操作系统作业说明文档

——文件系统模拟

学号：1950072

姓名：郑柯凡

指导老师：张惠娟

1. 项目背景

文件系统是操作系统中统一管理信息资源的一种软件，管理文件的存储、检索、更新，提供安全可靠的共享和保护手段，并且方便用户使用。对于操作系统而言，文件系统是必不可少的。

1.1 基本需求

1. 在内存中开辟一个空间作为文件存储器，在其上实现一个简单的文件系统。
2. 退出这个文件系统时，需要该文件系统的内容保存到磁盘上，以便下次可以将其回复到内存中来。
3. 采用一定的方式管理文件的物理结构、目录结构以及空闲的空间。
4. 提供如下操作：格式化、创建子目录、删除子目录、显示目录、更改当前目录、创建文件、打开文件、关闭文件、写文件、读文件、删除文件等等。

1.2 项目目的

1. 理解文件存储空间的管理。
2. 掌握文件的物理结构、目录结构和文件操作。
3. 实现简单的文件系统管理。
4. 加深文件系统实现过程的理解。

2.开发环境

- 操作系统：Windows 10
- 开发软件：Visual Studio 2019
- 开发语言：C#

3. 算法设计

3.1 物理结构的组织

本项目中采用综合模式的索引表管理文件的物理结构，即文件信息被存放在若干个不连续的物理块中，而这若干个不连续的物理块地址又按照文件顺序存放在多级索引中。使用这种组织方法将对每一个文件都建立一个多级索引表，记录文件各块在磁盘中的块编号，其中全部索引的第i个就指向了文件的第i块。虽然对每一个文件建立一个索引表将消耗大量的存储空间，且从磁盘中取文件也意味着大量的寻道操作，对时间也是一笔不小的开销，但使用这种组织方法具有既能顺序存取，又能随机存取；能够实现文件动态的更改；能充分利用磁盘空间的优点。

3.2 目录结构的组织

本项目采用目录项分解的树型目录组织文件控制块。目录项分解指将文件控制块FCB分成符号目录项和文件目录项两部分，符号目录项仅保留文件名和文件号用于搜索识别，而基本目录项存储了文件的详细信息及其物理地址，即上文提到的索引表。每次查找文件时，只需先查找到其符号目录项，再通过间接寻址的方式就能找到目标文件。使用该方法有效地缩减了目录项的大小，节省了内存空间，也减少了磁盘的访问次数，提高了搜索的时间。而采用树型目录的一个好处就是可以利用文件的路径快速地找到其文件控制块。

3.3 空闲空间的管理

本项目采用位图方法管理磁盘的空闲空间，即使用一串二进制位反映磁盘中各块的使用情况。申请物理块时只需先在位图中查找空闲的磁盘块，然后返回相应的磁盘块地址即可。位图的好处就在于仅用一串二进制位来表示磁盘分配情况，不依赖其他复杂的数据结构，空间消耗小，但适合各种物理结构，描述能力较好，性价比较高。同时，位运算在计算机中所需的时间也是很少的，能够节省空闲块的查询时间。

3.4 内存与磁盘的交互

该功能通过数据持久化机制实现，具体实现为使用C#提供的序列化方法，将程序中的类对象都声明为可序列化的后，就可以在程序结束时将文件管理系统的数据，即各个对象实例，用文件流的形式保存到磁盘中。下次运行程序时，只需要从磁盘中读取上一次的数据就可以恢复之前的运行环境。

4 核心代码说明

4.1 物理结构——索引组织——综合模式

- 因为综合模式中的多级索引中，各级索引具有很大的相似性，因此将单级索引单独建类，以便更好的组织多级索引结构。索引组织对每个文件都要建一个索引表，一个索引表有三级索引，索引中存放的是文件的块。若文件不大，则只会用到第一级索引；若文件超过了一级索引的容量大小，则一级索引的最后一项指向第二级索引的位置。第三级索引同理。下面列出了单级索引的代码实现以及索引表的属性。

```
namespace File_Management
{
    //物理结构_单级索引类
    class Index
    {
```

```
        public int[] index;
        public int index_id;
        public Index()
        {
            index = new int[100];
            index_id = 0;
        }
        public void addIndex(int data)
        {
            index[index_id] = data;
            index_id++;
        }
        public bool isfull()
        {
            return (index_id >= 100);
        }
    }
}
------------------------------------------------------------------------
----------------------
//索引表类（此处只列出了类的属性，函数方法详见代码）
//一级索引
private Index primaryindex;
//二级索引
private Index secondaryindex;
//三级索引
private Index thirdindex;
```

