文件系统实验报告 16281050 邢飞龙

实验简介

本实验要求在模拟的I/O系统之上开发一个简单的文件系统。

系统设计

关键结构设计

```
ldisk[10][1][10] ; //10个柱面，1个盘面，10个扇区，每个扇区512字节

//每个block大小为512kb，因此总块数为100块

block_graph[100]; //位图

struct File_discription{

    int index;

    int lenth;

    int position[3];//文件占用块

}//文件描述符结构体

typedef struct file {
    int index;
    string filename;//文件名
    int filelenth;
    string filecontent;
    int point;
};//文件结构体


file my_file[40];//文件索引
```

关键函数设计

1. create(): 根据指定的文件名创建新文件。

   功能:

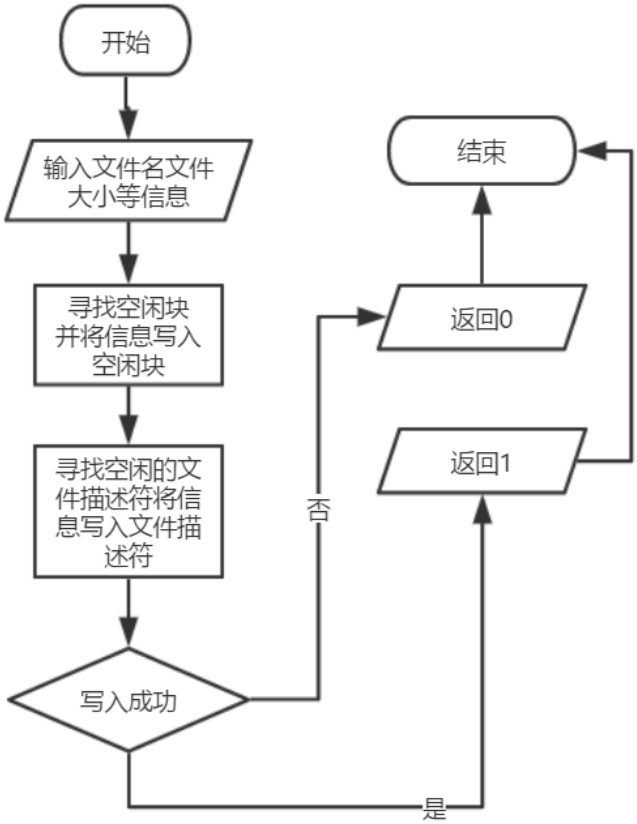
   设定文件名，文件长度，文件索引号

   寻找空闲块，写入空闲块，位图中相应位置1，

   寻找空闲描述符，写入描述符（文件大小，占用块）

返回状态信息

流程图：



代码

```
void create() {
    cout <<endl<< "请输入文件名：";
    string name;
    cin >> name;
    if (isexit(name)!=-1)
    {
        cout << "文件名已存在！" << endl;
    }
    else
    {
        cout << endl << "请输入文件大小：";
        int lenth;
        cin >> lenth;
        int num = lenth / 512 + 1;
        if (find_kblock(num) != -1) {
            int index = find_suoyin();
            my_file[index].filename = name;
            my_file[index].filelenth = lenth;
            my_file[index].index = index;
            my_file[index].point = 0;
            file_dis[index].index = index;
            file_dis[index].lenth = lenth;
            cout << endl << "创建成功";
```

```
        }
        else
        {
            cout << endl << "创建失败，空闲空间不足";
        }
    }
}
```

