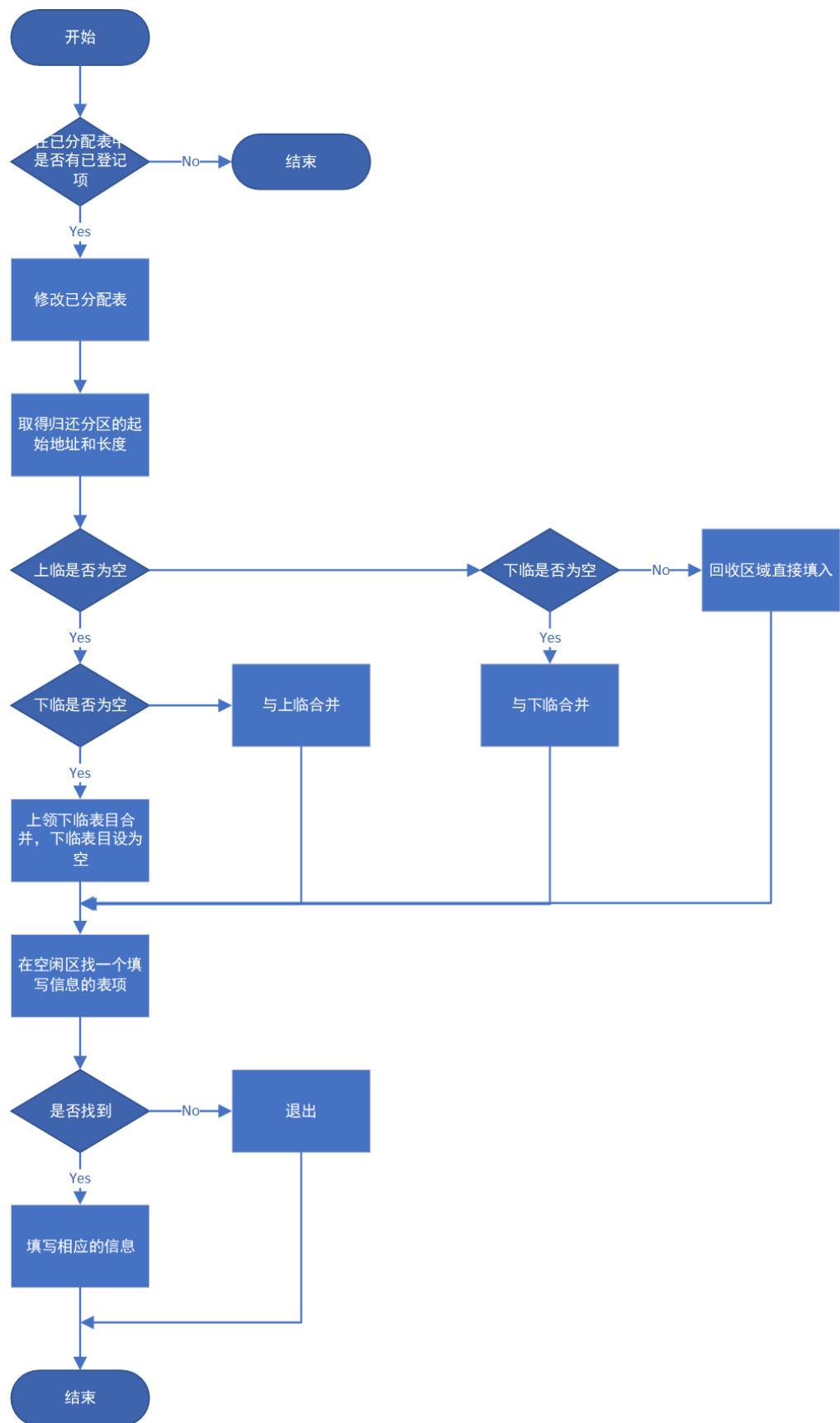
先在分配区找到已修改项，判断分配区和回收区的情况，对内存回收，将分配区复原

2. 算法流程图

1. 内存分配算法



2. 内存回收算法



3. 程序功能

本程序模拟了计算机中内存的分配和回收问题，即：存储管理。输入作业名和所需内存大小，该程序会自动为之分配内存，输入作业名相应的会对内存进行回收。

3. 调试并运行代码，写出结果。

1. 程序源代码

```
#include<stdio.h>
#include <dos.h>
```

```

#include<stdlib.h>
#include<conio.h>
#define n 10 /*假定系统允许的最大作业数为n，假定模拟实验中n值为10*/
#define m 10 /*假定系统允许的空闲区表最大为m，假定模拟实验中m值为10*/
#define minisize 100 /*空闲分区被分配时，如果分配后剩余的空间小于minisize，则将该空闲分区全部分配，若大于minisize，则切割分配*/
struct
{
    float address; /*已分配分区起始地址*/
    float length; /*已分配分区长度，单位为字节*/
    int flag; /*已分配区表登记栏标志，用"0"表示空栏目*/
} used_table[n]; /*已分配区表*/

struct
{
    float address; /*空闲区起始地址*/
    float length; /*空闲区长度，单位为字节*/
    int flag; /*空闲区表登记栏标志，用"0"表示空栏目，用"1"表示未分配*/
} free_table[m]; /*空闲区表*/

void allocate(char J,float xk) /*给J作业，采用最佳分配算法分配xk大小的空间*/
{
    int i,k;
    float ad;
    k=-1;
    for(i=0; i<m; i++) /*1遍历所有可用的空闲区，找到一个可分配内存的存储空间*/
        if(free_table[i].length>=xk&&free_table[i].flag==1)
            if(k==-1||free_table[i].length<free_table[k].length)
                k=i;
    if(k==-1)/*未找到可用空闲区，返回*/
    {
        printf("无可空闲区\n");
        return;
    }
    /*找到可用空闲区，开始分配：若空闲区大小与要求分配的空间差小于minisize大小，则空闲区全部分配；若空闲区大小与要求分配的空间差大于minisize大小，则从空闲区划出一部分分配*/
    if(free_table[k].length-xk<=minisize)
    {
        free_table[k].flag=0;
        ad=free_table[k].address;
        xk=free_table[k].length;
    }
    else
    {
        free_table[k].length=free_table[k].length-xk;
        ad=free_table[k].address+free_table[k].length;
    }

    /*修改已分配区表*/
    i=0;
    while(used_table[i].flag!=0&&i<n) /*2寻找一个可分配的分区*/
        i++;
    if(i>=n) /*无表目可填写已分配分区*/
    {
        printf("无表目填写已分分区，错误\n");
        /*修正空闲区表*/
        if(free_table[k].flag==0)
            /*前面找到的是整个空闲分区*/

