1. 课程设计任务、要求、目的

1.1 课程设计任务

- 首先分配一定容量的磁盘存储空间，作为文件存储空间；
- 建立相应的文件系统，使用隐式链接物理文件的系统；
- 解决文件的重名、共享和安全控制；
- 支持文件的“按名存取”；
- 为该文件系统设计相应的数据结构来管理目录、磁盘空闲空间、已分配空间等。
- 提供文件的创建、删除、移位、改名等功能。
- 提供良好的界面，可以显示磁盘文件系统的状态和空间的使用情况；
- 提供虚拟磁盘转储功能，可将信息存入磁盘，还可从磁盘读入内存；

1.2 课程设计目的和要求

系统应该包含两个部分，一个部分是按内核代码原则设计的固定分区分配存储管理系统，由一系列的函数组成；另一个部分是演示系统，调用固定分区分配存储管理系统的相应函数，以让其运行，同时提供系统的展示界面，可以是 GUI 或者字符界面，以展示系统的运行状态，显示系统的关键数据结构的内容。

具体包括：建立描述磁盘分配状况的数据结构；建立描述文件控制块，磁盘物理块,目录控制块的数据结构；对文件进行的操作,如读取与写入操作,文件删除与文件创建，目录删除与目录创建，文件和目录的重命名与防重名，切换用户等操作。

编写一个以图形化展示的 GUI 界面。

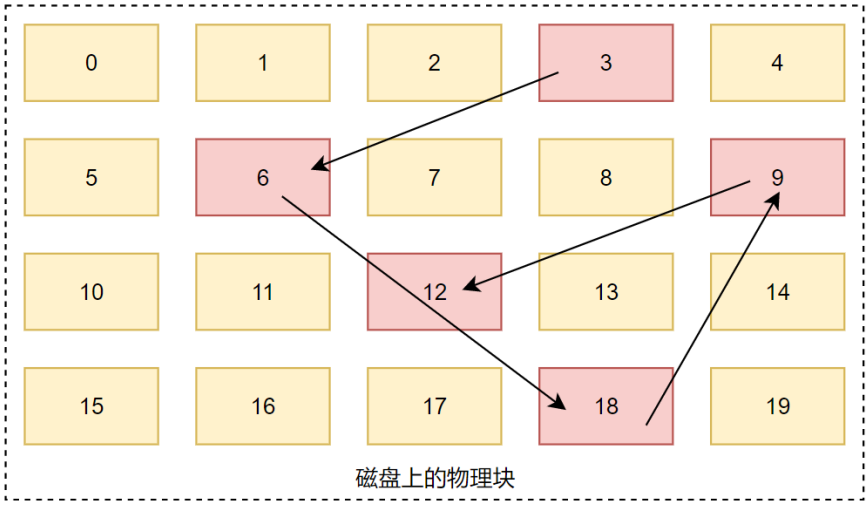
2. 开发环境

设备名称　Lanren1
处理器 Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz　2.40 GHz
机带 RAM　16.0 GB (15.8 GB 可用)
设备 ID　1A7A8D82-9DD6-43CF-949C-D3A0859533A6
产品 ID　00342-35488-62400-AAOEM
系统类型　64 位操作系统，基于 x64 的处理器
C/C++语言 IDE Visual studio 2019
运行平台为 x86 指令，int 指针类型为 4 字节
GUI 使用的库 使用 openGL 的窗口库 GLFW 和 Dear imGUI

3. 相关原理及算法

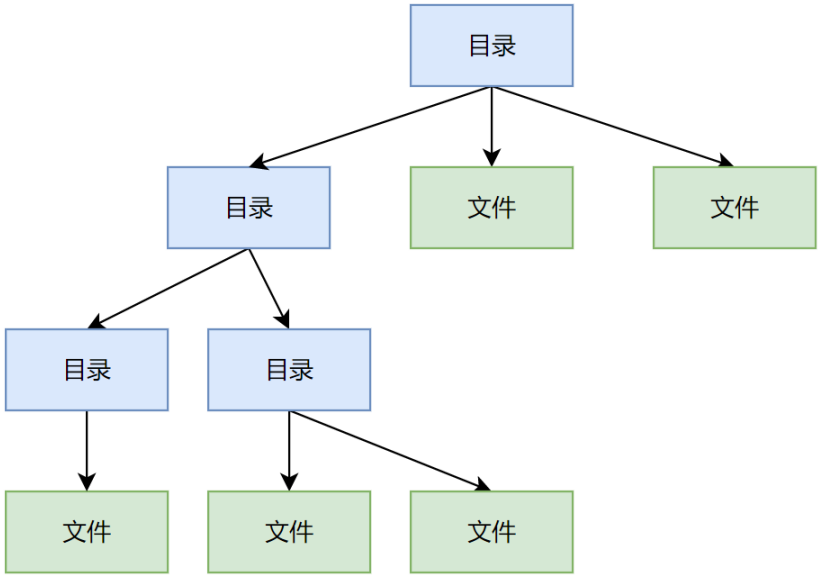
隐式链接文件系统中文件控制块 FCB 记录起始物理块与末尾物理块的编号，如下图所示，通过链式查找可将文件读取，除了文件的最后一个磁盘块之外，每个磁盘块都会保存指向下一个盘块的指针，这些指针对用户是透明的。

文件名	第一个物理块	最后一个物理块	...
XXX	3	12	...