2. destroy(filename): 删除指定文件

功能：

删除文件信息

释放磁盘空间

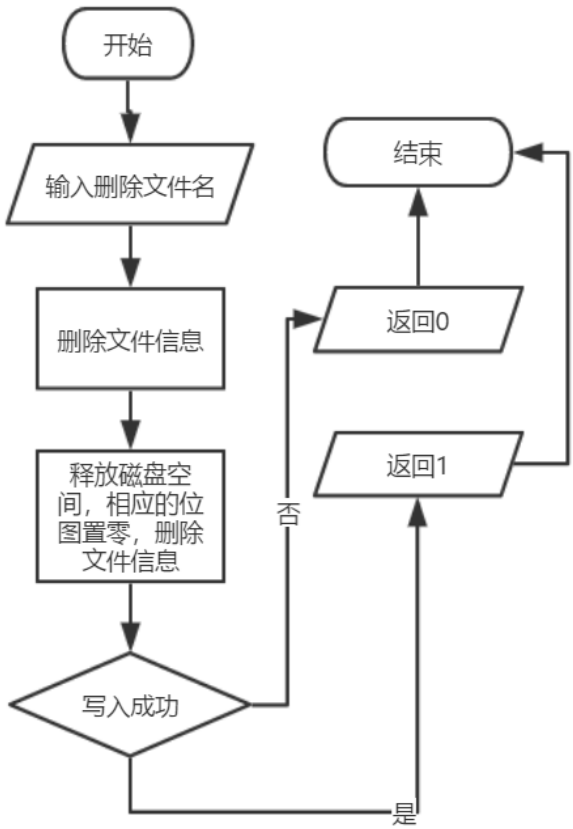
相应位图置0

删除文件描述符

删除文件信息

返回状态信息

流程图



代码

```
void destroy(string name) {
    if (isexit(name) == -1)
    {
        cout << "文件不存在！" << endl;
    }
    else
```

```
    {
        int temp_index = isexit(name);
        my_file[temp_index].filelenth = -1;
        my_file[temp_index].index = -1;
        my_file[temp_index].filecontent = "";
        my_file[temp_index].filename = "";
        my_file[temp_index].point = 0;
        file_dis[temp_index].index = -1;
        file_dis[temp_index].lenth = -1;
        for (int i = 0; i < 3; i++)
        {
            if (file_dis[temp_index].position[i] != -1) {
                block_graph[file_dis[temp_index].position[i]] = 0;
            }
        }
        cout << "文件已删除! " << endl;
    }
}
```

3. open(filename):

功能

打开文件。该函数返回的索引号可用于后续的read, write, lseek,或close操作。

代码

```
int open() {
    string name;
    cout << endl << "请输入要打开的文件名: ";
    cin >> name;
    if (isexit(name) == -1)
    {
        cout << "文件不存在! " << endl;
        return -1;
    }
    else
    {
        int temp_index = isexit(name);
        if (my_file[temp_index].isopen == 1)
        {
            cout << endl << "文件已打开";
        }
        else {
            cout <<endl<< "文件打开成功";
        }
        return temp_index;
    }
}
```

4. close(index):

功能

关闭指定文件。

代码

```
//文件关闭函数
void close(int index) {
    if (my_file[index].isopen==1)
    {
        my_file[index].isopen == 1;
        cout << endl << "文件关闭成功"<<endl;
    }
    else {
        cout << endl << "文件未打开"<<endl;
    }
}
```

5. read(index,count):

功能

从指定文件顺序读入*count*个字节。读操作从文件的读写指针指示的位置开始。

代码

```
//文件读取函数
string read(int index) {
    cout << endl << "请输入读取长度";
    int count;
    cin >> count;
    string temp = my_file[index].filecontent;
    if (my_file[index].point+count>=temp.length())
    {
        cout <<endl<< "文件长度不足";
        return "";
    }
    else
    {
        string temp2 = temp.substr(my_file[index].point, count);
        return temp2;
    }
}
```

6. write(index): **功能**

把输入字节顺序写入指定文件。写操作从文件的读写指针指示的位置开始。

代码

```
//文件写函数
void write(int index) {
    cout<<endl << "请输入写入内容";
    string temp;
    cin >> temp;
    my_file[index].filecontent =
my_file[index].filecontent.insert(my_file[index].point,temp);
}
```