```

```

        free_table[k].flag=1;
    else
    {
        /*前面找到的是某个空闲分区的一部分*/
        free_table[k].length=free_table[k].length+xk;
        return;
    }
}
else
{
    /*修改已分配表*/
    used_table[i].address=ad;
    used_table[i].length=xk;
    used_table[i].flag=j;
}
return;
}

/*主存分配函数结束*/
void reclaim(char j)
/*回收作业名为j的作业所占主存空间*/
{
    int i,k,j,s,t;
    float S,L;
    /*寻找已分配表中对应登记项*/
    s=0;
    while((used_table[s].flag!=j||used_table[s].flag==0)&&s<n)
        s++;
    if(s>=n)/*在已分配表中找不到名字为j的作业*/
    {
        printf("找不到该作业\n");
        return;
    }
    /*修改已分配表*/
    used_table[s].flag=0;
    /*取得归还分区的起始地址S和长度L*/
    S=used_table[s].address;
    L=used_table[s].length;
    j=-1;
    k=-1;
    i=0;
    /*寻找回收分区的空闲上下邻，上邻表目k，下邻表目j*/
    while(i<m&&(j==-1||k==-1))
    {
        if(free_table[i].flag==1)
        {
            if(free_table[i].address+free_table[i].length==S)k=i;/*找到上邻*/
            if(free_table[i].address==S+L)j=i;/*找到下邻*/
        }
        i++;
    }
    if(k!=-1)
        if(j!=-1)
            /* 3如果上邻表和下邻表空，将上邻下邻表目合并，并把下邻表目设为空*/
            {
                free_table[k].length=free_table[j].length+free_table[k].length+L;
                free_table[j].flag=0;
            }
}

```

```

else
    /*上邻空闲区，下邻非空闲区，与上邻合并*/
    free_table[k].length=free_table[k].length+L;
else if(j!=-1)
    /*上邻非空闲区，下邻为空闲区，与下邻合并*/
{
    free_table[j].address=S;
    free_table[j].length=free_table[j].length+L;
}
else
    /*上下邻均为非空闲区，回收区域直接填入*/
{
    /*在空闲区表中寻找空栏目*/
    t=0;
    while(free_table[t].flag==1&&t<m)
        t++;
    if(t>=m)/*空闲区表满，回收空间失败，将已分配表复原*/
    {
        printf("主存空闲表没有空间，回收空间失败\n");
        used_table[s].flag=j;
        return;
    }
    free_table[t].address=S;
    free_table[t].length=L;
    free_table[t].flag=1;
}
return;
}/*主存回收函数结束*/
int main( )
{
    int i,a;
    float xk;
    char J;
    /*空闲分区表初始化：*/
    free_table[0].address=10240; /*起始地址假定为10240*/
    free_table[0].length=10240; /*长度假定为10240，即10k*/
    free_table[0].flag=1; /*初始空闲区为一个整体空闲区*/

    for(i=1; i<m; i++)
        free_table[i].flag=0; /*其余空闲分区表项未被使用*/
    /*已分配表初始化：*/
    for(i=0; i<n; i++)
        used_table[i].flag=0; /*初始时均未分配*/
    while(1)
    {
        printf("*****");
        printf("\n\t\t选择功能项\n");
        printf("*****");
        printf("\n\n\t\t0-退出");
        printf("\n\t\t1-分配主存");
        printf("\n\t\t2-回收主存");
        printf("\n\t\t3-显示主存");
        printf("\n\n\t\t\t\t\t选择功能项(0~3) : \n");
        scanf("%d",&a);
        switch(a)
        {
            case 0:
                exit(0); /*a=0程序结束*/

```

```

    case 1:      /*a=1分配主存空间*/
        printf("输入作业名J和作业所需长度xk: \n");
        scanf("%*c%c%f",&J,&xk);
        allocate(J,xk); /*分配主存空间*/
        break;
    case 2:      /*a=2回收主存空间*/
        printf("输入要回收分区的作业名");
        scanf("%*c%c",&J);
        reclaim(J); /*回收主存空间*/
        break;
    case 3:      /*a=3显示主存情况*/
        /*输出空闲区表和已分配表的内容*/
        printf("输出空闲区表: \n起始地址 分区长度 标志\n");
        for(i=0; i<m; i++)

            printf("%6.0f%9.0f%6d\n",free_table[i].address,free_table[i].length,
            free_table[i].flag);
            printf(" 按任意键,输出已分配区表\n");
            getch();
            printf(" 输出已分配区表: \n起始地址 分区长度 标志\n");
            for(i=0; i<n; i++)
                if(used_table[i].flag!=0)

                    printf("%6.0f%9.0f%6c\n",used_table[i].address,used_table[i].length,
                    used_table[i].flag);
                    else

                        printf("%6.0f%9.0f%6d\n",used_table[i].address,used_table[i].length,
                        used_table[i].flag);
                        break;
            default:
                printf("没有该选项\n");
            }/*case*/
        }/*while*/
        return 1;
    }/*主函数结束*/

```

2. 运行测试

1.1 选择模式1-分配主存


```
C:\Users\86133\Desktop\2020010505_薛飞宇_实验3_III\实验源代码\任务1_2020010505_薛飞宇\bin\Debu...
*****
选择功能项
*****

0-退出
1-分配主存
2-回收主存
3-显示主存

选择功能项(0~3) :
1_
```

1.2 给作业a分配内存长度为999999，内存分配失败

```
C:\Users\86133\Desktop\2020010505_薛飞宇_实验3_III\实验源代码\...
*****
选择功能项
*****

0-退出
1-分配主存
2-回收主存
3-显示主存

选择功能项(0~3) :
1
输入作业名J和作业所需长度xk:
a
999999
无可空闲区
*****
选择功能项
*****

0-退出
1-分配主存
2-回收主存
3-显示主存

选择功能项(0~3) :
```

1.3 选择功能3-显示主存

```
C:\Users\86133\Desktop\2020010505_薛飞宇_实验3_III\实验源代码\...
999999
无可空闲区
*****
                选择功能项
*****

                0-退出
                1-分配主存
                2-回收主存
                3-显示主存

                                选择功能项(0~3) :
3
输出空闲区表:
起始地址 分区长度 标志
10240      10240      1
0          0          0
0          0          0
0          0          0
0          0          0
0          0          0
0          0          0
0          0          0
0          0          0
0          0          0
0          0          0
按任意键, 输出已分配区表
输出已分配区表:
起始地址 分区长度 标志
0          0          0
0          0          0
0          0          0
0          0          0
0          0          0
0          0          0
0          0          0
0          0          0
0          0          0
0          0          0
*****
                选择功能项
*****

                0-退出
                1-分配主存
```

1.4 给作业a分配内存长度800，分配成功

```
C:\Users\86133\Desktop\2020010505_薛飞宇_实验3_III\实验源代码\任务1_20...
选择功能项(0~3) :
1
输入作业名J和作业所需长度xk:
a
800
*****
选择功能项
*****

0-退出
1-分配主存
2-回收主存
3-显示主存

选择功能项(0~3) :
3
输出空闲区表:
起始地址 分区长度 标志
10240      9440      1
0          0        0
0          0        0
0          0        0
0          0        0
0          0        0
0          0        0
0          0        0
0          0        0
0          0        0
按任意键, 输出已分配区表
输出已分配区表:
起始地址 分区长度 标志
19680      800      a
0          0        0
0          0        0
0          0        0
0          0        0
0          0        0
0          0        0
0          0        0
0          0        0
0          0        0
*****
选择功能项
*****
```

