

实验	一	二	三	四	五	六	七	八	九	十	总评
成绩											

武汉大学计算机学院

课程实验(设计)报告

专业(班): 计算机学院信息安全 X 班

学号:

姓名: XX

课程名称: 操作系统实验

2013 年 5 月 30 日

实习一 处理器调度

一、实习内容

选择一个调度算法，实现处理器调度。

二、实习目的

本实习模拟在单处理器环境下的处理器调度，加深了解处理器调度的工作。

三、实习题目

设计一个按优先数调度算法实现处理器调度的程序。

四、实习内容

1. 设计思想

(1) 假定系统有 5 个进程，每个进程用一个 PCB 来代表。PCB 的结构为：

- 进程名——如 P1~P5。
- 指针——按优先数的大小把 5 个进程连成队列，用指针指出下一个进程 PCB 的首地址。
- 要求运行时间——假设进程需要运行的单位时间数。
- 优先数——赋予进程的优先数，调度时总是选取优先数大的进程先执行。
- 状态——假设两种状态：就绪和结束，用 R 表示就绪，用 E 表示结束。初始状态都为就绪状态。

(2) 开始运行之前，为每个进程确定它的“优先数”和“要求运行时间”。通过键盘输入这些参数。

(3) 处理器总是选择队首进程运行。采用动态改变优先数的办法，进程每运行 1 次，优先数减 1，要求运行时间减 1。

(4) 进程运行一次后，若要求运行时间不等于 0，则将它加入就绪队列，否则，将状态改为“结束”，退出就绪队列。

(5) 若就绪队列为空，结束，否则转到(3)重复。

要求能接受键盘输入的进程优先数及要求运行时间，能显示每次进程调度的情况，如哪个进程在运行，哪些进程就绪，就绪进程的排列情况。

2. 上机代码

```
#include <stdio.h>
#include <string.h>
#define num 5//5 个进程

struct PCB//进程控制块
{
    char ID;//进程名
    int runtime;//进程运行时间
    int pri;//进程优先级
    char state; //进程状态
};

struct PCB pcblist[num];//进程控制块数组
int fnum=0;//已运行完成的进程
struct PCB temp;//在冒泡排序时用于替换的 PCB

void chushihua()//初始化程序
{
    int i;
    for(i=0;i<num;i++)
    {
        printf("PCB[%d]:ID pri runtime \n",i+1);
        scanf("%s%d%d",&pcblist[i].ID,&pcblist[i].pri,&pcblist[i].runtime);
        pcblist[i].state='R';//将所有进程的状态初始化为就绪状态
    }
}
```

```

        getchar();
    }
}

void show()//显示当前进程状况程序
{
    int i;
    printf("\nID    pri    runtime    state\n");
    for(i=0;i<num;i++)
    {
        printf("%s%6d%9d        %s\n",&pcblist[i].ID,pcblist[i].pri,pcblist[i].runtime,&pcblist[i].state);
    }
    getchar();
}

void run()//主体执行部分程序，含将优先级进行冒泡排序及运行最高优先级的进程
{
    int i,j,k;
    int t=0;//所有程序运行总时间

    for(j=0;j<num;j++)//计算所有进程总的运行时间
    {
        if(pcblist[j].runtime==0)
            pcblist[j].state='F';
        t+=pcblist[j].runtime;
    }

    for(j=0;k<t;j++)
    {
        for(i=0;i<num-fnum;i++)//将状态为就绪的进程进行冒泡排序
            for(j=0;j<num-i-1-fnum;j++)
                if(pcblist[j].pri>pcblist[j+1].pri)
                {
                    temp=pcblist[j];
                    pcblist[j]=pcblist[j+1];
                    pcblist[j+1]=temp;
                }
        for(j=num-1;j>=0;j--)//在排好序的进程中找到为就绪的最大优先级的进程
        {
            if(pcblist[j].state=='R')
                break;
        }
        if(pcblist[0].state=='R')//修改当前运行的进程的各项参数

```

```

        {
            pcblist[j].pri=pcblist[j].pri-1;
            pcblist[j].runtime=pcblist[j].runtime-1;
            printf("%s",&pcblist[j].ID);
            if(pcblist[j].runtime==0)
            {
                pcblist[j].state='F';
                fnum++;
            }
        }
    show();

}
}

int main()//主函数
{
    chushihua();
    show();
    run();
    return 0;
}

