**北 京 林 业 大 学**

**22 学年— 23 学年第2 学期 操作系统 实习报告书**

专 业： 计算机 \_\_\_ 班 级： 计算机21-1

姓 名： 李嘉梁 学 号： 211302104

实习地点： 计算中心 辅导教师： 田萱

实习题目： 虚拟多分区多级文件系统

实习环境： IDEA

排版格式要求： 五号字，中文用宋体，英文使用Time New Roman，单倍行距，图表有正确题注和正文中的引用及说明

# 一、课程设计目标：

设计一个虚拟的多分区多目录的文件系统，支持具体的文件存储空间管理、文件的物理结构、目录结构和文件操作的实现，以加深对文件系统内部功能和实现过程的理解。该系统第一级为磁盘分区，第二级为各个分区下的文件和各级目录，支持对文件和目录的创建、删除、修改和查询等操作。

# 二、课程设计内容：

### 1、任务1——磁盘建立和分区管理

#### 方案描述：

按照每个内存空间大小：小则使用空闲表空闲管理 大则使用位示图空闲管理的规则进行空闲空间管理。

#### 设计思路：

设计两种空闲空间管理数据结构：空闲表和位示图，根据需求大小选择管理方式并初始化，以后按具体管理方式分别管理。

#### 关键代码解析：

1. 空闲表空闲空间管理方式（适合于小型空间管理）

public class Freetable extends FSM{  
 List<Pair<Integer, Integer>> freetable; //<第一个盘块号,盘块数>  
  
 public Freetable(int method,int diskAddress,int diskSize) {  
 this.method = method;  
 this.freetable = new ArrayList<Pair<Integer, Integer>>();  
 this.totNum = Utility.*ceilDivide*(diskSize, Constants.*BLOCK\_SIZE*);//块数=磁盘分区大小/向上整除/块大小  
 this.freetable.add(new Pair(diskAddress,totNum));  
 }  
  
 public List<Pair<Integer, Integer>> getFreetable() {  
 return freetable;  
 }  
  
 public void setFreetable(List<Pair<Integer, Integer>> freetable) {  
 this.freetable = freetable;  
 }  
  
}

1. 位示图空闲空间管理方式（适合于大型空间管理）

public class Bitmap extends FSM{  
 private int[][] bitmap;// 0表示空闲 1表示占用  
  
 public Bitmap(int method,int diskSize) {  
 this.method = method;  
 this.totNum = Utility.*ceilDivide*(diskSize, Constants.*BLOCK\_SIZE*);//块数=磁盘分区大小/向上整除/块大小  
 this.freeNum = Utility.*ceilDivide*(diskSize, Constants.*BLOCK\_SIZE*);//块数=磁盘分区大小/向上整除/块大小  
 this.bitmap = new int[totNum/Constants.*COLUMN\_COUNT*+1][Constants.*COLUMN\_COUNT*];  
 }  
  
 public int[][] getBitmap() {  
 return bitmap;  
 }  
  
 public void setBitmap(int[][] bitmap) {  
 this.bitmap = bitmap;  
 }  
  
}

1. 磁盘空间管理包括展示磁盘分区（showDisk）、创建磁盘分区（mkDisk）、切换磁盘分区（chgDisk）、移动目录中文件（moveDir）、释放目录内存（freeDir）、释放文件内存（freeFile）、写入磁盘分区（writeToDisk）、寻找空闲块（findEmpty）、显示空闲空间（showFree）。

public interface DiskService {  
 void initDisk(String[] initInfo);  
  
 */\*\*  
 \* 显示磁盘分区名称 修改日期 类型 容量  
 \*/* void showDisk(String diskName);  
  
 */\*\*  
 \* 创建磁盘分区  
 \*  
 \** ***@param*** *diskName 磁盘分区名  
 \** ***@param*** *diskAddress 磁盘地址  
 \** ***@param*** *diskSize 磁盘分区大小  
 \** ***@return*** *{****@link*** *Boolean}  
 \*/* public FCB mkDisk(String diskName, int diskAddress, int diskSize);  
  
 */\*\*  
 \* 切换磁盘分区  
 \* chgDisk .. 可切换上一级  
 \*  
 \** ***@param*** *diskName 磁盘分区名  
 \** ***@return*** *{****@link*** *Boolean}  
 \*/* Boolean chgDisk(String diskName);  
  
 */\*\*  
 \* (跨盘)转移目录中所有文件（借助栈）  
 \*  
 \** ***@param*** *fcb\_src FCB(源)  
 \** ***@param*** *diskName\_src 磁盘分区名(源)  
 \** ***@param*** *diskName\_des 磁盘分区名(目标)  
 \** ***@return*** *{****@link*** *Boolean}  
 \*/* Boolean moveDir(FCB fcb\_src, String diskName\_src, String diskName\_des);  
  
 */\*\*  
 \* 释放目录中所有文件占用内存（借助栈）  
 \*  
 \** ***@param*** *fcb FCB  
 \** ***@param*** *diskName 磁盘分区名  
 \** ***@return*** *{****@link*** *Boolean}  
 \*/* Boolean freeDir(FCB fcb, String diskName);  
  
 */\*\*  
 \* 释放文件空间 修改FAT表和位示图  
 \*  
 \** ***@param*** *fcb FCB  
 \** ***@param*** *diskName 磁盘分区名  
 \** ***@return*** *{****@link*** *Boolean}  
 \*/* Boolean freeFile(FCB fcb, String diskName);  
  
 */\*\*  
 \* 写入磁盘  
 \* 文件内容写入磁盘  
 \*  
 \** ***@param*** *content 文件内容  
 \** ***@param*** *diskName 磁盘分区名  
 \** ***@return*** *int 返回第一块的磁盘号  
 \*/* int writeToDisk(String content, String diskName);  
  
*/\*\*  
 \* 寻找空闲块  
 \*  
 \** ***@param*** *needNum 需求块数  
 \** ***@param*** *diskName 磁盘分区名  
 \** ***@return*** *int 块号  
 \*/*int findEmpty(int needNum, String diskName);  
  
*/\*\*  
 \* 显示空闲空间  
 \*  
 \** ***@param*** *diskName 磁盘分区名  
 \*/*void showFree(String diskName);

}

#### **InitDisk**命令说明：

###### 设计流程：

用户输入 [ 命令 + （磁盘分区名 + 磁盘分区大小）\*n ]

文件系统会检测输入的磁盘分区名和磁盘分区大小的合法性，

如果合法则按照每个内存空间大小“[0,1024)使用空闲表空闲管理 [1024,16384]使用位示图空闲管理”的规则进行空闲空间管理。

###### 关键代码解析：

检查命令合法性：

数组长度=1则输出命令提示

否则检查是否为大小>=3且满足2n+1的数组

case "InitDisk":/\*1\*/  
 if(inputs.length == 1 || inputs.length %2 == 0){ //>=3  
 System.*out*.println("InitDisk [DiskName#] [DiskSize]KB [DiskName#] [DiskSize]KB...");  
 System.*out*.println("提示: 磁盘分区名须以\"#\"结尾 磁盘分区大小须为[0,16384]的整数 [0,1024)使用空闲表空闲管理 [1024,16384]使用位示图空闲管理");  
 break;  
 }  
 *dataService*.initData();  
 *diskService*.initDisk(inputs);  
 break;

检查合法性：

1. 检查磁盘分区大小是否为整数
2. 预测加上磁盘分区大小后是否超出磁盘大小
3. 磁盘分区名是否以“#”结尾

@Override  
public void initDisk(String[] initInfo) {  
 Disk disk = Disk.*getINSTANCE*();  
  
 //磁盘分区表  
 Map<String, FCB> diskMap = disk.getDPT();  
 int address\_cnt=0;  
 int driveSize;  
 for (int i = 1; i < initInfo.length - 1; i++){  
 if(i%2 != 0){//0 [1 2] [3 4]  
 try {  
 driveSize = Integer.*parseInt*(initInfo[i + 1]);  
 }catch(NumberFormatException e){  
 System.*out*.println("[error] 磁盘分区大小非整数 请重新初始化");  
 return;  
 }  
 int needNum = Utility.*ceilDivide*(driveSize, Constants.*BLOCK\_SIZE*);//块数=磁盘分区大小/向上整除/块大小  
 if(address\_cnt+needNum-1 > Constants.*BLOCK\_COUNT*-1){  
 System.*out*.println("[error] 超出磁盘空间大小 请重新初始化");  
 return;  
 }  
 if(!initInfo[i].endsWith("#")){  
 System.*out*.println("[error] 磁盘分区名未以\"#\"结尾 请重新初始化");  
 return;  
 }  
 FCB fcb = *diskService*.mkDisk(initInfo[i],address\_cnt,driveSize);  
 diskMap.put(initInfo[i],fcb);  
 address\_cnt+=needNum;  
 }  
 }  
 disk.setDPT(diskMap);  
 System.*out*.println("[success] 磁盘分区初始化成功");  
  
 if(*dirService*.chgDir("/" + initInfo[1])) { //chgDisk is included  
 System.*out*.println("[success] 已切换至首个磁盘分区");  
 }  
}

创建磁盘分区：

检查合法性：

1. 判断磁盘分区名重复

按大小选择磁盘分区空闲管理方式（“[0,1024)使用空闲表空闲管理 [1024,16384]使用位示图空闲管理”）并将其存入磁盘分区首个物理块

@Override  
public FCB mkDisk(String diskName, int diskAddress, int diskSize) {  
 FCB curDir = Memory.*getInstance*().getCurDir();  
 List<FCB> children = curDir.getChildren();  
// //判空  
// if(Objects.isNull(diskName)){  
// System.out.println("[error] 磁盘分区名不可为空");  
// return null;  
// }

//判断重复  
 diskName = diskName.trim(); //去除首尾空格  
 for (FCB child : children) {  
 if(child.getFileName().equals(diskName)){  
 System.*out*.println("[error] 磁盘分区名重复 请重新命名");  
 return null;  
 }  
 }  
 //创建索引节点 创建FCB 文件大小为0 空文件  
 IndexNode indexNode = new IndexNode(diskSize, diskAddress, 0, new Date());  
 FCB fcb = new FCB(diskName, "DISK", indexNode, curDir, new LinkedList<>());  
 //在首块设置空闲空间管理  
 FSM fsm = diskSize>=Constants.*THRESHOLD*?new Bitmap(Constants.*BITMAP*,diskSize):new Freetable(Constants.*FREETABLE*,diskAddress,diskSize);  
 Disk.*getINSTANCE*().getBlocks()[diskAddress].setFsm(fsm);  
 //将文件控制块放入磁盘的fcb集合  
 Disk.*getINSTANCE*().getFcbList().add(fcb);  
 //修改父目录的文件项 加入父目录儿子集合  
 curDir.getIndexNode().addFcbNum();  
 curDir.getChildren().add(fcb);  
 System.*out*.println("[success] 创建磁盘分区 " + diskName + " 成功");  
 return fcb;  
}

###### 结果描述：

初始化磁盘分区为：A分区大小为xxM；B分区大小为yyM；B分区大小为yyM；C分区大小为yyM；... ...；要求至少两个磁盘分区。初始化完毕后默认进入首个分区，即以首个分区为当前分区,盘符为A#，B#。

#### **ChgDisk**命令说明：

###### 设计流程：

用户输入 [ 命令 + 磁盘分区名 ]

文件系统会检测输入的磁盘分区名的合法性，

如果合法则切换当前磁盘分区。

###### 关键代码解析：

检查命令合法性：

数组长度=1则输出命令提示

否则正常执行

case "ChgDisk":/\*2\*/  
 if(inputs.length == 1){  
 System.*out*.println("ChgDisk [DiskName#]");  
 break;  
 }  
 *diskService*.chgDisk(inputs[1]);  
 break;

先判断磁盘分区是否存在，如果存在则切换当前分区和路径，否则报错。

@Override  
public Boolean chgDisk(String diskName) {  
 String path = "/"+diskName;  
 FCB fcb = *dirService*.pathResolve(path);  
 //null 不存在  
 if(Objects.*isNull*(fcb)){  
 System.*out*.println("[error] 目标磁盘分区不存在");  
 return false;  
 }else {  
 Memory.*getInstance*().setCurDisk(fcb);  
 Memory.*getInstance*().setCurDir(fcb);  
 }  
 return true;  
}

###### 结果描述：

将当前分区切换到B分区。

### 2、任务1——磁盘空闲空间管理

#### 方案描述：

按照每个内存空间大小：小则使用空闲表空闲管理 大则使用位示图空闲管理的规则进行空闲空间管理。

#### 设计思路：

设计两种空闲空间管理数据结构：空闲表和位示图，根据需求大小选择管理方式并初始化，以后按具体管理方式分别管理。

#### 关键代码解析：

1. 空闲表空闲空间管理方式（适合于小型空间管理）

public class Freetable extends FSM{  
 List<Pair<Integer, Integer>> freetable; //<第一个盘块号,盘块数>  
  
 public Freetable(int method,int diskAddress,int diskSize) {  
 this.method = method;  
 this.freetable = new ArrayList<Pair<Integer, Integer>>();  
 this.totNum = Utility.*ceilDivide*(diskSize, Constants.*BLOCK\_SIZE*);//块数=磁盘分区大小/向上整除/块大小  
 this.freetable.add(new Pair(diskAddress,totNum));  
 }  
  
 public List<Pair<Integer, Integer>> getFreetable() {  
 return freetable;  
 }  
  
 public void setFreetable(List<Pair<Integer, Integer>> freetable) {  
 this.freetable = freetable;  
 }  
  