1.5 给作业b分配内存长度9000，分配成功

```
C:\Users\86133\Desktop\2020010505_薛飞宇_实验3_III\实验源代码\任务1_20...  —  □  ×

                                     选择功能项(0~3) :
1
输入作业名J和作业所需长度xk:
b
9000
*****
                        选择功能项
*****

                        0-退出
                        1-分配主存
                        2-回收主存
                        3-显示主存

                                     选择功能项(0~3) :
3
输出空闲区表:
起始地址  分区长度  标志
10240      440      1
0          0        0
0          0        0
0          0        0
0          0        0
0          0        0
0          0        0
0          0        0
0          0        0
0          0        0
按任意键, 输出已分配区表
输出已分配区表:
起始地址  分区长度  标志
19680      800      a
10680      9000     b
0          0        0
0          0        0
0          0        0
0          0        0
0          0        0
0          0        0
0          0        0
0          0        0
*****
                        选择功能项
```

1.6 给作业c分配内存长度400，分配成功

```
C:\Users\86133\Desktop\2020010505_薛飞宇_实验3_III\实验源代码\任务1_20...
输入作业名J和作业所需长度xk:
c
400
*****
                        选择功能项
*****

                        0-退出
                        1-分配主存
                        2-回收主存
                        3-显示主存

                                选择功能项(0~3) :
3
输出空闲区表:
起始地址 分区长度 标志
10240      440      0
   0        0      0
   0        0      0
   0        0      0
   0        0      0
   0        0      0
   0        0      0
   0        0      0
   0        0      0
   0        0      0
   0        0      0
按任意键, 输出已分配区表
输出已分配区表:
起始地址 分区长度 标志
19680      800      a
10680     9000      b
10240      440      c
   0        0      0
   0        0      0
   0        0      0
   0        0      0
   0        0      0
   0        0      0
   0        0      0
   0        0      0
*****
                        选择功能项
*****

                        0-退出
```

1.7 回收内存b

```
C:\Users\86133\Desktop\2020010505_薛飞宇_实验3_III\实验源代码\任务1_20...
2-回收主存
3-显示主存

选择功能项(0~3) :
2
输入要回收分区的作业名b
*****
选择功能项
*****

0-退出
1-分配主存
2-回收主存
3-显示主存

选择功能项(0~3) :
3
输出空闲区表:
起始地址 分区长度 标志
10680      9000      1
0          0        0
0          0        0
0          0        0
0          0        0
0          0        0
0          0        0
0          0        0
0          0        0
0          0        0
按任意键, 输出已分配区表
输出已分配区表:
起始地址 分区长度 标志
19680      800      a
10680      9000     0
10240      440      c
0          0        0
0          0        0
0          0        0
0          0        0
0          0        0
0          0        0
0          0        0
*****
选择功能项
```

1.8 给作业d分配内存长度100，分配成功

```
C:\Users\86133\Desktop\2020010505_薛飞宇_实验3_III\实验源代码\任务1_20...
输入作业名J和作业所需长度xk:
d
100
*****
                        选择功能项
*****

                        0-退出
                        1-分配主存
                        2-回收主存
                        3-显示主存

                                选择功能项(0~3) :
3
输出空闲区表:
起始地址 分区长度 标志
10680      8900      1
      0          0      0
      0          0      0
      0          0      0
      0          0      0
      0          0      0
      0          0      0
      0          0      0
      0          0      0
      0          0      0
按任意键, 输出已分配区表
输出已分配区表:
起始地址 分区长度 标志
19680      800      a
19580      100      d
10240      440      c
      0          0      0
      0          0      0
      0          0      0
      0          0      0
      0          0      0
      0          0      0
      0          0      0
*****
                        选择功能项
*****

                        0-退出
```

1.9 回收内存a

```
*****
                        选择功能项
*****

                        0-退出
                        1-分配主存
                        2-回收主存
                        3-显示主存

                                选择功能项(0~3) :
2
输入要回收分区的作业名a
```

1.10 回收内存c,d

```
*****
                          选择功能项
*****
                          0-退出
                          1-分配主存
                          2-回收主存
                          3-显示主存

                                          选择功能项(0~3) :
2
输入要回收分区的作业名c
*****
                          选择功能项
*****

                          0-退出
                          1-分配主存
                          2-回收主存
                          3-显示主存

                                          选择功能项(0~3) :
2
输入要回收分区的作业名d
```

1.11 全部内存回收完毕，显示内存分配情况


```
C:\Users\86133\Desktop\2020010505_薛飞宇_实验3_III\实验源代码\任务1_20...
0-退出
1-分配主存
2-回收主存
3-显示主存

选择功能项(0~3) :
3
输出空闲区表:
起始地址 分区长度 标志
10240      10240      1
19680      800       0
0          0         0
0          0         0
0          0         0
0          0         0
0          0         0
0          0         0
0          0         0
0          0         0
0          0         0
按任意键, 输出已分配区表
输出已分配区表:
起始地址 分区长度 标志
19680      800       0
19580      100       0
10240      440       0
0          0         0
0          0         0
0          0         0
0          0         0
0          0         0
0          0         0
0          0         0
0          0         0
*****
选择功能项
*****

0-退出
1-分配主存
2-回收主存
3-显示主存

选择功能项(0~3) :
```

任务2：设计测试实验

1.设计一个固定式分区分配的存储管理方案，并模拟实现分区的分配和回收过程。

设计一个固定式分区分配的存储管理方案，并模拟实现分区的分配和回收过程。假定每个作业都是批处理作业，且不允许动态申请内存。为实现分区的分配和回收，可设定一个分区表，按照表中的有关信息进行分配，并根据分区的分配和回收情况修改该表。

要求：

1. 描述模型、定义数据结构。

1. 已分配区表

```

struct
{
    float address;    /*已分配分区起始地址*/
    float length;     /*已分配分区长度，单位为字节*/
    int flag;         /*已分配区表登记栏标志，用"0"表示空栏目*/
} used_table[n];     /*已分配区表*/

```

2. 空闲区表

```

struct
{
    float address;    /*空闲区起始地址*/
    float length;     /*空闲区长度，单位为字节*/
    int flag;         /*空闲区表登记栏标志，用"0"表示空栏目，用"1"表示未分配*/
} free_table[m];     /*空闲区表*/