```

五、上机实验所用平台及相关软件

上机平台： Windows 7 +DEV- C++

六、运行结果

Q1— Q5 分别是 12345，经运行的顺序为

Q2Q2Q2Q3Q3Q1Q1Q4Q1Q5Q5Q1Q1Q5Q5

测试结果正确无误

```
I:\操作系统实验\1\1.exe
1 3 5
PCB[2]:ID pri runtime
2 6 3
PCB[3]:ID pri runtime
3 4 2
PCB[4]:ID pri runtime
4 2 1
PCB[5]:ID pri runtime
5 1 4

ID    pri    runtime    state
1      3        5        R
2      6        3        R
3      4        2        R
4      2        1        R
5      1        4        R

2
ID    pri    runtime    state
5      1        4        R
4      2        1        R
1      3        5        R
3      4        2        R
2      5        2        R
```

```
I:\操作系统实验\1\1.exe
5
ID    pri    runtime    state
5     -2        1        R
1     -2        0        F
4      1        0        F
3      2        0        F
2      3        0        F

5
ID    pri    runtime    state
5     -3        0        F
1     -2        0        F
4      1        0        F
3      2        0        F
2      3        0        F

ID    pri    runtime    state
5     -3        0        F
1     -2        0        F
4      1        0        F
3      2        0        F
2      3        0        F
```

七、实验心得

运用优先级调度算法，最根本的问题在于挑选优先级最高的就绪状态的进程，故而本程序采用符号 `fnum` 对已完成的程序数目进行计数，并采用冒牌排序法挑选出优先级最高的就绪状态的进程将之运行

并输出。经过本次试验，我更深的理解了操作系统按优先级的调度过程，我会继续努力深入学习。

实习二主存空间的分配和回收

一、实验内容

主存储器空间的分配和回收。

二、实习目的

通过本实习帮助理解在不同的存储管理方式下应怎样进行存储空间的分配和回收。

三、实习题目

可变分区管理方式下采用首次适应算法实现主存分配和回收

四、实习内容

1.设计思想

(1)可变分区方式是按作业需要的主存空间大小来分割分区的。当要装入一个作业时，根据作业需要的主存容量查看是否有足够的空闲空间，若有，则按需分配，否则，作业无法装入。假定内存大小为 128K，空闲区说明表格式为：

- 起始地址——指出空闲区的起始地址；
- 长度——一个连续空闲区的长度；
- 状态——有两种状态，一种是“未分配”状态；另一种是“空表目”状态，表示该表项目前没有使用。

(2)采用首次适应算法分配回收内存空间。运行时，输入一系列分

配请求和回收请求。要求能接受来自键盘的空间申请及释放请求，能显示分区分配及回收后的内存布局情况。

2. 上机代码

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

const int CANTUSE = 0;
const int CANUSE=1;
const int MSIZE = 128; //内存分区

struct MZone
{
    //空闲区起始地址
    int begin_addr;
    //一个连续空闲区的长度
    int length;
    //状态
    int state;

    //内存中任务名
    char task_name[32];
    //指向下一个空闲分区
    struct MZone *next;
};

//内存头指针
struct MZone *Mhead = NULL;

//showmemory 函数，显示当前内存分配情况
void showmemory()
{
    struct MZone *Mpoint = Mhead;
    printf("内存的使用情况\n");
    printf("beginaddr\tlength\tstate\ttask\n");

    while( NULL!=Mpoint)
    {
        printf("%dk\t\t",Mpoint->begin_addr);
```



```

printf("%dk\t",Mpoint->length);
if(Mpoint->state==CANTUSE)
printf("CANTUSE\t");
else

printf("free\t");

printf("%s\n",Mpoint->task_name);
Mpoint = Mpoint->next;
}

system("pause");

}
//Mininsert 函数，功能插入任务到空闲分区
int Mininsert(struct MZone* Mnew)
{

struct MZone *Zinsert = Mhead;
while( Zinsert->length<Mnew->length || !Zinsert->state)
{
if( NULL!=Zinsert->next )
{
Zinsert = Zinsert->next;
}
else
{
Zinsert = Zinsert->next;
break;
}
}

if( NULL==Zinsert )
{
return 0;
}

if( MSIZE == Zinsert->begin_addr+Mnew->length )
{
Zinsert->state = CANTUSE;
strcpy(Zinsert->task_name , Mnew->task_name);
Zinsert->next = NULL;
return 1;
}
}

```