7. lseek(index, pos): **功能**

   把文件的读写指针移动到*pos*指定的位置。*pos*是一个整数，表示从文件开始位置的偏移量。文件打开时，读写指针自动设置为0。每次读写操作之后，它指向最后被访问的字节的下一个位置。*lseek*能够在不进行读写操作的情况下改变读写指针能位置。

   代码

```
//文件读写指针移动
void lseek(int index) {
int pos;
    cout << endl << "请输读写指针位置; ";
    cin >> pos;
    my_file[index].point = pos;
    if (my_file[index].point < 0)
        my_file[index].point = 0;
    return;
}
```

8. directory () **功能**

   列表显示所有文件及其长度，以及其所占块的编号

   代码

```
//文件打印
void directory() {
    cout << endl << "--------------------文件列表--------------------" << endl;
    cout << "文件名" << "\t" << "文件长度"<<"\t"<<"文件所占块";
    for (int i = 0; i < 40; i++)
    {
        if (my_file[i].index != -1) {
            cout << endl << my_file[i].filename << "\t" << my_file[i].filelenth <<"B"
<< "\t";
            cout << "\t";
            for (int j = 0; j < 3; j++)
            {
                if (file_dis[i].position[j] != -1) {
                    cout << file_dis[i].position[j]<<" ";
                }
            }

        }
    }
    cout << endl << "-----------------------------------------------" << endl;
```

```
}
```

9. find_kblock(int num,int index) **功能**

   为建立的文件寻找空闲快并写入到空闲快中。

   代码

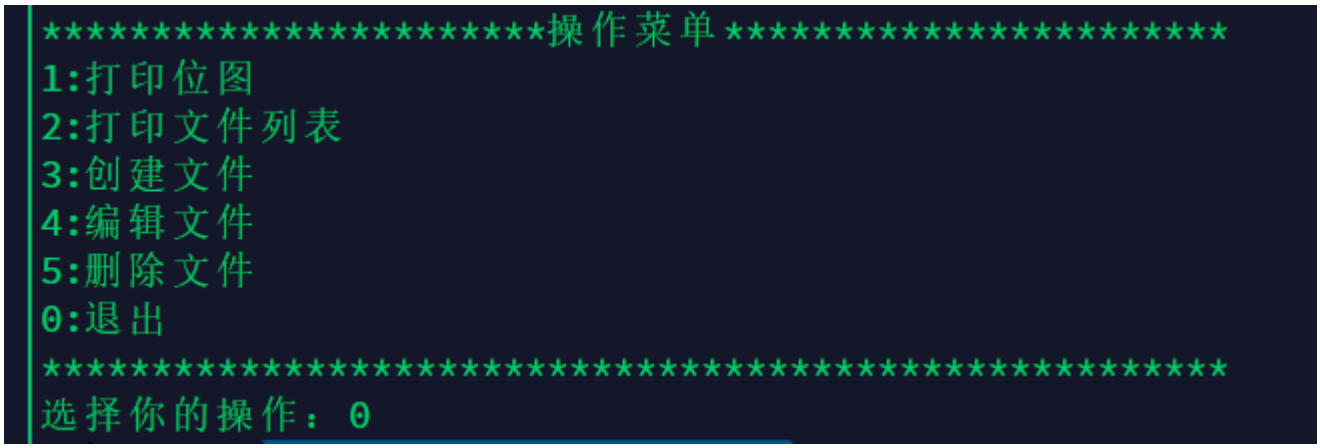
```
//寻找空闲块并写入
int find_kblock(int num,int index) {
    if (num == 1)
    {
        for (int i = 0; i < 100; i++)
        {
            if (block_graph[i] == 0) {
                block_graph[i] = 1;
                file_dis[index].position[0] = i;
                file_dis[index].position[1] = -1;
                file_dis[index].position[2] = -1;
                return i;
            }
        }
    }
    else if (num == 2) {
        for (int i = 0; i < 99; i++)
        {
            if (block_graph[i] == 0 && block_graph[i + 1] == 0)
            {
                block_graph[i] = 1;
                block_graph[i + 1] = 1;
                file_dis[index].position[0] = i;
                file_dis[index].position[1] = i + 1;
                file_dis[index].position[2] = -1;
                return i;
            }
        }
    }
    else
    {
        for (int i = 0; i < 98; i++)
        {
            if (block_graph[i] == 0 && block_graph[i + 1] == 0 && block_graph[i + 2]
== 0)
            {
                block_graph[i] = 1;
                block_graph[i + 1] = 1;
                block_graph[i + 2] = 1;
                file_dis[index].position[0] = i;
                file_dis[index].position[1] = i + 1;
                file_dis[index].position[2] = i + 2;

                return i;
            }
        }
```

```
    }
    return -1;
}
```