```

2. 写代码实现并运行调试。

```

#include<stdio.h>
#include <dos.h>
#include<stdlib.h>
#include<conio.h>
//#include<iostream.h>
#define n 10 /*假定系统允许的最大作业数为n，假定模拟实验中n值为10*/
#define m 10 /*假定系统允许的空闲区表最大为m，假定模拟实验中m值为10*/
#define minisize 100 /*空闲分区被分配时，如果分配后剩余的空间小于minisize，则将该空闲分区全部分配，若大于minisize，则切割分配*/
struct
{
    float address; /*已分配分区起始地址*/
    float length; /*已分配分区长度，单位为字节*/
    int flag; /*已分配区表登记栏标志，用"0"表示空栏目*/
} used_table[n]; /*已分配区表*/

struct
{
    float address; /*空闲区起始地址*/
    float length; /*空闲区长度，单位为字节*/
    int flag; /*空闲区表登记栏标志，用"0"表示空栏目，用"1"表示未分配*/
} free_table[m]; /*空闲区表*/

void allocate(char J, float xk) /*给J作业，采用最佳分配算法分配xk大小的空间*/
{
    int i, k;
    float ad;
    k=-1;
    for(i=0; i<m; i++)
        if(free_table[i].length>=xk&&free_table[i].flag==1)
            if(k==-1 || free_table[i].length<free_table[k].length)
                k=i;
    if(k==-1)/*未找到可用空闲区，返回*/
    {
        printf("无可用的空闲区\n");
        return;
    }
}

```

/*找到可用空闲区，开始分配：若空闲区大小与要求分配的空间差小于minisize大小，则空闲区全部分配；若空闲区大小与要求分配的空间差大于minisize大小，则从空闲区划出一部分分配*/

```
if (free_table[k].length - xk <= minisize)
{
    free_table[k].flag = 0;
    ad = free_table[k].address;
    xk = free_table[k].length;
}
else
{
    free_table[k].length = free_table[k].length - xk;
    ad = free_table[k].address + free_table[k].length;
}

/*修改已分配区表*/
i = 0;
while (used_table[i].flag != 0 && i < n)
    i++;
if (i >= n) /*无表目可填写已分配分区*/
{
    printf("无表目填写已分分区，错误\n");
    /*修正空闲区表*/
    if (free_table[k].flag == 0)
        /*前面找到的是整个空闲分区*/
        free_table[k].flag = 1;
    else
    {
        /*前面找到的是某个空闲分区的一部分*/
        free_table[k].length = free_table[k].length + xk;
        return;
    }
}
else
{
    /*修改已分配表*/
    used_table[i].address = ad;
    used_table[i].length = xk;
    used_table[i].flag = J;
}
return;
```

/*主存分配函数结束*/

void reclaim(char J)

/*回收作业名为J的作业所占主存空间*/

```
{
    int i, k, j, s, t;
    float S, L;
    /*寻找已分配表中对应登记项*/
    s = 0;
    while ((used_table[s].flag != J || used_table[s].flag == 0) && s < n)
        s++;
    if (s >= n) /*在已分配表中找不到名字为J的作业*/
    {
        printf("找不到该作业\n");
        return;
    }
    /*修改已分配表*/
    used_table[s].flag = 0;
```

```

/*取得归还分区的起始地址S和长度L*/
S=used_table[s].address;
L=used_table[s].length;
j=-1;
k=-1;
i=0;
/*寻找回收分区的空闲上下邻，上邻表目k，下邻表目j*/
while(i<m&&(j== -1 || k== -1))
{
    if(free_table[i].flag==1)
    {
        if(free_table[i].address+free_table[i].length==S)k=i; /*找到上邻*/
        if(free_table[i].address==S+L)j=i; /*找到下邻*/
    }
    i++;
}
if(k!=-1)
    if(j!=-1)
    {
        free_table[k].length=free_table[j].length+free_table[k].length+L;
        free_table[j].flag=0;
    }
    else
        /*上邻空闲区，下邻非空闲区，与上邻合并*/
        free_table[k].length=free_table[k].length+L;
else if(j!=-1)
    /*上邻非空闲区，下邻为空闲区，与下邻合并*/
    {
        free_table[j].address=S;
        free_table[j].length=free_table[j].length+L;
    }
    else
        /*上下邻均为非空闲区，回收区域直接填入*/
        {
            /*在空闲区表中寻找空栏目*/
            t=0;
            while(free_table[t].flag==1&&t<m)
                t++;
            if(t>=m)/*空闲区表满，回收空间失败，将已分配表复原*/
            {
                printf("主存空闲表没有空间，回收空间失败\n");
                used_table[s].flag=J;
                return;
            }
            free_table[t].address=S;
            free_table[t].length=L;
            free_table[t].flag=1;
        }
    return;
}/*主存回收函数结束*/
int main( )
{
    int i,a;
    float xk;
    char J;
    /*空闲分区表初始化：*/
    free_table[0].address=10240; /*起始地址假定为10240*/

