FTP实验报告

1. Server

1.1 命令实现

```
USER, PASS, SYST, TYPE, PASV, PORT, REST, RETR, STOR, APPE, LIST, CWD, PWD, MKD,
RMD, DELE, QUIT
```

- 用户认证：服务器支持用户登录，使用 USER 和 PASS 命令进行身份验证，目前只接受 anonymous的user，密码需要为邮箱格式
- 文件传输：支持文件的上传（STOR）、下载（RETR）、上传追加（APPE）等操作。
- 目录和文件操作：支持创建目录（MKD）、删除目录（RMD）和更改工作目录（CWD）、删除文件（DELE）等功能
- 文件列表：使用 LIST 命令，可以请求服务器列出当前目录中的文件。
- 数据连接模式：支持主动模式（PORT）和被动模式（PASV）的数据传输。
- 退出：QUIT

1.2 支持多个客户端登录

服务器通过指定端口和根目录启动。FTP 服务器采用多线程架构，能够同时处理多个客户端的连接。每当有新的客户端连接时，服务器会fork一个新的子进程来处理该连接，而主进程则继续监听其他客户端的请求，因此会有多个子进程处理client, 一个父进程处于阻塞状态

1.3 传输文件不阻塞服务器

接收到`RETR, STOR, APPE, LIST`指令并且连接数据连接后，`fork()`一个进程进行数据传输。同时，为了方便管理数据传输，把当前连接状态，连接类型，开始位置，ip, socket等信息存入一个结构体进行操作。

1.4 恢复中断传输

- 下载方面，通过`REST`设定重传点，然后`RETR`可以从重传点的位置进行重传，
- 上传方面，通过`LIST`查看上次上传的文件情况，再通过`APPE`操作可以进行重传

1.5 其他难点和思路

- 大小端转换：与python的socket连接不上，查阅资料发现要用`htons`函数 进行转换
- 文件权限控制：
    - 在处理 `RETR, STOR`和`APPE`时， 服务器会检查所有路径名中是否包括../和./，如果包括则会拒绝请求
    - 在处理 `CWD, MKD, RMD`时，遇到../和./的情况，使用 `realpath` 函数解析路径并返回其绝对路径，服务器会使用 `strncmp` 函数比较解析后的路径与根目录的前缀，确保它在根目录下，并且该目录存在。如果路径不合法或目录不为空，服务器会返回错误消息。

2.客户端

2.1 命令实现

```
USER, PASS, SYST, TYPE, PASV, PORT, REST, RETR, STOR, APPE, LIST, CWD, PWD, MKD,
RMD, DELE, QUIT
bye, cd, close, delete, get, help, lcd, lls, login, lpwd, ls, get, mkdir, mput,
open, put, pwd, quit, reget, reput, rmdir
```

客户端实现的命令主要包括各种FTP的VERB和linux类似的命令，

- 登录: 连接服务器的IP和端口后，可以发送USER，PASS指令登录, 也可以发送`login <ip> <port>`命令进行登入
- 文件操作：有些命令对于服务器进行操作，有些命令可以对客户端进行操作。VERB的对应功能类似于server, 此外还可以可以通过`cd, pwd, mkdir, rmdir, delelte, ls`等命令对服务器的路径和文件进行操作（跳转，当前路径，增加，删除，重命名）。在客户端工作路径上的操作对应为`lpwd, lls, lcd`,
- 帮助：`help`命令列出已经实现的各种命令，`help <command>`可以显示更详细的信息
- 退出与打开：`QUIT`和`close`关闭与当前服务器的传输，但仍在ftp环境中，`open <ip> <port>`可以对另一个服务器进行连接，`quit`和`bye`则是完全退出ftp环境

2.2 传输文件不阻塞客户端

`get, reget, mget, put, reput, mput`都可以在不阻塞客户端的情况下进行文件传输，即以上命令未完成时，客户端仍能够处理来自用户的其他命令。在客户端的 main 函数中，创建了一个 ThreadPoolExecutor 实例，最大工作线程数设置为 7，这个线程池允许同时处理多个文件传输任务，而不会阻塞主线程。每当用户输入一个文件传输命令时，客户端都会使用executor.submit 将方法及其参数提交给线程池

2.3 恢复中断的传输

- 下载中断（`reget`）：调用`ls`获取远程文件的信息，包括文件大小和最后修改时间，然后使用获取本地文件的大小和最后修改时间。如果本地文件<远程文件，并且本地文件的修改时间晚于远程文件的修改时间，则可以进行断点续传。调用`REST <local_size>`设置续传位置为本地文件的大小, 然后使用`PORT/PASV+RETR`进行重传。
- 上传中断（`reput`）：调用`ls`获取远程文件的信息，包括文件大小和最后修改时间，然后获取本地文件信息。如果本地文件>远程文件，并且本地文件的修改时间早于远程文件的修改时间，则可以进行断点续传。设定重传位置为远程文件大小，然后使用`PORT/PASV+APPE`进行重传

2.4 其他难点和思路

- 服务器回复的异步问题: 在执行 `RETR` 命令时，客户端需要从服务器接收文件数据并处理服务器的响应。在处理数据传输时，客户端可能会等待来自服务器的响应，而此时服务器已经返回了错误消息，后面不会再有有关传输结果的消息。为了避免程序在接收到错误响应后卡住，应在遇到 550 错误时立即关闭数据连接并返回错误。这确保了客户端能够及时响应错误，而不会继续等待数据传输。
- 设置响应的超时: 在`read_response`方法中，我设置了 socket 的超时，以防止客户端在等待服务器响应时无限期阻塞。通过设置超时，可以确保客户端在一定时间内未收到响应时能够进行适当的处理，例如重试或返回错误。

3.总结

本次实现的 FTP 服务器和客户端功能完整，能够满足基本的文件传输需求。通过多进程架构，服务器能够高效地处理多个客户端的请求；通过多线程架构，客户端可以不阻塞地处理各种用户的输入。并且实现了方便快捷的文件断点续传。后续可以考虑进一步优化性能和增加更多功能，如支持更复杂的文件操作和更安全的认证机制。