}

1. 位示图空闲空间管理方式（适合于小型空间管理）

public class Bitmap extends FSM{  
 private int[][] bitmap;// 0表示空闲 1表示占用  
  
 public Bitmap(int method,int diskSize) {  
 this.method = method;  
 this.totNum = Utility.*ceilDivide*(diskSize, Constants.*BLOCK\_SIZE*);//块数=磁盘分区大小/向上整除/块大小  
 this.freeNum = Utility.*ceilDivide*(diskSize, Constants.*BLOCK\_SIZE*);//块数=磁盘分区大小/向上整除/块大小  
 this.bitmap = new int[totNum/Constants.*COLUMN\_COUNT*+1][Constants.*COLUMN\_COUNT*];  
 }  
  
 public int[][] getBitmap() {  
 return bitmap;  
 }  
  
 public void setBitmap(int[][] bitmap) {  
 this.bitmap = bitmap;  
 }  
  
}

1. 磁盘空间管理包括展示磁盘分区（showDisk）、创建磁盘分区（mkDisk）、切换磁盘分区（chgDisk）、移动目录中文件（moveDir）、释放目录内存（freeDir）、释放文件内存（freeFile）、写入磁盘分区（writeToDisk）、寻找空闲块（findEmpty）、显示空闲空间（showFree）。

public interface DiskService {  
 void initDisk(String[] initInfo);  
  
 */\*\*  
 \* 显示磁盘分区名称 修改日期 类型 容量  
 \*/* void showDisk(String diskName);  
  
 */\*\*  
 \* 创建磁盘分区  
 \*  
 \** ***@param*** *diskName 磁盘分区名  
 \** ***@param*** *diskAddress 磁盘地址  
 \** ***@param*** *diskSize 磁盘分区大小  
 \** ***@return*** *{****@link*** *Boolean}  
 \*/* public FCB mkDisk(String diskName, int diskAddress, int diskSize);  
  
 */\*\*  
 \* 切换磁盘分区  
 \* chgDisk .. 可切换上一级  
 \*  
 \** ***@param*** *diskName 磁盘分区名  
 \** ***@return*** *{****@link*** *Boolean}  
 \*/* Boolean chgDisk(String diskName);  
  
 */\*\*  
 \* (跨盘)转移目录中所有文件（借助栈）  
 \*  
 \** ***@param*** *fcb\_src FCB(源)  
 \** ***@param*** *diskName\_src 磁盘分区名(源)  
 \** ***@param*** *diskName\_des 磁盘分区名(目标)  
 \** ***@return*** *{****@link*** *Boolean}  
 \*/* Boolean moveDir(FCB fcb\_src, String diskName\_src, String diskName\_des);  
  
 */\*\*  
 \* 释放目录中所有文件占用内存（借助栈）  
 \*  
 \** ***@param*** *fcb FCB  
 \** ***@param*** *diskName 磁盘分区名  
 \** ***@return*** *{****@link*** *Boolean}  
 \*/* Boolean freeDir(FCB fcb, String diskName);  
  
 */\*\*  
 \* 释放文件空间 修改FAT表和位示图  
 \*  
 \** ***@param*** *fcb FCB  
 \** ***@param*** *diskName 磁盘分区名  
 \** ***@return*** *{****@link*** *Boolean}  
 \*/* Boolean freeFile(FCB fcb, String diskName);  
  
 /\*\*  
 \* 文件内容写入磁盘分区 *\*  
 \** ***@param*** *content 文件内容  
 \** ***@param*** *diskName 磁盘分区名  
 \** ***@return*** *int 返回第一块的磁盘号  
 \*/* int writeToDisk(String content, String diskName);  
  
*/\*\*  
 \* 寻找空闲块  
 \*  
 \** ***@param*** *needNum 需求块数  
 \** ***@param*** *diskName 磁盘分区名  
 \** ***@return*** *int 块号  
 \*/*int findEmpty(int needNum, String diskName);  
  
*/\*\*  
 \* 显示空闲空间  
 \*  
 \** ***@param*** *diskName 磁盘分区名  
 \*/*void showFree(String diskName);

}

#### **ShowDisk**命令说明：

###### 设计流程：

用户输入 [ 命令 + 磁盘分区名 ]

文件系统会检测输入的磁盘分区名的合法性，

如果合法则展示当前磁盘分区。

###### 关键代码解析：

检查命令合法性：

数组长度=1则输出命令提示

否则正常执行

case "ShowDisk":/\*3\*/  
 if(inputs.length == 1){  
 System.*out*.println("ShowDisk [DiskName#]");  
 break;  
 }  
 *diskService*.showDisk(inputs[1]);  
 break;

先判断磁盘分区是否存在，如果存在则展示当前分区，否则报错。

@Override  
public void showDisk(String diskName) {  
  
 FCB fcb\_disk;  
 if(diskName.equals("")){  
 fcb\_disk = Memory.*getInstance*().getCurDisk();  
 }  
 else{  
 String path = "/"+diskName;  
 fcb\_disk = *dirService*.pathResolve(path);  
 //null 不存在  
 if(Objects.*isNull*(fcb\_disk)){  
 System.*out*.println("[error] 目标磁盘分区不存在");  
 return;  
 }  
 }  
  
 View view = new View();  
 System.*out*.println("名称\t\t\t\t总大小\t\t\t已用大小\t\t\t可用大小");  
  
 view.showFcb\_disk(fcb\_disk);  
 show\_free(diskName);  
}

通过磁盘分区的索引结点获得其大小信息

public void showFcb\_disk(FCB fcb){  
 SimpleDateFormat dateFormat= new SimpleDateFormat("yyyy/MM/dd HH:mm");  
 IndexNode indexNode = fcb.getIndexNode();//name-total-used-unused  
 System.*out*.printf("%-15s\t%-15s\t%-15s\t%-15s",  
// indexNode.getFcbNum(),  
 fcb.getFileName(),  
 indexNode.getSize() +" KB",  
 indexNode.getSize()-indexNode.getSize\_unused() +" KB",  
 indexNode.getSize\_unused() +" KB"  
 );  
 System.*out*.println();  
 }

###### 结果描述：

显示X分区的空间统计情况，包括磁盘分区名称、总空间大小，已用空间大小和可用空间大小

### 3、任务1——文件目录结构管理

#### 方案描述：

采用树形目录结构管理

#### 设计思路：

文件目录结构管理包括展示目录、创建目录、切换目录、删除目录、移动目录、路径解析、更新大小、展示绝对路径、树形展示目录操作。

#### 关键代码解析：

文件目录结构管理包括展示目录(dir)、创建目录(mkDir)、切换目录(chgDir)、删除目录(delete)、移动目录（move）、路径解析（pathResolve）、更新大小（updateSize）、展示绝对路径（pwd）、树形展示目录（treeDir）。

public interface DirService {  
 */\*\*  
 \* 显示当前目录下的文件名称 修改日期 类型 大小  
 \*  
 \** ***@param*** *dirPath 目录路径  
 \*/* void dir(String dirPath);  
  
 */\*\*  
 \* 创建目录  
 \*  
 \** ***@param*** *dirPath 目录路径  
 \** ***@return*** *{****@link*** *Boolean}  
 \*/* Boolean mkDir(String dirPath);  
  
 */\*\*  
 \* 切换目录  
 \* cd .. 可切换上一级  
 \*  
 \** ***@param*** *dirPath 目录路径  
 \** ***@return*** *{****@link*** *Boolean}  
 \*/* Boolean chgDir(String dirPath);  
  
 */\*\*  
 \* 删除目录  
 \*  
 \** ***@param*** *dirPath 目录路径  
 \** ***@return*** *{****@link*** *Boolean}  
 \*/* Boolean delete(String dirPath);  
  
 */\*\*  
 \* 移动目录  
 \*  
 \** ***@param*** *dirPath 文件路径  
 \** ***@param*** *newDirPath 新文件路径  
 \** ***@return*** *{****@link*** *Boolean}  
 \*/* Boolean move(String dirPath, String newDirPath);  
  
 */\*\*  
 \* 路径解析 查看是否存在该文件或目录  
 \*  
 \** ***@param*** *path 路径  
 \** ***@return*** *{****@link*** *FCB}  
 \*/* FCB pathResolve(String path);  
  
 */\*\*  
 \* 递归修改父目录大小  
 \*  
 \** ***@param*** *fcb FCB  
 \** ***@param*** *isAdd 添加文件 add  
 \*/* void updateSize(FCB fcb,Boolean isAdd,int new\_add);  
  
 */\*\*  
 \* 显示当前全目录路径  
 \* /ljl/a  
 \*  
 \** ***@param*** *fcb 指定目录  
 \** ***@return*** *{****@link*** *String} 全路径  
 \*/* String pwd(FCB fcb);  
  
 */\*\*  
 \* 显示当前目录  
 \* /a  
 \*/* void showPath();  
  
 */\*\*  
 \* 树状显示目录  
 \*  
 \** ***@param*** *dirPath 目录路径  
 \*/* void treeDir(String dirPath,int level);  
}

#### **MkDir**命令说明：

###### 设计流程：

用户输入 [ 命令 + 新目录路径 ]

文件系统会检测输入的新目录路径的合法性，

如果合法则创建新目录。

###### 关键代码解析：

检查命令合法性：

数组长度=1则输出命令提示

否则正常执行

case "MkDir":/\*4\*/  
 if(inputs.length == 1){  
 System.*out*.println("MkDir [DirPath]");  
 break;  
 }  
 *dirService*.mkDir(inputs[1]);  
 break;

检查路径合法性：

先判断新目录路径是否存在，如果存在则重命名（在目录名后+“-1”），否则继续。

再判断新目录路径父目录是否存在，如果不存在则报错，否则继续。

做后检查目录名是否为空，如是则报错，否则创建新目录。

@Override  
 public Boolean mkDir(String dirPath) {;  
  
 FCB temp\_fcb = *dirService*.pathResolve(dirPath);  
 String[] splitDir = dirPath.split("/");  
 String dirName = splitDir[splitDir.length - 1];  
 if(Objects.*nonNull*(temp\_fcb)){  
 dirName+="-1";  
 System.*out*.println("[error] 文件名重复 已重命名为\""+dirName+"\"");  
 }  
  
 */\*\*begin: 寻找新目录的父目录\*\*/* dirPath = dirPath.trim();  
 int index\_last\_slash = dirPath.lastIndexOf("/");  
 if(index\_last\_slash!=-1){  
 dirPath = dirPath.substring(0, index\_last\_slash);  
 if(dirPath.equals(".")||dirPath.equals("..")){ // ./D->.  
 dirPath+="/";  
 }  
 }  
 else{// D->./  
 dirPath="./";  
 }  
  
// System.out.println("fatherDirPath is : "+dirPath);  
  
 FCB curDir = Memory.*getInstance*().getCurDir();  
 FCB rootDir = Memory.*getInstance*().getRootDir();  
 FCB desDir = null;  
  
 if(dirPath.startsWith("./")){//1  
 dirPath = dirPath.substring(2);  
 if(dirPath.equals("")){  
 desDir = curDir;  
 }  
 else{  
 splitDir = dirPath.split("/");  
 FCB temp = curDir;  
 for (int i = 0; i < splitDir.length - 1; i++) {  
 //找到目标文件所在目录  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[i])){  
 temp = child;  
 continue;  
 }  
 }  
 }  
 //在该目录下找  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[splitDir.length - 1])){  
 desDir=child;  
 break;  
 }  
 }  
 }  
 }  
  
 else if(dirPath.startsWith("../")){//2  
  
 FCB temp;  
 if(curDir != rootDir){  
 temp = curDir.getFather();  
 }  
 else{  
 System.*out*.println("[error] 目标目录父目录不存在");  
 return false;  
 }  
 dirPath = dirPath.substring(3);  
 while (dirPath.startsWith("../")){  
 if(temp != rootDir){  
 temp = temp.getFather();  
 }  
 else{  
 System.*out*.println("[error] 目标目录父目录不存在");  
 return false;  
 }  
 dirPath = dirPath.substring(3);  
 }  
  
 if(dirPath.equals("")){  
 desDir=temp;  
 }  
 else {  
 splitDir = dirPath.split("/");  
 for (int i = 0; i < splitDir.length - 1; i++) {  
 //找到目标文件所在目录  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[i])){  
 temp = child;  
 continue;  
 }  
 }  
 }  
 //在该目录下找  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[splitDir.length - 1])){  
 desDir=child;  
 break;  
 }  
 }  
 }  
 }  
  
 else if((dirPath.startsWith("/"))){//3  
 //以/开头 从根目录逐层往下找  
 dirPath = dirPath.substring(1);  
 if(dirPath.equals("")){  
 desDir=rootDir;  
 }  
 else{  
 splitDir = dirPath.split("/");  
 FCB temp = rootDir;  
 for (int i = 0; i < splitDir.length - 1; i++) {  
 //找到目标文件所在目录  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[i])){  
 temp = child;  
 continue;  
 }  
 }  
 }  
 //在该目录下找  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[splitDir.length - 1])){  
 desDir=child;  
 break;  
 }  
 }  
 }  
 }  
  
 else if(dirPath.equals("..")){//4  
 //判断是不是已经在根目录  
 if(curDir != Memory.*getInstance*().getRootDir()){  
 //改变当前目录为父目录  
 desDir=curDir.getFather();  
 }  
 else{  
 System.*out*.println("[error] 目标目录父目录不存在");  
 return false;  
 }  
  