```

```

free_table[0].length=10240; /*长度假定为10240, 即10k*/
free_table[0].flag=1; /*初始空闲区为一个整体空闲区*/

for(i=1; i<m; i++)
    free_table[i].flag=0; /*其余空闲分区表项未被使用*/
/*已分配表初始化: */
for(i=0; i<n; i++)
    used_table[i].flag=0; /*初始时均未分配*/
while(1)
{
    printf("*****");
    printf("\n\t\t选择功能项\n");
    printf("*****");
    printf("\n\n\t\t0-退出");
    printf("\n\t\t1-分配主存");
    printf("\n\t\t2-回收主存");
    printf("\n\t\t3-显示主存");
    printf("\n\n\t\t\t\t\t选择功能项(0~3) : \n");
    scanf("%d",&a);
    switch(a)
    {
        case 0:
            exit(0); /*a=0程序结束*/
        case 1: /*a=1分配主存空间*/
            printf("输入作业名J和作业所需长度xk: \n");
            scanf("%*c%c%f",&J,&xk);
            allocate(J,xk); /*分配主存空间*/
            break;
        case 2: /*a=2回收主存空间*/
            printf("输入要回收分区的作业名");
            scanf("%*c%c",&J);
            reclaim(J); /*回收主存空间*/
            break;
        case 3: /*a=3显示主存情况*/
            /*输出空闲区表和已分配表的内容*/
            printf("输出空闲区表: \n起始地址 分区长度 标志\n");
            for(i=0; i<m; i++)

                printf("%6.0f%9.0f%6d\n",free_table[i].address,free_table[i].length,
                    free_table[i].flag);
            printf(" 按任意键,输出已分配区表\n");
            getch();
            printf(" 输出已分配区表: \n起始地址 分区长度 标志\n");
            for(i=0; i<n; i++)
                if(used_table[i].flag!=0)

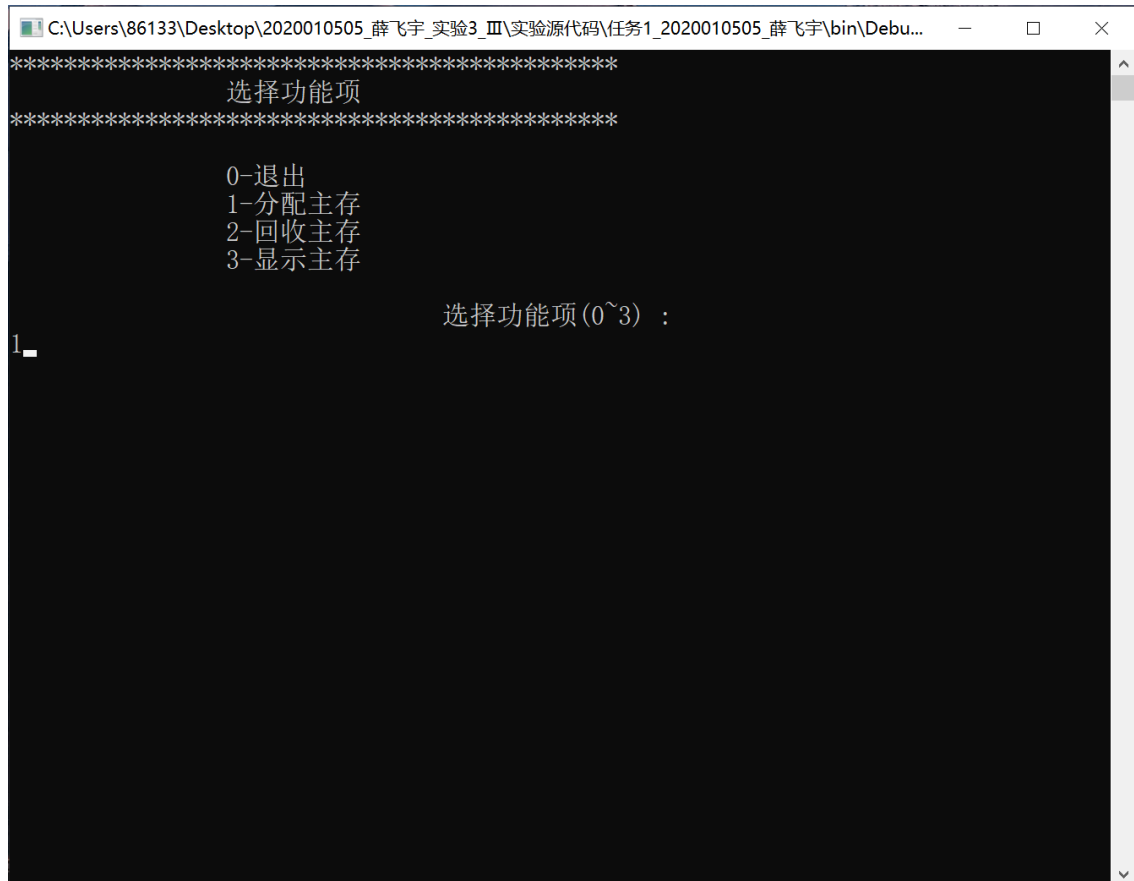
                    printf("%6.0f%9.0f%6c\n",used_table[i].address,used_table[i].length,
                        used_table[i].flag);
            else

                printf("%6.0f%9.0f%6d\n",used_table[i].address,used_table[i].length,
                    used_table[i].flag);
            break;
        default:
            printf("没有该选项\n");
    }/*case*/
}/*while*/
return 1;

```

```
}/*主函数结束*/
```

3. 给出运行的数据和结果。



The screenshot shows a Windows command prompt window with the following text:

```
C:\Users\86133\Desktop\2020010505_薛飞宇_实验3_III\实验源代码\任务1_2020010505_薛飞宇\bin\Debu...  
*****  
                        选择功能项  
*****  
  
                        0-退出  
                        1-分配主存  
                        2-回收主存  
                        3-显示主存  
  
                        选择功能项(0~3) :  
1_
```

```
C:\Users\86133\Desktop\2020010505_薛飞宇_实验3_III\实验源代码\...
*****
                        选择功能项
*****

                        0-退出
                        1-分配主存
                        2-回收主存
                        3-显示主存

                                选择功能项(0~3) :
1
输入作业名J和作业所需长度xk:
a
999999
无可用空闲区
*****
                        选择功能项
*****

                        0-退出
                        1-分配主存
                        2-回收主存
                        3-显示主存

                                选择功能项(0~3) :
```



```
C:\Users\86133\Desktop\2020010505_薛飞宇_实验3_III\实验源代码\任务1_20...
选择功能项(0~3) :
1
输入作业名J和作业所需长度xk:
a
800
*****
选择功能项
*****

0-退出
1-分配主存
2-回收主存
3-显示主存

选择功能项(0~3) :
3
输出空闲区表:
起始地址 分区长度 标志
10240      9440      1
0          0          0
0          0          0
0          0          0
0          0          0
0          0          0
0          0          0
0          0          0
0          0          0
0          0          0
按任意键, 输出已分配区表
输出已分配区表:
起始地址 分区长度 标志
19680      800      a
0          0          0
0          0          0
0          0          0
0          0          0
0          0          0
0          0          0
0          0          0
0          0          0
0          0          0
*****
选择功能项
*****
```

2.设计一个可变式分区分配的存储管理方案并定义数据结构、写代码运行并调试。

1. 模拟实现分区的分配和回收过程
2. 分区的管理算法可选择：首次适应算法、循环首次适应算法和最佳适应算法三种之一，或者自行定义——采用最佳适应算法。

```
#include<stdio.h>
#include <dos.h>
#include<stdlib.h>
#include<conio.h>
//#include<iostream.h>
```

```

#define ALLOCATE_ERROR -1
#define ALLOCATE_SUCCESS 0
#define n 10 /*假定系统允许的最大作业数为n，假定模拟实验中n值为10*/
#define m 10 /*假定系统允许的空闲区表最大为m，假定模拟实验中m值为10*/
#define minisize 100 /*空闲分区被分配时，如果分配后剩余的空间小于minisize，则将该空闲
分区全部分配，若大于minisize，则切割分配*/
struct
{
    float address; /*已分配分区起始地址*/
    float length; /*已分配分区长度，单位为字节*/
    int flag; /*已分配区表登记栏标志，用"0"表示空栏目*/
} used_table[n]; /*已分配区表*/
struct
{
    float address; /*空闲区起始地址*/
    float length; /*空闲区长度，单位为字节*/
    int flag; /*空闲区表登记栏标志，用"0"表示空栏目，用"1"表示未分配*/
} free_table[m]; /*空闲区表*/
void allocate ( char j, float xk ) /*给j作业，采用最佳分配算法分配xk大小的空间*/
{
    int i, k;
    float ad;
    k = -1;
    for ( i = 0; i < m; i++ ) /*1
        if ( free_table[i].length >= xk && free_table[i].flag == 1 )
            if ( k == -1 || free_table[i].length < free_table[k].length )
                k = i;
    if ( k == -1 ) /*未找到可用空闲区，返回*/
    {
        printf ( "无可用空闲区\n" );
        return;
    }
    /*找到可用空闲区，开始分配：若空闲区大小与要求分配的空间差小于minisize大小，则空闲区
    全部分配；若空闲区大小与要求分配的空间差大于minisize大小，则从空闲区划出一部分分配*/
    if ( free_table[k].length - xk <= minisize )
    {
        free_table[k].flag = 0;
        ad = free_table[k].address;
        xk = free_table[k].length;
    }
    else
    {
        free_table[k].length = free_table[k].length - xk;
        ad = free_table[k].address + free_table[k].length;
    }

    /*修改已分配区表*/
    i = 0;
    while ( used_table[i].flag != 0 && i < n ) /*2
        i++;
    if ( i >= n ) /*无表目可填写已分配分区*/
    {
        printf ( "无表目填写已分分区，错误\n" );
        /*修正空闲区表*/
        if ( free_table[k].flag == 0 )
            /*前面找到的是整个空闲分区*/
            free_table[k].flag = 1;
    }
}

