FTP实验文档

姓名: 游凯超

学号: 2016013234

班级:软件61

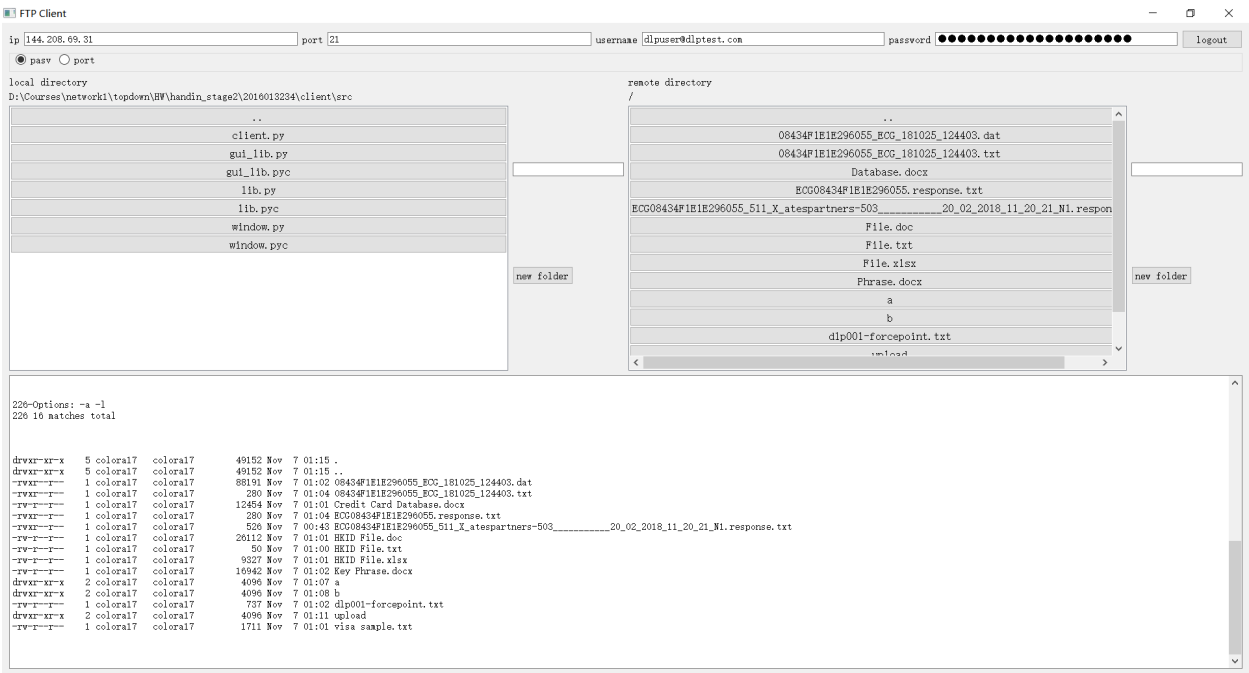
1. 实现的功能

server：

1. 能够接受USER, PASS, RETR, STOR, QUIT, SYST, TYPE, PORT, PASV, MKD, CWD, PWD, LIST, RMD,RNFR, RNTO,REST,APPE命令
2. 能够同时服务多个客户端
3. 能够传输大文件
4. 能够实现文件断点续传（REST命令和APPE命令）

client:

1. 能够与服务器进行连接，发送USER, PASS, RETR, STOR, QUIT, SYST, TYPE, PORT, PASV, MKD, CWD, PWD, LIST, RMD, RNFR, RNTO命令
2. 能够上传/下载FTP服务器上的文件
3. GUI界面



上方是输入服务器的信息，登录后左侧显示当前目录中的文件/目录，右侧显示服务器端的相关信息。最下方有输出窗口，所有的命令记录及其输出都显示在这里。点击文件按钮即可下载/上传文件，点击目录按钮即可进入该目录。在输入框输入文件夹名称，点击"new folder"即可创建文件夹。

上方还有单选框来指定数据传输的模式。

1. 代码结构简介

client方面:

GUI整体基于PyQt5实现。

```
|_client.py             GUI入口文件
|_cmd.py                命令行的client
|_window.py             窗口设计文件,涉及窗口的布局
|_lib.py                心库文件，包装FTP命令给外部使用(被gui和命令行程序复用)
|_gui_lib.py            gui的一些库函数集合,包含目录树视图（DirectoryView）,登录/登出函数槽
```

server方面:

```
|_common.h        头文件
|_global.c        全局变量的定义区
|_handlers.c      各种命令的处理函数
|_lib.c           辅助函数
|_server.c        主文件，创建服务器，监听并派生线程
|_Makefile        Makefile
```