 }  
  
 else {//5->1  
 //在当前目录下找  
 splitDir = dirPath.split("/");  
 FCB temp = curDir;  
 for (int i = 0; i < splitDir.length - 1; i++) {  
 //找到目标文件所在目录  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[i])){  
// System.out.println(splitDir[i]);  
 temp = child;  
 continue;  
 }  
 }  
 }  
 //在该目录下找  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[splitDir.length - 1])){  
// System.out.println(splitDir[splitDir.length - 1]);  
 desDir=child;  
 break;  
 }  
 }  
 }  
  
 if(desDir==null){  
 System.*out*.println("[error] 目标目录父目录不存在");  
 return false;  
 }  
  
 */\*\*end: 寻找新目录的父目录\*\*/* List<FCB> children = desDir.getChildren();  
 //判空  
 if(Objects.*isNull*(dirName)){  
 System.*out*.println("[error] 目录名不可为空");  
 return false;  
 }  
// //判断重复  
// dirName = dirName.trim(); //去除首尾空格  
// for (FCB child : children) {  
// if(child.getFileName().equals(dirName)){  
// dirName+="-1";  
// System.out.println("[error] 目录名重复 已重命名为\""+dirName+"\"");  
// break;  
// }  
// }  
 //创建索引节点 创建FCB 文件大小为0 空文件  
 IndexNode indexNode = new IndexNode(0, -1, 0, new Date());  
 FCB fcb = new FCB(dirName, "DIR", indexNode, desDir, new LinkedList<>());  
 //将文件控制块放入磁盘的fcb集合  
 Disk.*getINSTANCE*().getFcbList().add(fcb);  
 //修改父目录的文件项 加入父目录儿子集合  
 desDir.getIndexNode().addFcbNum();  
 desDir.getChildren().add(fcb);  
 System.*out*.println("[success] 创建目录成功！");  
 return true;  
 }

###### 结果描述：

创建指定目录xx

#### **DelDir** 命令说明：

###### 设计流程：

用户输入 [ 命令 + 目录路径 ]

文件系统会检测输入的目录路径的合法性，

如果合法则删除目录。

###### 关键代码解析：

检查命令合法性：

数组长度=1则输出命令提示

否则正常执行

case "DelDir":/\*5\*/  
 if(inputs.length == 1){  
 System.*out*.println("DelDir [DirPath]");  
 break;  
 }  
 *dirService*.delete(inputs[1]);  
 break;

检查路径合法性：

先判断目录路径是否存在，不存在则报错，否则继续。

然后判断目录名是否为根目录，如是则报错，否则继续。

然后判断是否为磁盘分区，如是则报错，否则继续。

然后判断是否为文件，如是则报错，否则继续。

然后判断是否非空，如是则报错，否则继续。

然后判断是否为当前目录，如是则报错，否则继续。

最后从FCB集合中去除 修改父目录文件项 修改父目录儿子结点 递归修改父目录文件大小

@Override  
 public Boolean delete(String dirPath) {  
 //判断是否存在  
 FCB fcb = *dirService*.pathResolve(dirPath);  
 if(Objects.*isNull*(fcb)){  
 System.*out*.println("[error] 目标目录不存在");  
 return false;  
 }  
 //根目录判断  
 else if(fcb == Memory.*getInstance*().getRootDir()){  
 System.*out*.println("[error] 无法删除根目录");  
 return false;  
 }  
 //磁盘分区判断  
 else if(fcb.getType().equals("DISK")){  
 //type DISK 磁盘  
 System.*out*.println("[error] 无法删除磁盘");  
 return false;  
 }//文件判断  
 else if(fcb.getType().equals("FILE")){  
 //type FILE 文件  
 System.*out*.println("[error] 无法删除文件");  
 return false;  
 }  
  
// //判断是否打开 打开要先关闭  
// //判断是否在openFileList中  
// String fill\_path = dirService.pwd(fcb);  
// List<OpenFile> openFileList = Memory.getInstance().getOpenFileList();  
// OpenFile toWriteFile = null;  
// for (OpenFile openFile : openFileList) {  
// if(openFile.getFilePath().equals(fill\_path)){  
// toWriteFile = openFile;  
// }  
// }  
// if(Objects.nonNull(toWriteFile)){  
// System.out.println("[error] 文件被打开 请先关闭");  
// return false;  
// }  
 //重复确认  
 String choice = null;  
 while (true){  
 System.*out*.println("确认删除该目录？（Y/N）");  
 choice = *scanner*.nextLine();  
 if(choice.equals("Y")) break;  
 if(choice.equals("N")) {  
 System.*out*.println("[success] 已取消删除！");  
 return false;  
 }  
 }  
  
 //非空目录判断  
 if(fcb.getIndexNode().getSize() != 0 || fcb.getIndexNode().getFcbNum() != 0){  
  
 if(fcb.getType().equals("DIR")){//非空文件夹  
 //type DIR 目录  
 //借助栈删除目录  
// diskService.freeDir(fcb);  
 System.*out*.println("[error] 文件夹非空 无法删除");  
 return false;  
 }  
// else {//非空文件  
// //清空磁盘  
// diskService.freeFile(fcb);  
// }  
 }  
 //如果是空目录 不允许是当前目录  
 if(fcb == Memory.*getInstance*().getCurDir()){  
 System.*out*.println("[error] 无法删除当前目录 请先退出当前目录！");  
 return false;  
 }  
 //从FCB集合中去除 修改父目录文件项 修改父目录儿子结点  
 Disk.*getINSTANCE*().getFcbList().remove(fcb);  
 fcb.getFather().getIndexNode().subFcbNum();  
 fcb.getFather().getChildren().remove(fcb);  
 //递归修改父目录文件大小  
 *dirService*.updateSize(fcb,false,-1);  
 System.*out*.println("[success] 删除成功");  
 return true;  
 }

###### 结果描述：

删除空目录xx

#### **(2)Dir** 命令说明：

###### 设计流程：

用户输入 [ 命令 + 目录路径 ]

文件系统会检测输入的目录路径的合法性，

如果合法则展示目录。

###### 关键代码解析：

检查命令合法性：

数组长度=1则展示当前目录

否则展示指定目录

case "Dir":/\*6\*/  
 if(inputs.length == 1){  
 *dirService*.dir("");  
 }  
 else {  
 *dirService*.dir(inputs[1]);  
 }  
 break;

检查路径合法性：

先判断目录路径是否存在，不存在则报错，否则继续。

然后判断是否为文件，如是则报错，否则继续。

最后显示xx目录下的文件和目录详情包括文件名称、修改时间、文件类型、文件大小

@Override  
public void dir(String path) {  
 Memory memory = Memory.*getInstance*();  
 FCB fcb = null;  
 if(path.trim().equals("")){  
 fcb = memory.getCurDir();  
 }  
 else{  
 fcb = *dirService*.pathResolve(path);  
 }  
 if(Objects.*nonNull*(fcb)){  
 if(fcb.getType().equals("DIR")||fcb.getType().equals("DISK")){  
 List<FCB> children = fcb.getChildren();  
 View view = new View();  
 System.*out*.println("名称\t\t\t\t修改日期\t\t\t\t类型\t\t\t\t大小");  
 for (int i = 0; i < children.size(); i++) {  
 FCB child = children.get(i);  
 view.showFcb(child);  
 }  
 }  
 else{  
 System.*out*.println("[error] 无法展示非目录或磁盘");  
 }  
 }  
 else {  
 System.*out*.println("[error] 目标目录不存在");  
 }  
}

public void showFcb(FCB fcb){  
 SimpleDateFormat dateFormat= new SimpleDateFormat("yyyy/MM/dd HH:mm");  
 IndexNode indexNode = fcb.getIndexNode();//name-time-type-size  
 System.*out*.printf("%-15s\t%-15s\t%-15s\t%-15s",  
// indexNode.getFcbNum(),  
 fcb.getFileName(),  
 dateFormat.format(indexNode.getUpdateTime()),  
 fcb.getType(),  
 indexNode.getSize() +" KB"  
 );  
 System.*out*.println();  
 }

###### 结果描述：

显示xx目录下的文件和目录详情包括文件名称、修改时间、文件类型、文件大小

#### **(3)ChgDir** 命令说明：

###### 设计流程：

用户输入 [ 命令 + 目录路径 ]

文件系统会检测输入的目录路径的合法性，

如果合法则切换目录。

###### 关键代码解析：

检查命令合法性：

数组长度=1则输出命令提示

否则正常执行

case "ChgDir":/\*7\*/  
 if(inputs.length == 1){  
 System.*out*.println("ChgDir [DirPath]");  
 break;  
 }  
 *dirService*.chgDir(inputs[1]);  
 break;

检查路径合法性：

先判断目录路径是否存在，不存在则报错，否则继续。

然后判断是否为文件，如是则报错，否则继续。

最后修改当前目录和当前磁盘分区（如有必要）。

@Override  
 public Boolean chgDir(String path) {  
 //解析路径  
 FCB desDir = *dirService*.pathResolve(path);  
 //null 不存在  
 if(Objects.*isNull*(desDir)){  
 System.*out*.println("[error] 目标目录不存在");  
 return false;  
 }else if(desDir.getType().equals("FILE")){  
 //type FILE 不是磁盘或目录文件  
 System.*out*.println("[error] 无法进入文件");  
 return false;  
 }else {  
 //type DISK DIR 切换到对应磁盘分区 目录  
 Memory.*getInstance*().setCurDir(desDir);  
 //更改磁盘分区  
 String[] splitDir = pwd(desDir).split("/");  
  
// System.out.println(splitDir.length);  
  
 if(splitDir.length>1){  
 FCB desDisk = *dirService*.pathResolve("/"+splitDir[1]);  
 if(Objects.*isNull*(desDisk)){  
 System.*out*.println("[error] 磁盘分区更换错误");  
 return false;  
 }  
 Memory.*getInstance*().setCurDisk(desDisk);  
 }else{  
 Memory.*getInstance*().setCurDisk(null);  
 }  
 }  
 return true;  
 }

###### 结果描述：

切换当前目录到xx下。

#### **(4)TreeDir** 命令说明：

###### 设计流程：

用户输入 [ 命令 + 目录路径 ]

文件系统会检测输入的目录路径的合法性，

如果合法则展示目录。

###### 关键代码解析：

检查命令合法性：

数组长度=1则展示当前目录

否则展示指定目录

case "TreeDir":/\*8\*/  
 if(inputs.length == 1){  
 *dirService*.treeDir("",0);  
 }  
 else {  
 *dirService*.treeDir(inputs[1],0);  
 }  
 break;

检查路径合法性：

先判断目录路径是否存在，不存在则报错，否则继续。

然后判断是否为文件，如是则报错，否则继续。

最后展示前目录。

@Override  
public void treeDir(String path,int level){  
 Memory memory = Memory.*getInstance*();  
 FCB fcb = null;  
 if(path.trim().equals("")){  
 fcb = memory.getCurDir();  
 }  
 else{  
 fcb = *dirService*.pathResolve(path);  
 }  
 if(Objects.*nonNull*(fcb)){  
 if(fcb.getType().equals("DIR")||fcb.getType().equals("DISK")){  
 treeDirImpl(fcb,level);  
 }  
 else{  
 System.*out*.println("[error] 无法展示非目录或磁盘");  
 }  
 }  
 else {  
 System.*out*.println("[error] 目标目录不存在");  
 }  
}

public void treeDirImpl(FCB fcb,int level) {  
 StringBuffer str = new StringBuffer("");  
 for (int i = 0; i < level; i++) {  
 str.append(" "); //每一个层次缩进四个空格  
 }  
 str.append("├─");//└─  
 if (fcb.getType().equals("FILE")) {  
 System.*out*.println(str + fcb.getFileName());  
 return;  
 }  
  
 System.*out*.println(str + fcb.getFileName());// 先打印目录出来  
 List<FCB> children = fcb.getChildren();// 取出该目录下所有文件以及文件夹  
 int i = 0;  
 // 开始遍历每一个目录项  
 for (i = 0; i < children.size(); i++) {  
 FCB child = children.get(i);  
 treeDirImpl(child, level + 1);  
 }  
  
}

###### 结果描述：

树状显示当前分区或目录下的xx目录的文件和目录结构。

#### **(5)MoveDir** 命令说明：

###### 设计流程：

用户输入 [ 命令 + 目录路径 ]

文件系统会检测输入的目录路径的合法性，

如果合法则切换目录。

###### 关键代码解析：

检查命令合法性：

数组长度<3则输出命令提示

否则正常执行

case "MoveDir":/\*9\*/  
if(inputs.length < 3){  
 System.*out*.println("MoveDir [DirPath] [NewDirPath]");  
 break;  
 }  
 *dirService*.move(inputs[1],inputs[2]);  
 break;