```

```

        else
        {
            /*前面找到的是某个空闲分区的一部分*/
            free_table[k].length = free_table[k].length + xk;
            return;
        }
    }
    else
    {
        /*修改已分配表*/
        used_table[i].address = ad;
        used_table[i].length = xk;
        used_table[i].flag = 1;
    }
    return;
}

/*主存分配函数结束*/
void allocateFirstFit ( char JobName, double xk ) //采用首次适应算法
{
    int k;
    float address_new;
    int i;
    for ( i = 0; i < m; i++ )
        if ( free_table[i].length >= xk && free_table[i].flag == 1 )
            if ( k == -1 || free_table[i].length < free_table[k].length )
                k = i;
    if ( k == -1 )
    {
        printf ( "空闲区无可分配内存" );
        exit ( -1 );
    }
    /*找到可用空闲区，开始分配：若空闲区大小与要求分配的空间差小于minisize大小，则空闲区
    全部分配；若空闲区大小与要求分配的空间差大于minisize大小，则从空闲区划出一部分分配*/
    if ( free_table[k].length - xk <= minisize )
    {
        free_table[k].flag = 0; //已经被完全分配，以后不能再被分配
        address_new = free_table[k].address; //分配区的首地址
        xk = free_table[k].length; //xk为该块完全分配的值
    }
    else
    {
        free_table[k].length = free_table[k].length - xk; //该块还有剩余
        address_new = free_table[k].address + free_table[k].length;
    }

    //修改已分配区表
    i = 0;
    while ( used_table[i].flag != 0 && i < n ) /*2 找到分配区的一个表项用来填写该
    分配区*/
        i++;
    if ( i >= n ) /*无表目可填写已分配分区*/
    {
        printf ( "无表目填写已分分区，错误\n" );
        /*修正空闲区表， 又可以继续分配*/
        if ( free_table[k].flag == 0 )
            /*前面找到的是整个空闲分区*/
            free_table[k].flag = 1;
        else

```

```

    {
        /*前面找到的是某个空闲分区的一部分*/
        free_table[k].length = free_table[k].length + xk;
        return;
    }
}
else
{
    //修改已分配表
    used_table[i].address = address_new;
    used_table[i].length = xk;
    used_table[i].flag = 1; //
}
return;
}
void reclaim ( char J )
/*回收作业名为J的作业所占主存空间*/
{
    int i, k, j, s, t;
    float S, L;
    /*寻找已分配表中对应登记项*/
    s = 0;
    while ( ( used_table[s].flag != J || used_table[s].flag == 0 ) && s < n
)
        s++;
    if ( s >= n ) /*在已分配表中找不到名字为J的作业*/
    {
        printf ( "找不到该作业\n" );
        return;
    }
    /*修改已分配表*/
    used_table[s].flag = 0;
    /*取得归还分区的起始地址S和长度L*/
    S = used_table[s].address;
    L = used_table[s].length;
    j = -1;
    k = -1;
    i = 0;
    /*寻找回收分区的空闲上下邻，上邻表目k，下邻表目j*/
    while ( i < m && ( j == -1 || k == -1 ) )
    {
        if ( free_table[i].flag == 1 )
        {
            if ( free_table[i].address + free_table[i].length == S ) k = i;
            /*找到上邻*/
            if ( free_table[i].address == S + L ) j = i; /*找到下邻*/
        }
        i++;
    }
    if ( k != -1 )
        if ( j != -1 )
            /* 3 上邻与下邻空闲区，与上下邻合并*/
            {
                free_table[k].length = free_table[j].length +
free_table[k].length + L;
                free_table[j].flag = 0;
            }
        else

```

[illegible]

```

printf ( "输入作业名J和作业所需长度xk: \n" );
scanf ( "%*c%c%f", &J, &xk );
allocateFirstFit ( J, xk ); /*分配主存空间*/
break;
case 2:      /*a=2回收主存空间*/
printf ( "输入要回收分区的作业名" );
scanf ( "%*c%c", &J );
reclaim ( J ); /*回收主存空间*/
break;
case 3:      /*a=3显示主存情况*/
/*输出空闲区表和已分配表的内容*/
printf ( "输出空闲区表: \n起始地址 分区长度 标志\n" );
for ( i = 0; i < m; i++ )
    printf ( "%6.0f%9.0f%6d\n", free_table[i].address,
free_table[i].length, free_table[i].flag );
printf ( " 按任意键,输出已分配区表\n" );
getch();
printf ( " 输出已分配区表: \n起始地址 分区长度 标志\n" );
for ( i = 0; i < n; i++ )
    if ( used_table[i].flag != 0 )
        printf ( "%6.0f%9.0f%6c\n", used_table[i].address,
used_table[i].length, used_table[i].flag );
    else
        printf ( "%6.0f%9.0f%6d\n", used_table[i].address,
used_table[i].length, used_table[i].flag );
    break;
default:
    printf ( "没有该选项\n" );
}/*case*/
}/*while*/
return 1;
}/*主函数结束*/

```

3. 设计并调试一个段页式存储管理的地址转换的模拟程序。

1. 设计段表、页表。

1. 段表

```

struct Parag
{
    int state; //段的状态
    int length; //段的长度,表示有几个页,并假设一个页占用一个内存
    int PTID; //页表始址
}parag[4];

```

2. 页表

```

struct Page
{
    int state; //状态
    int MMID; //物理块号
}page[100];

```

2. 测试运行

```
#include <stdio.h>
```

```

#include <stdlib.h>

#define n 4 //段表的长度
//段表
struct Parag
{
    int state; //段的状态
    int length; //段的长度,表示有几个页,并假设一个页占用一个内存
    int PTID; //页表始址
}parag[4];

//页表
struct Page
{
    int state; //状态
    int MMID; //物理块号
}page[100];

void print(int did, int ptid)
{
    int i;
    printf("段表的信息: \n");
    printf("段号\t| 状态\t| 段表大小\t| 页表始址|\n");
    for(i=0; i < n; i++)
        printf("%d\t| %d\t| %d\t| %d\t|\n", i, parag[i].state,
parag[i].length, parag[i].PTID);
    printf("所用表的信息: \n");
    printf("页号\t| 状态\t| 物理块号|\n");
    for(i = ptid; i<ptid+parag[did].length; i++)
        printf("%d\t| %d\t| %d\t|\n", i-ptid, page[i].state,
page[i].MMID);
}

void Init()
{
    int i, d=0;
    //初始化段表
    for(i=0; i < n; i++)
    {
        parag[i].length=i+4; //段的长度依次为4 5 6 7
        parag[i].state=1;
        parag[i].PTID=d;
        d+=parag[i].length;
    }
    //初始化页表
    for(i = 0; i < d; i++)
    {
        page[i].state=1;
        page[i].MMID=(i+7)%d; //确定物理块号
    }
}

void Transform(int DID, int PID, int Address)
{
    if(DID >= n)
    {
        printf("段越界! \n");
        return;
    }
}