```

else
{
strcpy(Zinsert->task_name , Mnew->task_name);
Zinsert->length = Mnew->length;
struct MZone *Ztail = (struct MZone *)malloc(sizeof(struct MZone));

Ztail->next=Zinsert->next;
Zinsert->next = Ztail;
Zinsert->state = CANTUSE;
memset( Ztail, 0, sizeof(char)*32 );
Ztail->begin_addr = Zinsert->begin_addr + Mnew->length;
Ztail->state = 1;

if(Ztail->next==NULL)
Ztail->length =MSIZE-Ztail->begin_addr;
else
Ztail->length =Ztail->next->begin_addr-Ztail->begin_addr;
return 1;
}
}

//memoallocate 函数，用于分配内存
void memoallocate(void)
{
struct MZone *Mnew = (struct MZone*)malloc(sizeof(struct MZone));
printf("输入要分配内存大小(kb):\n");
scanf("%d",&Mnew->length);
printf("输入任务名:\n");
scanf("%s",&Mnew->task_name);
Minsert(Mnew)?printf("分配内存成功\n"):printf("没有符合大小的空闲分区，内存分配失败。 \n");
system("pause");
free(Mnew);
}

```

```

//Mreturn 函数，功能回收内存
int Mreturn(char taskname[])
{
struct MZone *front = NULL;
struct MZone *position = Mhead;
struct MZone *tail = Mhead->next;

```

```

while( 0!=strcmp(position->task_name,taskname) )
{
front = position;
if( NULL!=position->next )
{
position = position->next;
}
else
{
position = NULL;
break;
}
tail = position->next;
}

if( NULL==position )
{
printf("内存中没有此任务！");
}
else
{

if( NULL!=tail&&NULL!=front )
{

if( front->state&&tail->state )
{

front->length = front->length + position->length + tail->length;
front->next = tail->next;
free(position);
free(tail);

}

if( front->state&&!tail->state )
{
front->length = front->length + position->length;
front->next = position->next;
free(position);
}

if( !front->state&&tail->state )
{

```

```

position->length = position->length + tail->length;
memset( position->task_name, 0, sizeof(char)*32 );
position->next = tail->next;
position->state = 1;

free(tail);

}
    if( !front->state&&!tail->state )
    {
memset( position->task_name, 0, sizeof(char)*32 );
position->state = 1;
    }
}
if( NULL!=tail&&NULL==front )
{
    if( !tail->state )
    {
memset( position->task_name, 0, sizeof(char)*32 );
position->state = 1;

    }
    else
    {

memset( position->task_name, 0, sizeof(char)*32 );
position->length = position->length + tail->length;
position->next = NULL;
position->state=1;
free(tail);

    }
}
if( NULL==tail&&NULL!=front )
{
    if(front->state)
    {
front->length = front->length + position->length;
front->next = NULL;
free(position);
    }
    else
    {
memset( position->task_name, 0, sizeof(char)*32 );

```

```

position->state = 1;

}
}
if( NULL==tail&&NULL==front )
{
memset( position->task_name, 0, sizeof(char)*32 );
position->state = 1;

}
printf("内存回收成功! \n");
}
}

//memoreturn 函数，用于回收内存
void memoreturn()
{
char tname[32];
printf("输入要收回的任务名\n");
scanf("%s",&tname);
Mreturn(tname);
system("pause");
}

int main()
{
int func_ = 0;
//初始化 Mhead
Mhead = (struct MZone*)malloc(sizeof(struct MZone));
Mhead->begin_addr = 0;
Mhead->length = MSIZE;
Mhead->state = 1;

memset(Mhead->task_name, 0, sizeof(char)*32 );
Mhead->next = NULL;

while( 1 )
{
printf("*****首次适应算法实现主存分配和回收系统（内存MSIZE）*****");
printf("|1:查看内存分配情况\n");
printf("|2:申请分配内存\n");
printf("|3:申请回收内存\n");
printf("|4:退出程序\n");

```

