FTP实验报告

唐建宇 2017012221

1.提交文件结构

```
server/
    main.c            // 服务器入口
    server.h          // 公共头文件，声明所有函数和全局变量
    process.c         // 接受并分发不同请求
    ftp-commands.c    // 每个请求对应的处理函数
    file_operation.c  // 处理文件操作的函数库
    utils.c           // 用到的其他函数
client/
    main.py           // PyQt5程序入口及槽函数
    client.py         // 封装每一种任务的处理函数
    multi_thread.py   // 多线程类，实现非阻塞用
    dialog.py         // 弹出输入对话框类
    mainWindow.py     // 主界面ui
    inputDialog       // 弹出的对话框ui
```

2.实现功能

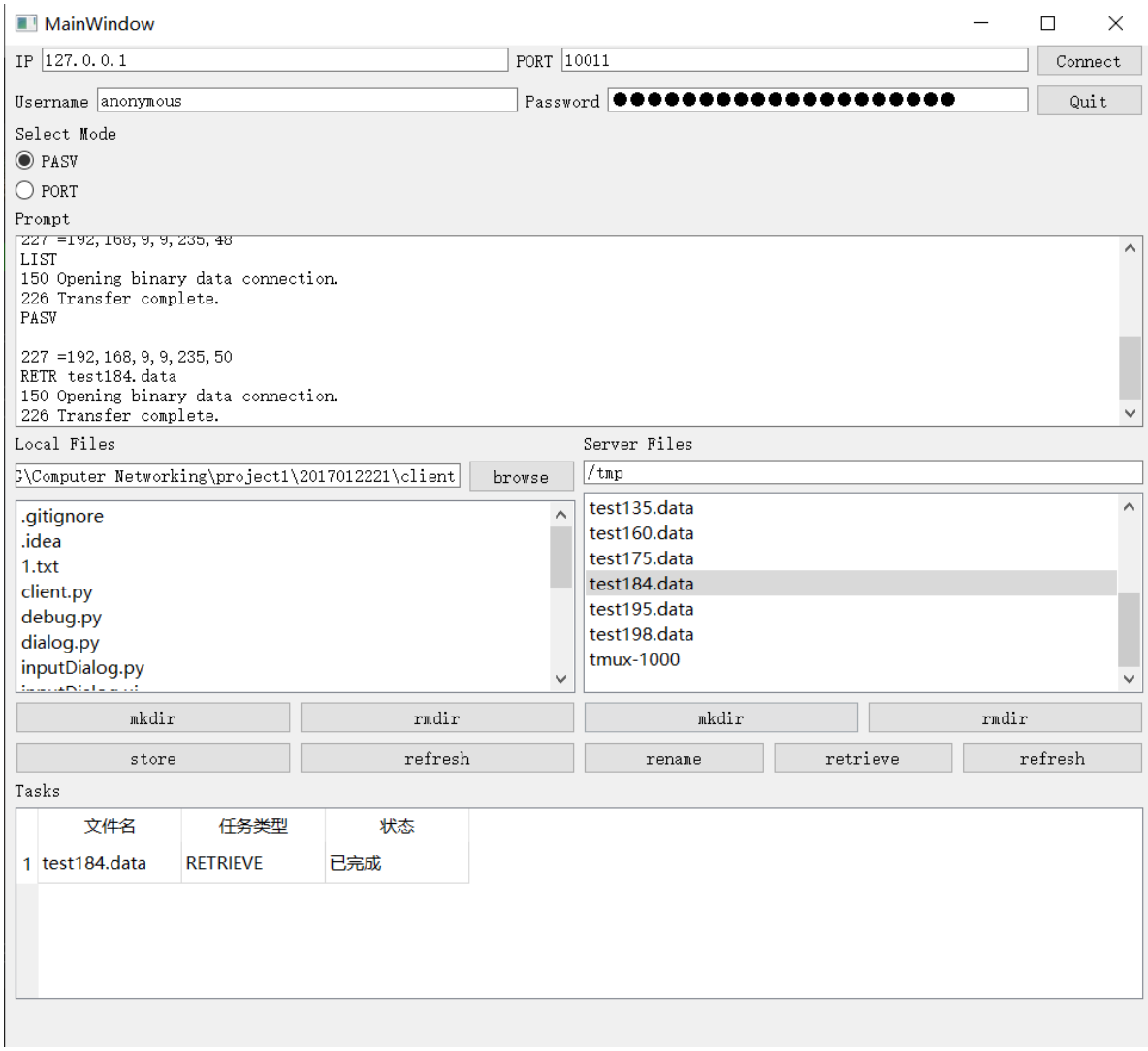
服务端

- 支持 USER, PASS, RETR, STOR, QUIT, SYST, TYPE, PORT, PASV, MKD, CWD, PWD, LIST, RMD,RNFR, RNTO,REST 命令
- 支持多客户端连接
- 传输大文件不阻塞服务器
- 支持上传和下载的断点续传

客户端

- 使用 USER, PASS, RETR, STOR, QUIT, SYST, TYPE, PORT, PASV, MKD, CWD, PWD, LIST, RMD,RNFR, RNTO,REST 命令提供服务
- 支持连接标准ftp服务器，如vsftpd
- 传输大文件不阻塞界面
- 支持断点续传
- 友好的界面

3.客户端图形界面操作指南



如图所示，上方为登录区，输入地址、端口、用户名、密码登录；其下为两种模式的选择；再下方是控制台，显示当前发送及接收的指令；中间是主要的操作区，左右两侧分别显示服务端和客户端列表和操作，点击其下方的按钮即可执行对应任务，其中store, retrieve, rename需要现在文件列表选中文件再点击；最下方是任务栏，显示文件传输任务及其状态。