4.2 目录结构——树型目录（多级目录）

- 本项目中采用树型目录的目录组织方法，同时采用目录项分解法的优化方法，将FCB分解成符号目录项和基本目录项两部分，并只对符号目录项建树，有效地缩减了目录树的大小。符号目录项中只储存文件号和文件名，而基本目录项中存储了文件的完整信息，如文件大小、修改时间等等。此处展示了符号目录项类，以及它的建树过程。

```
namespace File_Management
{
    //符号目录项
    [Serializable]
    public class SymFCB
    {
        public enum FileType { folder, txt };
        //文件名
        public string fileName;
        //文件号
        public int file_id;
        //文件类型
        public FileType fileType;
        //符号目录项的树结构
        public SymFCB father = null, son = null, next = null, pre = null;
        //计数
        public static int file_counter = 0;

        //构造函数
        public SymFCB()
        {
            this.fileType = FileType.folder;
```

```
            this.file_id = file_counter++;
        }

        //赋值构造函数
        public SymFCB(string fileName,FileType fileType)
        {
            this.fileName = fileName;
            this.fileType = fileType;
            this.file_id = file_counter++;
        }

        //添加符号目录项
        public void addSonItem(SymFCB newItem)
        {
            //空文件
            if(this.son == null)
            {
                this.son = newItem;
                newItem.father = this;
            }
            //文件非空
            else
            {
                SymFCB temp = this.son;
                while(temp.next != null)
                {
                    temp = temp.next;
                }
                temp.next = newItem;
                newItem.pre = temp;
            }
        }

        //删除符号目录项
        public void remove()
        {
            if(pre==null)
            {
                father.son = next;
            }
            else if (pre != null)
            {
                pre.next = next;
            }
        }
    }
}
```