检查源目录路径合法性：

先判断目录路径是否存在，不存在则报错，否则继续。

然后判断是否为根目录，如是则报错，否则继续。

然后判断是否为磁盘，如是则报错，否则继续。

然后判断是否为文件，如是则报错，否则继续。

然后判断是否为当前目录，如是则报错，否则继续。

检查目标目录路径合法性：

先判断目录路径是否存在，不存在则报错，否则继续。

然后判断目标目录中是否有存在与源目录重名情况，存在则重命名+“-1”，否则继续

然后从原父目录中除名

然后递归修改原父目录文件大小

然后源目录修改父目录为目标目录

然后修改目标目录的文件项 把源目录加入目标目录儿子集合

然后递归修改目标目录目录文件大小

最后判断是否存在跨磁盘分区情况，存在则需要以新的方式存入另一磁盘分区并释放原来的内存。

@Override  
public Boolean move(String dirPath, String newDirPath){  
 //判断是否存在  
 FCB srcDir = *dirService*.pathResolve(dirPath);  
 if(Objects.*isNull*(srcDir)){  
 System.*out*.println("[error] 源文件不存在");  
 return false;  
 }  
 //根目录判断  
 else if(srcDir == Memory.*getInstance*().getRootDir()){  
 System.*out*.println("[error] 无法移动根目录");  
 return false;  
 }  
 //磁盘分区判断  
 else if(srcDir.getType().equals("DISK")){  
 //type DISK 磁盘  
 System.*out*.println("[error] 无法移动磁盘");  
 return false;  
 }

//文件判断  
else if(srcDir.getType().equals("FILE")){  
 //type FILE 文件  
 System.*out*.println("[error] 无法移动文件");  
 return false;  
}

//不允许是当前目录  
 else if(srcDir == Memory.*getInstance*().getCurDir()){  
 System.*out*.println("[error] 无法删除当前目录 请先退出当前目录！");  
 return false;  
 }  
  
 */\*\*begin: 寻找目标目录\*\*/* newDirPath = newDirPath.trim();  
 FCB curDir = Memory.*getInstance*().getCurDir();  
 FCB rootDir = Memory.*getInstance*().getRootDir();  
 FCB desDir = null;  
  
 if(newDirPath.startsWith("./")){//1  
 newDirPath = newDirPath.substring(2);  
 if(newDirPath.equals("")){  
 desDir = curDir;  
 }  
 else{  
 String[] splitDir = newDirPath.split("/");  
 FCB temp = curDir;  
 for (int i = 0; i < splitDir.length - 1; i++) {  
 //找到目标文件所在目录  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[i])){  
 temp = child;  
 continue;  
 }  
 }  
 }  
 //在该目录下找  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[splitDir.length - 1])){  
 desDir=child;  
 break;  
 }  
 }  
 }  
 }  
  
 else if(newDirPath.startsWith("../")){//2  
  
 FCB temp;  
 if(curDir != rootDir){  
 temp = curDir.getFather();  
 }  
 else{  
 System.*out*.println("[error] 目标目录不存在");  
 return false;  
 }  
 newDirPath = newDirPath.substring(3);  
 while (newDirPath.startsWith("../")){  
 if(temp != rootDir){  
 temp = temp.getFather();  
 }  
 else{  
 System.*out*.println("[error] 目标目录不存在");  
 return false;  
 }  
 newDirPath = newDirPath.substring(3);  
 }  
  
 if(newDirPath.equals("")){  
 desDir=temp;  
 }  
 else {  
 String[] splitDir = newDirPath.split("/");  
 for (int i = 0; i < splitDir.length - 1; i++) {  
 //找到目标文件所在目录  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[i])){  
 temp = child;  
 continue;  
 }  
 }  
 }  
 //在该目录下找  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[splitDir.length - 1])){  
 desDir=child;  
 break;  
 }  
 }  
 }  
 }  
  
 else if((newDirPath.startsWith("/"))){//3  
 //以/开头 从根目录逐层往下找  
 newDirPath = newDirPath.substring(1);  
 if(newDirPath.equals("")){  
 desDir=rootDir;  
 }  
 else{  
 String[] splitDir = newDirPath.split("/");  
 FCB temp = rootDir;  
 for (int i = 0; i < splitDir.length - 1; i++) {  
 //找到目标文件所在目录  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[i])){  
 temp = child;  
 continue;  
 }  
 }  
 }  
 //在该目录下找  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[splitDir.length - 1])){  
 desDir=child;  
 break;  
 }  
 }  
 }  
 }  
  
 else if(newDirPath.equals("..")){//4  
 //判断是不是已经在根目录  
 if(curDir != Memory.*getInstance*().getRootDir()){  
 //改变当前目录为父目录  
 desDir=curDir.getFather();  
 }  
 else{  
 System.*out*.println("[error] 目标目录不存在");  
 return false;  
 }  
  
 }  
  
 else {//5=1  
 //在当前目录下找  
 String[] splitDir = newDirPath.split("/");  
 FCB temp = curDir;  
 for (int i = 0; i < splitDir.length - 1; i++) {  
 //找到目标文件所在目录  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[i])){  
 temp = child;  
 continue;  
 }  
 }  
 }  
 //在该目录下找  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[splitDir.length - 1])){  
 desDir=child;  
 break;  
 }  
 }  
 }  
  
 if(desDir==null){  
 System.*out*.println("[error] 目标目录不存在");  
 return false;  
 }

*/\*\*end: 寻找目标目录\*\*/  
  
  
  
 /\*\*begin: 把文件转移到目标目录下\*\*/* String temp\_1=pwd(srcDir);  
 String temp\_2=pwd(desDir);  
 String[] splitDir\_1 = temp\_1.split("/");  
 String[] splitDir\_2 = temp\_2.split("/");  
  
 //判断重复  
 List<FCB> children = desDir.getChildren();  
 String dirName=srcDir.getFileName();  
 for (FCB child : children) {  
 if(child.getFileName().equals(dirName)){  
 dirName+="-1";  
 System.*out*.println("[error] 目录名重复 已重命名为\""+dirName+"\"");  
 break;  
 }  
 }  
  
 //从原父目录中除名  
 srcDir.getFather().getIndexNode().subFcbNum();  
 srcDir.getFather().getChildren().remove(srcDir);  
 //递归修改原父目录文件大小  
 *dirService*.updateSize(srcDir,false,-1);  
  
 //子目录修改父目录  
 srcDir.setFileName(dirName);  
 srcDir.setFather(desDir);  
 srcDir.getIndexNode().setUpdateTime(new Date());  
  
 //修改目标目录的文件项 加入目标目录儿子集合  
 desDir.getIndexNode().addFcbNum();  
 desDir.getChildren().add(srcDir);  
 //递归修改目标目录目录文件大小  
 *dirService*.updateSize(srcDir,true,-1);  
  
 */\*\*end: 把文件转移到目标目录下\*\*/  
  
 /\*\*begin:(跨盘)转移文件地址\*\*/* if(!splitDir\_1[1].equals(splitDir\_2[1])){//目标目录跨盘  
 //空文件判断  
 if(srcDir.getIndexNode().getSize() != 0 || srcDir.getIndexNode().getFcbNum() != 0){  
 if(srcDir.getType().equals("DIR")){//非空文件夹  
 //type DIR 目录  
 *diskService*.moveDir(srcDir, splitDir\_1[1], splitDir\_2[1]);//转移内部文件  
 }  
// else {//非空文件  
// //清空磁盘  
// diskService.freeFile(fcb);  
// }  
 }  
 }  
  
 */\*\*end:(跨盘)转移文件地址\*\*/* System.*out*.println("[success] 转移目录成功！");  
  
 return true;  
 }

###### 结果描述：

把指定路径下的目录xx删除，同时复制到yy目录下。

### 4、任务1——文件外存分配管理

#### 方案描述：

文件物理结构采用显式连接（FAT表）

#### 设计思路：

文件外存分配管理包括创建文件，打开文件，读取文件，写入文件，关闭文件，删除文件，拷贝文件，导出文件

#### 关键代码解析：

文件外存分配管理包括创建文件（create），打开文件（open），读取文件（read），写入文件（write），关闭文件（close），删除文件（delete），拷贝文件（copy），导出文件（export）

public interface FileService {  
  
 */\*\*  
 \* 创建文件  
 \*  
 \** ***@param*** *filePath 文件路径  
 \** ***@return*** *{****@link*** *Boolean}  
 \*/* Boolean create(String filePath);  
  
 */\*\*  
 \* 打开文件  
 \*  
 \** ***@param*** *filePath 文件路径  
 \** ***@return*** *{****@link*** *Boolean}  
 \*/* Boolean open(String filePath);  
  
 */\*\*  
 \* 读取文件  
 \*  
 \** ***@param*** *filePath 文件路径  
 \** ***@return*** *{****@link*** *Boolean}  
 \*/* Boolean read(String filePath);  
  
 */\*\*  
 \* 写入文件  
 \*  
 \** ***@param*** *filePath 文件路径  
 \** ***@return*** *{****@link*** *Boolean}  
 \*/* Boolean write(String filePath);  
  
 */\*\*  
 \* 关闭文件  
 \*  
 \** ***@param*** *filePath 文件路径  
 \** ***@return*** *{****@link*** *Boolean}  
 \*/* Boolean close(String filePath);  
  
 */\*\*  
 \* 删除文件  
 \*  
 \** ***@param*** *filePath 文件路径  
 \** ***@return*** *{****@link*** *Boolean}  
 \*/* Boolean delete(String filePath);  
  
 */\*\*  
 \* 复制  
 \*  
 \** ***@param*** *filePath 文件路径  
 \** ***@param*** *newDirPath 新目录路径  
 \** ***@return*** *{****@link*** *Boolean}  
 \*/* Boolean copy(String filePath,String newDirPath);  
  
 */\*\*  
 \* 导出  
 \*  
 \** ***@param*** *filePath 文件路径  
 \** ***@param*** *exDirPath 外目录路径  
 \** ***@return*** *{****@link*** *Boolean}  
 \*/* Boolean export(String filePath,String exDirPath) throws IOException;  
}

#### **Create** 命令说明：

###### 设计流程：

用户输入 [ 命令 + 文件路径 ]

文件系统会检测输入的文件路径的合法性，

如果合法则创建文件。

###### 关键代码解析：

检查命令合法性：

数组长度=1则输出命令提示

否则正常执行

case "Create":/\*10\*/  
 if(inputs.length == 1){  
 System.*out*.println("Create [FilePath]");  
 break;  
 }  
 *fileService*.create(inputs[1]);  
 break;

检查路径合法性：

先判断目录路径是否存在，存在则重命名+“-1”，否则继续。

然后判断目标目录父目录是否存在，如不存在则报错，否则继续。

然后判断文件名是否为空，如是则报错，否则继续。

最后创建新文件。

@Override  
 public Boolean create(String filePath) {  
  
  
 FCB temp\_fcb = *dirService*.pathResolve(filePath);  
 String[] splitDir = filePath.split("/");  
 String fileName = splitDir[splitDir.length - 1];  
 if(Objects.*nonNull*(temp\_fcb)){  
 fileName+="-1";  
 System.*out*.println("[error] 文件名重复 已重命名为\""+fileName+"\"");  
 }  
  
 */\*\*begin: 寻找新文件的父目录\*\*/* filePath = filePath.trim();  
 int index\_last\_slash = filePath.lastIndexOf("/");  
 if(index\_last\_slash!=-1){  
 filePath = filePath.substring(0, index\_last\_slash);  
 if(filePath.equals(".")||filePath.equals("..")){ // ./F->.  
 filePath+="/";  
 }  
 }  
 else{// F->./  
 filePath="./";  
 }  
  
// System.out.println("fatherDirPath is : "+filePath);  
  
 FCB curDir = Memory.*getInstance*().getCurDir();  
 FCB rootDir = Memory.*getInstance*().getRootDir();  
 FCB desDir = null;  
  
 if(filePath.startsWith("./")){//1  
 filePath = filePath.substring(2);  
 if(filePath.equals("")){  
 desDir = curDir;  
 }  
 else{  
 splitDir = filePath.split("/");  
 FCB temp = curDir;  
 for (int i = 0; i < splitDir.length - 1; i++) {  
 //找到目标文件所在目录  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[i])){  
 temp = child;  
 continue;  
 }  
 }  
 }  
 //在该目录下找  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[splitDir.length - 1])){  
 desDir=child;  
 break;  
 }  
 }  
 }  
 }  
  
 else if(filePath.startsWith("../")){//2  
  
 FCB temp;  
 if(curDir != rootDir){  
 temp = curDir.getFather();  
 }  
 else{  
 System.*out*.println("[error] 目标目录父目录不存在");  
 return false;  
 }  
 filePath = filePath.substring(3);  
 while (filePath.startsWith("../")){  
 if(temp != rootDir){  
 temp = temp.getFather();  
 }  
 else{  
 System.*out*.println("[error] 目标目录父目录不存在");  
 return false;  
 }  
 filePath = filePath.substring(3);  
 }  
  
 if(filePath.equals("")){  
 desDir=temp;  
 }  
 else {  
 splitDir = filePath.split("/");  
 for (int i = 0; i < splitDir.length - 1; i++) {  
 //找到目标文件所在目录  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[i])){  
 temp = child;  
 continue;  
 }  
 }  
 }  
 //在该目录下找  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[splitDir.length - 1])){  
 desDir=child;  
 break;  
 }  
 }  
 }  
 }  
  
 else if((filePath.startsWith("/"))){//3  
 //以/开头 从根目录逐层往下找  
 filePath = filePath.substring(1);  
 if(filePath.equals("")){  
 desDir=rootDir;  
 }  
 else{  
 splitDir = filePath.split("/");  
 FCB temp = rootDir;  
 for (int i = 0; i < splitDir.length - 1; i++) {  
 //找到目标文件所在目录  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[i])){  
 temp = child;  
 continue;  
 }  
 }  
 }  
 //在该目录下找  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[splitDir.length - 1])){  
 desDir=child;  
 break;  
 }  
 }  
 }  
 }  
  
 else if(filePath.equals("..")){//4  
 //判断是不是已经在根目录  
 if(curDir != Memory.*getInstance*().getRootDir()){  
 //改变当前目录为父目录  
 desDir=curDir.getFather();  
 }  
 else{  
 System.*out*.println("[error] 目标目录父目录不存在");  
 return false;  
 }  
  