```

printf("*****\n*****");
scanf("%d",&func_);
switch( func_ )
{
case 1 :showmemory();break;
case 2 :memoallocate();break;
case 3 :memoreturn();break;
case 4 :return 1;
}
system("cls");
}
}

```

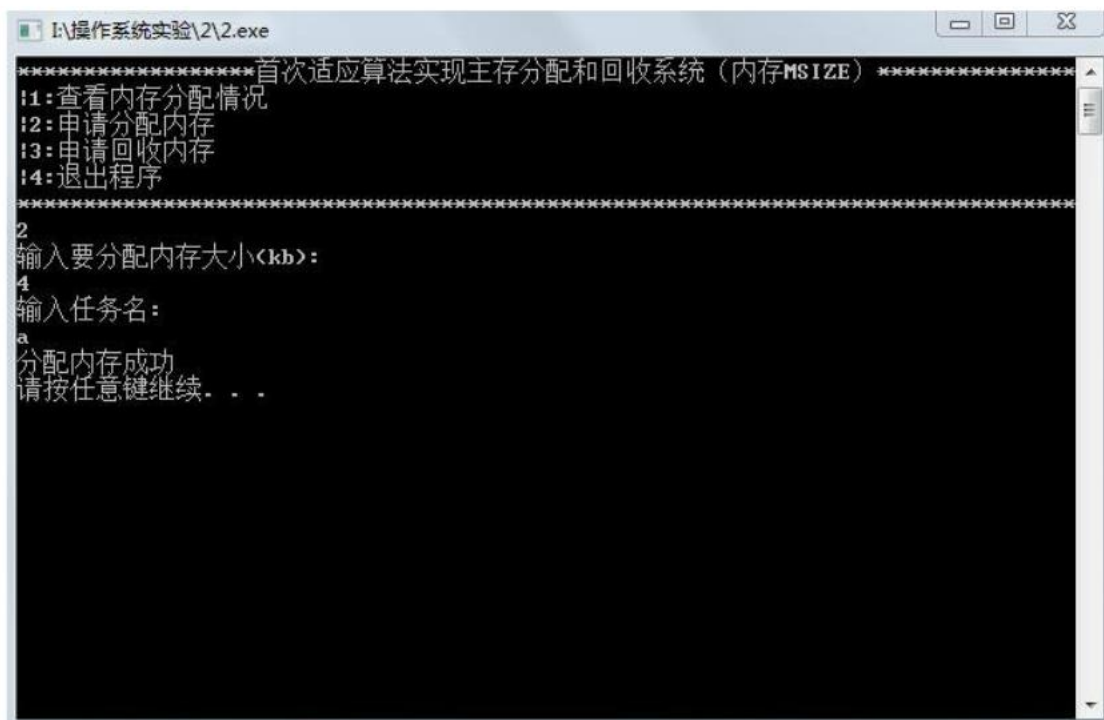
五、上机实验所用平台及相关软件

上机平台： Windows 7 + DEV- C++

六、 运行结果

依次建立进程 1 2 3 均分配 10 个内存单元,并进行重命名和超范围测试,并成功。位视图显示正确。删除 2 号后,位视图修改正确。试验成功。

1. 分配内存



```

I:\操作系统实验\2\2.exe
*****首次适应算法实现主存分配和回收系统（内存MSIZE）*****
!1:查看内存分配情况
!2:申请分配内存
!3:申请回收内存
!4:退出程序
*****
2
输入要分配内存大小(kb):
4
输入任务名:
a
分配内存成功
请按任意键继续...

```

```
I:\操作系统实验\2\2.exe
*****首次适应算法实现主存分配和回收系统（内存MSIZE）*****
!1:查看内存分配情况
!2:申请分配内存
!3:申请回收内存
!4:退出程序
*****
2
输入要分配内存大小(kb):
145
输入任务名:
b
没有符合大小的空闲分区，内存分配失败。
请按任意键继续. . .
```

2. 查看内存分配情况:

```
I:\操作系统实验\2\2.exe
*****首次适应算法实现主存分配和回收系统（内存MSIZE）*****
!1:查看内存分配情况
!2:申请分配内存
!3:申请回收内存
!4:退出程序
*****
1
内存的使用情况
beginaddr    length  state  task
0k           4k    CANUSE a
4k          124k    free
请按任意键继续. . .
```

3. 回收内存:

```
I:\操作系统实验\2\2.exe
*****首次适应算法实现主存分配和回收系统（内存MSIZE）*****
!1:查看内存分配情况
!2:申请分配内存
!3:申请回收内存
!4:退出程序
*****
3
输入要收回的任务名
a
内存回收成功!
请按任意键继续. . .
```

```
I:\操作系统实验\2\2.exe
*****首次适应算法实现主存分配和回收系统（内存MSIZE）*****
!1:查看内存分配情况
!2:申请分配内存
!3:申请回收内存
!4:退出程序
*****
3
输入要收回的任务名
b
内存中没有此任务! 请按任意键继续. . .
```

4. 回收内存后查看内存状况:


```
E:\操作系统实验\2\2.exe
*****首次适应算法实现主存分配和回收系统（内存MSIZE）*****
!1: 查看内存分配情况
!2: 申请分配内存
!3: 申请回收内存
!4: 退出程序
*****
1
内存的使用情况
beginaddr      length  state  task
0k             128k   free
请按任意键继续. . .
```

七. 心得体会

此代码主要分为三个板块，查看，分配和回收，主要问题在于回收时要考虑是否有两个或三个空闲的块，若有，则将其合并。本程序主要应用指针完成各块的链接，经过本次试验，我进一步加深了对使用指针方式表示和实现主存分配回收的理解，同时也得到了将系统理论实践的机会，让我不断进步。

实习三 磁盘存储空间的分配和回收

一、实习内容

模拟磁盘空闲空间的表示方法，以及模拟实现磁盘空间的分配和回收。

二、实习目的

磁盘初始化时把磁盘存储空间分成许多块（扇区），这些空间可以

被多个用户共享。用户作业在执行期间常常要在磁盘上建立文件或把已经建立在磁盘上的文件删去，这就涉及到磁盘存储空间的分配和回收。一个文件存放到磁盘上，可以组织成顺序文件（连续文件）、链接文件（串联文件）、索引文件等，因此，磁盘存储空间的分配有两种方式，一种是分配连续的存储空间，另一种是可以分配不连续的存储空间。怎样有效地管理磁盘存储空间是操作系统应解决的一个重要问题，通过本实习使学生掌握磁盘存储空间的分配和回收算法。

三、实习题目

用位示图管理磁盘存储空间

四、实习内容

1. 设计思想

(1) 为了提高磁盘存储空间的利用率，可在磁盘上组织成链接文件、索引文件，这类文件可以把逻辑记录存放在不连续的存储空间。为了表示哪些磁盘空间已被占用，哪些磁盘空间是空闲的，可用位示图来指出。位示图由若干字节构成，每一位与磁盘上的一块对应，“1”状态表示相应块已占用，“0”状态表示该块为空闲。位示图的形式与实习二中的位示图一样，但要注意，对于主存储空间和磁盘存储空间应该用不同的位示图来管理，绝不可混用。

(2) 申请一块磁盘空间时，由分配程序查位示图，找出一个为“0”的位，计算出这一位对应块的磁盘物理地址，且把该位置成占用状态“1”。假设现在有一个盘组共8个柱面，每个柱面有2个磁道（盘面），每个磁道分成4个物理记录。那么，当在位示图中找到某一字节的某

一位为“0”时，这个空闲块对应的磁盘物理地址为：

柱面号=字节号

磁道号= 位数 / 4

物理记录号= 位数 % 4

(3) 归还一块磁盘空间时，由回收程序根据归还的磁盘物理地址计算出归还块在位示图中的对应位，把该位置成“0”。按照(2)中假设的盘组，归还块在位示图中的位置计算如下：

字节号=柱面号

位数=磁道号×4+物理记录号

(4) 设计申请磁盘空间和归还磁盘空间的程序。

要求能接受来自键盘的空间申请及释放请求，要求能显示或打印程序运行前和运行后的位示图；分配时把分配到的磁盘空间的物理地址显示或打印出来，归还时把归还块对应于位示图的字节号和位数显示或打印出来。

2. 上机代码

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

struct weishitu //一张位示图表
{
    int bit[64] ;
    char name[64];
}weishitu;

struct weishitu *head = NULL; //指向位示图的指针

void showweishitu() //显示位示图
{
```

```

printf("位示图如下 (0 表示块可用): \n");
for(int i = 0; i<64;i++)
{
if( 0==i%8)
{
printf("\n");
}
printf("%2d:%d\t",i+1,head->bit[i]);
}
}

void fenpei() //分配位示图
{
int n;
int i;
int busy=0; //位示图有多少个 1
char name;

printf("请输入你所需的物理块数:\n");
scanf("%d",&n);
printf("请输入这些文件的名字:\n");
scanf("%s",&name);
for( i=0;i<64;i++)
{
busy =busy + head->bit[i]; //计算位示图有多少个 1
}
if( 64-busy<n ) //判断空闲区是否满足需要
{
printf("内存不足，分配失败！");
}
else
{
for(i=0 ; i<64; i++)
{
if( n!=0)
if( 0==head->bit[i])
{
printf(" 已分配： 第 %d 个柱面， 第 %d 个磁道 , 第 %d 个物理记录\n",i/8+1,i%8/4+1,(i%8)%4+1);
head->bit[i] = 1;
head->name[i]=name;
n--;