```

```

    }
    if(parag[DID].length <= PID)
    {
        printf("页越界\n");
        return;
    }
    int ptid = parag[DID].PTID;
    int ptid2 = ptid+PID;
    int mmid = page[ptid2].MMID;
    int newAddress = 1024*mmid+Address;
    printf("转换前的地址: 段号: %d, 页号: %d, 页内地址: %d\n",DID, PID,
Address);
    print(DID,ptid);//显示段表和所用页表的信息
    printf("转换后的地址为: %d", newAddress);
}

int main()
{
    int DID, PID, Address;
    Init();//初始化页表和段表
    printf("请输入要转换的地址: \n");
    printf("请输入段号: \n");
    scanf("%d",&DID);
    printf("请输入页号: \n");
    scanf("%d",&PID);
    printf("请输入页内地址: \n");
    scanf("%d", &Address);
    Transform(DID, PID, Address);
    return 0;
}

```

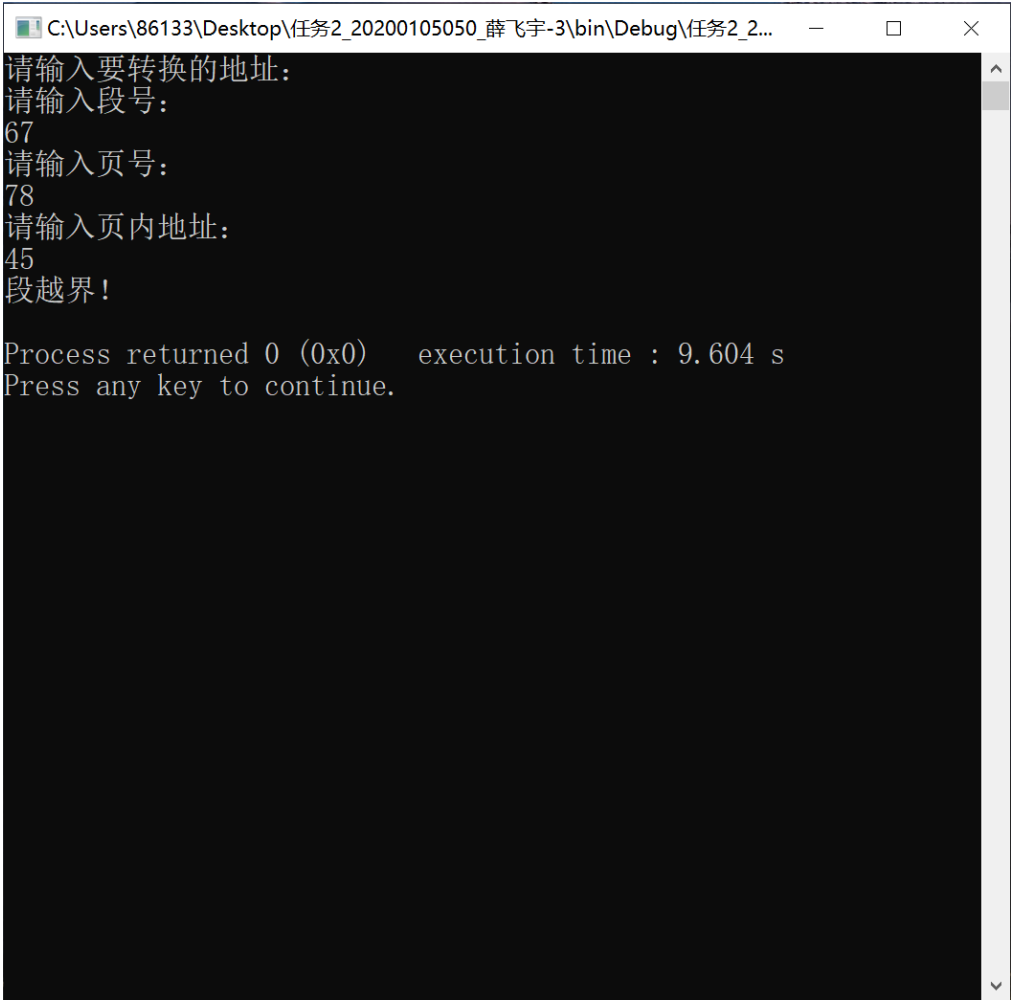
1. 正常执行：给出一个有代表性的地址，通过查找段页表后得到转换的地址。打印转换前的地址、相应的段表、页表、转换后的地址

```

C:\Users\86133\Desktop\任务2_20200105050_薛飞宇-3\bin\Debug\任务2_20200105050_薛飞宇-3.exe
请输入要转换的地址:
请输入段号:
1
请输入页号:
1
请输入页内地址:
2
转换前的地址: 段号: 1, 页号: 1, 页内地址: 2
段表的信息:
段号    状态    段表大小    页表起始址
0        1        4           0
1        1        5           4
2        1        6           9
3        1        7          15
所用表的信息:
页号    状态    物理块号
0        1       11
1        1       12
2        1       13
3        1       14
4        1       15
转换后的地址为: 12290
Process returned 0 (0x0)    execution time : 7.288 s
Press any key to continue.

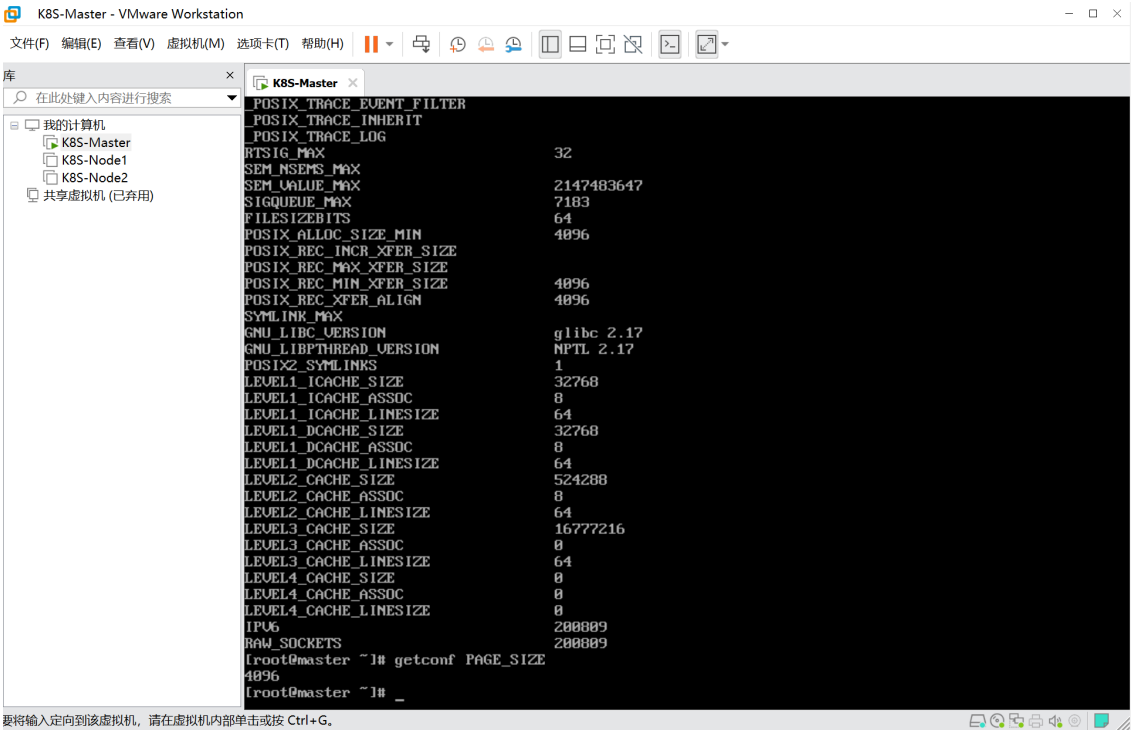
```