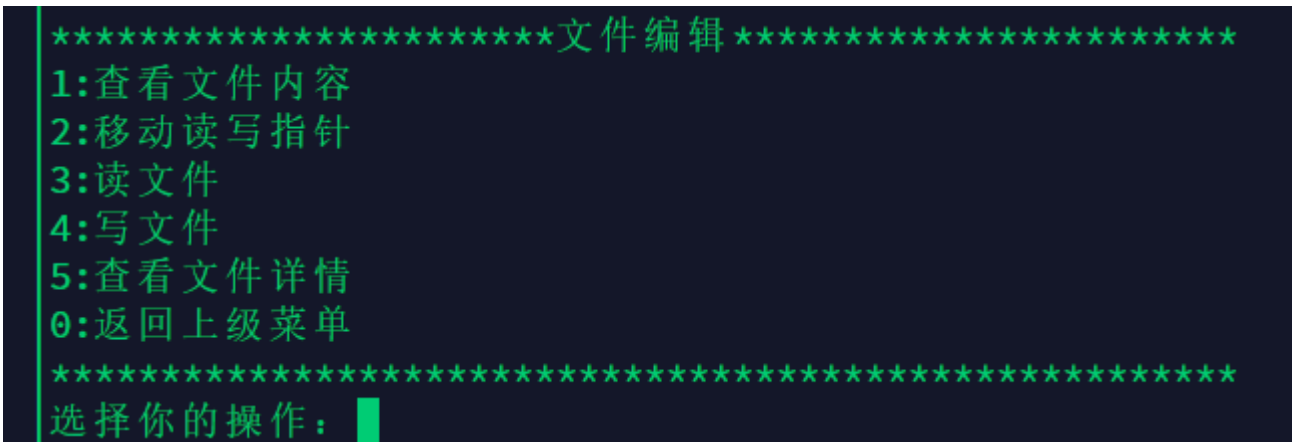
运行结果展示和分析：

功能目录

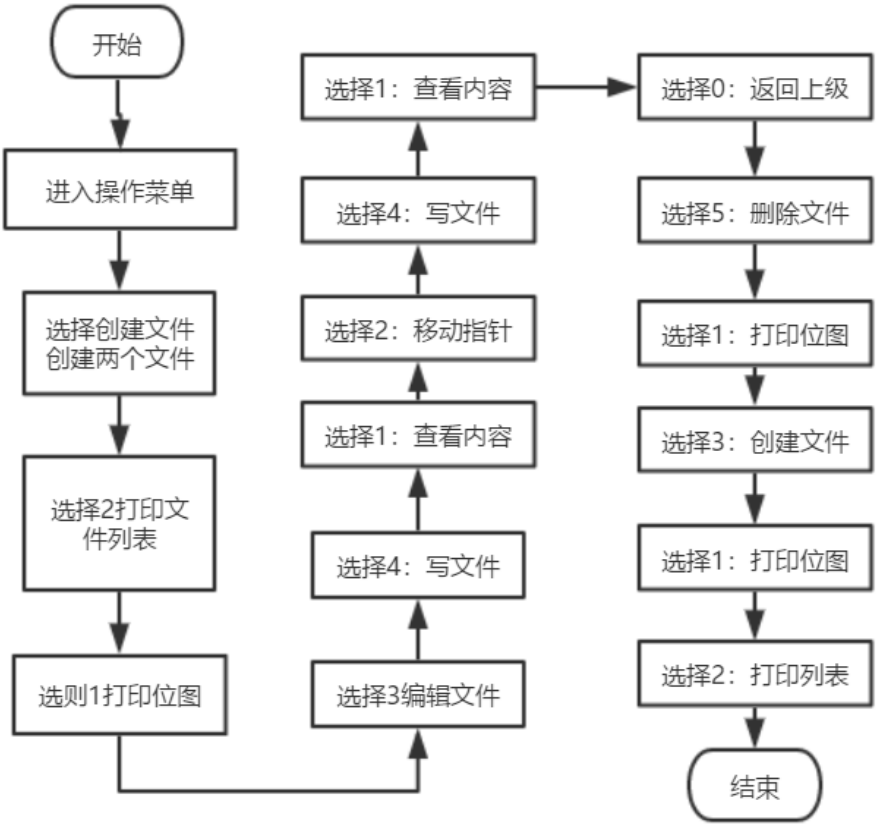
主操作目录



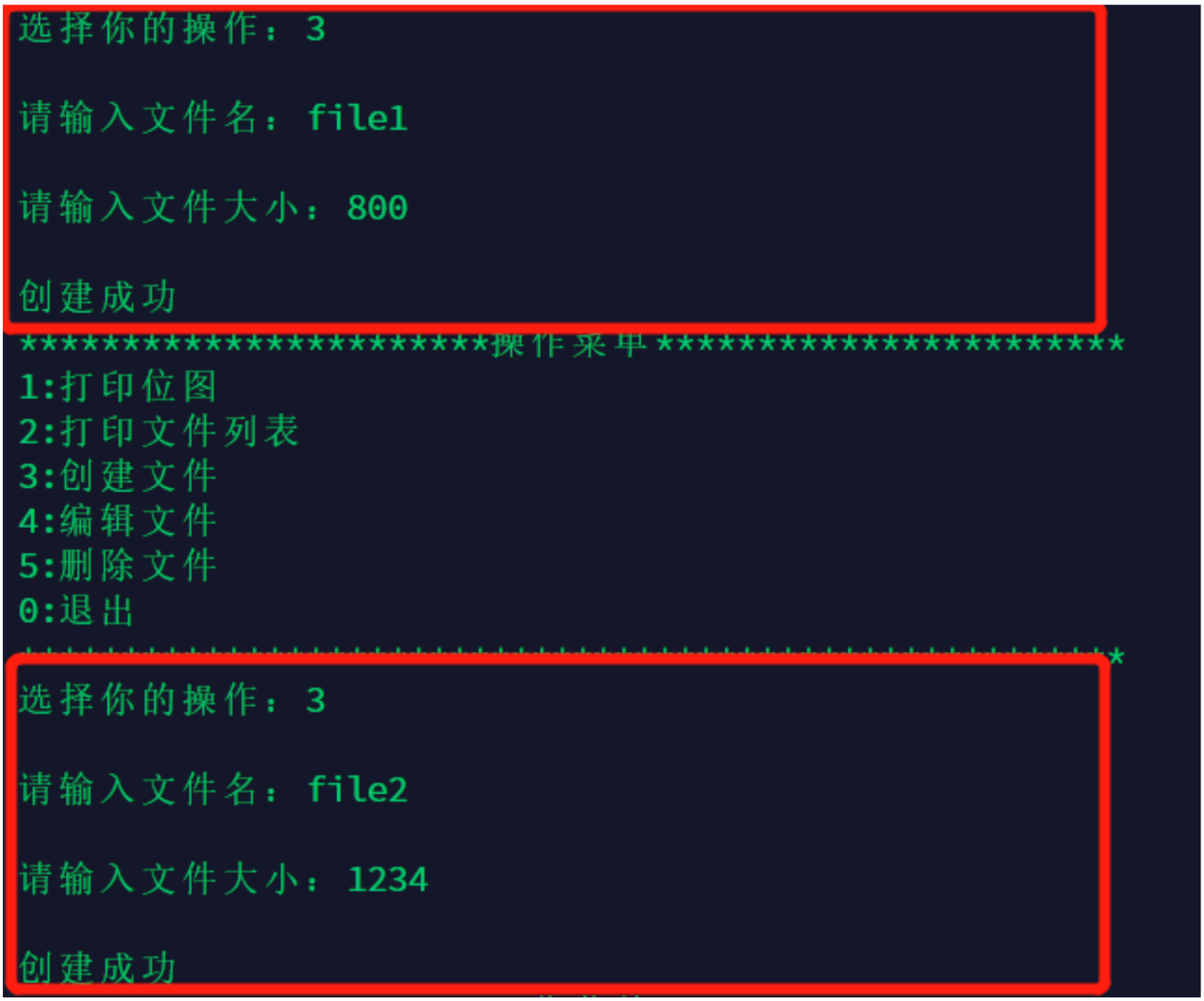
文件编辑目录



操作演示流程

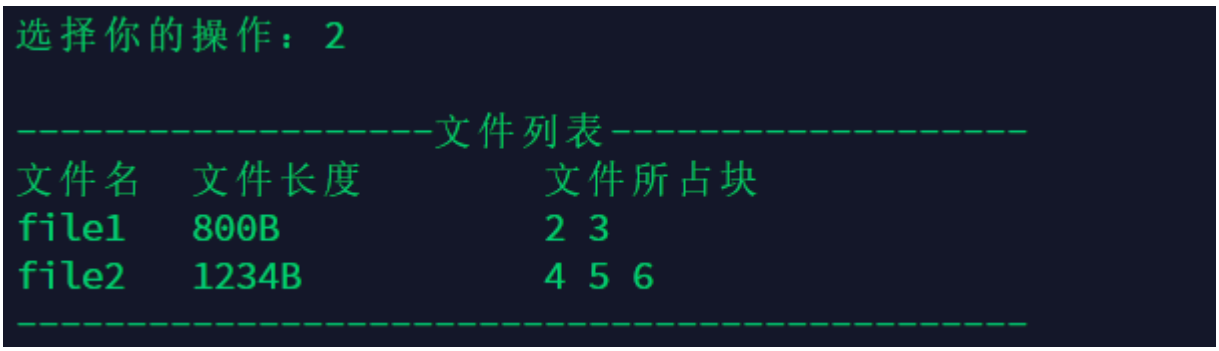

开始
进入操作菜单
选择创建文件
创建两个文件
选择2打印文
件列表
选则1打印位图
选择3编辑文件
选择4：写文件
选择1：查看内容
选择2：移动指针
选择1：查看内容
选择4：写文件
选择1：查看内容
选择0：返回上级
选择5：删除文件
选择1：打印位图
选择3：创建文件
选择1：打印位图
选择2：打印列表
结束

操作截图及解释



创建两个文件，一个大小为800B，一个大小1234B，每个块占512B，因此第一个文件占用两个块，第二个文件占用了3个块。由于第一块和第二块已经被位图和文件描述符占用，因此内存分配从第三块开始。

打印文件目录如图：



文件占用块的情况符合预期，打印位图：

```
选择你的操作：1

----------------------位图----------------------
1  1  1  1  1  1  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  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  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  0

------------------------------------------------
```

