FTP阶段1实验报告

FTP命令实现

实现了题目要求中”USER”, ”PASS”, ”RETR”, ”STOR”, ”QUIT”, ”SYST”, ”TYPE”, ”PORT”, ”PASV”, ”MKD”, ”CWD”, ”PWD”, ”LIST”, ”RMD”, ”ABOR”,”RNFR”, ”RNTO”等命令。

在此基础上，为了方便完成第二阶段的端点续传、UI界面，补充实现了”REST”,”APPE”,”STAT”,”DELE”命令（STAT必须带一个文件名作为参数，用于获取远程文件的信息）。

使用说明

首先进入server/src文件夹，运行

```
sudo ./server -port 2000 -root /tmp
```

其中-port指定了控制连接的端口号，-root指定了将哪个文件夹用于客户端的根目录。然后进入client/src文件夹，按照client/doc中的说明运行客户端。

代码说明

以下列出核心代码的路径和功能。

```
.
├── client
│   └── src
│       ├── client.py：实现ftp客户端的核心逻辑
│       ├── cmd.py： ftp的命令行程序
│       └── util.py：在实现ftp客户端时常用的函数库
├── server
│   └── src
│       ├── Makefile
│       ├── handle.c： 对不同的ftp命令进行处理
│       ├── handle.h
│       ├── server：ftp服务器程序
│       ├── server.c： 启动ftp服务器，并开始等待服务器的连接
│       ├── util.c： 在实现handle.c时常用的函数库
│       └── util.h：ftp服务器常用结构体和函数的声明
```

└—— udp

困难点与创新点

对多用户的支持

采用多进程的方式。在主进程中监听客户端的请求，为每个请求分配一个新的进程。由于每个登录请求只会被分配一个进程，因此在进行文件传输时，不会响应客户端新的控制命令。而其余客户端对应着其他进程，因此它们的控制命令和数据传输不会受到影响。

目录跳转时路径的处理

由于RETR、STOR、CWD、PWD、LIST、RMD等命令中均涉及到文件（夹）路径的问题，因此在util.c中实现了一系列与路径转换相关的函数。其核心思想如下：

i. 对同一个文件路径，在服务器看来它的路径名为SP,在客户端看来它的路径名为CP。
ii. 设远程用户登录时的根目录，在服务器看来它的路径为root。
iii. 若SP是相对路径，则SP=CP。
iv. 若SP是绝对路径，则SP=root+CP。（”+”表示字符串的连接）

在CP转换为SP后，就可以很方便的调用操作系统提供的函数库进行文件的读写。而将SP转换为CP后，就可以向客户端提供文件信息。

随机端口的生成

在PASV模式下服务器需要随机生成一个端口号，但是不同用户在登录时，有极低的概率会出现端口的冲突。为了防止这样的情况出现，在util.c中实现了is_port_used方法，判断某个端口是否被占用，如果已被占用，则端口号会重新随机生成。

访问权限的控制

为了防止客户端通过”cd ..”等操作访问到服务器的其他文件夹，需要在路径跳转时进行判断。实际实现中采用了一种非常简单的方式：

a. 跳转到客户端指定的文件夹；
b. 设客户端根目录在服务器上的绝对路径为P1，当前文件夹在服务器上的绝对路径为P2；
c. 判断P1为P2的前缀是否满足，如果不满足，则自动跳转回客户端的根目录。

即使用户在根目录下输入类似”cd dir1/../..”之类的命令，也能很好地防止其访问到服务器的其他文件夹。

断点续传的实现

为了实现断点续传，在服务器端增加了REST、APPE、STAT等命令。过程如下（只列出思路，忽略了文件大小的校验）。

当客户端进行断点下载时：

i. 客户端向服务器发送REST命令；
ii. 服务器根据REST命令得知该从文件的哪个位置开始读取数据；
iii. 客户端发送RETR命令，开始文件下载。

当客户端进行断点上传时：

i. 客户端向服务器发送STAT命令，获得服务器上相应文件的信息，从中提取出文件大小remote_file_size；
ii. 客户端将文件读指针指向remote_file_size；
iii. 客户端发送APPE命令，开始文件的上传。

客户端逻辑与交互界面的分离

为了阶段2更快地进行UI界面的设计，将客户端核心逻辑抽离出来（client.py)。

命令行程序（cmd.py)进行的工作主要是对用户输入进行解析，然后映射到客户端的相关方法。