 }  
  
 else {//5->1  
 //在当前目录下找  
 splitDir = filePath.split("/");  
 FCB temp = curDir;  
 for (int i = 0; i < splitDir.length - 1; i++) {  
 //找到目标文件所在目录  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[i])){  
 temp = child;  
 continue;  
 }  
 }  
 }  
 //在该目录下找  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[splitDir.length - 1])){  
 desDir=child;  
 break;  
 }  
 }  
 }  
  
 if(desDir==null){  
 System.*out*.println("[error] 目标目录父目录不存在");  
 return false;  
 }  
  
 */\*\*end: 寻找新文件的父目录\*\*/* List<FCB> children = desDir.getChildren();  
  
 //判空  
 if(Objects.*isNull*(fileName)) {  
 System.*out*.println("[error] 文件名不可为空");  
 return false;  
 }  
// //判断重复  
// fileName = fileName.trim(); //去除首尾空格  
// for (FCB child : children) {  
// if(child.getFileName().equals(fileName)){  
// fileName+="-1";  
// System.out.println("[warning] 文件名重复 已重命名为\""+fileName+"\"");  
// break;  
// }  
// }  
 //创建索引节点 创建FCB 文件大小为0 空文件  
 IndexNode indexNode = new IndexNode(0, -1, 0, new Date());  
 FCB fcb = new FCB(fileName, "FILE", indexNode, desDir, null);  
 //将文件控制块放入磁盘的fcb集合  
 Disk.*getINSTANCE*().getFcbList().add(fcb);  
 //修改父目录的文件项 加入父目录儿子集合  
 desDir.getIndexNode().addFcbNum();  
 desDir.getChildren().add(fcb);  
 System.*out*.println("[success] 创建文件成功");  
 return true;  
 }

###### 结果描述：

创建文件filexx。

#### **Copy** 命令说明：

###### 设计流程：

用户输入 [ 命令 + 文件路径 ]

文件系统会检测输入的文件路径的合法性，

如果合法则拷贝文件。

###### 关键代码解析：

检查命令合法性：

数组长度<3 则输出命令提示

否则正常执行

case "Copy":/\*11\*/  
 if(inputs.length < 3){  
 System.*out*.println("Copy [FilePath] [NewDirPath]");  
 break;  
 }  
 else if(inputs.length == 2){  
 *fileService*.copy(inputs[1],inputs[1]);  
 break;  
 }  
 *fileService*.copy(inputs[1],inputs[2]);  
 break;

检查源文件路径合法性：

先判断路径是否存在，不存在则报错，否则继续。

然后判断是否非文件，如是则报错，否则继续。

检查目标目录路径合法性：

先判断目录路径是否存在，不存在则报错，否则继续。

然后判断目标目录中是否有存在与源目录重名情况，存在则重命名+“-1”，否则继续

然后创建新文件，读出源文件内容，写入新文件

然后将文件控制块放入磁盘的fcb集合

然后修改父目录的文件项 加入父目录儿子集合

@Override  
 public Boolean copy(String filePath, String newDirPath){  
 //判断是否存在  
 FCB fcb = *dirService*.pathResolve(filePath);  
 if(Objects.*isNull*(fcb)){  
 System.*out*.println("[error] 源文件不存在");  
 return false;  
 }  
 else if(!fcb.getType().equals("FILE")){  
 System.*out*.println("[error] 不能拷贝非文件");  
 return false;  
 }  
  
 */\*\*begin: 寻找目标目录\*\*/* newDirPath = newDirPath.trim();  
 FCB curDir = Memory.*getInstance*().getCurDir();  
 FCB rootDir = Memory.*getInstance*().getRootDir();  
 FCB desDir = null;  
  
 if(newDirPath.startsWith("./")){//1  
 newDirPath = newDirPath.substring(2);  
 if(newDirPath.equals("")){  
 desDir = curDir;  
 }  
 else{  
 String[] splitDir = newDirPath.split("/");  
 FCB temp = curDir;  
 for (int i = 0; i < splitDir.length - 1; i++) {  
 //找到目标文件所在目录  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[i])){  
 temp = child;  
 continue;  
 }  
 }  
 }  
 //在该目录下找  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[splitDir.length - 1])){  
 desDir=child;  
 break;  
 }  
 }  
 }  
 }  
  
 else if(newDirPath.startsWith("../")){//2  
  
 FCB temp;  
 if(curDir != rootDir){  
 temp = curDir.getFather();  
 }  
 else{  
 System.*out*.println("[error] 目标目录不存在");  
 return false;  
 }  
 newDirPath = newDirPath.substring(3);  
 while (newDirPath.startsWith("../")){  
 if(temp != rootDir){  
 temp = temp.getFather();  
 }  
 else{  
 System.*out*.println("[error] 目标目录不存在");  
 return false;  
 }  
 newDirPath = newDirPath.substring(3);  
 }  
  
 if(newDirPath.equals("")){  
 desDir=temp;  
 }  
 else {  
 String[] splitDir = newDirPath.split("/");  
 for (int i = 0; i < splitDir.length - 1; i++) {  
 //找到目标文件所在目录  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[i])){  
 temp = child;  
 continue;  
 }  
 }  
 }  
 //在该目录下找  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[splitDir.length - 1])){  
 desDir=child;  
 break;  
 }  
 }  
 }  
 }  
  
 else if((newDirPath.startsWith("/"))){//3  
 //以/开头 从根目录逐层往下找  
 newDirPath = newDirPath.substring(1);  
 if(newDirPath.equals("")){  
 desDir=rootDir;  
 }  
 else{  
 String[] splitDir = newDirPath.split("/");  
 FCB temp = rootDir;  
 for (int i = 0; i < splitDir.length - 1; i++) {  
 //找到目标文件所在目录  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[i])){  
 temp = child;  
 continue;  
 }  
 }  
 }  
 //在该目录下找  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[splitDir.length - 1])){  
 desDir=child;  
 break;  
 }  
 }  
 }  
 }  
  
 else if(newDirPath.equals("..")){//4  
 //判断是不是已经在根目录  
 if(curDir != Memory.*getInstance*().getRootDir()){  
 //改变当前目录为父目录  
 desDir=curDir.getFather();  
 }  
 else{  
 System.*out*.println("[error] 目标目录不存在");  
 return false;  
 }  
  
 }  
  
 else {//5=1  
 //在当前目录下找  
 String[] splitDir = newDirPath.split("/");  
 FCB temp = curDir;  
 for (int i = 0; i < splitDir.length - 1; i++) {  
 //找到目标文件所在目录  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[i])){  
 temp = child;  
 continue;  
 }  
 }  
 }  
 //在该目录下找  
 for (FCB child : temp.getChildren()) {  
 if(child.getFileName().equals(splitDir[splitDir.length - 1])){  
 desDir=child;  
 break;  
 }  
 }  
 }  
  
 if(desDir==null){  
 System.*out*.println("[error] 目标目录不存在");  
 return false;  
 }  
  
 */\*\*end: 寻找目标目录\*\*/  
  
  
 /\*\*begin: 在目标目录下创建复制文件\*\*/* //判断重复  
 List<FCB> children = desDir.getChildren();  
 String fileName=fcb.getFileName();  
 for (FCB child : children) {  
 if(child.getFileName().equals(fileName)){  
 fileName+="-1";  
 System.*out*.println("[error] 文件名重复 已重命名为\""+fileName+"\"");  
 break;  
 }  
 }  
 //创建索引节点 创建FCB 文件大小为0 空文件  
 IndexNode indexNode = new IndexNode(0, -1, 0, new Date());  
 FCB fcb\_new = new FCB(fileName, "FILE", indexNode, desDir, null);  
  
 /\*begin: read from old file and write into new file\*/  
  
 StringBuilder content = new StringBuilder();  
  
 //read  
 int[] fat = Memory.*getInstance*().getFat();  
 Block[] disk = Disk.*getINSTANCE*().getBlocks();  
 //从磁盘读取  
 if(fcb.getIndexNode().getSize() == 0){  
 //空文件  
 }  
 else{  
// int temp = fat[fcb.getIndexNode().getFirst\_block()];  
 int temp = fcb.getIndexNode().getFirst\_block();  
// while (fat[temp] != -1){  
 while (temp != -1){  
 //遍历读出  
 content.append(disk[temp].getContent());  
 temp = fat[temp];  
 }  
// content.append(disk[temp].getContent());  
 }  
 System.*out*.println("[success] 读出成功");  
  
  
 String temp\_1=*dirService*.pwd(fcb\_new);  
 String[] splitDir\_1 = temp\_1.split("/");  
 String diskName=splitDir\_1[1];  
  
  
 //write  
 int size = content.toString().toCharArray().length;//文件大小为输入字数  
 int first = *diskService*.writeToDisk(content.toString(),diskName);  
 //将文件指向第一块  
 fcb\_new.getIndexNode().setFirst\_block(first);  
 //修改索引结点大小  
 fcb\_new.getIndexNode().setSize(size);  
 //修改父目录项 以及一直递归修改父目录的大小  
 *dirService*.updateSize(fcb\_new,true,-1);  
 System.*out*.println("[success] 写入成功");  
  
 /\*end: read from old file and write into new file\*/  
  
 //将文件控制块放入磁盘的fcb集合  
 Disk.*getINSTANCE*().getFcbList().add(fcb\_new);  
 //修改父目录的文件项 加入父目录儿子集合  
 desDir.getIndexNode().addFcbNum();  
 desDir.getChildren().add(fcb\_new);  
 System.*out*.println("[success] 复制文件成功");  
 */\*\*end: 在目标目录下创建复制文件\*\*/* return true;  
 }

###### 结果描述：

复制文件filexx 成一个新文件fileyy。

#### **Delete** 命令说明：

###### 设计流程：

用户输入 [ 命令 + 文件路径 ]

文件系统会检测输入的文件路径的合法性，

如果合法则删除文件。

###### 关键代码解析：

检查命令合法性：

数组长度=1 则输出命令提示

否则正常执行

case "Delete":/\*12\*/  
 if(inputs.length == 1){  
 System.*out*.println("Delete [FilePath]");  
 break;  
 }  
 *fileService*.delete(inputs[1]);  
 break;

检查文件路径合法性：

先判断路径是否存在，不存在则报错，否则继续。

然后判断是否非文件，如是则报错，否则继续。

然后从FCB集合中去除 修改父目录文件项 修改父目录儿子结点

然后递归修改父目录文件大小

@Override  
 public Boolean delete(String filePath) {  
 //判断是否存在  
 FCB fcb = *dirService*.pathResolve(filePath);  
 if(Objects.*isNull*(fcb)){  
 System.*out*.println("[error] 目标文件不存在");  
 return false;  
 }  
 else if(!fcb.getType().equals("FILE")){  
 System.*out*.println("[error] 不能删除非文件");  
 return false;  
 }  
 //判断是否打开 打开要先关闭  
 //判断是否在openFileList中  
 String fill\_path = *dirService*.pwd(fcb);  
 List<OpenFile> openFileList = Memory.*getInstance*().getOpenFileList();  
 OpenFile toWriteFile = null;  
 for (OpenFile openFile : openFileList) {  
 if(openFile.getFilePath().equals(fill\_path)){  
 toWriteFile = openFile;  
 }  
 }  
 if(Objects.*nonNull*(toWriteFile)){  
 System.*out*.println("[error] 文件被打开 请先关闭");  
 return false;  
 }  
 //重复确认  
 String choice = null;  
 while (true){  
 System.*out*.println("确认删除该文件（Y/N）");  
 choice = *scanner*.nextLine();  
 if(choice.equals("Y")) break;  
 if(choice.equals("N")) {  
 System.*out*.println("[success] 已取消删除");  
 return false;  
 }  
 }  
 //非空文件判断  
 if(fcb.getIndexNode().getSize() != 0 || fcb.getIndexNode().getFcbNum() != 0){  
// if(fcb.getType().equals("DIR")){//非空文件夹  
// //type DIR 目录  
// //借助栈删除目录  
//// diskService.freeDir(fcb);  
// System.out.println("[error] 文件夹非空 无法删除");  
// return false;  
// }else {  
// //清空磁盘  
// diskService.freeFile(fcb);  
// }  
  