位图显示前7块被占用。下面选择file1进行编辑：

```
*************************操作菜单*************************
1:打印位图
2:打印文件列表
3:创建文件
4:编辑文件
5:删除文件
0:退出
**********************************************************
选择你的操作：4

请输入要打开的文件名：file1

文件打开成功

************************文件编辑************************
1:查看文件内容
2:移动读写指针
3:读文件
4:写文件
5:查看文件详情
0:返回上级菜单
**********************************************************
选择你的操作：
```

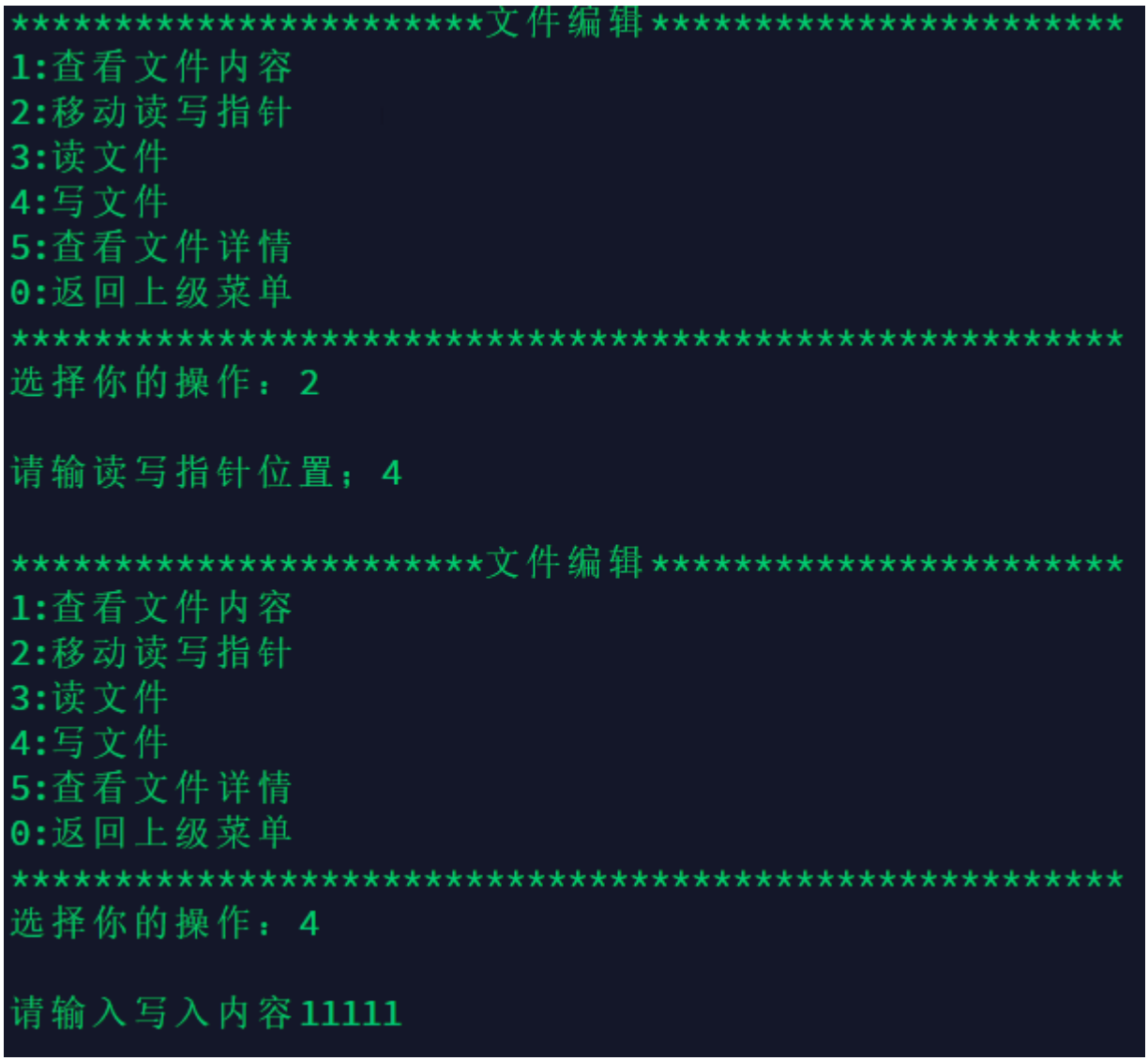
选择4，写文件，文件读写指针位于0；

```
选择你的操作：4

请输入写入内容xingfeilong16281050
```