4.实现方式

服务端

- 采用select函数处理多用户的连接和请求，异步地执行每个用户的每个请求；
- 传输文件时，使用多线程，额外开一个线程进行文件传输，防止大文件阻塞服务器；
- 通过REST命令获取断点以支持断点续传。

客户端

- 基于PyQt5实现图形界面；
- 传输文件时，继承PyQt5的QThread额外开一个线程进行传输，并通过信号机制实时更新界面上的任务栏；
- 通过服务器返回的状态码判断操作是否成功，并用提示对话框提醒用户操作结构。

5.技术难点与解决方案

1) 多客户端连接与非阻塞

根据project guide，实现多客户端的连接有两种解决方案，即多线程和select函数，为了避免可能的线程管理造成的困扰，使程序结构更加清晰简洁，我选择了select方式。这里遇到问题在于，虽然通过select，系统帮我维护了请求的队列，但是处理这些请求还是需要一个一个顺序地处理，与真正的异步还是不同。这在普通的指令上并没有体现出问题，但根据我实际测试发现，传输大文件依然会阻塞服务器。因此，在传输文件时，我额外创建一个临时的线程进行传输，主线程直接返回以处理下一个请求，文件传输完毕后，线程自动关闭，因此也不需要对线程进行额外的管理，保持了代码的简洁性。

同理，在客户端上，一旦出现大文件传输，整个界面就会卡住，直到传输完毕才能操作，这样用户体验必然很差。因此我同样采用了多线程传输文件的方法，同时通过任务栏，提示当前任务的状态。

2) PyQt的调试问题

在PyQt的执行与纯python执行有些不同，主要是对于错误信息的提示，导致程序会直接出现卡死并退出，却不出现任何提示，这给调试增加了困难；后来意识到是因为python不会帮助我们捕捉并提示程序运行中抛出的异常，可能出现异常的操作需要try except进行手动捕捉并输出。同理，在最终给用户的程序中，对于如connect这样的函数，连接失败是很常见的情况，但因为他并不是返回一个代表失败的返回值，而是抛出异常，因此必须捕捉并提示用户，也使程序继续执行不被卡死、退出。

3) 局域网IP地址问题

在调试时，我们自己的