

实验八、Socket 编程实验报告

组号: _____

姓名: _____ 学号: _____ 班级: _____

姓名: _____ 学号: _____ 班级: _____

一、 实验名称

- 实现一个简单的聊天程序；
- 实现基于 TCP 协议的 FTP 客户端和服务器程序。

二、 实验原理

客户端通过 Socket 模块将请求信息发给服务器，服务器在本地组织好数据后，再使用 Socket 模块将响应数据发回给客户端。

三、 实验目的

- 掌握 Sockets 的相关基础知识，学习 Sockets 编程的基本函数和模式、框架；
- 掌握 UDP、TCP 协议及 Client/Server 和 P2P 两种模式的通信原理；
- 掌握 Socket 编程框架。

四、 实验内容

1. 聊天程序

基础功能包括验证用户登录、用户间文字聊天、用户间传输文件；高级功能包括用户注册、保存历史聊天记录、数据传输加密、在线用户群聊，多线程处理、利用 Tkinter 模块进行 GUI 开发。

2. FTP 程序

基础功能包括用户和口令的认证、目录的各种操作（进入、创建、删除、列表）、文件的各种操作（上传、下载、追加、更新、重命名、删除）；高级功能包

括 NAT 穿透、主动和被动模式、断点续传、多文件的删除。

五、实验实现

1. 人员分工

2. 聊天程序实验设计

(一) 协议

本网络聊天程序依赖的传输层协议是 TCP，采用 C/S 模式，服务器端能够并发响应来自多个客户端的请求。然而 TCP 是流式协议，会将多次连续发送数据量小，并且时间间隔短的数据一次性打包发送，导致客户端之间对于消息的分割会有很大的问题。

为了解决该问题，需要对客户端每次发送的消息长度进行描述，保证客户端在接受到任意一次消息时就知道此次的消息报文长度，以便客户端能从字节流中分割出该消息内容。本程序使用 python 中的 `struct` 模块来将每次的消息报文定义为报头（存放报文长度）和数据部分（存放真实消息内容），用 `struct.pack('>H', len(data)) + data` 来封装报文，进行打包压缩传输；用 `struct.unpack('>H', socket.recv(2))[0]` 解析出报头，获取此次消息的真实长度，然后使用 `socket.recv()` 函数来获取消息的全部真实内容。

在消息发送交互过程中服务器充当中转站，服务器端将自己接收到的 `data` 字节流划分为字典，能够很精确地捕捉到客户端之间交互的具体请求和应答，根据对应指令对相应的 `socket` 链接对象进行相应的操作，而对于客户端登录、注册及在线等状态转换过程本程序具体用相应的函数来封装，根据服务器端所接收的状态反馈来及时地调用相关函数来进行更新。

(二) UI 设计

本网络聊天程序的客户端部分为用户提供了可视化界面界面，基于 python 中

的 `tkinter` 模块搭建，含有账号输入框、密码输入框、登录按钮、注册按钮。登录界面由图 8-1 所示，聊天界面由图 8-2 所示



图 8-1 登录界面



图 8-2 聊天窗口界面

(三) 框架结构

服务器端采用 `socketserver` 的 `BaseRequestHandler` 类，由本程序对该类的重定义和若干处理函数构成，使用 `socketserver.ThreadingTCPServer('0.0.0.0', 8888), Handler`) 可自动处理并发请求，循环地从半连接池中取出链接请求与其建立双向

链接，拿到链接对象进行通信循环，从而实现并发响应，即每有一个客户端请求连接时，都会 new 一个 `BaseRequestHandler` 类对象，然后在一个线程中处理相关请求，而在这个过程中服务器将 ip 地址半连接池设置成 0.0.0.0 表示任意地址，相当于客户端与服务器端的 `socket` 连接只绑定端口。若干处理函数被 `socketserver` 的 `BaseRequestHandler` 类重写方法 `handle` 所调用，致使服务器端在接收到命令指令后能处理登录请求、注册请求、获取所有已登录用户的列表、获取连接中的用户与其他用户的聊天记录、将连接中的用户的的消息发给其期望接收的用户、将连接中的用户的发送文件请求发给其期望接收的用户。

客户端使用了 `tkinter` 模块，具有可视化界面，在程序实现过程中，用若干处理函数实现具体功能，而将不同的过程按照 `tkinter` 模块划分为若干事件，结合 `tkinter` 模块用具体函数封装来实现，包括有登录按钮点击事件：当登录按钮点击时向服务端请求登录，如果登录成功则关闭登录页面，开启聊天页面；注册按钮点击事件：当注册按钮点击时向服务端请求注册，得到回应后显示回应的消息（注册成功或注册失败、账号已存在等消息）；发送按钮点击事件：进行客户端之间的消息通信；发送文件按钮点击事件：客户端之间进行文件传输。而处理函数有刷新所有已登录用户列表函数：当开启聊天页面或收到服务端发来的新用户登录/登出的消息时刷新用户列表；将聊天记录加入聊天记录显示框函数：当用户刚登录时显示 Public 聊天聊天记录，当用户点击其他用户与其一对一聊天时显示与其的聊天记录，当点击用户列表中的某用户时，显示与其一对一聊天的窗口；接收服务端消息函数：该函数运行在一个独立的线程中，不断接收服务端发来的消息。

在客户端与服务器端进行发送报文时，最重要的是 `utils` 工具程序文件，在该程序中实现了对 `socket` 链接对象进行发送消息和接收消息，并且也实现了对消息的加密和解密，分割和打包，是客户端和服务器端程序实现的基础。

3. 聊天程序关键代码描述

（一）消息加密解密

为了方便实现，本程序采用对称加密算法 AES，用 `key` 对 `data` 加密，则用同一个 `key` 对密文解密。`encrypt` 函数对数据进行加密，在该函数中首先随机生成长度等于 AES 块大小的不可重复的密钥向量 `code`，使用秘钥 `key` 和密钥向量

`code` 以及 `MODE_CFB` 模式来初始化 AES 对象，在传输时将密钥向量 `code` 加到加密后的密文前面一起传输。在加密时，明文长度必须为 16 的倍数，如果长度不为 16 的位数则就需要补足为 16 的倍数。

使用 `decrypt` 函数对数据进行解密，解密的过程使用与加密过程使用的相同秘钥来解密。对于加密解密的实现都已封装在 AES 模块，可以直接调用使用。

（二）登录验证及注册

登录过程需要从文件中加载所有已注册用户的账号和密码对应的 MD5 值信息调用处理函数 `validate` 进行认证，验证用户账户密码的 MD5 值是否和文件中的值相同，如果不相同则将无法登录，需要调用处理函数 `register` 进行用户注册，注册后用户的信息将保存到文件中，能够在下次直接登录。对保存用户信息的文件内容是自动生成的，类似于数据库，但与数据库不同的时该文件是不可见的，用户无法打开，只能用作登录时进行验证。

（三）Socket 连接发送及接收数据

对于要发送的数据及接收的数据是采用 `struct` 模块的 `pack` 和 `unpack` 函数进行打包和解析，将数据报文定义为报头和数据部分，以用来解决 TCP 传输过程中的粘包问题。发送数据 `send` 函数及接收数据 `recv` 函数是客户端和服务器端各功能实现的基础，承担了管理客户端与服务器端 `socket` 连接通道的任务，保证该通道既能接收到彼此的通信指令，还进行消息传送和文件传输。