 String temp\_1=*dirService*.pwd(fcb);  
 String[] splitDir\_1 = temp\_1.split("/");  
 String diskName=splitDir\_1[1];  
  
 *diskService*.freeFile(fcb,diskName);  
 }  
// //如果是空目录 不允许是当前目录  
// if(fcb == Memory.getInstance().getCurDir()){  
// System.out.println("[error] 无法删除当前目录 请先退出当前目录！");  
// return false;  
// }  
 //从FCB集合中去除 修改父目录文件项 修改父目录儿子结点  
 Disk.*getINSTANCE*().getFcbList().remove(fcb);  
 fcb.getFather().getIndexNode().subFcbNum();  
 fcb.getFather().getChildren().remove(fcb);  
 //递归修改父目录文件大小  
 *dirService*.updateSize(fcb,false,-1);  
 System.*out*.println("[success] 删除成功");  
 return true;  
 }

###### 结果描述：

删除文件filexx。

#### **Write** 命令说明：

###### 设计流程：

用户输入 [ 命令 + 文件路径 ]

文件系统会检测输入的文件路径的合法性，

如果合法则写文件。

###### 关键代码解析：

检查命令合法性：

数组长度=1 则输出命令提示

否则正常执行

case "Write":/\*13\*/  
 if(inputs.length == 1){  
 System.*out*.println("Write [FilePath]");  
 break;  
 }  
 *fileService*.open(inputs[1]);  
 *fileService*.write(inputs[1]);  
 *fileService*.close(inputs[1]);  
 break;

首先打开文件

检查文件路径合法性：

先判断路径是否存在，不存在则报错，否则继续。

然后判断是否非文件，如是则报错，否则继续。

然后判断文件是否已有内容，如有则判断追加还是覆盖写入，否则继续。

写入磁盘

然后从FCB集合中去除 修改父目录文件项 修改父目录儿子结点

然后递归修改父目录文件大小

最后关闭文件

@Override  
 public Boolean write(String filePath) {  
 //判断是否存在  
 FCB fcb = *dirService*.pathResolve(filePath);  
 if(Objects.*isNull*(fcb)){  
 System.*out*.println("[error] 目标文件不存在");  
 return false;  
 }else if(!fcb.getType().equals("FILE")){  
 //type DISK DIR 不是普通文件  
 System.*out*.println("[error] 无法写非文件");  
 return false;  
 }else {  
 //type FILE 普通文件  
 //判断是否在openFileList中  
 String fill\_path = *dirService*.pwd(fcb);  
 List<OpenFile> openFileList = Memory.*getInstance*().getOpenFileList();  
 OpenFile toWriteFile = null;  
 for (OpenFile openFile : openFileList) {  
 if(openFile.getFilePath().equals(fill\_path)){  
 toWriteFile = openFile;  
 }  
 }  
 if(Objects.*nonNull*(toWriteFile)){  
 StringBuilder content = new StringBuilder();  
 System.*out*.println("请输入要写入的内容 [...^]:");  
 //获取用户输入 输入//end结束  
 while (true){  
 String nextLine = *scanner*.nextLine();  
 if(nextLine.endsWith("^")){  
 content.append(nextLine,0,nextLine.length()-1);  
 break;  
 }else {  
 content.append(nextLine);  
 content.append("\n");  
 }  
 }  
 String choice = null;  
 if(fcb.getIndexNode().getSize() == 0){  
 //空文件 默认覆盖  
 choice = "1";  
 }else {  
 //有内容 让用户选择写入模式  
 while (true){  
 System.*out*.println("原文件有内容 请选择覆盖写[1]/追加写[2]:");  
 choice = *scanner*.nextLine();  
 if(choice.equals("1") || choice.equals("2")){  
 break;  
 }  
 }  
 }  
 int[] fat = Memory.*getInstance*().getFat();  
 int size = content.toString().toCharArray().length;//文件大小为输入字数  
  
 String temp\_1=*dirService*.pwd(fcb);  
 String[] splitDir\_1 = temp\_1.split("/");  
 String diskName=splitDir\_1[1];  
  
 if(choice.equals("1")){  
 //覆盖写入  
 //1.如果不是空文件 则清空之前占据的盘块  
 if(fcb.getIndexNode().getSize() != 0){  
 *diskService*.freeFile(fcb,diskName);  
 }  
 //2.重新写入  
 int first = *diskService*.writeToDisk(content.toString(),diskName);  
 //3.将文件指向第一块  
 fcb.getIndexNode().setFirst\_block(first);  
 //4.修改索引结点大小  
 fcb.getIndexNode().setSize(size);  
 //修改父目录项 以及一直递归修改父目录的大小  
 *dirService*.updateSize(fcb,true,-1);  
 }else {  
 //追加写入  
 //1.从第一块往下找 直到-1的块的块号  
// int temp = fat[fcb.getIndexNode().getFirst\_block()];  
 int temp = fcb.getIndexNode().getFirst\_block();  
 int temp2 = temp;  
// while (fat[temp] != -1){  
 while (temp != -1){  
 temp2 = temp;  
 temp = fat[temp];  
 }  
 //2.写入要追加的内容  
// content.insert(0,'\n');  
 int append\_begin = *diskService*.writeToDisk(content.toString(),diskName);  
 //3.修改最后一块指向新的内容  
 fat[temp2]=append\_begin;  
 //4.修改索引结点大小 加上原来的  
 int size\_origin = fcb.getIndexNode().getSize();  
 fcb.getIndexNode().setSize(size + size\_origin);  
 //修改父目录项 以及一直递归修改父目录的大小  
 *dirService*.updateSize(fcb,true,size);  
 }  
 System.*out*.println("[success] 写入成功");  
 return true;  
 }else {  
 System.*out*.println("[error] 文件未打开 请先打开");  
 return false;  
 }  
// }  
 }  
 }

###### 结果描述：

打开文件filexx并支持写入保存

#### **Read** 命令说明：

###### 设计流程：

用户输入 [ 命令 + 文件路径 ]

文件系统会检测输入的文件路径的合法性，

如果合法则读文件。

###### 关键代码解析：

检查命令合法性：

数组长度=1 则输出命令提示

否则正常执行

case "Read":/\*14\*/  
 if(inputs.length == 1){  
 System.*out*.println("Read [FilePath]");  
 break;  
 }  
 *fileService*.open(inputs[1]);  
 *fileService*.read(inputs[1]);  
 *fileService*.close(inputs[1]);  
 break;

首先打开文件

检查文件路径合法性：

先判断路径是否存在，不存在则报错，否则继续。

然后判断是否非文件，如是则报错，否则继续。

然后判断文件是否已有内容，如无则输出empty，否则读出内容。

最后关闭文件

@Override  
 public Boolean read(String filePath) {  
 //判断是否存在  
 FCB fcb = *dirService*.pathResolve(filePath);  
 if(Objects.*isNull*(fcb)){  
 System.*out*.println("[error] 目标文件不存在");  
 return false;  
 }else if(!fcb.getType().equals("FILE")){  
 //type DISK DIR 不是普通文件  
 System.*out*.println("[error] 无法读非文件");  
 return false;  
 }else {  
 //type FILE 普通文件  
 //判断是否在openFileList中  
 String fill\_path = *dirService*.pwd(fcb);  
 List<OpenFile> openFileList = Memory.*getInstance*().getOpenFileList();  
 OpenFile toWriteFile = null;  
 for (OpenFile openFile : openFileList) {  
 if(openFile.getFilePath().equals(fill\_path)){  
 toWriteFile = openFile;  
 }  
 }  
 if(Objects.*nonNull*(toWriteFile)){  
 int[] fat = Memory.*getInstance*().getFat();  
 Block[] disk = Disk.*getINSTANCE*().getBlocks();  
 //从磁盘读取  
 System.*out*.println("--------BEGIN--------");  
 if(fcb.getIndexNode().getSize() == 0){  
 System.*out*.println("<!--EMPTY FILE-->");  
 System.*out*.println("---------END---------");  
 return false;  
 }  
// int temp = fat[fcb.getIndexNode().getFirst\_block()];  
 int temp = fcb.getIndexNode().getFirst\_block();  
// while (fat[temp] != -1){  
 while (temp != -1){  
 //遍历输出  
 System.*out*.print(disk[temp].getContent());  
 temp = fat[temp];  
 }  
// System.out.print(disk[temp].getContent());  
 System.*out*.println();  
 System.*out*.println("---------END---------");  
 }else {  
 System.*out*.println("[error] 文件未打开 请先打开");  
 return false;  
 }  
 }  
 return true;  
 }

###### 结果描述：

打开文件filexx并输出显示

#### **Export** 命令说明：

###### 设计流程：

用户输入 [ 命令 + 文件路径 ]

文件系统会检测输入的文件路径的合法性，

如果合法则创建文件。

###### 关键代码解析：

检查命令合法性：

数组长度<3 则输出命令提示

否则正常执行

case "Export":/\*15\*///C:\Users\JIALIANGLI\Desktop  
 if(inputs.length < 3){  
 System.*out*.println("Export [FilePath] [ExDirPath]");  
 break;  
 }  
 *fileService*.export(inputs[1],inputs[2]);  
 break;

检查源文件路径合法性：

先判断路径是否存在，不存在则报错，否则继续。

然后判断是否非文件，如是则报错，否则继续。

检查外部目录路径合法性：

先判断外部路径是否存在，不存在则报错，否则继续。

然后判断目标目录中是否有存在与源目录重名情况，存在则重命名+“-1”，否则继续

然后创建新文件，读出源文件内容，写入新文件

@Override  
 public Boolean export(String filePath, String exDirPath) throws IOException {  
 //判断是否存在  
 FCB fcb = *dirService*.pathResolve(filePath);  
 if(Objects.*isNull*(fcb)){  
 System.*out*.println("[error] 源文件不存在");  
 return false;  
 }  
 else if(!fcb.getType().equals("FILE")){  
 System.*out*.println("[error] 不能导出非文件");  
 return false;  
 }  
  
  
 */\*\*begin: 在目标文件夹下创建复制文件\*\*/* /\*begin: read from old file and write into new file\*/  
 //判断重复  
 String fileName=fcb.getFileName();  
 File f = new File(exDirPath);  
 File[] listFile = f.listFiles();// 取出该目录下所有文件以及文件夹  
 int i = 0;  
 // 开始遍历每一个目录项  
 for (i = 0; i < listFile.length; i++) {  
 f = listFile[i];  
 if (f.isFile()&&f.getName().equals(fileName+".txt")) {  
 fileName+="-1";  
 System.*out*.println("[error] 文件名重复 已重命名为\""+fileName+".txt"+"\"");  
 break;  
 }  
 }  
  
 exDirPath = exDirPath +"\\"+fileName+".txt";;  
// System.out.println(newFilePath);  
 File file=new File(exDirPath);  
 file.createNewFile();  
 FileWriter fileWriter = new FileWriter(exDirPath);  
 BufferedWriter bufferedWriter = new BufferedWriter(fileWriter);  
 StringBuilder content = new StringBuilder();  
  
 //read  
 int[] fat = Memory.*getInstance*().getFat();  
 Block[] disk = Disk.*getINSTANCE*().getBlocks();  
 //从磁盘读取  
 if(fcb.getIndexNode().getSize() == 0){  
 //空文件  
 }  
 else{  
// int temp = fat[fcb.getIndexNode().getFirst\_block()];  
 int temp = fcb.getIndexNode().getFirst\_block();  
// while (fat[temp] != -1){  
 while (temp != -1){  
 //遍历读出  
 content.append(disk[temp].getContent());  
 temp = fat[temp];  
 }  
// content.append(disk[temp].getContent());  
 }  
 System.*out*.println("[success] 读出成功");  
  
 //write  
 bufferedWriter.write(content.toString());  
 bufferedWriter.flush();  
 bufferedWriter.close();  
 System.*out*.println("[success] 写入成功");  
  
 /\*end: read from old file and write into new file\*/  
  
 System.*out*.println("[success] 导出文件成功");  
 */\*\*end: 在目标文件夹下创建复制文件\*\*/* return true;  
 }

###### 结果描述：

把虚拟磁盘上的文件filexx导出到外部硬盘exdir目录下。

### 5、任务2——Pintos分析

#### 通用链表的基本使用：

###### 设计流程：

使用pintos通用链表数据结构中提供的函数完成多个tcb结构体的创建、插入到mylist链表中。

###### 关键代码解析：

步骤1：新建 thread 结构体。

步骤2：为tid字段赋值。

每个新创建的thread结构体的 tid 字段依次赋值为数组 mytid中的对应数值（mytid为myinsert的第二个参数）。

步骤3：插入到链表中。

实验中的代码已经在原有 thread 结构体基础上增加了一个链表元素字段 myelem，并声明了一个链表mylist，插入过程中使用自定义的字段 myelem 作为链表元素挂载到 mylist 中。

/\*

创建 5 个 thread 结构体，为每个结构体的 tid 字段依次赋值为第二个参数数组 mytid 中的对应数值，并依次将其插入到第一个参数list 中，插入过程中要求使用自定义的字段 myelem 作为链表元素挂载到 list 中。

