计算机网络 套接字编程报告

吴梓建 2020011319

一、实验环境

1、server：Ubuntu 22.04.1（VMWare）；C

2、client：Windows 11； Python

二、服务器搭建

1、服务器可接收的基本指令

指令 <参数>	功能	实现方式
USER <username>	输入用户名	记录 username，通过 checkUser()检查用户名是否存在
PASS <password>	输入密码	记录 password，通过 checkPass()检查密码与用户名是否对应
SYST	返回版本	返回对应信息
TYPE <type>	指定传输的数据类型，但本次仅支持二进制传输	返回对应信息
PORT <ip, port>	指定主动模式传输	根据客户端输入提取出 IP 地址和端口并记录
PASV	指定被动模式传输	服务器记录随机端口，创建监听套接字，并返回 IP 和端口
RETR <file>	从服务器上下载文件	服务器根据端口信息创建文件传输套接字，与客户端连接后以二进制模式将本地文件传输
STOR <file>	向服务器上传文件	服务器根据端口信息创建文件传输套接字，与客户端连接后以二进制模式接收客户端传来的文件
LIST	列出工作目录下的文件和文件夹	服务器根据端口信息创建文件传输套接字，与客户端连接后逐一读取工作目录下的文件
MKD <dir>	创建文件夹	检查当前工作目录，若无同名文件夹，在当前工作目录下创建文件夹

CWD <dir>	修改工作目录	检查当前工作目录，若存在该路径，则修改工作路径为当前工作目录+指定路径（不支持“../”）
PWD	显示当前工作目录	返回工作目录
RMD <dir>	删除文件夹	检查当前工作目录，若存在有权修改的对应文件夹则删除
RNFR <dir>	修改文件/文件夹名（前）	检查当前工作目录，若存在有权修改的对应文件则记录地址
RNTO <dir>	修改文件/文件夹名（后）	访问记录的地址，修改文件名
QUIT	退出程序	停止接受指令，断开连接

2、主要难点

文件传输套接字的创建和连接。FTP 支持主动传输和被动传输，尽管文档中将二者描述为相似的行为，但在实际构造服务器的过程中需要采取不同操作（文档对 PORT 进行了一定讲解，对 PASV 则没有太多涉及，但 PASV 在服务器构建中更加复杂）。主动传输下，服务器先构建对应的套接字，当客户端发出文件传输指令时，服务器先发送 150 开头的返回信息，再尝试连接该套接字；被动传输下，服务器需要先构建监听套接字，向客户端提供 IP 和端口，再在客户端发送文件传输指令后进行连接，等待客户端连接上以后才响应（根据提供的 FTP server 测试应该是这样）。次序不同在服务器构建上造成了许多问题，也导致调试时多次出现客户端无法连接服务器的情况。

信息的接收和发送。信息收发的各类操作出现了很多问题，尽管客户端发送的正确报文都是类似的格式，但在手动调试时难以保证每次输入的都是合理的数据，因此调试过程中会经常出现各类错误，例如 PORT 指令后接的 IP 和端口格式不规范导致服务器无法处理、在 PORT 模式设置失败后直接运行 RETR 导致无法创建套接字等。此外，RETR 和 STOR 指令下，服务器会连续发送两条回复信息，也常出现服务器在两条信息之间卡死，导致客户端卡死的情况。对于地址和大文件处理等，采用了指针运算，指针地址变化也导致过指针无法释放的问题。信息接收和发送看似比较简单，但需要处理各类异常情况的话还是很复杂的。

三、客户端搭建

1、 开发思路

由于客户端允许采用不同语言，故可以使用相对比较简单的 Python 开发，也不再需要运行虚拟机进行调试。同时，服务器的开发过程加深了我对 FTP 传输的理解，在客户端开发时也更加得心应手。考虑到客户端实际需要处理的信息较少，将指令根据返回信息分为 6 种情况：

- PORT：主动模式下客户端需要进行类似服务器的操作，创建监听套接字并在发送文件传输指令后等待服务器连接；
- PASV：被动模式下客户端需要记录服务器发送的 IP 和端口信息，便于创建文件传输套接字连接；
- RETR：客户端根据模式选择连接还是等待连接文件传输套接字，待收到服务器准备信息后开始下载文件，文件下载完毕后立刻关闭连接，之后输出服务器返回信息；
- STOR：处理方式类似 RETR，但改为上传文件；
- QUIT：接收到服务器响应后停止接收指令，关闭连接；
- 其他：一问一答，同时监测回复是否为 223 登录成功，登录成功时才允许处理前述 QUIT 外的指令。

2、 主要难点

客户端开发的主要难点在于与服务器的协同。客户端不需要像服务器一样处理大量信息，但客户端的响应模式需要服从于服务器运行（如文件传输时主动模式和被动模式的响应），其运转逻辑高度依赖于服务器的回复。我在开发时选择先开发一个简单的一问一答式客户端，以调试服务器；后续发现客户端对不同指令的响应需求，便在此基础上增加条件判断和参数。由于客户端对服务器的依赖，在客户端调试过程中不容易判断问题所在，多次出现客户端由于没有及时应答服务器的阻塞函数而卡死的情况。为了弄清楚客户端的错误来源，有必要看到服务器内部的运转逻辑，因此客户端开发也依赖于服务器开发。

此外，由于实验难度较大且任务繁重，本次实验仅完成了基础要求。以前曾使用 MFC 进行 C++的 GUI 开发，Python 的 GUI 开发其实比较简单，很遗憾本次未能实现，希望之后有机会可以尝试一下。

最后，感谢实验中为我提供指导和解惑的老师、助教和同学们！