发送数据前会在报头加上指明数据大小的一个两字节数，保证服务器端在接收到数据时能够准确分割出不同段的消息报文。接收数据时先接收这个两字节数，获取将要接收的数据包的大小，然后接收这个大小的数据作为本次接收的数据包，这样就防止了不同数据包之间基于 TCP 传输的干扰。

（四）聊天记录管理

本程序对每条聊天记录以 key-value 形式进行保存，其中 `key` 为 (`sender, receiver`)，`value` 为 (`sender, time, msg`)，在这个过程中需要使用 Python 中 `Pickle` 模块的 `dump()` 方法和 `load()` 方法进行序列化的读出和写入打开的文件对象，

`load_history` 函数从文件中加载出目前登录用户与所有用户的所有聊天记录，`append_history` 函数把一条聊天记录存入内存中，`get_history` 函数返回某用户对某用户的聊天记录，`save_history` 函数将所有用户的所有聊天记录保存到文件中。这些函数的都是按照时间顺序进行保存和加载聊天记录。

(五) 登录界面

基于 `tkinter` 模块搭建，含有账号输入框、密码输入框、登录按钮、注册按钮。对于该过程的实现首先需要设立 `tkinter` 对象，进而添加组件来修饰该界面，用 `tkinter.Canvas()` 方法来加载登录界面的背景图片，用账户与密码文字标签 `tkinter.Label()` 方法在界面设有账户和密码文本字样，用账户与密码输入框 `tkinter.Entry()` 方法在界面实现用户能够进行输入用户名和密码，用 `tkinter.Button()` 方法来显示登录和注册按钮，用 `tkinter.place()` 方法来对相关内容在界面中的相对位置进行了控制。为了实现进行登录和注册功能，还需要用 `on_btn_login_clicked()`、`on_btn_reg_clicked()` 方法实现点击登录和注册按钮就进行 `socket` 连接来调用相关方法来完成相应动作，在这个过程中都是 `tkinter` 模块进行配合以提示用户相关操作响应内容。

(六) 聊天窗口界面

基于 `tkinter` 模块搭建，含有其他已登录用户列表显示框、聊天记录显示框、发送消息输入框、发送消息按钮，发送文件按钮。对于该部分的实现和以上功能的实现类似，此处不再重复作详细说明。

(七) 文件传输

由于采用 C/S 交互模式，在客户端之间传输文件时，收发文件都是经过服务器来中转。以客户端 A 向客户端 B 传输文件为例解释本聊天程序文件传输实现的思路及特点，在实现过程中我的构想是以当前客户端的文件夹为执行单位，传输过来的文件会传送到对应客户端的工作目录下，在 public 聊天模式下也会发给自己能够导致冲突，无法做到正常通信，因此我所实现的传输文件都是一对一传输的，此时客户端 A 首先应该在聊天框的在线列表中点击客户端 B 的用户名

进入一对聊天室模式，根据 `on_session_select()` 函数的设置，这时发送文件的按钮属性变为 `normal`，当点下次按钮会在其工作目录及操作系统上的其他目录寻找要传送的文件，点击确认后会率先向服务器发送 `file_request`，服务器接收后会记录下当前文件的大小及文件名和所要传输文件的另一客户端 B 的信息，服务器向客户端 B 发送相关信息来获得客户端 B 的允许确认，在客户端 B 确认后会与服务器的另一端口建立 `socket` 连接来接收文件，而客户端 A 在收到服务器端发来的 `file_accept` 后开始向服务器端传送文件，且服务器端向客户端 B 传输文件，当传输完成后，客户端 A 聊天窗口弹出文件发送成功，客户端 B 弹出文件接收文件成功，到此，整个文件传输过程顺利结束。

4. FTP 程序实验设计

(一) 协议

本 FTP 程序依赖的传输层协议是 TCP。由于 TCP 是基于字节流的通信协议，因此当发送方连续向接收方发送多个包时，接收方在一次收取中收到的数据包数量可能大于 1，这被称为粘包现象。与之相对的，接收方也可能只收到一个数据包的部分，这被称为半包。有关粘包和半包的直观描述见图 8-3 所示。

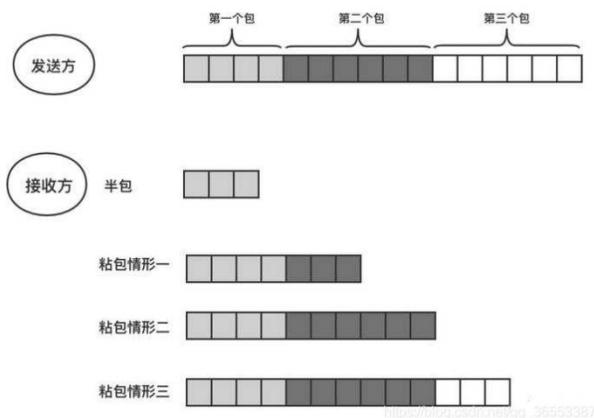


图 8-3 粘包与半包

为解决粘包和半包问题，接收方的上层需要区分出报文边界。基于这个思想，衍生出了三种解决粘包的方法，分别是固定数据包的长度、以指定的字符串为包的结束标志、使用包头+包体格式。值得一提的是，FTP 协议中的控制命令就采用了第二种方法，以 “`\r\n`” 作为控制命令的结尾。

在应用层方面，本 FTP 程序共支持 APPE、CDUP 等 38 条控制命令。客户

端通过组合不同的控制命令来实现不同的功能，服务器则对收到的特定控制命令做出特定的反应。

(二) UI 设计

本 FTP 程序的客户端部分为用户提供了命令行界面，42 种命令的含义和格式如表 8-1 所示。

表 8-1 客户端支持的全部命令

命令	含义	格式
!	run local command	!<command>
append	append to a file	append <localfile> [remotefile]
ascii	set ascii transfer type	ascii
binary	set binary transfer type	binary
bye	terminate ftp session and exit	bye
cd	change remote working directory	cd <remotedirectory>
close	terminate ftp session	close
debug	toggle/set debugging mode	debug
delete	delete remote file	delete <remotefile>
dir	list contents of remote directory	dir [<remotedirectory>] [<localfile>]
disconnect	terminate ftp session	disconnect
get	receive file	get <remotefile> [<localfile>]
help	print local help information	help
lcd	change/show local working directory	lcd [<directory>]
literal	send arbitrary ftp command	literal <argument> []
ls	list abbreviated contents of remote directory	ls [<remotedirectory>] [<localfile>]
mdelete	delete multiple files	mdelete <remotefile>[...]
mdir	list contents of multiple remote directories	mdir <remotefile>[...] <localfile>
mget	get multiple files	mget <remotefile>[]
mkdir	make directory on the remote machine	mkdir <directory>
mls	list contents of multiple remote directories	mls <remotefile>[] <localfile>
mput	send multiple files	mput <localfile>[]
newer	get file if remote file is newer than local file	newer <remotefile> [<localfile>]
open	connect to remote ftp	open <computer>