```

```

}
}
printf("分配成功! \n");
}
}

```

```

void huishou() //回收文件
{
char hname;
int i;
int flag=0;//标志回收是否成功
printf("请输入你要回收的文件名\n");
scanf("%s",&hname);
for( i=0;i<64;i++)
{
if(head->name[i]==hname )
{

head->bit[i] = 0;
printf("回收第%d 字节的第%d 位的物理块\n",i/8+1,i%8+1);
flag++;
}
}
if(flag==0)
printf("回收失败，你输入的文件不存在\n");
}

```

```

int main () //主函数
{
int i;
head=(struct weishitu*)malloc(sizeof(struct weishitu)); //初始化
for( i=0;i<64;i++)
{
head->bit[i]=0;
head->name[i]=0;
}
char choose = 0;
while( 1 )
{
printf("a:查看磁盘位示图\n");
printf("b:申请分配磁盘空间\n");
printf("c:申请回收磁盘空间\n");
printf("d:退出程序\n");

```

```

scanf("%s",&choose);
switch( choose )
{
case 'a' :showweishitu();break;
case 'b' :fenpei();break;
case 'c' :huishou();break;
case 'd' :return 0;
}
system("pause");
system("cls");
}

}

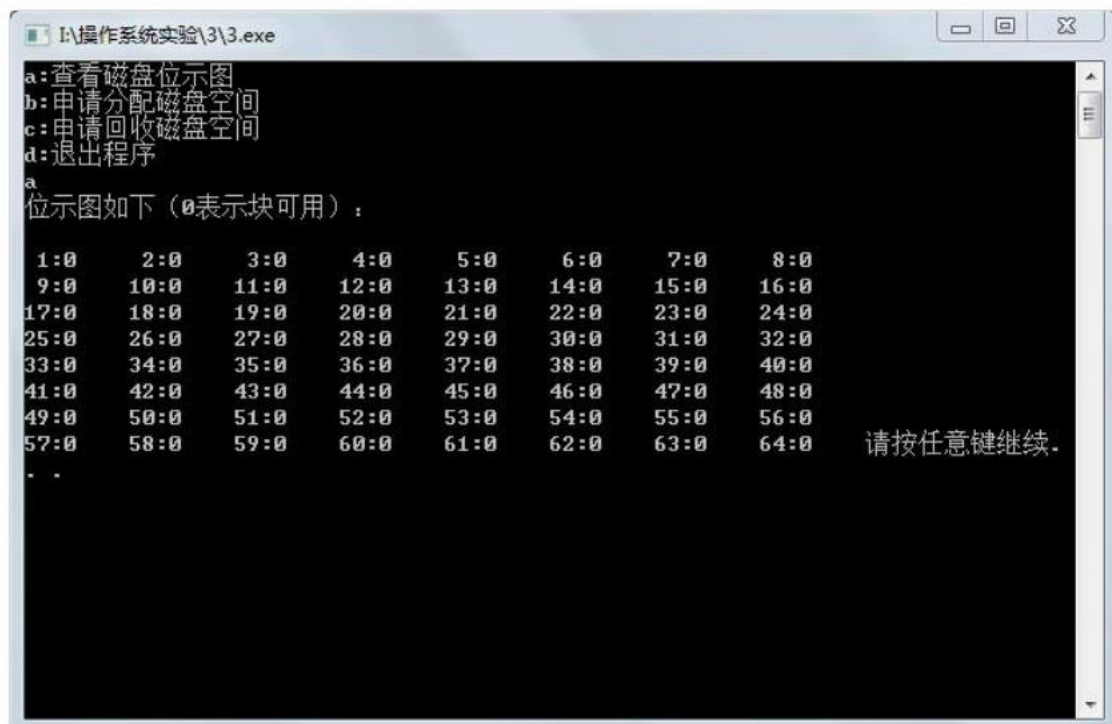
```