\*/

void myinsert(struct list \*list,tid\_t mytid[5])

{

    list\_init(list);

    for(int i=0;i<5;i++)

  {

      /\*\*\*begin 补全以下代码\*\*\*/

      struct thread \*t = (struct thread \*)malloc(sizeof(struct thread));

      t->tid=mytid[i];

      list\_push\_back(list,&t->myelem);

    /\*\*end\*\*/

  }

}

###### 结果描述：

完成多个 tcb 的创建、插入到自定义链表 mylist 中。

#### 通用链表的首地址计算：

###### 设计流程：

遍历mylist链表，并对其中每个元素所在的thread节点执行第一个参数名对应的函数func，第二个参数aux为传给func的参数。

###### 关键代码解析：

通过调用list\_entry宏，计算出各链表节点的首地址（即每个tcb的首地址/指针t），并将返回的t作为参数传入到thread\_for\_each的第一个参数func中。

void

thread\_foreach(thread\_action\_func \*func, void \*aux)

{

  struct list\_elem \*e;

  for (e = list\_begin (&mylist); e != list\_end (&mylist);

       e = list\_next (e))

    {

      struct thread \*t ;

      /\*\*\*\*   BEGIN  补全以下代码\*\*\*\*\*/

      t = list\_entry (e, struct thread, myelem);

      /\*\*\*\*   END  \*\*\*\*\*/

      func (t, aux);

    }

}

###### 结果描述：

通过链表节点中的链表元素，获取到链表节点的首地址

#### 基于函数指针的通用链表操作：

###### 设计流程：

创建5个tcb结构体，为每个结构体的tid字段依次赋值为数组mytid中的对应数值，并依次将其有序插入到自定义链表mylist中（调用已经定义好的list\_insert\_ordered），插入过程中要求使用自定义的字段myelem作为链表元素挂载到mylist中

###### 关键代码解析：

按照tid大小定义偏序关系，使用这种关系有序插入。

/\*  自定义比较函数\*/

bool

mycompare(const struct list\_elem \*a, const struct list\_elem \*b, void \*aux)

{

    /\*\*\*begin  补全以下代码\*\*\*/

    const struct thread \*t1 = list\_entry (a, struct thread, myelem);

    const struct thread \*t2 = list\_entry (b, struct thread, myelem);

    return t1->tid<t2->tid;

  /\*\*\*end\*\*\*/

}

/\*

创建 5 个 tcb 结构体，为每个结构体的 tid 字段赋值为第二个参数数组 mytid 中的数值，并【按序】将其插入到第一个参数list 中，插入过程中要求使用自定义的字段 myelem 作为链表元素挂载到 list 中。

\*/

void myinsert(struct list \*list,tid\_t mytid[5])

{

    list\_init(list);

    for(int i=0;i<5;i++)

  {

      /\*\*\*begin 补全以下代码\*\*\*/

      struct thread \*t = (struct thread \*)malloc(sizeof(struct thread));

      t->tid=mytid[i];

      list\_insert\_ordered(list,&t->myelem,&mycompare,NULL);

    /\*\*end\*\*/

  }

}

###### 结果描述：

将5个tcb结构体有序插入到自定义链表mylist中

#### Pintos的编译、运行和调试：

###### 设计流程：

打印提示信息后，再打印一行信息“Booting my pintos!”。

请修改代码，重新编译内核，实现该信息的打印，并通过在

/data/workspace/myshixun/pintos/src/threads

目录下运行命令

pintos -- run alarm-multiple | tee 1.txt

将运行结果保存。

###### 关键代码解析：

printf(“Booting my pintos!”)

输出“Booting my pintos!”

###### 结果描述：

输出“Booting my pintos!”

# 三、心得体会

### 1、已完成任务总结

按**任务1和任务2两大分类**分别阐述总结已经完成的功能和设计，并使用思维导图介绍已经实现的功能和设计逻辑。

任务1：

1. 已经完成的功能和设计：
2. 磁盘空间管理包括展示磁盘分区（showDisk）、创建磁盘分区（mkDisk）、切换磁盘分区（chgDisk）、移动目录中文件（moveDir）、释放目录内存（freeDir）、释放文件内存（freeFile）、写入磁盘分区（writeToDisk）、寻找空闲块（findEmpty）、显示空闲空间（showFree）

（2）文件目录结构管理包括展示目录(dir)、创建目录(mkDir)、切换目录(chgDir)、删除目录(delete)、移动目录（move）、路径解析（pathResolve）、更新大小（updateSize）、展示绝对路径（pwd）、树形展示目录（treeDir）。

（3）文件外存分配管理包括创建文件（create），打开文件（open），读取文件（read），写入文件（write），关闭文件（close），删除文件（delete），拷贝文件（copy），导出文件（export）

（4）数据存储管理包括初始化（initData），加载数据（loadData），保存数据（saveData）

（5）其它业务逻辑包括打开文件（open），关闭文件（close），帮助（help），退出（exit）

2. 实现的功能和设计逻辑：（图1，图2）

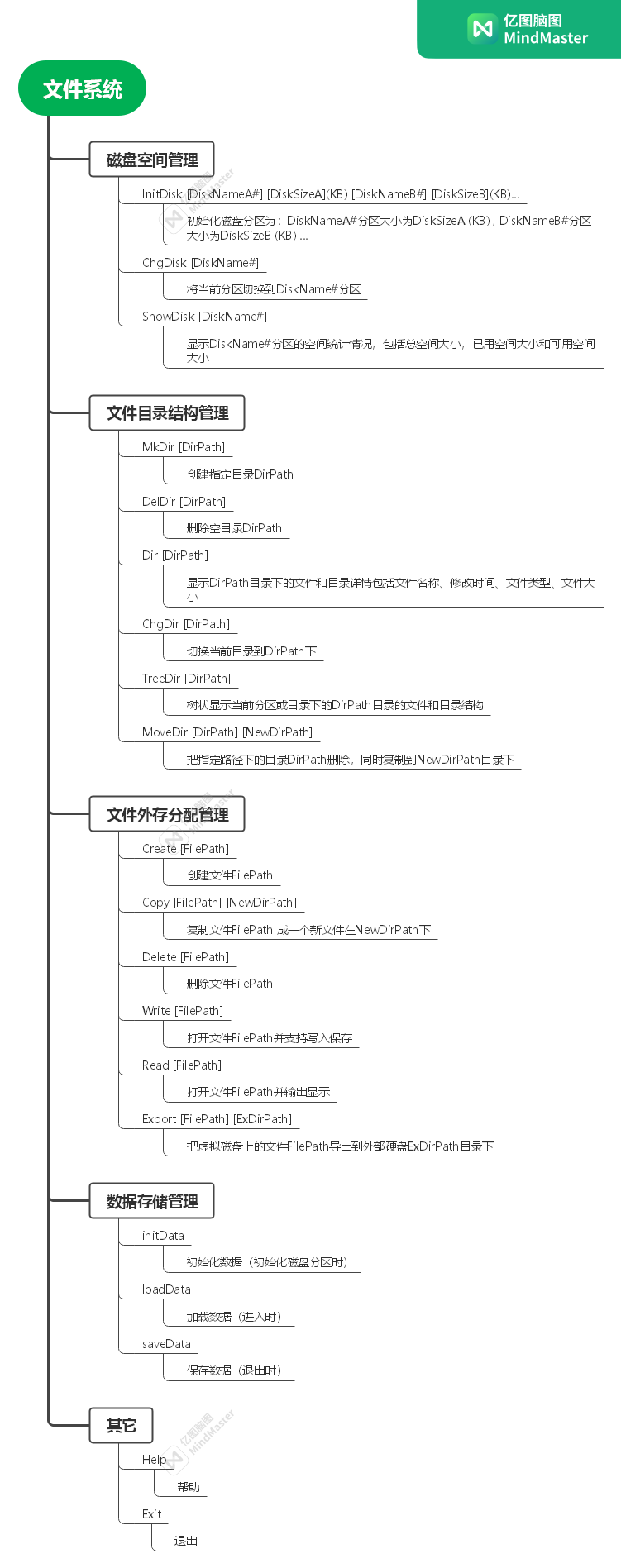


图1 任务一实现功能

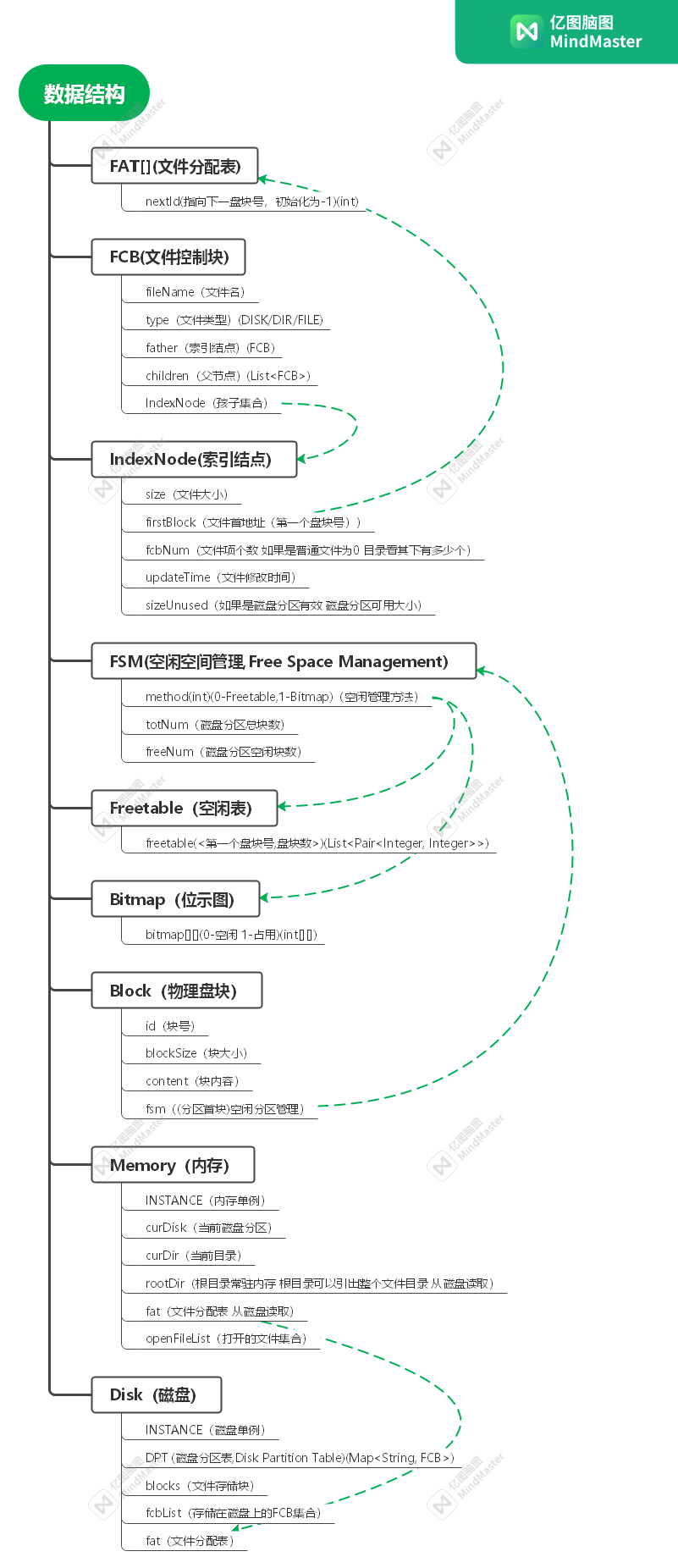


图2 任务一设计逻辑

\*注：

1. FCB:

每个文件（普通文件和目录）都有文件控制块FCB，凭借FCB才可以形成树形文件目录。本系统的FCB除了文件名文件类型以外的大部分内容都由索引结点进行封装，加速读取速率。每个FCB有父节点和孩子集合，采用双亲孩子表示法进行树结构的组织。

1. IndexNode:

文件的索引结点存放文件大小，存放在的物理块首地址，文件项数（如果是目录才有效，文件为0），文件创建者，修改时间，未用空间大小（如果是磁盘分区才有效）。目录项增加索引结点的方式进行封装，大量节省目录的空间，加快访问速率。

1. 内存和磁盘中均有FAT表：

内存中常驻FAT表，当前目录FCB，根目录FCB，以及打开的文件集合，本系统实现了文件多开，通过设置openFileList记录每一个打开的文件，还可以防止文件多开以及防止退出文件未保存。

1. open:

用于打开文件，用于在对文件进行操作前需要先打开文件，这样可以方便操作系统分配资源使用权比如外设的管理。打开文件包括解析文件路径是否存在，解析是否是普通文件，判断是否重复打开，然后封装为openFile加入到openFileList中。系统可选择打开多个文件。

1. close：

用于关闭文件，只有关闭文件才可以释放资源使用权，别的进程才可以进行使用该资源。关闭过程包括：解析文件路径是否存在，解析是否是普通文件，判断是否在openFileList中。