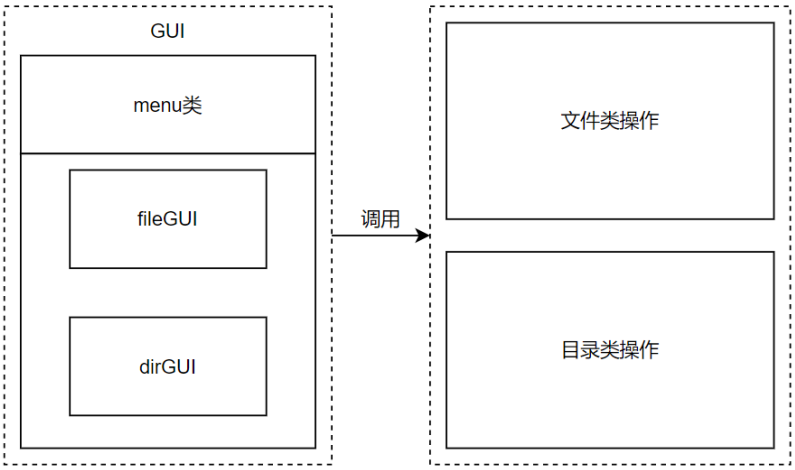
文件控制块与目录控制块均可保存于磁盘中，占有相同大小的空间，便于存储与磁盘中

文件控制块 FCB 保存文件相关操作需要的数据；目录控制块 DIR 保存目录中文件链与目录链的指针等等数据，用于完成目录操作；逻辑上文件系统为一个树状结构，顶层为根目录，根目录无法移动和删除。

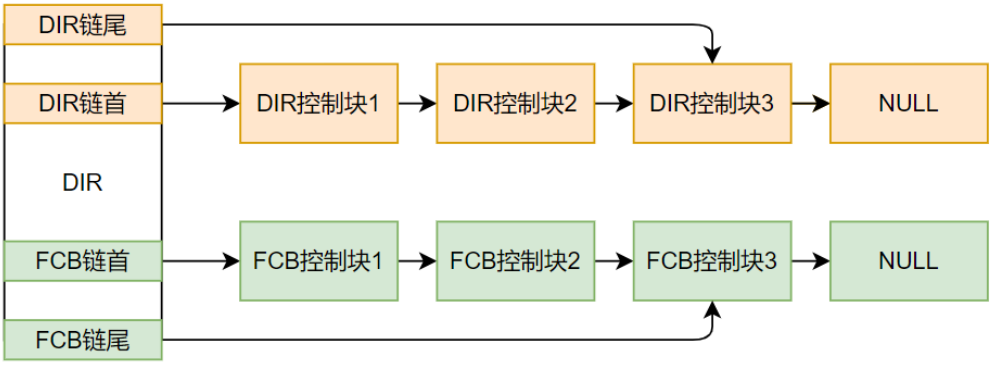


4. 系统结构和主要的算法设计思路

整个文件系统由**GUI 代码**与**文件系统代码**组成，为文件系统在磁盘中分配了一个 10000KB 大小的 block 文件，一个物理块为 1KB 大小，共包含物理块 10000 个，物理块的刷写与读入使用了 C 语言的文件读写函数来模拟 I/O，由两个函数完成：getDiskBlock(),sendBackDisk(),对 GUI 使用时透明的,GUI 调用系统代码给出的接口进行对应操作。



为实现目录在逻辑上的树形结构，在目录下的所有文件的FCB组织为双向链表，FCB结构体中包含链表的上一项FCB指针和下一项FCB指针；在目录下的所有目录同样也组织为链表，目录控制块DIR结构体中包含上一项DIR的指针和下一项的DIR指针;当为链首或链尾时用NULL指向空。



文件系统中主要操作都围绕文件控制块与目录控制块展开，向GUI界面提供的接口包括:

文件创建，检查文件是否在目录中重名，请求FCB的空间，向空闲块请求分配物理块，若成功分配物理块则将创建的文件添加到父目录中，最后将更改过的信息刷写回磁盘。

文件删除，读取文件的FCB控制块，从起始物理块开始归还给空闲块，链式查询所有物理块直到最后一个物理块归还完成，然后将FCB所占空间归还，将更改完的信息刷写回磁盘。

文件重命名，读取文件的FCB控制块至内存中，更改FCB的文件名，刷写回磁盘。

目录创建，检查目录是否在当前目录下重名，请求DIR空间，将信息初始化，最后将修改的信息刷写回磁盘。

目录删除，读取目录的DIR，链式查询目录下的所有目录DIR控制块并删除，链式查询目录下的所有文件FCB控制块并删除，将DIR所占空间归还，将更改的信息刷写回磁盘。

目录重命名，读取目录的 DIR 控制块至内存中，更改 DIR 的目录名，刷写回磁盘。

GUI 演示部分：

由 C++编写，因为 IMGUI 的代码写在渲染每一帧的循环里，所以与其他 GUI 不同，需要采用即时渲染的方式写，每一个窗口以及操作都需要在 if 语句中实现，所有我将组件抽象出来分为了几个主要的控件，用 C++的类封装了起来，共三个主要类：

fileGUI 类包含 **FCB 指针**，根据 FCB 内容渲染文件项，包括文件名，创建时间，创建用户编号；支持打开，修改，剪切，删除，创建快捷方式等操作；

dirGUI 类包含 **DIR 指针**，根据 DIR 信息渲染目录，包括目录名，创建时间，支持打开，剪切，删除，创建快捷方式等操作；

menu 类用于渲染主窗口，menu 根据当前目录控制块 **current** 实时更新需要渲染的所有 **fileGUI** 和 **dirGUI** 项，将这些项加入 vector 中(**files 和 dirs**)，调用这些对象的 **show** 方法渲染至主窗口中，每当发生更改时才重新更新 **files** 或者 **dirs**；支持返回上一级目录，创建文件，创建目录，切换用户，粘贴文件或目录等操作。

