# MFile文件上传系统设计

成员：曾拉娜 何文涛

## 程序概述

文件上传系统采用CS模式设计，文件系统作为服务端，负责管理和处理文件系统的所有操作。客户端采用QT编写，实现通过自定义消息结构经过网络传输，可视化浏览、管理文件系统。

## 文件系统设计思想概述

本程序设计的文件系统基于在物理磁盘上生成的一个固定大小的文件，在文件上进行区域划分，实现对文件的存储和管理。

本程序设计的文件系统大小为67600384Bytes(64.47MB)，主要划分为3个区域：数据存储块管理区，文件目录区（包括根目录区）和数据存储区。

主要的设计架构如下：



1、数据存储块管理区作用是管理文件系统的每一个物理盘块的数据存储情况，包括注明各个物理盘块的使用情况和物理盘块在存储数据时的连接情况（文件分块存储）。程序定义了一个数据结构FAT，负责存储物理盘块的使用情况，useflag=0表示数据存储块没被占用，可以写入；useflag=1表示数据存储块被占用，不能写入数据；useflag=2表示数据存储块用于存储目录信息。Nextblock用于标记文件在分块存储时的数据内容连接，下载文件时按存储的块内容读出。

Struct FAT

{

int useflag;

int nextblock;

};

2、文件目录区作用是存储了文件系统所有目录信息，用于管理目录间的层级结构，记录目录下的文件信息。程序使用了dirFile结构体存储目录信息，单个目录信息是由定义的FCB结构体负责存储。FCB可以存储目录里的项目信息，包括文件或文件夹。FCB存储了包括文件名/文件夹名、用户名、上传时间、项目类型、文件大小、父目录存储块号、当前存储块号。父目录存储块号记录了当前目录的父目录的存储块号（包括文件夹或文件），当前存储块号则是记录本目录所存储在的数据存储块号（FAT的下标）。要说明的是，每个dirFile的fcb[0]存储的是整个目录的相关信息，主要有本目录项的上一级目录的数据存储块号，本目录里已经存在项目数量(needBlockCount)。

Struct dirFile

{

struct FCB fcb[BlockFcbCount];//能存储128个目录项，每个占64字节

};

struct FCB

{

char fname[18]; //文件名

char time\_upload[18];//上传时间

char sername[8];

char type;

int size; //文件大小

int fatherBlockNum; //当前的父目录盘块号

int currentBlockNum; //当前的盘块

int needBlockCount;//

};

3、数据存储区的作用就是存储文件的数据内容，不包括目录。数据存储区有数据块组成，程序设计的块大小为2048 Bytes（存储2048个字符），共有32688块数据块。数据块的存储情况由数据块存储管理区(FAT)负责管理。

4、整个文件系统的数据存储信息（不包括数据存储区）是由结构体DISK负责存储，存储的内容就如上所说，这里再概述一下。FAT区共管理文件系统的数据存储块32768块，其中前80块每块的大小为8192B，负责记录了文件系统的FAT和目录区。而FAT区共占用32个数据存储块（每块大小为8192B）。

接着是根目录的数据区，由root负责存储，占用1个数据存储块（8192B），能存储128项文件和文件夹信息。

最后的是data数据，每行占用一个数据存储块，能够存储128项文件和文件夹信息，共能存储47个目录(dirFile)。

Struct DISK

{

FAT FAT1[BlockCount]; //FAT1 盘块 0---31

struct dirFile root; //根目录 32

char data[47][BlockSize];//用于存储目录 2048为长度 33-79

}

## 客户端功能描述：

1. **客户端主要的类结构设计**

* MainPage类：

接收用户的点击事件

通过信号触发SocketHandler单例模式对象的处理函数

接受SocketHandler单例模式对象的信号，更新界面，并提示用户操作结果

* SocketHandler类：

接收MainPage类发送的信号，进行相应的请求消息打包

通过Socket成员对象发送打包好的信息、上传文件内容

通过Socket成员对象接收服务器发送的响应消息，或者文件下发

发送信号通知MainPage相应的操作结果

* Socket类：

通过socket连接服务器

发送SocktHandler打包好的信息

根据文件路径读取文件并上传给服务器

接收服务器的响应消息

接收服务器下发的文件内容并根据文件路径存贮到本地

**客户端UML图如下所示**：



**2、自定义消息结构设计**

Server->Client

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | ResType | | State共有的 State特有的 | | | | | | fileId | 备注 |
| response | RES\_NEW\_DIR | 11 | FAIL | 1 | OK | 0 | NO\_SPACE | 2 | 新建的文件夹ID | 新建文件夹 |
| RES\_DEL\_FILE | 12 | NO\_CLEAN | 2 | / | 删除文件或文件夹 |
| RES\_UPLOAD\_FILE | 13 | NO\_SPACE | 2 | 新上传的文件ID | 上传文件 |
| RES\_CHG\_POS | 15 | NO\_SPACE | 2 | 新文件夹下的Id |  |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | resType | | Size | FileID | FileType | FileSize | FileName | UserName | Date | …… | 备注 |
| response | RES\_CHG\_CUR\_DIR | 16 | 文件数量 | 一个文件或文件夹的信息 | | | | | | / | 文件目录 |

Client->Server

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | msgType | | FileId | Size | filename | UserName |
| Request | REQ\_NEW\_DIR | 21 | / | / | 文件夹名 | 上传者名 |
| REQ\_DEL\_FILE | 22 | 删除的id | / | / | / |
| REQ\_UPLOAD\_FILE | 23 | / | 文件大小 | 文件名 | 上传者名 |
| REQ\_DOWNLOAD\_FILE | 24 | 下载的文件id | / | / | / |
| REQ\_CHG\_POS\_START | 25 | 移动文件的id | / | / | / |
| REQ\_CHG\_POS\_END | 26 | / | / | / | / |
| REQ\_CHG\_CUR\_DIR | 27 | 更换目录的id设置 | / | / | / |

## 程序整体架构

文件系统作为server端，处理client端的连接和请求。服务器端分两个线程执行，分别完成接受用户连接和处理客户端请求。



* **文件上传操作**

客户端选择需要上传的文件，先通过自定义消息结构告知服务器端，然后通过网络socket上传文件内容。



* **文件下载设计**

客户端选择需要下载的文件，通过自定义消息结构告知服务器端文件系统。服务器端则获取文件内容，通过网络传输到客户端，客户端将文件保存到本地。



* **文件夹或文件移动设计**

客户端选择需要移动的文件或文件夹，然后定位到需要移动到的新目录，通过自定义消息结构告知服务器，文件系统完成对移动，并将目录信息更新。



## 系统实现的功能及简单使用说明

**功能**：

1、通过网络上传文件，下载文件；

2、可视化浏览网络文件系统，实现目录切换和目录显示，实现文件或文件夹的删除操作；

3、实现文件夹或文件在不同目录之间的移动。

**使用说明：**

MFile文件系统在Linux下运行，文件系统的文件为/home/myfiles，文件系统使用的端口号为8500。服务器启动后，客户端填写好服务端的IP和端口号后，进行连接。

第一次使用时文件系统为空，所以根目录下为空。点击create新建文件夹；upload上传本地文件；download下载选中的文件；move为移动选中的文件或文件夹，然后切换到你想要移动到的目录下点击move完成移动；delete删除文件或文件夹；在客户端的底部可以看到文件夹或文件的相关信息；双击文件夹进入文件夹，点击back实现目录的切换显示。具体可参看测试文档。

退出客户端直接按叉键，退出服务器使用ctrl+c。