4.3 空闲空间的管理——位图

- 设置两个数组，一个块数组记录内存块中的信息，另一个布尔数组模拟一串二进制位，其中True代表相应的块为空，False代表相应的块已被占据。

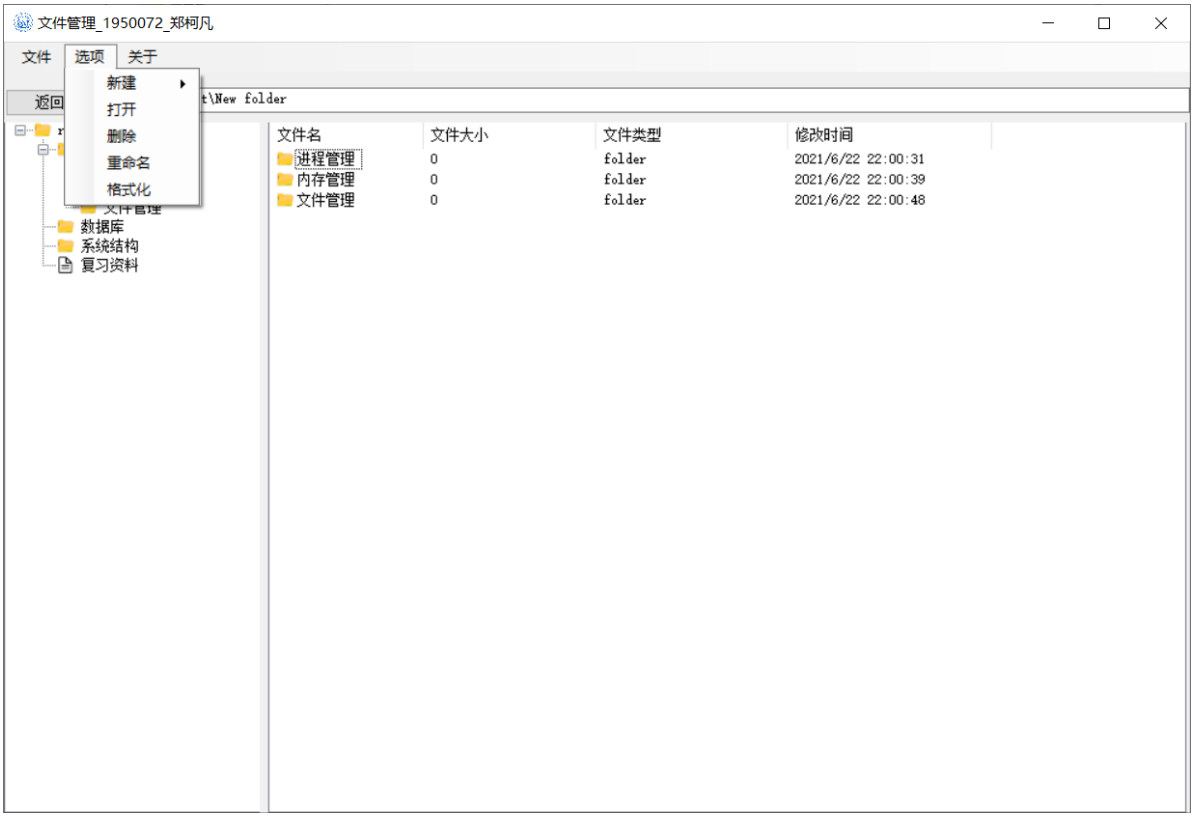
```
namespace File_Management
{
    //使用位图管理块
    [Serializable]
```

```
    public class BitMap
    {
        public static int Capcity = 1000000;
        private Block[] blocks = new Block[Capcity];
        private bool[] bitMap = new bool[Capcity];
        private int bit_id = 0;

        //位图构造函数
        public BitMap()
        {
            //所有块都空闲
            for (int i = 0; i < Capcity; i++)
            {
                bitMap[i] = true;
            }
        }

        //获取某一块的信息
        public string getBlock(int i)
        {
            return blocks[i].getInfo();
        }

        //找到第一个空的块
        public int allocateBlock(string data)
        {
            bit_id %= Capcity;
            int tempPointer = bit_id;
            while (true)
            {
                if (bitMap[tempPointer])
                {
                    blocks[tempPointer] = new Block();
                    blocks[tempPointer].setInfo(data);
                    bit_id = tempPointer + 1;
                    return tempPointer;
                }
                else
                {
                    tempPointer = (tempPointer + 1) % Capcity;
                }
                //遍历一整边还未找到，说明没有空的
                if (tempPointer == bit_id)
                {
                    break;
                }
            }
            return -1;
        }

        //取出一块
        public void withdraw(int index)
        {
            bitMap[index] = true;
        }

        //取出多块
        public void withdraw(List<int> indexs)
```

```
63          {
64              foreach(int i in indexs)
65              {
66                  bitMap[i] = true;
67              }
68          }
69
70          //建立一个索引表来保存一个文件块的数字
71          public IndexTable write(string data)
72          {
73              IndexTable table = new IndexTable();
74              while (data.Count() > 16)
75              {
76                  table.addIndex(allocateBlock(data.Substring(0, 15)));
77                  data = data.Remove(0, 15);
78              }
79              table.addIndex(allocateBlock(data));
80
81              return table;
82          }
83      }
84  }
```

5. 项目演示

5.1 界面展示



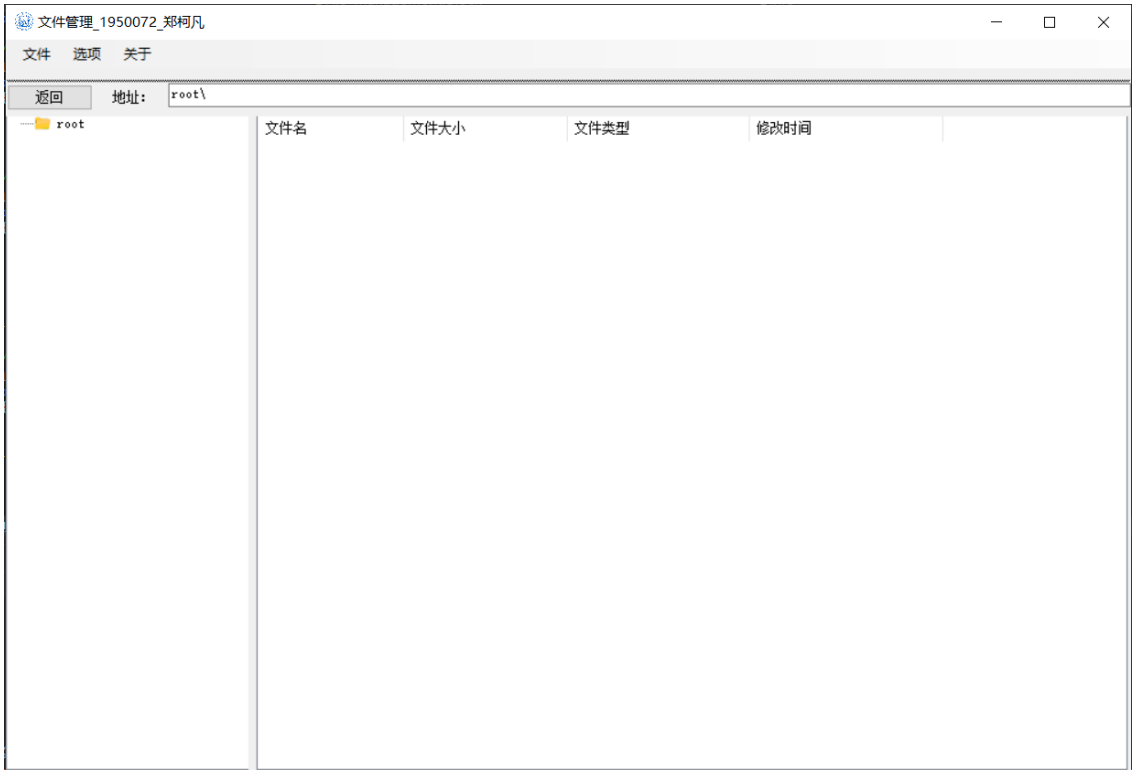
整个项目界面主要划分为3个部分，分别为顶部菜单栏、左侧目录树、以及右侧列表视图。