2. 实现方式简介

client: lib.py 采用命令分发的方式，命令被抽象成一个统一的类，有一个invoke函数可以调用，出错则返回Exception对象，否则返回数据。 在命令行模式下， 程序读取用户输入，查找合适的命令，调用其invoke方法，完成一次执行。

server: 监听来自client的连接，每收到一个连接就派生出一个线程来处理本次所有消息。

3. 难点

3.1 client如何判断server的回复已经结束

server接受client的命令时，一个命令就是一行，很容易判断。但client接受server的命令的时候，发一个命令过去，得到的回复可能是多行。查阅文档知，以"xxx-yyy"开头的回复，不代表回复结束。"xxx yyy"才是代表一次回复结束。所以，对于client来说，应该直到一次读取以\r\n结束且这一行是以xxx+ 空格开始。

3.2 如何判断一次连接已经结束

在python上，接受消息的代码为

```
import socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((ip, port))
sock.recv(n)
```

关键在于其中的`sock.recv(n)`,这一行是否是阻塞的？如何判断本次连接已经关闭？

经过自己动手实验后，我发现：

1. 如果本次连接已经关闭，`sock.recv(n)`立马返回，且返回值为0
2. 如果本次连接还没有关闭，`sock.recv(n)`从缓冲区中尝试着读n个字节的数据。不足n个则直接返回读到的所有数据。**如果当前没有数据可以读，则阻塞直到有数据。**

这里和文件读写不一样。普通的文件读写，如果我们要求读1024字节，即调用`read(fd, buffer, 1024)`，如果返回的值小于1024我们就可以判断文件已经读完了。之所以是这样，是因为文件是一个确定大小的buffer，而TCP连接是一个字节流，没有开始和结束的概念，只有当连接被关闭了，你才读不到数据了。

不过在Linux上，socket相关的有一个是WAITALL flag, 这样，`read`不读取到要求的size是不会停止的，这样子处理一下，socket看起来就和普通的文件流完全一样了。但是这并不适用于服务器，因为一个命令可能小于1024字节，如果直接阻塞地读取1024个字节，那么就无法正确地处理命令了。

如果我们统一一下对文件的读写标准，即使最后一次读到了末尾，只读到了不到1024个字节，我们再尝试着读一下，这样返回的肯定是0字节。于是，我们可以把文件读完了和数据连接的数据传送完了统一为`read(fd, buffer, 1024) == 0`.

3.3 端口占用没有及时释放

在Linux上运行server，第一次运行结束之后用Ctrl+C关闭，第二次再用同样的命令运行，发现上一次的程序还没有释放端口。看来Linux并不会在程序运行结束之时主动释放端口，有一定的延时。所以，我们可以为server加入对SIGINT信号的处理函数，接受到Ctrl + C的信号的时候手动关闭server端的监听socket以释放资源。

3.4 与python的socket连不上

助教给的代码，直接运行autograde脚本，发现二者没有通讯。查阅资料知，这是因为网络上端口号用的大小端表示和机器不一样导致的。即端口号需要用`htons`函数来进行转换。

3.5 编写socket相关代码时的调试方法

C代码当然是用gdb来调试。但是server需要client来连，client怎么模拟呢？在我的开发过程中，用telnet来模拟客户端发起连接，用nc来监听连接（用于port模式时）。这样，我避免了用C再写一个client并且开两个gdb来调试的尴尬。

另外，张洋同学给我分享了一个公共的FTP server，极大提高了我的开发效率。该server的信息为：

ip : 144.208.69.31 port:21 username: dlpuser@dlptest.com password: e73jzTRTNqCN9PYAAjjn

3.6 server中拒绝对root上级目录的访问

出于安全性考虑，server只能让client访问root下的内容。故对于命令中含有的文件名参数，在代码实现中，我先将这个文件名的绝对路径构造出来，再确保root是这个路径的子串。这样，就确保了用户的操作都是在root目录之下的。代码见server下lib.c中join_path函数。

3.7 获取本机ip

给公共DNS 8.8.8.8 发送一个UDP包，根据其地址来确定本机ip，并缓存下来，下次再需要获取本机ip时直接使用。