命令	含义	格式
passive	enter/exit passive transfer mode	passive
prompt	toggles prompt mode on and off	prompt
put	send one file	put <localfile> [<remotefile>]
pwd	print working directory on remote machine	pwd
quit	terminate ftp session and exit	quit
quote	send arbitrary ftp command	quote <argument>[]
recv	receive file	recv <remotefile> [<localfile>]
reget	get file restarting at end of local file	reget <remotefile> [<localfile>]
remotehelp	displays help for remote commands	remotehelp [<command>]
rename	rename file	rename <filename> <newfilename>
restart	restart file transfer at bytecount	restart <position>
send	send one file	send <localfile> [<remotefile>]
size	show size of remote file	size <remotefile>
status	show current status	status
system	show remote system type	system
type	set file transfer type	type [<typename>]
user	send new user information	user <username> [<password>] [<account>]

(三) 框架结构

本 FTP 程序的

客户端程序主要由一个 `shell` 函数和若干底层函数构成。`shell` 函数的主体是一个无限循环，每循环一次处理一条用户输入的命令，处理过程会用到一些底层函数。当 `shell` 函数发现用户输入了无效命令时会提示“?Invalid command”，并不做任何处理。

服务器程序主要由主函数、处理函数和若干底层函数构成。在主函数中，程序会时刻监听 21 端口，并通过循环处理连接到 21 端口客户端。每当客户端发起连接，主函数都会调用一次 `fork`，并令子进程运行处理函数。换言之，服务器程序是通过多进程来支持多用户连接的。

处理函数与客户端程序中的 `shell` 函数较为相似，同样使用了循环，每次处理一条客户端发来的控制命令。当客户端发来无效的控制命令时，服务器会返回

“500 Unknown Command.”，并不做其他处理。此外，处理函数每次循环都会重新设置一个 300 秒的倒计时器，若客户端在 300 秒内未发来任何控制命令，本进程便会收到一个闹钟中断信号。通过 `signal` 函数更改进程对信号的响应方式，实现了服务器超时后自动断开与客户端的连接。

(四) NAT 穿透模块

由于生活中使用的 NAT 大都为 Symmetric NAT，故我选用 TURN 服务器来实现 NAT 穿透，思路是使用一台具有公网 IP 的服务器为两台位于 NAT 之后的主机转发报文。不可避免地，我需要对标准的 FTP 程序做一些修改。例如，服务器程序不能再通过监听 21 号端口来判断是否有客户端请求服务，客户端的主动模式也不能再使用。考虑到测试部分要求使用自编的客户端程序与标准服务器程序通信，因此我将提供两个版本的客户端和服务器程序，分别支持和不支持 NAT 穿透。

5. FTP 程序关键代码描述（点击小标题可跳转到附录代码对应部分，再点击函数名即可返回）

(一) 客户端程序

a. client 类简述

此类的成员函数为 `shell` 函数和若干底层函数，成员变量记录了控制套接字的 FD、断点续传偏移量、客户端网卡名称、服务器地址信息、与服务器的连接状态、传输模式（字符或二进制）、数据传送模式（主动或被动）和多文件操作时是否开启提示。

b. 获得服务器上特定文件的大小

当 `shell` 函数检测到用户输入 “`size <remotefile>`” 时，会调用底层函数 “`ftp_size`”。`ftp_size` 会通过控制套接字向服务器 21 端口发送控制命令 “`SIZE <remotefile>\r\n`”，再通过控制套接字获取服务器返回的信息，并将其显示在屏幕上。

除了 size 命令之外，还有很多其他命令（如 ascii 、 quit 等）也使用了类似的运行机制，此处不再重复做详细说明。

c. 主动模式下创建数据连接

主动模式下，某些底层函数需要用到数据连接时便会调用 data_actv 函数，调用者将会得到监听套接字的 FD，在合适的时候调用者从监听套接字中接受一个连接，便可与服务器的 20 端口建立连接。data_actv 函数会首先创建一个套接字，并将其随机与一个端口绑定（记端口值为 port），并开始监听。之后函数通过控制套接字向服务器 21 端口发送控制命令“PORT a,b,c,d,e,f\r\n”，其中“a.b.c.d”为客户端 IP 地址、e = port / 256、f = port % 256，再通过控制套接字获取服务器返回的信息。若信息状态码为 200，函数则返回监听套接字的 FD；否则返回-1。

d. 被动模式下创建数据连接

被动模式下，某些底层函数需要用到数据连接时便会调用 data_pasv 函数，调用者将会得到已经与服务器数据端口建立好连接的套接字 FD。data_pasv 函数首先通过控制套接字向服务器 21 端口发送控制命令“PASV\r\n”，正常情况收到的反馈是“227 Entering Passive Mode (a,b,c,d,e,f).\r\n”。之后函数创建一个数据套接字，与服务器的 e * 256 + f 端口建立连接，最后将该套接字的 FD 返回。

e. 下载文件

当 shell 函数检测到用户输入“recv <remotefile> [<localfile>]”或“get <remotefile> [<localfile>]”时（<localfile>的缺省值为<remotefile>），会调用底层函数“ftp_recv”。ftp_recv 首先建立数据连接，之后根据断点续传偏移量 offset 的值决定是否向服务器发送 REST，然后向服务器发送“RETR <remotefile>\r\n”，准备从数据连接接收文件。接收文件时，函数同样根据 offset 的值决定是否移动初始文件指针，随后用循环不断从数据连接中接收数据，并将其写入本地文件（若使用 ASCII 传输模式会先将数据中的所有“\r\n”改为“\n”）。数据传输完成后，函数还会输出传输速度等统计信息。

上传文件的过程与下载文件类似，此处不再详述。

f. 更新文件

当 shell 函数检测到用户输入“newer <remotefile> [<localfile>]”时(<localfile>的缺省值为<remotefile>)，会调用底层函数“ftp_newer”。ftp_newer 会首先检测本地是否存在名为<localfile>的文件，不存在则直接执行“get <remotefile> [<localfile>]”；若存在，则通过控制连接向服务器发送“MDTM <remotefile>\r\n”，服务器将返回<remotefile>的修改时间。若<remotefile>的修改时间早于<localfile>的修改时间，则提示“Local file <localfile> is newer than remote file <remotefile>.”，并不做其他任何操作；否则执行“get <remotefile> [<localfile>]”。

g. 列出远程目录的内容

当 shell 函数检测到用户输入“dir [<remotedirectory>] [<localfile>]”时(<remotedirectory>和<localfile>的缺省值均为空)，会调用底层函数“ftp_dir”。ftp_dir 函数会先建立数据连接，再通过控制连接向服务器发送“LIST <remotedirectory>\r\n”，并按照一定的顺序从两个连接中接收信息。对于从数据连接中接收的数据，若<localfile>为空或“-”，则将它们显示在屏幕上；否则函数会创建名为<localfile>的文件，将接收到的数据写入其中。

ls 命令的实现过程与 dir 命令非常相似，仅将发送的控制命令 LIST 改成了 NLST，此处不再详细叙述。

h. 多文件下载

当 shell 函数检测到用户输入“mget <remotefile>[]”时，会调用底层函数“ftp_mget”。ftp_mget 会为<remotefile>[]列表中的每一个元素 filename 建立数据连接，并通过控制连接向服务器发送“NLST filename\r\n”，记录下所有获得正常回应的 filename。对于被记录下的文件名，ftp_mget 会分别为它们调用 ftp_recv，将文件下载到本地。

i. 为支持 NAT 穿透做出的修改

为支持 NAT 穿透做出的修改主要涉及建立控制连接和建立数据连接。

对于控制连接的建立，客户端创建套接字后不再向服务器的 21 端口发起连

接，而是向 TURN 服务器的特定端口发起连接，并发送“CONC a.b.c.d\r\n”，其中 a.b.c.d 是目标 FTP 服务器的公网 IP。若目标 FTP 服务器已经与 TURN 服务器建立连接，TURN 服务器将向客户端发送“250 port\r\n”。客户端收到后会新建一个套接字，连接到 TURN 服务器的 port 端口，之后要发向 FTP 服务器 21 端口的信息都发到 TURN 服务器的 port 端口，TURN 服务器将完成转发工作。