- 顶部菜单栏：该部分给用户提供了和系统交互的按钮。文件栏中用户可以选择保存当前文件或者加载历史文件；选项栏中，系统提供了新建、打开、删除、重命名文件或文件夹以及格式化整个文件系统的功能选项，用户可以自由进行操作；关于栏中介绍了关于本项目的一些信息。
- 左侧目录树：左侧目录树部分将整个文件系统的结构可视化，使用户能更好地观察文件系统的结构。

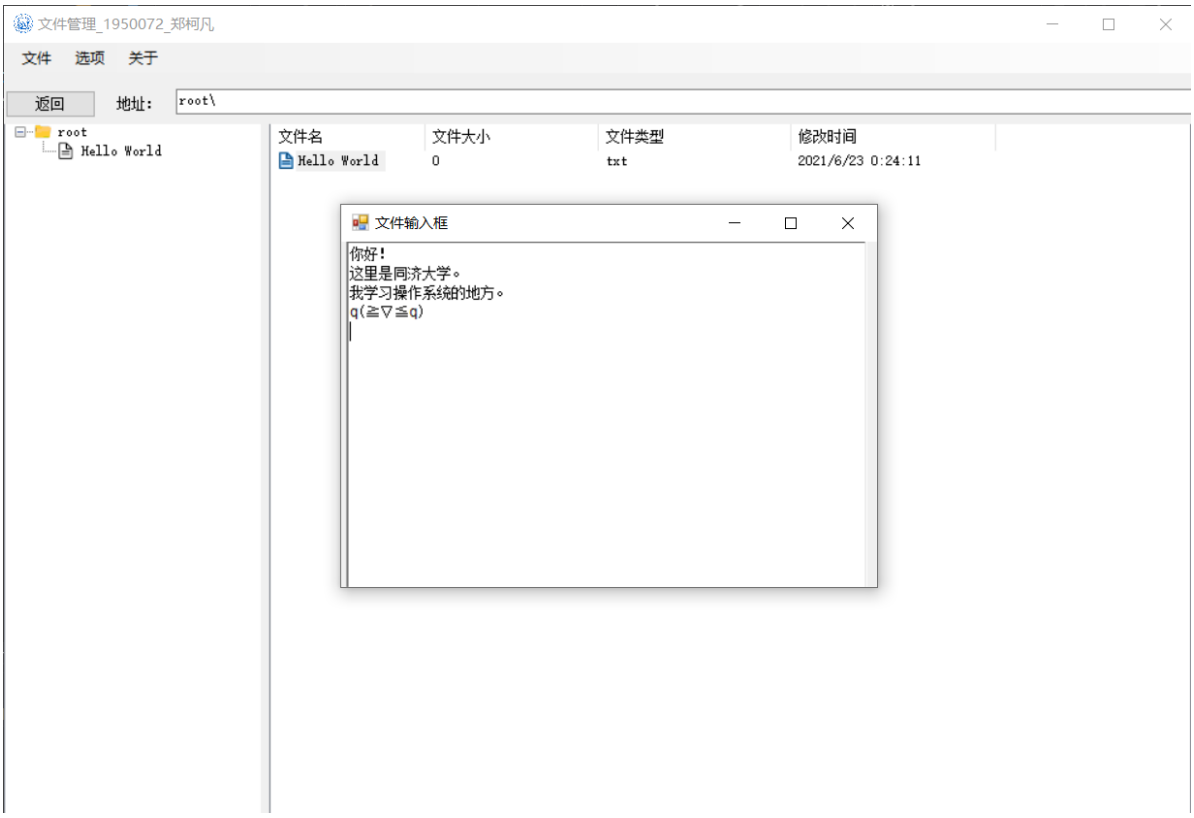
- 右侧列表视图：右侧列表视图首先给用户展示了当前的目录地址以及该目录下的文件，用户可以选择后退操作回到上一级目录，也可以双击列表视图中的文件或文件夹打开，进入下一级目录或进行编辑。此外，用户也可以通过右击各个文件进行不同的操作，操作与选项栏中一致。