五、上机实验所用平台及相关软件

上机平台： Windows 7 + DEV- C++

六、运行结果

1.查看位示图：



2.申请分配磁盘空间

```
I:\操作系统实验\3\3.exe
已分配: 第2个柱面, 第1个磁道, 第4个物理记录
已分配: 第2个柱面, 第2个磁道, 第1个物理记录
已分配: 第2个柱面, 第2个磁道, 第2个物理记录
已分配: 第2个柱面, 第2个磁道, 第3个物理记录
已分配: 第2个柱面, 第2个磁道, 第4个物理记录
已分配: 第3个柱面, 第1个磁道, 第1个物理记录
已分配: 第3个柱面, 第1个磁道, 第2个物理记录
已分配: 第3个柱面, 第1个磁道, 第3个物理记录
已分配: 第3个柱面, 第1个磁道, 第4个物理记录
已分配: 第3个柱面, 第2个磁道, 第1个物理记录
已分配: 第3个柱面, 第2个磁道, 第2个物理记录
已分配: 第3个柱面, 第2个磁道, 第3个物理记录
已分配: 第3个柱面, 第2个磁道, 第4个物理记录
已分配: 第4个柱面, 第1个磁道, 第1个物理记录
已分配: 第4个柱面, 第1个磁道, 第2个物理记录
已分配: 第4个柱面, 第1个磁道, 第3个物理记录
已分配: 第4个柱面, 第1个磁道, 第4个物理记录
已分配: 第4个柱面, 第2个磁道, 第1个物理记录
已分配: 第4个柱面, 第2个磁道, 第2个物理记录
已分配: 第4个柱面, 第2个磁道, 第3个物理记录
已分配: 第4个柱面, 第2个磁道, 第4个物理记录
已分配: 第5个柱面, 第1个磁道, 第1个物理记录
已分配: 第5个柱面, 第1个磁道, 第2个物理记录
分配成功!
请按任意键继续. . .
```

```
I:\操作系统实验\3\3.exe
a: 查看磁盘位示图
b: 申请分配磁盘空间
c: 申请回收磁盘空间
d: 退出程序
b
请输入你所需的物理块数:
68
请输入这些文件的名字:
b
内存不足, 分配失败! 请按任意键继续. . .
```

3. 申请回收磁盘空间


```
I:\操作系统实验\3\3.exe
回收第2字节的第3位的物理块
回收第2字节的第4位的物理块
回收第2字节的第5位的物理块
回收第2字节的第6位的物理块
回收第2字节的第7位的物理块
回收第2字节的第8位的物理块
回收第3字节的第1位的物理块
回收第3字节的第2位的物理块
回收第3字节的第3位的物理块
回收第3字节的第4位的物理块
回收第3字节的第5位的物理块
回收第3字节的第6位的物理块
回收第3字节的第7位的物理块
回收第3字节的第8位的物理块
回收第4字节的第1位的物理块
回收第4字节的第2位的物理块
回收第4字节的第3位的物理块
回收第4字节的第4位的物理块
回收第4字节的第5位的物理块
回收第4字节的第6位的物理块
回收第4字节的第7位的物理块
回收第4字节的第8位的物理块
回收第5字节的第1位的物理块
回收第5字节的第2位的物理块
请按任意键继续...
```

```
I:\操作系统实验\3\3.exe
a: 查看磁盘位示图
b: 申请分配磁盘空间
c: 申请回收磁盘空间
d: 退出程序
c
请输入你要回收的文件名
d
回收失败，你输入的文件不存在
请按任意键继续...
```

4.回收后的位示图



七、心得体会

本程序采用位示图分配磁盘空间，主要考虑位示图与物理块的对应关系，比较简单。通过本次实验，我进一步加深了对连续磁盘空间的分配回收过程的理解，同时也得到了将系统理论实践的机会，以后还将更加深入的理解这部分的问题，不断加强能力。