对于数据连接的建立，TURN 服务器会禁止使用主动模式。当客户端向 TURN 服务器的 port 端口发送“PASV\r\n”时，TURN 服务器将会返回“250 dataport\r\n”。客户端收到后建立数据套接字，连接到 TURN 服务器的 dataport 端口，之后通过数据连接发送和接收数据都由此数据套接字完成。

(二) 服务器程序

a. 接收客户端连接

服务器程序的主函数持续监听 21 端口，每收到一个客户端的连接建立请求，就创建一个新进程为其提供服务。

b. 通过文件名获取文件详细信息

当服务器收到 LIST 命令后，不仅要获得目标文件夹下的所有文件名，还需要通过文件名获得文件的权限、创建者、修改时间等详细信息，此功能使用了 stat 函数实现。以文件名为第一参数调用 stat 函数后，便可得到以结构体形式给出的文件详细信息。对结构体的各成员变量进行处理，即可生成详细信息字符串。

c. 对服务器其他功能的一些说明（此处无超链接）

事实上，服务器大部分功能的实现逻辑与客户端相似。例如，服务器对 PASV 命令的响应方式就十分像客户端在主动模式下建立连接的过程，服务器对 RETR 命令的响应方式也与客户端的 send 命令较为一致。因此，本部分将不再重复叙述服务器这些功能的实现细节。

d. 为支持 NAT 穿透做出的修改

为支持 NAT 穿透做出的修改主要涉及建立控制连接和建立数据连接。

对于控制连接的建立，服务器不再监听 21 号端口，而是向 TURN 服务器的特定端口发起连接，并发送“CONS\r\n”，正常情况下 TURN 服务器将返回一个端口号，服务器再向这个端口发起连接。之后每当有客户端想要连接到服务器时，TURN 服务器就会通过新连接向服务器发送一个端口号，服务器发送到这个端口的控制信息都会被 TURN 服务器转发给客户端。

对于数据连接的建立，我新增了一条名为 DATA 的控制命令。当 TURN 服务器发现客户端想要与服务器建立数据连接时，会向服务器发送“DATA port\r\n”，服务器向 port 端口发送的数据信息都会被 TURN 服务器转发给客户端。。

(三) TURN 服务器

TURN 服务器程序由一个主函数和四个数据转发函数构成。

四个数据转发函数分别负责转发客户端到服务器的控制信息、服务器到客户端的控制信息、客户端到服务器的数据信息和服务器到客户端的数据信息。后三个函数实现起来较为简单，都是循环地从一个端口接收信息，再通过另一个端口发送出去。而第一个函数还需要判断客户端发来的控制命令是否为 PORT 和 PASV，原因是 TURN 服务器需要对数据连接建立命令做特殊的处理。若客户端发送了 PORT 命令，TURN 服务器将直接返回“510 NAT traversal service doesn't support active mode.\r\n”，不做其他处理。若客户端发送了 PASV 命令，TURN 服务器将随机选择两个端口开始监听，并分别将端口值告知客户端和服务器。两端口都收到连接请求后，创建两个子进程分别让其调用两个数据信息转发函数。

主函数则持续监听某一特殊端口，对于每个与此端口建立连接的终端，接收一条它们发来的消息。若该消息的前缀为“CONS”（FTP 服务器发来的），则返回一个随机的端口，并在此端口开始监听，接收到连接请求后，在哈希表中存储一条对方公网 IP 与套接字 FD 的对应信息；若该消息的前缀为“CONC”（FTP 客户端发来的），则查看哈希表中是否存在键值为消息中 IP 的数据。如果不存在，直接向客户端回复“500 Target Not Found.\r\n”，否则随机选择两个端口开始监听，并分别将端口值告知客户端和服务器。两端口都收到连接请求后，创建两个子进程分别让其调用两个控制信息转发函数。

六、 测试及结果分析

1. 聊天程序

在测试过程中，采用如下网络拓扑，其中云服务器 ESC 和 PC 都为 Linux 系统环境，ESC 为 Ubuntu 20.04，PC 为 Ubuntu 18.04。