GUI 中还包括错误提醒的各类窗口，对使用的用户友好，包括重名错误窗口，路径错误窗口，权限错误窗口，剪切板空警告窗口等等。

5. 程序实现---主要数据结构

文件控制块 FCB:文件控制块的数据结构如下，保存文件操作所需的数据

```
typedef struct FCB//68 字节
{
    u16 fcbBlock;           //fcb 所在物理块
    u16 firstBlock;         //第一个物理块号
```

```
    u16 lastBlock;          //最后一个物理块号

    char name[11];          //文件名
    char nameLen;
    char last[5];           //后缀名
    char lastLen;

    char createTime[4];    //创建文件
    int createTime_year;
    char lastWriteTime[4]; //文件最后修改时间
    int lastWriteTime_year;
    uInt FileSize;          //文件的大小 字节为单位

    myBOOL rw;               //文件可读可写 rw=1 时可读可写, rw=0 时仅可读
    myBOOL isShortCut;        //是否为快捷方式文件
    u16 UserEnable;         //每一位对应一个用户是否可以读写

    struct FCB *preFcb;            //目录的上一个兄弟文件指针
    struct FCB *nextFcb;           //目录的下一个兄弟文件指针
    struct DIR *directory;         //文件所在目录

    uInt rPoint;            //读指针, 一字节为单位
    uInt wPoint;            //写指针, 一字节为单位
}fcb;
```

目录控制块DIR:目录控制块的数据结构, 包含了目录操作所需的数据, FCB链首与链尾, DIR链首与链尾, 以及前后的指针。

```
typedef struct DIR//68 字节
{
    int dirBlock;          //DIR 控制块所属的物理块
    int DIRBLOCK;          //放置该目录下文件 fcb 的起始物理块
    char name[11];         //目录名
    char nameLen;

    char createTime[4];    //目录的创建时间
    int createTime_year;
    char alterTime[4];     //目录的修改时间
    int alterTime_year;

    struct FCB *fcbHead;   //目录下的 FCB 链头
    struct FCB *fcbEnd;    //目录下的 FCB 链尾
    struct DIR *dirHead;   //目录下的目录链头
    struct DIR *dirEnd;    //目录下的目录链尾
    struct DIR *preDir;    //上一个兄弟目录的指针
    struct DIR *nextDir;   //下一个兄弟目录的指针
```

```
    eqlSizeContainer* freeHead;//空闲 fcb 或者 dir 链头

    struct DIR *parent;    //上级目录指针

}dir;
```

dirGUI 控件实现的类

```
class dirGUI
{
private:
    dir* dp;        //文件控制块指针
    int index;      //dirs中的索引
    bool dirOpen;   //打开操作绑定
    bool showChangeWindow;  //修改操作绑定变量
    bool showConfirmWindow; //确认操作绑定变量
    bool showMoveWindow;    //移动操作绑定变量
    bool choose;            //选项窗口绑定变量
public:
    dirGUI(dir* _dp, int _index);
    void changeWindow();    //修改窗口
    void confirmWindow();       //确认窗口
    void chooseWindow();    //选项窗口
    void show();            //控件样式
};
```

fileGUI 控件实现的类

```
class fileGUI
{
private:
    fcb* fp;
    int index;
    bool showFileEditWindow;
    bool showChangeWindow;
    bool showConfirmWindow;
    bool showMoveWindow;
    bool choose;
    bool chooseS;
    //fcb指针一个
public:
    fileGUI(fcb* _fp, int _index);
    void show()
    void chooseSWindow();
    void chooseWindow();
    void editWindow();
    void changeWindow();
```

```
    void confirmWindow();
};
```

公共数据结构,用于渲染的对象,在menu中使用

```
vector<fileGUI> files = {};
vector<dirGUI> dirs = {};
```