2. 段错误：段越界



任务3：页结构分析

1. 使用getconf查看计算机中页的大小，如下图，为4096



2. 参考伙伴算法，编写一个内存模拟管理程序

```
#include <stdio.h>
```

```

#include <stdlib.h>

struct Info
{
    int value;
    int count;
    int begin;
    int end;
};

void init(struct Info info[])
{
    int i;
    for(i=0; i<11; i++)
    {
        if(i == 0) info[i].value = 1;
        else info[i].value = info[i-1].value*2;
        info[i].count = 0;
        info[i].begin = 0;
        info[i].end = 0;
    }
    info[10].count = 1;
    info[10].end = 1023;
}

void print(struct Info info[])
{
    int i,j;
    printf("|");
    for (i=0; i<11; i++){
        if (info[10].count > 0){
            printf("%d|",info[10].value);
            break;
        }
        if (info[i].count > 0){
            printf("%d|",info[i].value);
        }else if(info[i+1].count > 0 && i!=9){
            printf("*|");
        }else if(i==9 && info[i].count == 0){
            printf("*|");
        }
    }
}

void buddy(struct Info info[], int num)
{
    int i, j, k;
    for(i=0; i<11; i++)
    {
        if(info[i].value >= num)
        {
            if(info[i].count == 1)
            {
                info[i].count = 0;
                printf("Allocate pages sucefully!\n");
                return ;
            }
            else{

```

```

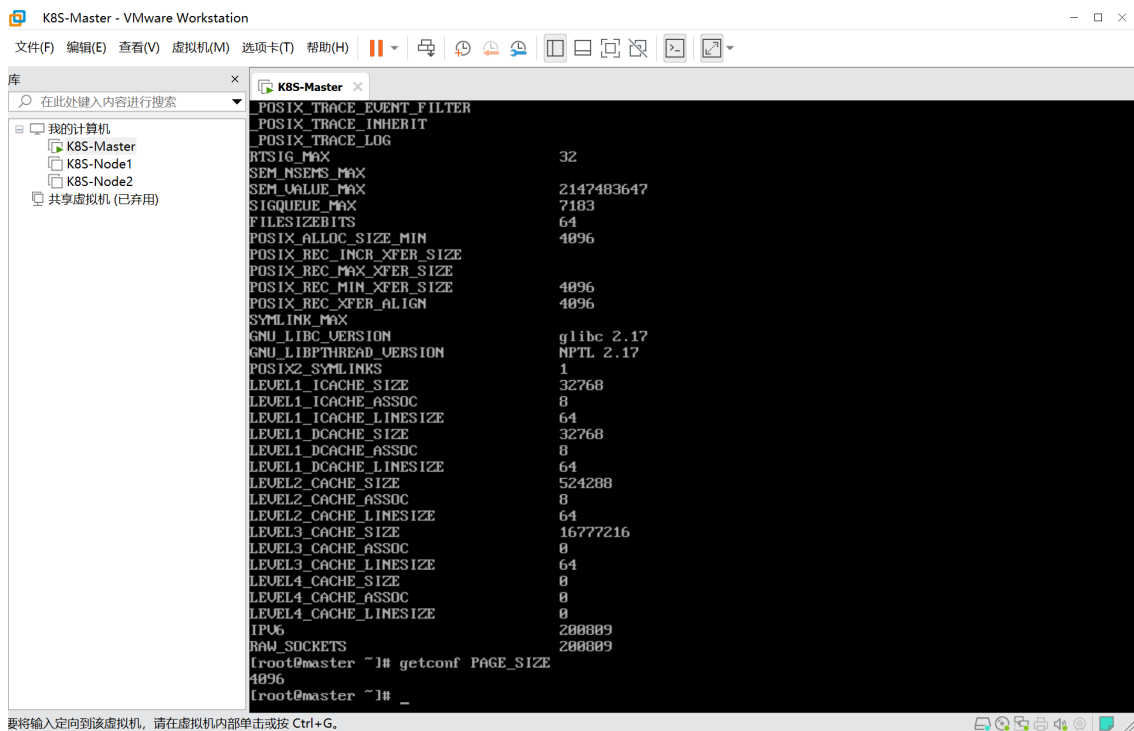
        for(j=i; j<11; j++)
        {
            if(info[j].count == 1)
            {
                info[j].count = 0;           //把该页拆分开
                for(k=i; k<j; k++)
                {
                    info[k].count = 1;
                    info[k].begin = info[k].value;
                    info[k].end = info[k].begin*2 - 1;
                }
                printf("Allocate pages succefully!\n");
                return ;
            }
            if(j == 10)
            {
                printf("Allocate pages failed!\n");
                return ;
            }
        }
    }
}

int main()
{
    int i, num;
    struct Info info[11];
    printf("Init:\n");
    init(info);

    print(info);
    while(1)
    {
        printf("\nPlease input a num:");
        scanf("%d", &num);
        fflush(stdin);
        if(num > 1023) printf("The num is too big , try littler\n");
        else
        {
            buddy(info, num);
            print(info);
        }
    }
    system("pause");
    return 0;
}

```

测试运行，结果如下



四、实习总结

本次实验模拟操作系统对存储器管理；加深了解存储管理的工作原理；理解存储管理的原理及数据结构；在任务1中，我理解了“最佳分配算法”算法，先在空闲区找一块内存，然后给这块内存分配内存；模拟了回收算法，先在分配区找到已修改项，判断分配区和回收区的情况，对内存回收，将分配区复原。然后，在任务2中，我设计一个固定式分区分配的存储管理方案，并模拟实现分区的分配和回收过程；设计一个可变式分区分配的存储管理方案，并定义数据结构、写代码运行并调试。编写和调试内存管理调度模拟程序；掌握存储管理算法。最后，我在任务3中阅读了“伙伴算法”，加深对Linux存储管理的理解。