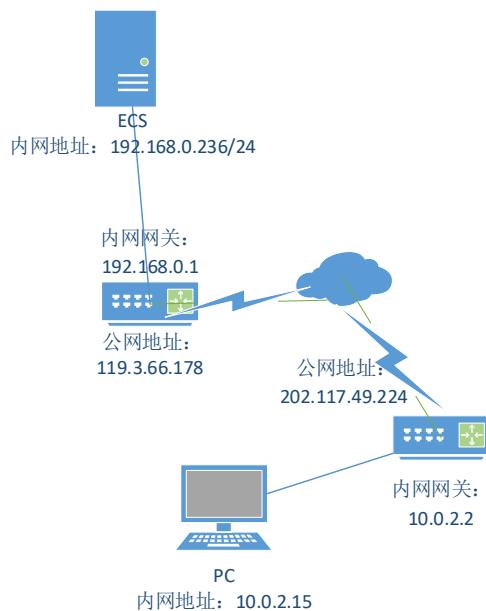


图 8-4 网络聊天程序测试网络拓扑图

在云服务器上启动服务器端程序后，在 PC 上进行客户端连接。如图 8-5 所示，正在登录客户端 test1



图 8-5 test1 登录

在登录验证成功后，test1 首先进入聊天室，如图 8-6 所示

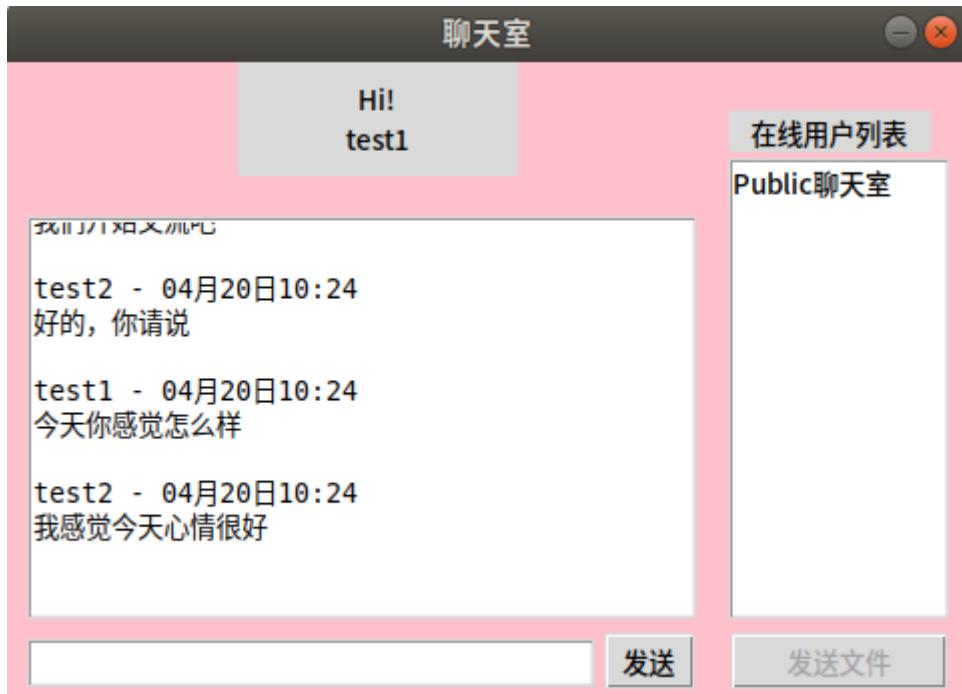


图 8-6 test1 进入聊天室

按照上述过程，正常登录 test2,test1 和 test2 进行群聊，如图 8-7 所示



图 8-7 test1 和 test2 进行群聊

接下来，test1 与 test2 开始一对聊天，如图 8-8 所示



图 8-8 test1 和 test2 进行一对聊天

接下来，test1 向 test2 传输文件，首先要获取 test2 的允许请求，如图 8-9 所示



图 8-9 test1 向 test2 发送文件，向 test2 获取请求

如果 test2 不想接收该文件，直接点 No 即可，test1 会收到如图 8-10 所示的消息提醒



图 8-10 test1 向 test2 发送文件, test2 拒绝

如果 test2 想接收该文件, 直接点 Yes 即可, test1 会收到如图 8-11 所示的消息提醒, 文件传输成功



图 8-11 test1 向 test2 发送文件, test2 接收, 文件传输成功

为验证注册功能，首先登录从未注册过的 test3，弹出如下警示，如图 8-12 所示



图 8-12 未注册 test3 登陆失败

对 test3 进行注册，如图 8-13 所示，显示注册成功



图 8-13 注册 test3 成功

test3 注册成功后进行登录，可加载之前的 public 聊天记录，如图 8-14 所示



图 8-14 test3 加载 public 聊天记录

Test3 在 public 聊天窗口发送消息，test1 和 test2 都进行显示，如图 8-15 所示



图 8-15 test3 进行 public 聊天

2. FTP 程序

受限于篇幅，仅在“普通客户端与普通服务器连接”测试并抓包分析客户端的全部功能。“普通客户端与标准服务器连接”和“带 NAT 穿透的客户端与带 NAT 穿透的服务器连接”中只选择客户端的部分功能进行测试，这主要是为了展示客户端和 NAT 穿透模块能够正常运作。

三项测试的网络拓扑均如图 8-16 所示。在四台主机/服务器中，ESC1 和 ESC2 使用了 Linux 系统（Ubuntu 20.04），PC 使用了 Linux 系统（Ubuntu 21.10）。

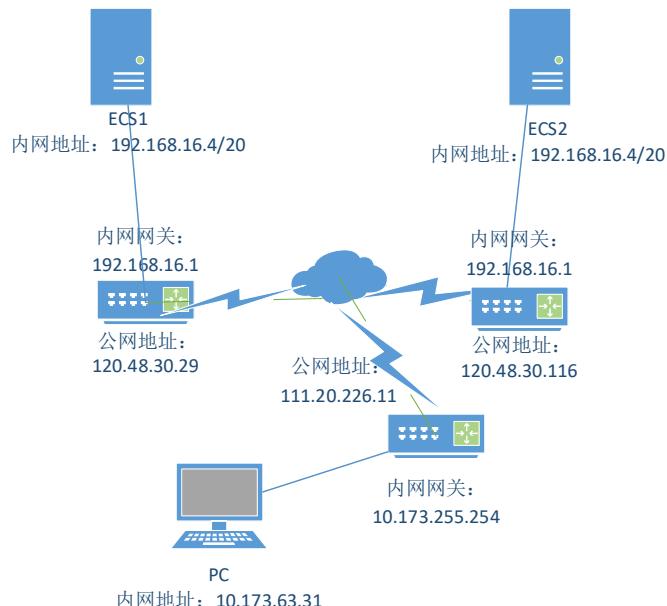


图 8-16 FTP 测试网络拓扑图

(一) 普通客户端与标准服务器连接

在 ESC1 上开启 vsFTPd 服务，并在 PC 上运行客户端程序，并指定目标服务器为 120.48.30.29。如图 8-17 所示，正确输入用户名和密码后即可登入服务器。

```
root@hayate:/home/hayate/exp8# ./cl wlo1 120.48.30.29
Connected to 120.48.30.29.
220 (vsFTPD 3.0.3)
Name (120.48.30.29:root): ubuntu
331 Please specify the password.
Password:
230 Login successful.
215 UNIX Type: L8
200 Switching to Binary mode.
ftp> []
```

图 8-17 登录

列出目录、下载等 FTP 命令的运行结果如图 8-18 所示，发现控制连接和两种数据连接均能正确发挥作用，说明自编的客户端满足协议标准。

```
ftp> ls
200 Switching to ASCII mode.
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
Client.cpp
Server.cpp
cl
226 Directory send OK.
ftp> passive
Passive mode: on;
ftp> get Server.cpp download.cpp
local: download.cpp remote: Server.cpp
227 Entering Passive Mode (120,48,30,29,73,184).
150 Opening BINARY mode data connection for Server.cpp (28555 bytes).
226 Transfer complete.
28555 bytes received in 0.04 secs (791.9611 kB/s)
ftp> !ls
'7-6 袁龙骧 苏贵林 实验八报告.docx'  ClientNAT.cpp  Server.cpp
cl                           download.cpp  ServerNAT.cpp
Client.cpp                   NAT_SER.cpp
ftp> []
```

图 8-18 运行部分命令

(二) 普通客户端与普通服务器连接

本项测试由 ESC1 运行服务器程序，PC 运行客户端程序。

由于某些命令（如 `mkdir` 等）难以观测运行结果，因此本节中可能会出现同时测试多条命令的情况，亦可能出现测试某条命令的过程中出现其他命令的情况。

此外，本节中只对会与服务器产生交互的命令进行抓包分析，对于本地命令（如 lcd 等）仅展示运行结果。

另一方面，为了测试主动模式和被动模式的正确性，在运行需要建立数据连接的命令时，我将等概率地从两种模式中选择一种使用。

(1) !