6. 程序实现---主要程序清单

目录操作:

```
//************************************************
//函数名：dirCreate
//参 数：父亲目录 名字
//返回值：目录指针
//说 明：创建目录
//************************************************
dir* dirCreate(dir *parent, char name[11], char nameLen)
{
    dir *p = NULL;
    eqlSizeContainer* q = NULL;
    int dirBlock;
    //申请空闲数据项
    q = eqlscGet(parent);
    //申请空闲物理块
    uInt index = BlockGet();

    if (q != NULL && index != 10000)
    {
        //刷新所属物理块块号
        dirBlock = q->num;
        p = (dir*)q;
        p->dirBlock = dirBlock;
        //创建的 dir 控制块须写回 脏化所在物理块
        isDirtyBlock[dirBlock] = ture;
        //链表建立
        array_base[index]->P_after = blockEnd;
        array_base[index]->P_before = blockEnd;
        //记录该目录下文件 fcb 的起始物理块
        p->DIRBLOCK = index;
        //分割物理块为空闲数据项加入空闲链表
        p->freeHead = (eqlSizeContainer*)(array_base[index]->Data);
        p->freeHead = p->freeHead + 14;
        p->freeHead->P_after = NULL;
        p->freeHead->num = index;
        for (int i = 0;i < 14;i++)
        {
```

```
            p->freeHead->P_before = p->freeHead - 1;
            p->freeHead = p->freeHead - 1;
            p->freeHead->P_after = p->freeHead + 1;
            p->freeHead->num = index;
        }
        p->freeHead->P_before = NULL;
        //初始化部分变量
        p->fcbHead = NULL;
        p->fcbEnd = NULL;
        p->dirHead = NULL;
        p->dirEnd = NULL;
        p->parent = parent;
        copyArray(p->name, name, 11);
        p->nameLen = nameLen;
        //time
        time_t t;
        time(&t);
        struct tm* localTime;
        localTime = localtime(&t);
        p->createTime_year = localTime->tm_year;
        p->createTime[0] = (char)localTime->tm_mon;
        p->createTime[1] = (char)localTime->tm_mday;
        p->createTime[2] = (char)localTime->tm_hour;
        p->createTime[3] = (char)localTime->tm_sec;


        if (parent->dirHead == NULL)
        {
            parent->dirHead = p;
            parent->dirEnd = p;
            p->preDir = NULL;
            p->nextDir = NULL;
        }
        else
        {
            //父亲目录的 dir 链不为空则填在最后故新 dir 控制块前的 dir 控制块须写回 脏化
所在物理块
            isDirtyBlock[parent->dirEnd->dirBlock] = ture;
            parent->dirEnd->nextDir = p;
            p->preDir = parent->dirEnd;
            parent->dirEnd = p;
            p->nextDir = NULL;
        }
```

```
        //整个 p 目录写回外存
        if (closeDir(p) == false)
        {
            return NULL;
        }
        //父亲目录中被修改的物理块写回
        sendBackDirDirtyBlocks(parent);

        //parent 控制块写回
        if (sendBackDirBlock(parent->parent, parent->dirBlock) == false)
        {
            return NULL;
        }
    }
    else
    {
        //出错 根据出错归还资源
        if (q != NULL)
        {
            eqlscSendback(parent,q,q->num);
        }
        if(index != 10000)
        {
            //返还物理块
            BlockSendback(index);
            //释放内存
            free(array_base[index]);
            //修改指针
            array_base[index] = NULL;
        }
    }
    return p;
}
//************************************************
//函数名：dirDelete
//参 数：目录指针
//返回值：bool 成功与否
//说 明：目录删除
//************************************************
myBOOL dirDelete(dir *d)
{
    //将 d 目录读入内存
    openDir(d);
```

```
//更新原来 parent 目录的链接
if (d->preDir!=NULL)    //若不是链首
{
    d->preDir->nextDir = d->nextDir;    //前一个目录指向当前 d 的后一目录
    //修改须写回 脏化所在物理块
    isDirtyBlock[d->preDir->dirBlock] = ture;
}
else
{
    d->parent->dirHead = d->nextDir;
    if (d->nextDir != NULL)
        d->nextDir->preDir = NULL;
}
if (d->nextDir != NULL) //若不是链尾
{
    d->nextDir->preDir = d->preDir;     //后一个目录指向当前 d 的前一目录
    //修改须写回 脏化所在物理块
    isDirtyBlock[d->nextDir->dirBlock] = ture;
}
else
{
    d->parent->dirEnd = d->preDir;
    if (d->preDir != NULL)
        d->preDir->nextDir = NULL;
}
//递归回收
//回收目录项
dir *preD = d->dirHead;
dir *nextD;
fcb *preF = d->fcbHead;
fcb *nextF;
while (preD != NULL)
{
    nextD = preD->nextDir;
    dirDelete(preD);
    preD = nextD;
}
//回收文件项
while (preF != NULL)
{
    nextF = preF->nextFcb;
    fileDelete(preF);
    preF = nextF;
}
```

```
    int CURBLOCK;
    //回收本目录
    while (d->DIRBLOCK != blockEnd)
    {
        CURBLOCK = array_base[d->DIRBLOCK]->P_after;
        //返还物理块
        BlockSendback(d->DIRBLOCK);
        //释放内存
        free(array_base[d->DIRBLOCK]);
        //清空指针
        array_base[d->DIRBLOCK] = NULL;
        d->DIRBLOCK = CURBLOCK;
    }

    dir* parent = d->parent;
    eqlscSendback(parent, (eqlSizeContainer*)d, d->dirBlock);
    //父亲目录中被修改的物理块写回
    sendBackDirDirtyBlocks(parent);
    //父亲目录所在物理块写回
    if (sendBackDirBlock(parent->parent, parent->dirBlock) == false)
        return false;

    return ture;
}
//*************************************************
//函数名：dirRename
//参 数：dir 指针 目录名 名字长度
//返回值：bool 成功与否
//说 明：目录重命名
//*************************************************
myBOOL dirRename(dir* d, char name[11], char nameLen)
{
    copyArray(d->name, name, 11);
    d->nameLen = nameLen;
    //写回
    sendBackDirBlock(d->parent, d->dirBlock);
    return ture;
}
```