1. initData：

当系统找不到序列化的虚拟磁盘文件，就会进行初始化操作。初始化包括初始化磁盘数据，FCB集合数据，根目录数据，FAT表，内存数据。

1. loadData：

当存储路径文件存在时，系统通过字节流加载磁盘数据，同时将磁盘中的根目录、当前目录、FAT表读入内存。

1. saveData：

当退出系统时，系统自动将内存数据持久化到磁盘中，再将磁盘存储到虚拟磁盘文件中。

任务2：（图3，图4）

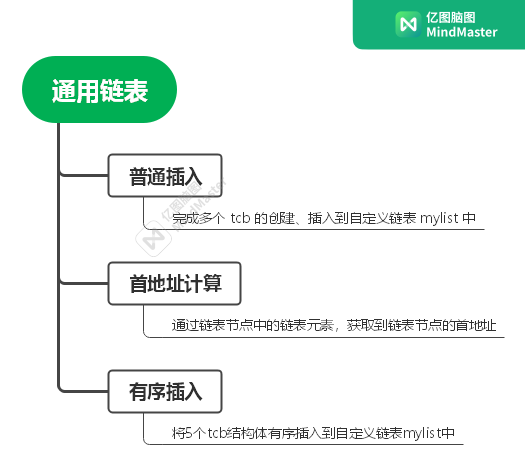


图3 任务二实现功能和设计逻辑



图4 任务二实现功能和设计逻辑

### 2、尚存在的问题

按**任务1和任务2两大分类**分别阐述未实现的功能或存在的未解决问题或系统设计的局限性（例如该系统支持的最大文件名为128个字符，最大文件Size为64K等）。

任务1：

1. 磁盘初始化只支持KB为单位的整数大小的磁盘分区创建
2. 创建或转移目录或文件时，如果父目录为不存在的目录时，不支持创建
3. 位示图寻找空闲空间需从头遍历，效率比较低

任务2：无

### 3、结论分析

要求从知识、能力和素养三方面进行阐述分析。**（此部分要求2000字以上）**

#### 1）知识方面

主要阐述方式：通过查阅教材\*\*（或参考书\*\*、网络资源\*\*、论文\*\*等）和深入分析，巩固和加深了哪些重要的知识点，解决了课程设计中的哪些关键问题。

必写以下几点内容：

1. 磁盘分区存储管理的设计和相应的内存读写结构
2. 两种空闲分区的物理存储设计和相应的内存读写结构
3. 目录结构的物理存储设计和相应的内存读写结构
4. 文件结构的物理存储设计和相应的内存读写结构

通过查阅教材《2023年操作系统考研复习指导》和深入分析，巩固和加深了关于文件控制块和索引结点、文件的操作、文件的逻辑结构、文件的物理结构（连续分配、链接分配（显式链接、隐式链接）、索引分配）、目录结构（树形目录结构）、目录的操作、目录实现、文件系统结构、文件系统布局（文件系统在磁盘中的结构，文件系统在内存中的结构）、外存空闲空间管理（空闲表法、空闲链表法、位示图法、成组链接法）、虚拟文件系统、分区和安装、磁盘的管理，解决了课程设计中的文件控制块和索引结点的设计、文件分配表的设计、文件系统布局（文件系统在磁盘中的结构，文件系统在内存中的结构）的设计、外存空闲空间管理（空闲表法、位示图法）的设计、磁盘分区的设计。

通过查阅网络资源（https://blog.csdn.net/wx\_assa/article/details/103834318?

ops\_request\_misc=%257B%2522request%255Fid%2522%253A%2522168896728316800188519441%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request\_id=168896728316800188519441&biz\_id=0&utm\_medium=distribute.pc\_search\_result.none-task-blog-2~all~sobaiduend~default-2-103834318-null-null.142^v88^control,239^v2^insert\_chatgpt&utm\_term=%E6%96%87%E4%BB%B6%E7%AE%A1%E7%90%86%E7%B3%BB%E7%BB%9F%E4%BD%8D%E7%A4%BA%E5%9B%BE&spm=1018.2226.3001.4187）和深入分析，巩固和加深了对位示图法的理解，解决了课程设计中的位示图设计问题。

通过查阅网络资源（https://blog.csdn.net/FHNCSDN/article/details/120272627?

ops\_request\_misc=%257B%2522request%255Fid%2522%253A%2522168896731116800213065544%2522%252C%2522scm%2522%253A%252220140713.130102334.pc%255Fall.%2522%257D&request\_id=168896731116800213065544&biz\_id=0&utm\_medium=distribute.pc\_search\_result.none-task-blog-2~all~first\_rank\_ecpm\_v1~rank\_v31\_ecpm-8-120272627-null-null.142^v88^control,239^v2^insert\_chatgpt&utm\_term=%E6%96%87%E4%BB%B6%E7%AE%A1%E7%90%86%E7%B3%BB%E7%BB%9F%E4%BD%8D%E7%A4%BA%E5%9B%BE&spm=1018.2226.3001.4187）和深入分析，巩固和加深了对文件系统目录结构、磁盘分区的理解，解决了课程设计中的文件系统目录结构、磁盘分区设计问题。

通过查阅网络资源（https://blog.csdn.net/qq\_41587740/article/details/109055716?

ops\_request\_misc=%257B%2522request%255Fid%2522%253A%2522168896731116800213065544%2522%252C%2522scm%2522%253A%252220140713.130102334.pc%255Fall.%2522%257D&request\_id=168896731116800213065544&biz\_id=0&utm\_medium=distribute.pc\_search\_result.none-task-blog-2~all~first\_rank\_ecpm\_v1~rank\_v31\_ecpm-7-109055716-null-null.142^v88^control,239^v2^insert\_chatgpt&utm\_term=%E6%96%87%E4%BB%B6%E7%AE%A1%E7%90%86%E7%B3%BB%E7%BB%9F%E4%BD%8D%E7%A4%BA%E5%9B%BE&spm=1018.2226.3001.4187）和深入分析，巩固和加深了对各种文件存储空间管理的理解，解决了课程设计中的空闲表和位示图设计问题。

通过查阅网络资源（https://github.com/Hjyheart/Project-for-os-fileSystem，https://github.com/szr712/FileSystem，

[https://github.com/SunnyBoy-WYH/OS-File，https://github.com/spice-wolf/FileManagementSystem，https://github.com/HellooYing/DocManage，](https://github.com/SunnyBoy-WYH/OS-File，https:/github.com/spice-wolf/FileManagementSystem，https:/github.com/HellooYing/DocManage，)

<https://gitee.com/tzecochan/FileSystem>

<https://gitee.com/loulabi/CLinux>

https://gitee.com/tzecochan/FileSystem

）和深入分析，巩固和加深了对各种文件系统框架设计的理解，解决了课程设计中的空闲表和文件系统框架设计问题。

1. 磁盘分区存储管理的设计和相应的内存读写结构：

文件控制块上可以找到磁盘分区的首物理块地址，首物理块上存了磁盘分区信息。

先找对应磁盘分区的空闲空间管理方式，再按照对应方式进行读写。

1. 两种空闲分区的物理存储设计和相应的内存读写结构：

文件控制块上可以找到磁盘分区的首物理块地址，首物理块上存了空闲分区管理表。

先找对应磁盘分区的空闲空间管理方式，再按照对应方式进行读写。

1. 目录结构的物理存储设计和相应的内存读写结构：

文件控制块上可以找到目录的首物理块地址，首物理块上存了目录信息。

先找对应磁盘分区的空闲空间管理方式，再按照对应方式进行读写。

1. 文件结构的物理存储设计和相应的内存读写结构：

文件控制块上可以找到文件的首物理块地址，首物理块上存了文件信息。

先找对应磁盘分区的空闲空间管理方式，再按照对应方式进行读写。

#### 2） 能力方面

主要阐述方式：通过本次课设，提升了哪些能力。

可以从算法设计能力、算法实现能力、算法分析能力、代码调试能力、解决复杂工程问题的能力、文献检索能力、创新能力等方面进行阐述。

通过本次课设，提升了算法设计能力、算法实现能力、算法分析能力、代码调试能力、解决复杂工程问题的能力、文献检索能力、创新能力。

1. 算法设计能力：通过对课本知识的反复消化，从而复现抽象算法，既考验了我对课本知识的理解和掌握，也考验了我算法从抽象到具体的设计能力。
2. 算法实现能力：对设计出来的伪代码框架进行代码实现，考验了我算法从抽象到具体的实现能力。
3. 算法分析能力：为了复现算法，需要对课本算法的反复消化和理解，考验了我的算法分析能力。
4. 代码调试能力：锻炼了我用IDEA调试程序的方法。
5. 解决复杂过程问题的能力：如何设计整体框架，分解文件系统的各个模块，自顶向下的设计程序。
6. 文献检索能力：提高了对具体问题具体分析然后有效检索CSDN，Github，Gitee的能力
7. 创新能力：包括对原本教材上一些算法在实践后的改动和融合，对抽象算法的反复打磨修改后形成自己独特思路的创新。

#### 3）素养方面

主要阐述方式：通过本次课设，提升了哪些个人的综合素养。下面提供的4点内容必须包括，其他方面可以根据自身的理解从多个角度进行简述，具体内容根据个人实际情况总结即可。

必写以下几点内容：

（1）通过本次课设，谈谈如何将专业知识直接服务于我国的科技发展与建设，如何将个人理想与国家的发展需求结合在一起。

（2）通过总结分析课设中空闲分区管理采用不同解决方案，谈谈解决实际问题时，策略选择对实现效率的决定性作用，谈谈对精益求精的工匠精神和终身学习意识的深刻认识。

（3）通过对课设中对PintOS的研习，谈谈对系统编程能力培养和发展的认知。

（4）通过阅读和学习《软件工程师职业道德规范和标准》，结合课设的具体内容，谈谈作为软件工程师在软件开发中应该遵守的职业道德规范。

例如，下面给出的几条职业道德规范。（不限于以下几条，可以从《软件工程师职业道德规范和标准》任选一条或多条进行简述。）

*3.06 只要条件许可，就应当采取最合适的专业标准去完成手头的任务，除非有道德或者技术上的正当理由来支持你不这么做。*

*3.11 确保项目文档齐全，包括所有发现的问题和解决的方法。。*

*3.13 留心只用合乎道德和法律的手段去使用准确的数据，并且只按照被适当授权的方式去使用这些数据。*

*3.14 维护数据的完整性，注意过期和有问题的数据。*

*3.15 对于任何形式的软件维护工作，要具备同开发新软件时一样的专业精神。*

通过本次课设，提升了热爱祖国、科技报国、工匠精神、终身学习、系统编程能力、软件工程师职业道德等综合素养。

（1）通过本次课设，我认识到文件系统作为家家户户电脑操作系统的重要组成部分，关系到无数人民的使用效率和体验，如果能设计出我国独创的文件系统，将不会被西方国家在技术上垄断，将会有完整的服务体系惠及人民，这就是将专业知识直接服务于我国的科技发展与建设以及将个人理想与国家的发展需求结合在一起。

（2）通过总结分析课设中空闲分区管理采用不同解决方案，在解决实际问题时，策略选择对实现效率有决定性作用，这在各种算法的时间效率和空间效率的对比上体现很明显，如简单的先来先服务或纯粹的暴力遍历无法满足现实在效率上的需求，并且造成大量的资源浪费，在现实中将是对人力物力的耗费，甚至是环境资源的破坏，最后效果并不好，因此策略选择对实现效率有决定性作用，经过算法优化后再行动会推动效率提升效果。谈谈对精益求精的工匠精神和终身学习意识的深刻认识。

（3）通过对课设中对PintOS的研习，我对系统编程能力培养和发展有了更深的认知。PintOS是斯坦福大学操作系统课程的课程项目，通过PintOS的自学，我能够更清晰地理解操作系统中的链表、线程到底是怎么一回事，需要考虑哪些因素：优先级的循环更新，嵌套调度，时间片的考量，关于线程锁的概念，以及更复杂的队列调度算法，做为系统的基本数据结构，为什么这么设计、设计的巧妙之处、这样设计带来的好处、怎么样适应算法对我的系统编程能力培养和发展有很大的启发。

（4）通过阅读和学习《软件工程师职业道德规范和标准》，我学习到了作为软件工程师在软件开发中应该遵守的职业道德规范。

3.06 只要条件许可，就应当采取最合适的专业标准去完成手头的任务，除非有道德或者技术上的正当理由来支持你不这么做。

我以我的学习水平尽可能的复现和创新了课本上的算法，使得实践上该算法的实际运行效率效果是最好的。

3.11 确保项目文档齐全，包括所有发现的问题和解决的方法。

我在项目文档报告中提出了我发现的问题和解决的方法。

3.13 留心只用合乎道德和法律的手段去使用准确的数据，并且只按照被适当授权的方式去使用这些数据。

我设计的文件系统只有导出文件是可以访问到外部磁盘的文件名信息，其它地方不侵犯用户的隐私，这是合理合法的。

3.14 维护数据的完整性，注意过期和有问题的数据。

我把数据持久化到“filesystem.txt”,如果加载数据有问题则会提醒用户重新初始化。

3.15 对于任何形式的软件维护工作，要具备同开发新软件时一样的专业精神。

当出现新问题时要有足够的耐心去改善软件，维护其实就是完善，使这个软件更健壮，更安全。