选择查看文件内容：

```
**********文件编辑**********
1:查看文件内容
2:移动读写指针
3:读文件
4:写文件
5:查看文件详情
0:返回上级菜单
**************************************************
选择你的操作：1

文件内容为：xingfeilong16281050
```

选择移动读写指针在进行文件写入：



查看文件内容：

可以看到刚刚在读写指针位置（位置4）继续向后写入的11111被成功写入。

选择读文件：

```
**********文件编辑**********
1:查看文件内容
2:移动读写指针
3:读文件
4:写文件
5:查看文件详情
0:返回上级菜单
**************************************************
选择你的操作：3

请输入读取长度7
读取内容:feilong
```

由于读写指针停留在位置9，因此向后读7个就是字符串“feilong”，符合预期！

选择返回上级菜单：

```
选择你的操作：0

**********操作菜单**********
1:打印位图
2:打印文件列表
3:创建文件
4:编辑文件
5:删除文件
0:退出
**************************************************
选择你的操作：
```

选择文件删除：删除file1：

```
**********操作菜单**********
1:打印位图
2:打印文件列表
3:创建文件
4:编辑文件
5:删除文件
0:退出
**************************************************
选择你的操作：5

请输入要删除的文件名：file1
文件已删除！
```

查看此时的文件列表和位图：

```
选择你的操作：2

--------------------文件列表--------------------
文件名   文件长度          文件所占块
file2    1234B            4 5 6
------------------------------------------------

************************操作菜单************************
1:打印位图
2:打印文件列表
3:创建文件
4:编辑文件
5:删除文件
0:退出
****************************************************
选择你的操作：1

---------------------位图---------------------
1  1  0  0  1  1  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  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  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  0

------------------------------------------------
```

可以看到file1被删除后原来占用的空间被释放出来了。

选择新建文件：file3 大小：1455B

查看文件目录和位图：

```
选择你的操作：2

------------------文件列表------------------
文件名   文件长度          文件所占块
file3    1455B           7 8 9
file2    1234B           4 5 6
--------------------------------------------

**********************操作菜单**********************
1:打印位图
2:打印文件列表
3:创建文件
4:编辑文件
5:删除文件
0:退出
****************************************************
选择你的操作：1

--------------------位图--------------------
1  1  0  0  1  1  1  1  1  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
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  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

--------------------------------------------
```

由于文件file3占用3个块，而2，3两块不够file3空间，因此file3选择了7，8，9三块使用，如上图。

如果我们再创建一个小于512B的文件file4，则file4会占用第2块：

```
选择你的操作：3

请输入文件名：file4

请输入文件大小：222

创建成功
```

```
选择你的操作：2

--------------------文件列表--------------------
文件名   文件长度          文件所占块
file3    1455B            7 8 9
file2    1234B            4 5 6
file4    222B             2
------------------------------------------------
```

```
选择你的操作：1

----------------------位图----------------------
1  1  1  0  1  1  1  1  1  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
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  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

------------------------------------------------
```

本系统采用的是首次适应算法，通过实验验证成功！

退出系统：

```
**********操作菜单**********
1:打印位图
2:打印文件列表
3:创建文件
4:编辑文件
5:删除文件
0:退出
**************************************************
选择你的操作：0
```