文件操作部分：

```
//*************************************************
//函数名：fileCreate
//参 数：父亲目录 名字后缀名 等
```

```
//返回值：fcb 指针
//说 明：创建文件
//************************************************
fcb* fileCreate(dir *parent, char fileName[11], char nameLen, char last[5], char
lastLen, myBOOL rw, u16 UserEnable, myBOOL isShortCut)
{
    eqlSizeContainer* q = NULL;
    int fcbBlock;
    fcb *p;
    uInt index;
    //为 fcb 先分配一个空闲数据项
    q = eqlscGet(parent);

    if (q != NULL)
    {
        //设置 fcb 的属性
        fcbBlock = q->num;
        p = (fcb*)q;
        p->fcbBlock = fcbBlock;
        //修改须写回 脏化所在物理块
        isDirtyBlock[fcbBlock] = ture;
        p->rw = rw;
        //大小不包括 fcb
        p->FileSize = 1;
        p->isShortCut = isShortCut;
        p->UserEnable = UserEnable;
        p->directory = parent;

        p->rPoint = 0;
        p->wPoint = 0;
        p->nameLen = nameLen;
        p->lastLen = lastLen;
        copyArray(p->name, fileName, 11);
        copyArray(p->last, last, 5);
        //time
        time_t t;
        time(&t);
        struct tm* localTime;
        localTime = localtime(&t);
        p->createTime_year = localTime->tm_year;
        p->createTime[0] = (char)localTime->tm_mon;
        p->createTime[1] = (char)localTime->tm_mday;
        p->createTime[2] = (char)localTime->tm_hour;
        p->createTime[3] = (char)localTime->tm_sec;
```

```
    //目录的属性设置
    if (parent->fcbHead == NULL)    //目录文件为空
    {
        parent->fcbHead = p;    //初始化为链头
        parent->fcbEnd = p;     //链尾也是
        p->preFcb = NULL;   //第一个文件前后指针均为NULL
        p->nextFcb = NULL;
    }
    else
    {
        //修改须写回 脏化所在物理块
        isDirtyBlock[parent->fcbEnd->fcbBlock] = ture;
        parent->fcbEnd->nextFcb = p;    //接到链尾上
        p->preFcb = parent->fcbEnd;     //向前链接
        parent->fcbEnd = p; //设置为链尾
        p->nextFcb = NULL;  //后指针设为空
    }
    //写回
    sendBackDirDirtyBlocks(parent);
    sendBackDirBlock(parent->parent, parent->dirBlock);

}
else    //返回失败
{
    return NULL;
}

index = BlockGet();
if (index != 10000)
{
    p->firstBlock = index;
    p->lastBlock = index;
    array_base[index]->P_before = blockEnd;
    array_base[index]->P_after = blockEnd;
    //修改第一块第一个字节为0表示字符串结束
    array_base[index]->Data[0] = 0;
}
else    //出现失败,则要回收已分配的物理块
{
    fileDelete(p);
    return NULL;
}
//写回
```

```
    sendBackDisk((char*)array_base[index], index, ture);

    return p;
}
//************************************************
//函数名：fileDelete
//参 数：fcb 指针
//返回值：bool 成功与否
//说 明：删除文件
//************************************************
myBOOL fileDelete(fcb *file)
{

    //删除文件块
    u16 pre = file->firstBlock;
    BLOCK_structure* tmp;
    u16 next;

    //物理块读入内存
    while (pre != blockEnd)
    {
        tmp = (BLOCK_structure*)getDiskBlock(pre);
        if (tmp == NULL)
            return false;

        next = tmp->P_after;
        BlockSendback(pre);
        free(tmp);
        pre = next;
    }

    //更新原来 parent 目录的链接
    if (file->preFcb != NULL)    //若不是链首
    {
        file->preFcb->nextFcb = file->nextFcb;    //前一个 fcb 指向当前 fl 的后一 fcb
        //修改须写回 脏化所在物理块
        isDirtyBlock[file->preFcb->fcbBlock] = ture;
    }
    else    //是链首
    {
        file->directory->fcbHead = file->nextFcb;
        if (file->nextFcb != NULL)
            file->nextFcb->preFcb = NULL;
    }
```

```
    if (file->nextFcb != NULL) //若不是链尾
    {
        file->nextFcb->preFcb = file->preFcb;     //后一个目录指向当前 d 的前一目录
        //修改须写回 脏化所在物理块
        isDirtyBlock[file->nextFcb->fcbBlock] = ture;
    }
    else    //是链尾
    {
        file->directory->fcbEnd = file->preFcb;
        if (file->preFcb != NULL)
            file->preFcb->nextFcb = NULL;
    }

    dir* parent = file->directory;
    //回收 fcb 的数据项
    eqlscSendback(parent, (eqlSizeContainer*)file, file->fcbBlock);
    //父亲目录中被修改的物理块写回
    sendBackDirDirtyBlocks(parent);
    //父亲目录所在物理块写回
    if (sendBackDirBlock(parent->parent, parent->dirBlock) == false)
        return false;

    file = NULL;
    return ture;
}
//************************************************
//函数名：fileRename
//参 数：fcb 指针 文件名 后缀名
//返回值：bool 成功与否
//说 明：文件重命名
//************************************************
myBOOL fileRename(fcb *file, char name[11], char nameLen, char lastName[5], char
lastLen)
{
    copyArray(file->name, name, 11);
    copyArray(file->last, lastName, 5);
    file->nameLen = nameLen;
    file->lastLen = lastLen;
    //写回
    sendBackDirBlock(file->directory, file->fcbBlock);
    return ture;
}
```

I/O 操作部分：

```
//************************************************
//函数名：getDiskBlock
//参 数：获取的物理块号 从 0 开始
//返回值：char 指针 数组长度为 1024 共 1024 字节
//说 明：磁盘文件的物理块分配至内存
//************************************************
char* getDiskBlock(int blockNum)
{
    if (blockNum > 10000)
    {
        return NULL;
    }
    FILE* f = fopen("block", "rb");
    if (f != NULL)
    {
        char* buf = malloc(1024);
        if (buf == NULL)
            return NULL;

        fseek(f, blockNum * 1024, SEEK_SET);
        fread(buf, 1024, 1, f);
        fclose(f);
        return buf;
    }
    else
    {
        return NULL;
    }
}
//************************************************
//函数名：sendBackDisk
//参 数：char 数组指针 物理块号 blocknum 是否清空指针
//返回值：BOOL
//说 明：回收物理块 并将其中信息刷写回磁盘中的文件
//************************************************
myBOOL sendBackDisk(char* buf, int blockNum, myBOOL isDelete)
{
    if (buf == NULL)
    {
        return false;
    }

    FILE* f = fopen("block", "rb+");
    if (f != NULL)
```

```
    {
        fseek(f, blockNum * 1024, SEEK_SET);
        fwrite(buf, 1024, 1, f);
        //撤销 buf
        free(buf);
        if (isDelete)
        {
            //清除指针
            array_base[blockNum] = NULL;
        }
        fclose(f);
        return ture;
    }
    else
    {
        return false;
    }
}
```