!命令能够在任何情况下运行本地命令，图 8-19 展示了使用!命令列出本地文件和在本地创建文件夹。

```
ftp> !ls -l
总用量 1272
-rwxrwxrwx 1 hayate hayate 913191 4月 18 11:16 '7-6 袁龙骥 苏贵林 实验八报告.docx'
-rwxr-xr-x 1 root root 149464 4月 18 10:39 cl
-rwxrwxrwx 1 hayate hayate 69320 4月 14 16:46 Client.cpp
-rwxrwxrwx 1 hayate hayate 69452 4月 13 11:00 ClientNAT.cpp
-rw-r--r-- 1 root root 27318 4月 18 10:55 download.cpp
-rwxrwxrwx 1 hayate hayate 11876 4月 15 13:15 NAT_SER.cpp
-rwxrwxrwx 1 hayate hayate 28555 4月 13 11:08 Server.cpp
-rwxrwxrwx 1 hayate hayate 28453 4月 13 11:04 ServerNAT.cpp

ftp> !mkdir mk
ftp> !ls -l
总用量 1276
-rwxrwxrwx 1 hayate hayate 913191 4月 18 11:16 '7-6 袁龙骥 苏贵林 实验八报告.docx'
-rwxr-xr-x 1 root root 149464 4月 18 10:39 cl
-rwxrwxrwx 1 hayate hayate 69320 4月 14 16:46 Client.cpp
-rwxrwxrwx 1 hayate hayate 69452 4月 13 11:00 ClientNAT.cpp
-rw-r--r-- 1 root root 27318 4月 18 10:55 download.cpp
drwxrwxr-x 2 hayate hayate 4096 4月 18 11:18 mk
-rwxrwxrwx 1 hayate hayate 11876 4月 15 13:15 NAT_SER.cpp
-rwxrwxrwx 1 hayate hayate 28555 4月 13 11:08 Server.cpp
-rwxrwxrwx 1 hayate hayate 28453 4月 13 11:04 ServerNAT.cpp

ftp> 
```

图 8-19 !命令

(2) append 命令

append 命令能够在指定服务器文件的末尾追加指定本地文件的内容。

如图 8-20 所示，当客户端未连接服务器时，调用 append 命令会提示错误。

```
ftp> append a.txt
Not connected.
ftp> 
```

图 8-20 未连接服务器时调用 append 命令

连接上服务器后，在本地创建如图 8-21(a)所示的 a.txt 文件（服务器根目录下不存在同名文件）。如图 8-21 (b)所示，首先运行“append a.txt”，服务器得到了内容如图 8-21 (a)所示的 a.txt 文件。再运行一次“append a.txt”，服务器上 a.txt 文件的内容发生了如图 8-21 (c)所示的变化。最后在服务器上创建如图 8-21 (d)所

示的 b.txt 文件，执行“append a.txt b.txt”后，服务器上 a.txt 文件的内容发生了如图 8-21 (e) 所示的变化。

aaa
aaaa

(a) a.txt

```
ftp> append a.txt
local: a.txt remote: a.txt
---> PORT 10,173,63,31,223,99
200 PORT command successful. Consider using PASV.
---> APPE a.txt
150 Ok to send data.
226 Transfer complete.
9 bytes sent in 0.00 secs (82.9157 kB/s)
ftp> append a.txt
local: a.txt remote: a.txt
---> PORT 10,173,63,31,218,177
200 PORT command successful. Consider using PASV.
---> APPE a.txt
150 Ok to send data.
226 Transfer complete.
9 bytes sent in 0.00 secs (137.3291 kB/s)
ftp> append a.txt b.txt
local: a.txt remote: b.txt
---> PORT 10,173,63,31,208,45
200 PORT command successful. Consider using PASV.
---> APPE b.txt
150 Ok to send data.
226 Transfer complete.
9 bytes sent in 0.00 secs (133.1676 kB/s)
ftp> 
```

(b) 执行命令

aaa
aaaa
aaa
aaa

(c) 发生变化的 a.txt

bbbb
bbb

(d) b.txt

```
bbbb  
bbb  
aaa  
aaaa
```

(e) 发生变化的 b.txt

图 8-21 正常运行 append 命令

图 8-22 所示的报文截获结果说明了 append 命令执行过程的正确性。

```
4620.356572 10.173.63.31 120.48.30.29 FTP 92Request: PORT 10,173,63,31,223,99  
4720.384216 120.48.30.29 10.173.63.31 FTP 117Response: 200 PORT command successful  
4920.384401 10.173.63.31 120.48.30.29 FTP 78Request: APPE a.txt  
5320.457735 120.48.30.29 10.173.63.31 FTP 88Response: 150 Ok to send data.  
5520.461126 10.173.63.31 120.48.30.29 FTP-DATA 75FTP Data: 9 bytes (PORT) (APPE a.txt)  
6120.527502 120.48.30.29 10.173.63.31 FTP 90Response: 226 Transfer complete.  
11334.878833 10.173.63.31 120.48.30.29 FTP 93Request: PORT 10,173,63,31,218,177  
11534.905866 120.48.30.29 10.173.63.31 FTP 117Response: 200 PORT command successful  
11734.906054 10.173.63.31 120.48.30.29 FTP 78Request: APPE a.txt  
12134.955662 120.48.30.29 10.173.63.31 FTP 88Response: 150 Ok to send data.  
12434.956251 10.173.63.31 120.48.30.29 FTP-DATA 75FTP Data: 9 bytes (PORT) (APPE a.txt)  
12934.979505 120.48.30.29 10.173.63.31 FTP 90Response: 226 Transfer complete.  
352150.828274 10.173.63.31 120.48.30.29 FTP 92Request: PORT 10,173,63,31,208,45  
353150.854747 120.48.30.29 10.173.63.31 FTP 117Response: 200 PORT command successful  
355150.855011 10.173.63.31 120.48.30.29 FTP 78Request: APPE b.txt  
361150.923158 120.48.30.29 10.173.63.31 FTP 88Response: 150 Ok to send data.  
364150.924858 10.173.63.31 120.48.30.29 FTP-DATA 75FTP Data: 9 bytes (PORT) (APPE b.txt)  
366150.966655 120.48.30.29 10.173.63.31 FTP 90Response: 226 Transfer complete.
```

图 8-22 运行 append 时截获的报文

此外，append 命令仅接受 1-2 个参数。如图 8-23 所示，参数不足时客户端会在给予一次补全机会后报错，参数过多时则仅取前两个参数。图 8-23 亦反应了当指定的本地文件不存在时，append 命令不会继续执行。

```
ftp> append  
(local-file)  
usage: append local-file remote-file  
ftp> append a b c  
local: a remote: b  
local: a: No such file or directory  
ftp>
```

图 8-23 异常调用 append 命令

(3) ascii

ascii 命令能够指定传输模式为 ASCII 模式。

如图 8-24 所示，当客户端未连接服务器时，调用 ascii 命令会提示错误。

```
ftp> ascii  
Not connected.  
ftp>
```

图 8-24 未连接服务器时调用 ascii 命令

如图 8-25 所示, 连接上服务器后再调用 ascii 命令, 发现传输模式成功切换, 图 8-26 所示的报文展示了 ascii 命令的执行过程。

```
ftp> ascii  
---> TYPE A  
200 Switching to ASCII mode.
```

图 8-25 正确调用 ascii 命令

```
10.000000 10.173.63.31 120.48.30.29 FTP      74 Request: TYPE A  
20.030377 120.48.30.29 10.173.63.31 FTP      96 Response: 200 Switching to ASCII mode.
```

图 8-26 正确调用 ascii 命令时截获的报文

(4) binary

binary 命令能够指定传输模式为二进制模式。

如图 8-27 所示, 当客户端未连接服务器时, 调用 binary 命令会提示错误。

```
ftp> binary  
Not connected.  
ftp> 
```