5.2 操作说明

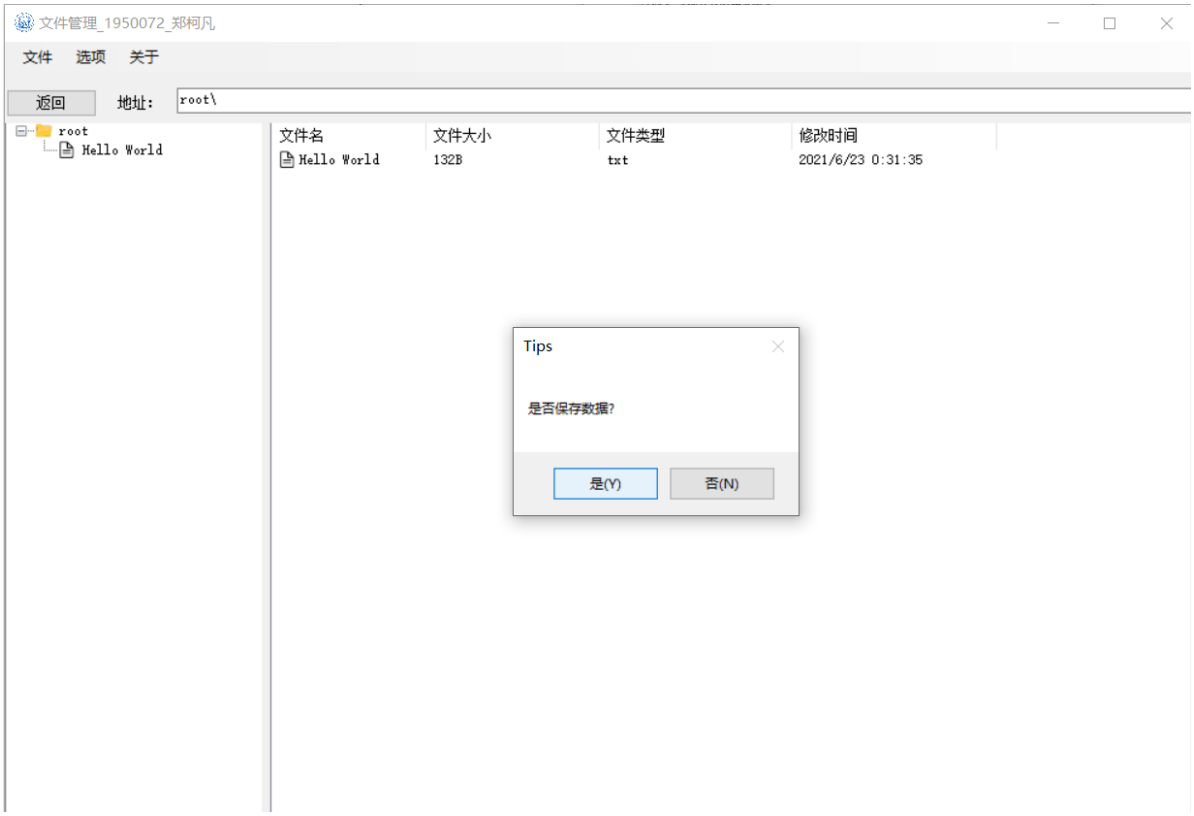
- 双击File_Management.exe，进入文件管理系统。



- 选择你想进行的操作：创建目录、编辑文件、重命名文件或文件夹等等。



- 保存文件后退出程序，目录及文件信息会保存至磁盘中，可在下次进入文件管理系统时载入。



6. 不足与改进

1. 本次在文件目录结构的组织、物理结构的组织、空闲空间的管理方面都各只使用了一种方法实现，之后会分别采用不同的方法实现该文件系统，深入学习文件系统的各项管理方法。
2. 本项目距离真实的文件系统还有很大的差距，用户与文件系统的交互空间还很大，计划后期实现更多的功能，如：文件搜索等。