GUI 部分：

```
    void menu::show()
    {
        if (filesChanged)   //当filesChanged时，刷新files数组的内容
        {
            files.clear();
            fcb* f = current->fcbHead;
            while (f != NULL)
            {
                fileGUI fl(f, files.size());    //创建新的对象
                files.push_back(fl);
                f = f->nextFcb;
            }
            filesChanged = 0;   //下次不再刷新
        }
        if (dirsChanged)    //同上刷新dirs的内容
        {
            dirs = {};
            dir* d = current->dirHead;
            while (d != NULL)
            {
                dirGUI dt(d, dirs.size());
                dirs.push_back(dt);
                d = d->nextDir;
            }
```

```
            dirsChanged = 0;
        }
        ImGui::SetNextWindowPos(ImVec2(200, 100));
        ImGui::Begin("file system");
        if (ImGui::Button(u8"返回"))
        {
            closeDir(current);
            if (current == root)
            {
                exit(0);
            }
            current = current->parent;
            currentPath.pop_back();
            dirsChanged = 1;
            filesChanged = 1;
        }ImGui::SameLine();
        string sPath = space(2);
        for (int i = 0; i < currentPath.size(); i++)
        {
            sPath += currentPath[i];
        }
        sPath += space(2);
        ImGui::Text(sPath.c_str()); ImGui::SameLine();
        int num = log2(currentUserNum);
        ImGui::Text(("\t用户编号"+to_string(num)+"\t").c_str());
ImGui::SameLine();
        ImGui::Text("\t\t\t\t\t\t\t"); ImGui::SameLine();
        if (ImGui::Button(u8"新建文件"))
        {
            this->showNewFile = 1;  //更新绑定变量
        }ImGui::SameLine();
        if (ImGui::Button(u8"新建目录"))
        {
            this->showNewDir = 1;   //更新绑定变量
        }ImGui::SameLine();
        if (ImGui::Button(u8"切换用户"))
        {
            this->showChangeUser = 1;   //更新绑定变量
        }ImGui::SameLine();
        if (ImGui::Button(u8"粘贴到目录"))
        {
            if (tmpFcb != NULL)
            {
                if(fileMovePart2(current) == 1)
```

```
            filesChanged = 1;   //需要刷新队列
    }
    else if (tmpDir != NULL)
    {
        if (dirMovePart2(current) == 1)
            dirsChanged = 1;    //刷新队列变量
    }
    else
    {
        showTempNULLWindow = 1; //剪切板为空
    }
}
//根据绑定的变量进行指定的操作
if (this->showNewDir)
{
    newDirWindow();
}
if (this->showNewFile)
{
    newFileWindow();
}
if (this->showChangeUser)
{
    changeUser();
}
if (showUserEnableErrorWindow)
{
    userEnableErrorWindow();
}
if (showRwErrorWindow)
{
    rwError();
}
if (showTempNULLWindow)
{
    tempErrorWindow();
}
if (showCopyErrorWindow)
{
    CopyErrorWindow();
}
if (showSameNameWindow == 1)
{
    sameNameWindow();
```

```
        }
        //磁盘使用量的显示
        float progress = (float)usedBlockNum/10000.0f, progress_dir = 1.0f;
        ImGui::ProgressBar(progress, ImVec2(0.0f, 0.0f));
        ImGui::SameLine(0.0f, ImGui::GetStyle().ItemInnerSpacing.x);
        ImGui::Text(u8"磁盘使用率");
        float progress_saturated = ((progress) < (0.0f) ? (0.0f) : (progress) >
(1.0f) ? (1.0f) : (progress));
        char buf[32];

        string s = to_string((int)(progress_saturated * 10000)) + "/" + "10000";
        for (int i = 0; i < s.size(); i++)
        {
            buf[i] = s[i];
            buf[i + 1] = '\0';
        }
        ImGui::ProgressBar(progress, ImVec2(0.f, 0.f), buf);
        ImGui::SameLine(0.0f, ImGui::GetStyle().ItemInnerSpacing.x);
        ImGui::Text(u8"磁盘使用量");

        //渲染fileGUI和dirGUI
        for (int i = 0; i < files.size(); i++)
        {
            files[i].show();
        }
        for (int i = 0; i < dirs.size(); i++)
        {
            dirs[i].show();
        }
        ImGui::End();
    }
```

dirGUI 部分代码

```
    void dirGUI::show()
    {
        ImGui::Spacing();
        string time = space(8) + to_string(dp->createTime_year+1900) + "/" +
to_string(dp->createTime[0]+1) + "/" + to_string(dp->createTime[1]) + " " +
to_string(dp->createTime[2]) + ":" + to_string(dp->createTime[3]) + space(8);
        ImGui::Separator();
        ImGui::Dummy(ImVec2(10, 5));
        ImGui::Text(u8"目录"); ImGui::SameLine();
        int spaceW = 0;
        spaceW = (8 - this->dp->nameLen) * 1.5 + 38;    //计算空格长度
        if (spaceW < 0) spaceW = 38;
```

```
        ImGui::Text((space(12)+ dp->name
+space(spaceW)).c_str());ImGui::SameLine();
        ImGui::Text(time.c_str()); ImGui::SameLine();
        if (ImGui::Button((u8"D选项" + to_string(index)).c_str()))
        {
            ImGui::SetNextWindowPos(ImGui::GetMousePos());
            choose = 1;
        }
        ImGui::Dummy(ImVec2(10, 5));
        if (this->choose)
        {
            chooseWindow();
        }
        if (this->dirOpen)
        {
            //打开目录操作
            openDir(dp);
            currentPath.push_back(string(dp->name) + "/");
            current = dp;
            dirOpen = 0;
            //dirs更改
            dirsChanged = 1;
            filesChanged = 1;
        }
        if (this->showChangeWindow) //更改窗口
        {
            changeWindow();
        }
        if (this->showConfirmWindow)    //确认窗口
        {
            confirmWindow();
        }
    }
```

fileGUI部分代码:

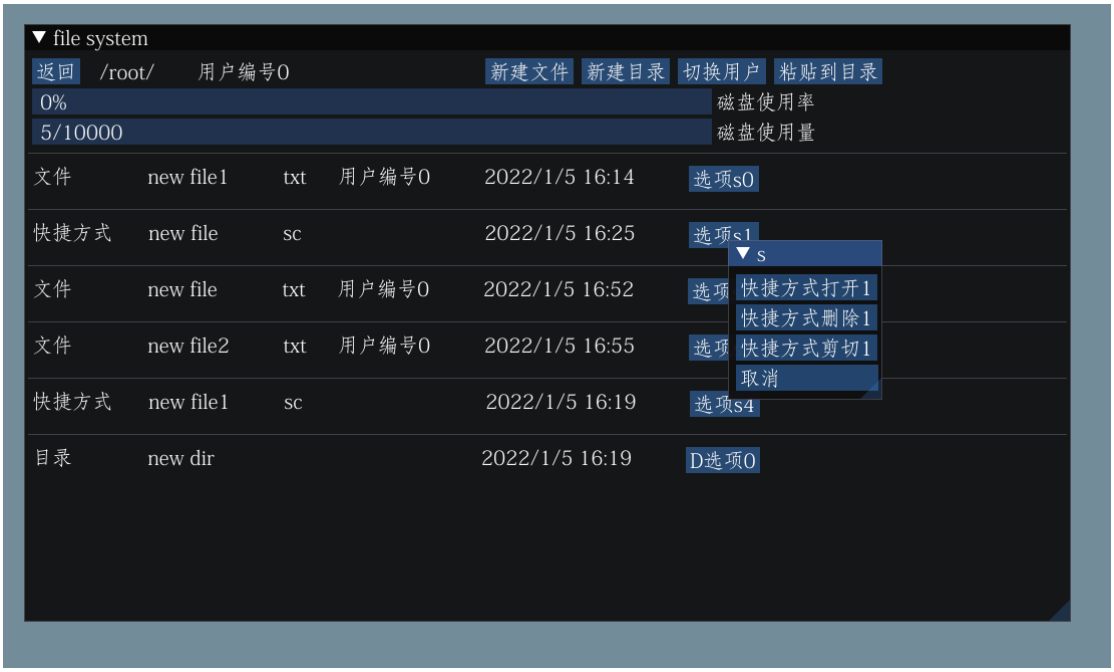
```
    void fileGUI::show()
    {
        if (fp->isShortCut)
        {
            ImGui::Spacing();
            string time = space(27) + to_string(fp->createTime_year + 1900) + "/"
+ to_string(fp->createTime[0] + 1) + "/" + to_string(fp->createTime[1]) + " " +
to_string(fp->createTime[2]) + ":" + to_string(fp->createTime[3]) + space(8);
            int num = log2(fp->UserEnable);
            ImGui::Separator();
```

```
            ImGui::Dummy(ImVec2(10, 5));
            ImGui::Text(u8"快捷方式"); ImGui::SameLine();
            int spaceW = 6;
            spaceW = (9 - this->fp->nameLen) * 2 + 6;
            if (spaceW < 0) spaceW = 6;
            ImGui::Text((space(5) + fp->name + space(spaceW)).c_str());
ImGui::SameLine();
            ImGui::Text((space(4) + fp->last + space(4)).c_str());
ImGui::SameLine();
            ImGui::Text(time.c_str()); ImGui::SameLine();

            if (ImGui::Button((u8"选项s" + to_string(index)).c_str()))
            {
                ImGui::SetNextWindowPos(ImGui::GetMousePos());
                chooseS = 1;
            }
            ImGui::Dummy(ImVec2(10, 5));
            if (this->chooseS)
            {
                chooseSWindow();
            }
            if (this->showConfirmWindow)
            {
                confirmWindow();
            }
        }
        else
        {
            ImGui::Spacing();
            string time = space(8) + to_string(fp->createTime_year + 1900) + "/" +
to_string(fp->createTime[0] + 1) + "/" + to_string(fp->createTime[1]) + " " +
to_string(fp->createTime[2]) + ":" + to_string(fp->createTime[3]) + space(8);
            int num = log2(fp->UserEnable);
            ImGui::Separator();
            ImGui::Dummy(ImVec2(10, 5));
            ImGui::Text(u8"文件"); ImGui::SameLine();
            int spaceW = 6;
            spaceW = (9 - this->fp->nameLen) * 2 + 6;
            if (spaceW < 0) spaceW = 6;
            ImGui::Text((space(12) + fp->name + space(spaceW)).c_str());
ImGui::SameLine();
            ImGui::Text((space(4) + fp->last + space(4)).c_str());
ImGui::SameLine();
            ImGui::Text((u8"用户编号" + to_string(num)).c_str());
```

```
ImGui::SameLine();
            ImGui::Text(time.c_str()); ImGui::SameLine();
            if (ImGui::Button((u8"选项s" + to_string(index)).c_str()))
            {
                ImGui::SetNextWindowPos(ImGui::GetMousePos());
                choose = 1;
            }
            ImGui::Dummy(ImVec2(10, 5));
            if (this->choose)
            {
                chooseWindow();
            }
            if (this->showFileEditWindow)
            {
                editWindow();
            }
            if (this->showChangeWindow)
            {
                changeWindow();
            }
            if (this->showConfirmWindow)
            {
                confirmWindow();
            }
        }
    }
```

7. 程序运行的主要界面和结果截图



快捷方式操作选项

目录操作选项


▼ file system
返回
/root/
用户编号0
新建文件
新建目录
切换用户
粘贴到目录
0%
磁盘使用率
5/10000
磁盘使用量
文件 new file1 txt 用户编号0 2022/1/5 16:14
快捷方式 new file sc 2022/1/5 16:25
文件 new file txt 用户编号0 2022/1/5 16:52
文件 new file2 txt 用户编号0 2022/1/5 16:55
快捷方式 new file1 sc 2022/1/5 16:19
目录 new dir 2022/1/5 16:19
D选项0
▼ f
文件打开0
文件修改0
文件删除0
文件剪切0
创建快捷方式0
取消

文件操作选项


▼ file system
返回
/root/
用户编号0
新建文件
新建目录
切换用户
粘贴到目录
0%
磁盘使用率
5/10000
磁盘使用量
文件 new file1 txt 用户编号0 2022/1/5 16:14 选项s0
快捷方式 new file sc 2022/1/5 16:25 选项s1
文件 new file txt 选项s2
文件 new file2 txt 选项s3
快捷方式 new file1 sc 选项s4
目录 new dir 选项0
▼ 创建新文件
new file
name
9
size
txt
last
4
lastSize
确认文件
取消

新建文件


▼ file system
返回 /root/ 用户编号0
新建文件 新建目录 切换用户 粘贴到目录
0%
磁盘使用率
5/10000
磁盘使用量
文件 new file1 txt 用户编号0 2022/1/5 16:14 选项s0
▼ 创建新目录
new dir name
8 - + size
确认目录 取消
快捷方式 2022/1/5 16:25 选项s1
文件 2022/1/5 16:52 选项s2
文件 2022/1/5 16:55 选项s3
快捷方式 new file1 sc 2022/1/5 16:19 选项s4
目录 new dir 2022/1/5 16:19 D选项0