图 8-27 未连接服务器时调用 binary 命令

如图 8-28 所示, 连接上服务器后再调用 binary 命令, 发现传输模式成功切换, 图 8-29 所示的报文展示了 ascii 命令的执行过程。

```
ftp> binary  
---> TYPE I  
200 Switching to Binary mode.
```

图 8-28 正确调用 binary 命令

```
65.735955 10.173.63.31 120.48.30.29 FTP      74 Request: TYPE I  
75.762357 120.48.30.29 10.173.63.31 FTP      97 Response: 200 Switching to Binary mode.
```

图 8-29 正确调用 binary 命令时截获的报文

(5) bye & quit

bye 和 quit 是两条名字不同, 但代码完全一样、功能也均为断开与服务器的连接并退出客户端的命令, 故本节中仅对 bye 命令进行测试。

如图 8-30 所示, 当客户端未连接服务器时, 调用 bye 命令将直接退出客户端。

```
ftp> bye  
hayate@hayate:~/exp8$
```

图 8-30 未连接服务器时调用 bye 命令

如图 8-31 所示，连接上服务器后再调用 bye 命令，客户端将先向服务器发送“QUIT\r\n”再退出，图 8-32 所示的报文展示了 bye 命令的执行过程。

```
ftp> bye  
---> QUIT  
221 Goodbye.  
hayate@hayate:~/exp8$
```

图 8-31 连接服务器时调用 bye 命令

10.000000	10.173.63.31	120.48.30.29	FTP	72 Request: QUIT
20.026477	120.48.30.29	10.173.63.31	FTP	80 Response: 221 Goodbye.

图 8-32 连接服务器时调用 bye 命令截获的报文

(6) cd & pwd

cd 命令能够切换服务器的工作目录，pwd 命令则能显示服务器当前的工作目录。

如图 8-33 所示，当客户端未连接服务器时，调用 cd 命令和 pwd 命令均会提示错误。

```
ftp> cd  
Not connected.  
ftp> pwd  
Not connected.  
ftp>
```

图 8-33 未连接服务器时调用 cd 命令和 pwd 命令

如图 8-34 所示，连接上服务器后再调用 cd 命令和 pwd 命令，发现切换到已存在文件夹的命令能够正确执行，切换到不存在文件夹的命令则不能执行，图 8-35 所示的报文展示了上述过程。

```

ftp> pwd
---> PWD
257 "/home/ubuntu" is the current directory
ftp> cd /home
---> CWD /home
250 Directory successfully changed.
ftp> pwd
---> PWD
257 "/home" is the current directory
ftp> cd ubuntu/notexist
---> CWD ubuntu/notexist
550 Failed to change directory.
ftp> pwd
---> PWD
257 "/home" is the current directory
ftp> cd ubuntu
---> CWD ubuntu
250 Directory successfully changed.
ftp> pwd
---> PWD
257 "/home/ubuntu" is the current directory
ftp> []

```

图 8-34 连接服务器时调用 cd 命令和 pwd 命令

10.000000	10.173.63.31	120.48.30.29	FTP	71Request: PWD
20.024856	120.48.30.29	10.173.63.31	FTP	111Response: 257 "/home/ubuntu" is the curr
43.611011	10.173.63.31	120.48.30.29	FTP	77Request: CWD /home
53.634988	120.48.30.29	10.173.63.31	FTP	103Response: 250 Directory successfully cha
718.890129	10.173.63.31	120.48.30.29	FTP	71Request: PWD
818.915133	120.48.30.29	10.173.63.31	FTP	104Response: 257 "/home" is the current dir
1030.434172	10.173.63.31	120.48.30.29	FTP	87Request: CWD ubuntu/notexist
1130.565820	120.48.30.29	10.173.63.31	FTP	99Response: 550 Failed to change directory
1332.172095	10.173.63.31	120.48.30.29	FTP	71Request: PWD
1432.207134	120.48.30.29	10.173.63.31	FTP	104Response: 257 "/home" is the current dir
2935.301097	10.173.63.31	120.48.30.29	FTP	78Request: CWD ubuntu
3035.325091	120.48.30.29	10.173.63.31	FTP	103Response: 250 Directory successfully cha
3237.146133	10.173.63.31	120.48.30.29	FTP	71Request: PWD
3337.171229	120.48.30.29	10.173.63.31	FTP	111Response: 257 "/home/ubuntu" is the curr

图 8-35 连接服务器时调用 cd 命令和 pwd 命令截获的报文

(7) close & disconnect

close 和 disconnect 是两条名字不同，但代码完全一样、功能也均为断开与服务器连接的命令，故本节中仅对 close 命令进行测试。

如图 8-36 所示，当客户端未连接服务器时，调用 close 命令会提示错误。

```

ftp> close
Not connected.
ftp> []

```

图 8-36 未连接服务器时调用 close 命令

如图 8-37 所示，连接上服务器后再调用 `close` 命令，客户端会切断与服务器的连接，但并未像 `bye` 命令一样退出客户端，图 8-38 所示的报文展示了上述过程。

```
ftp> close
---> QUIT
221 Goodbye.
ftp> [ ]
```

图 8-37 连接服务器时调用 `close` 命令

10.000000	10.173.63.31	120.48.30.29	FTP	72 Request: QUIT
20.026374	120.48.30.29	10.173.63.31	FTP	80 Response: 221 Goodbye.

图 8-38 连接服务器时调用 `close` 命令截获的报文

(8) debug

`debug` 命令用于切换调试模式。

如图 8-39 所示，打开调试模式后，用户能看到客户端向服务器发送的底层命令，关闭调试模式则看不到这些底层命令。

```
ftp> debug
Debugging on.
ftp> pwd
---> PWD
257 "/home/ubuntu" is the current directory
ftp> debug
Debugging off.
ftp> pwd
257 "/home/ubuntu" is the current directory
ftp> [ ]
```

图 8-39 调试模式开启与否的区别

(9) delete

`delete` 命令能够删除服务器上的文件。

如图 8-40 所示，当客户端未连接服务器时，调用 `delete` 命令会提示错误。

```
ftp> delete a
Not connected.
ftp> [ ]
```

图 8-40 未连接服务器时调用 `delete` 命令

如图 8-41 所示，连接上服务器后再调用 `delete` 命令，能够成功删除服务器

上已存在的文件，而删除文件夹和不存在的文件则会失败，图 8-42 所示的报文展示了上述过程。

```
ftp> dir
---> TYPE A
200 Switching to ASCII mode.
---> PORT 10,173,63,31,193,53
200 PORT command successful. Consider using PASV.
---> LIST
150 Here comes the directory listing.
drwxr-xr-x    2 root      root          4096 Apr 18 14:18 b
-rw-r--r--    1 root      root           30 Apr 18 14:18 a.txt
226 Directory send OK.
ftp> delete a.txt
---> DELE a.txt
250 Delete operation successful.
ftp> delete b
---> DELE b
550 Delete operation failed.
ftp> delete c.txt
---> DELE c.txt
550 Delete operation failed.
ftp> dir
---> PORT 10,173,63,31,152,99
200 PORT command successful. Consider using PASV.
---> LIST
150 Here comes the directory listing.
drwxr-xr-x    2 root      root          4096 Apr 18 14:18 b
226 Directory send OK.
ftp> 
```