新建目录


▼ file system
返回 /root/ 用户编号0
新建文件 新建目录 切换用户 粘贴到目录
0%
磁盘使用率
5/10000
磁盘使用量
文件 new file1 txt 用户编号0 2022/1/5 16:14 选项s0
快捷方式 new file sc
▼ 更改用户
1 - + size
确认编号 取消更改
文件 new file txt 用户编号0
文件 new file2 txt 用户编号0
快捷方式 new file1 sc
目录 new dir 2022/1/5 16:19 D选项0

更改用户编号

文件内容修改

文件更改属性

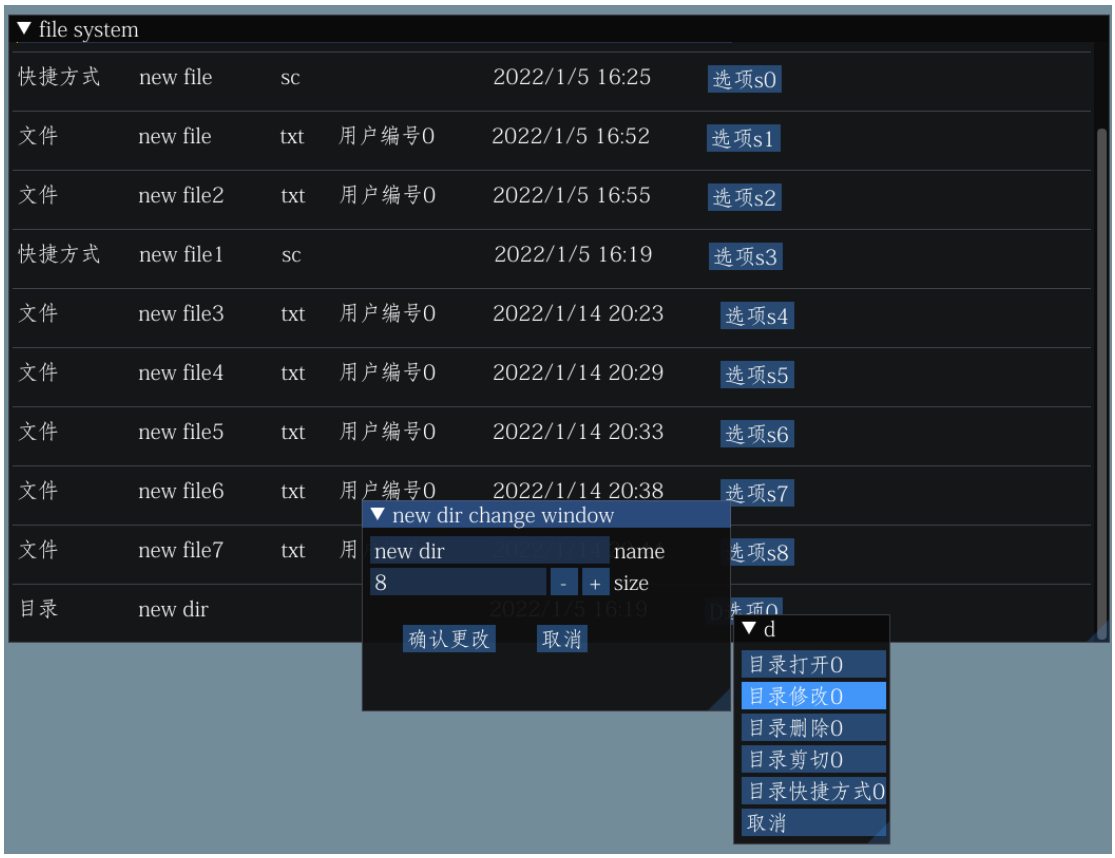

file system
返回
/root/
用户编号0
新建文件
新建目录
切换用户
粘贴到目录
0%
磁盘使用率
4/10000
磁盘使用量
快捷方式 new file sc 2022/1/5 16:25 选项s0
文件 new file txt 用户编号0 2022/1/5 16:52 选项s1
文件 new file2 txt 用户编号0 2022/1/5 16:55 选项s2
快捷方式 new file1 sc 2022/1/5 16:19 选项s3
目录 new dir 2022/1/5 16:19
new file confirm wind
确认
取消

文件删除


file system
返回
/root/
用户编号0
新建文件
新建目录
切换用户
粘贴到目录
0%
磁盘使用率
4/10000
磁盘使用量
快捷方式 new file sc 2022/1/5 16:25 选项s0
文件 new file txt 用户编号0 2022/1/5 16:52 选项s1
文件 new file2 txt 选项s2
快捷方式 new file1 sc 选项s3
目录 new dir 选项0
创建新文件
new file
name
9
txt
4
Size
错误
发生重名错误
更改名字

重名错误

多个文件



目录修改

8. 总结和感想体会

本次课程设计学习了设计一个软件系统的基本步骤，从最开始的确定需求，确定使用的工具与依赖库，确定这个系统有哪些部分组成，目录控制块与文件控制块分开编写，空闲块的管理，到代码的编写，测试，debug 解决各种各样的问

题,一步步下来确实花了不少时间，但确实收获了很多，对链式文件系统的一些优点与缺点也有了更加深刻的了解;整个文件系统从最开始的概念不太清楚到一步步搭建,不断地确定需求也不断地发现新的需求。

通过代码编写，深入了解了文件系统的各类操作，文件和目录的操作，最开始的版本因为没有认真阅读课程设计要求，完成后发现写的是一个基于内存的文件系统，没有经过磁盘的刷写，后来重新确定后发现刷写回磁盘是一个主要难点。

GUI 部分也是第一次用这套 ImGui,最开始不怎么习惯这套编写方式,容易按键按下去没有反应或者只在一帧中显示。最后也是根据示例一步步写出正确的代码。

这次课程设计受益匪浅，我学习到了很多,也总结很多经验。