图 8-41 连接服务器时调用 delete 命令

10.000000	10.173.63.31	120.48.30.29	FTP	74Request: TYPE A
20.045178	120.48.30.29	10.173.63.31	FTP	96Response: 200 Switching to ASCII mode.
40.045460	10.173.63.31	120.48.30.29	FTP	92Request: PORT 10,173,63,31,193,53
50.089179	120.48.30.29	10.173.63.31	FTP	117Response: 200 PORT command successful. C
60.089361	10.173.63.31	120.48.30.29	FTP	72Request: LIST
90.159092	120.48.30.29	10.173.63.31	FTP	105Response: 150 Here comes the directory l
110.161391	120.48.30.29	10.173.63.31	FTP-DATA	190FTP Data: 124 bytes (PORT) (LIST)
170.245837	120.48.30.29	10.173.63.31	FTP	90Response: 226 Directory send OK.
2010.522229	10.173.63.31	120.48.30.29	FTP	78Request: DELE a.txt
2110.567473	120.48.30.29	10.173.63.31	FTP	100Response: 250 Delete operation successfu
2314.792234	10.173.63.31	120.48.30.29	FTP	74Request: DELE b
2414.835235	120.48.30.29	10.173.63.31	FTP	96Response: 550 Delete operation failed.
2619.097097	10.173.63.31	120.48.30.29	FTP	78Request: DELE c.txt
2719.139182	120.48.30.29	10.173.63.31	FTP	96Response: 550 Delete operation failed.
2921.434065	10.173.63.31	120.48.30.29	FTP	92Request: PORT 10,173,63,31,152,99
3021.479275	120.48.30.29	10.173.63.31	FTP	117Response: 200 PORT command successful. C
3221.479453	10.173.63.31	120.48.30.29	FTP	72Request: LIST
3521.557334	120.48.30.29	10.173.63.31	FTP	105Response: 150 Here comes the directory l
4421.602338	120.48.30.29	10.173.63.31	FTP	90Response: 226 Directory send OK.

图 8-42 连接服务器时调用 delete 命令截获的报文

此外，`delete` 命令仅接受一个参数。如图 8-43 所示，参数不足时客户端会在给予一次补全机会后报错，参数过多时则仅取前一个参数。

```
ftp> delete
(remote-file)
usage: delete remote-file
ftp> delete
(remote-file) a
---> DELE a
250 Delete operation successful.
ftp> delete a b c d
---> DELE a
550 Delete operation failed.
ftp> []
```

图 8-43 异常调用 `delete` 命令

(10) dir

`dir` 命令用于列出服务器上目录或文件的详细信息。

如图 8-44 所示，当客户端未连接服务器时，调用 `dir` 命令会提示错误。

```
ftp> dir
Not connected.
ftp> []
```

图 8-44 未连接服务器时调用 `dir` 命令

`dir` 可接受 0-2 个参数，输入参数过多时将仅使用前两个参数。第一个参数指明了目标目录或文件，使用“`-`”表示服务器当前工作目录；第二个参数指明了保存详细信息的位置，使用“`-`”表示将信息输出在屏幕上。两参数的缺省值均为“`-`”。

图 8-45 展示了各种调用 `dir` 命令的方式，图 8-46 所示的报文展示了 `dir` 命令的执行过程。

```
ftp> dir
---> TYPE A
200 Switching to ASCII mode.
---> PASV
227 Entering Passive Mode (120,48,30,29,151,225).
---> LIST
150 Here comes the directory listing.
drwxr-xr-x    2 root      root          4096 Apr 18 14:18 b
-rw-r--r--    1 root      root          22 Apr 18 14:45 a
226 Directory send OK.
ftp> dir -a -
---> PASV
227 Entering Passive Mode (120,48,30,29,210,195).
---> LIST -a
150 Here comes the directory listing.
drwxr-xr-x    2 root      root          4096 Apr 18 14:18 b
drwxr-xr-x    3 root      root          4096 Apr 18 14:45 .
drwxr-xr-x    9 root      root          4096 Apr 18 14:45 ..
-rw-r--r--    1 root      root          22 Apr 18 14:45 a
226 Directory send OK.
ftp> dir / dirres.txt
output to localfile: dirres.txt? y
---> PASV
227 Entering Passive Mode (120,48,30,29,194,235).
---> LIST /
150 Here comes the directory listing.
226 Directory send OK.
ftp> [ ]
```

(a) 调用 dir 命令

```

drwxr-xr-x  2 root   root      4096 Apr 23 15:34 mnt
drwxr-xr-x  40 root   root      800  Apr 18 14:17 run
drwxr-xr-x 200 root   root     4000 Apr 17 09:20 dev
dr-xr-xr-x  13 root   root      0 Apr 17 09:16 sys
drwxr-xr-x  3 root   root     4096 Apr 17 09:25 srv
dr-xr-xr-x 254 root   root      0 Apr 17 09:16 proc
drwxr-xr-x  2 root   root     4096 Apr 23 15:34 libx32
drwxr-xr-x  3 root   root     4096 Jan 27 14:49 snap
drwxr-xr-x 15 root   root     4096 Apr 17 09:23 boot
drwxr-xr-x  5 root   root     4096 Jan 27 14:40 media
drwxrwxrwx  7 root   root     4096 Apr 18 13:38 tmp
drwx----- 2 root   root    16384 Jan 27 14:40 lost+found
drwxr-xr-x 70 root   root     4096 Apr 17 10:18 lib
drwxr-xr-x  2 root   root     4096 Apr 23 15:34 lib32
drwxr-xr-x 11 root   root     4096 Apr 17 09:24 var
drwxr-xr-x  3 root   root     4096 Mar 17 18:08 lib64
drwxr-xr-x 13 root   root     4096 Mar 17 18:05 usr
drwxr-xr-x  3 root   root     4096 Apr 17 09:17 home
drwxr-xr-x 147 root   root    4096 Apr 18 10:42 etc
drwx----- 7 root   root     4096 Apr 18 14:45 root
drwxr-xr-x 860 root   root   28672 Apr 18 10:41 bin
drwxr-xr-x  7 root   root     4096 Apr 17 09:03 opt
drwxr-xr-x 325 root   root   12288 Apr 17 09:25 sbin

```

(b) 调用 dir 命令后生成的 dirres.txt

图 8-45 连接服务器时调用 dir 命令

10.000000	10.173.63.31	120.48.30.29	FTP	74Request: TYPE A
20.041203	120.48.30.29	10.173.63.31	FTP	96Response: 200 Switching to ASCII mode.
40.041453	10.173.63.31	120.48.30.29	FTP	72Request: PASV
50.067161	120.48.30.29	10.173.63.31	FTP	117Response: 227 Entering Passive Mode (12
90.093276	10.173.63.31	120.48.30.29	FTP	72Request: LIST
100.120203	120.48.30.29	10.173.63.31	FTP	105Response: 150 Here comes the directory
110.120684	120.48.30.29	10.173.63.31	FTP-DATA	186FTP Data: 120 bytes (PASV) (LIST)
170.189226	120.48.30.29	10.173.63.31	FTP	90Response: 226 Directory send OK.
1910.963207	10.173.63.31	120.48.30.29	FTP	72Request: PASV
2010.989401	120.48.30.29	10.173.63.31	FTP	117Response: 227 Entering Passive Mode (12
2511.013489	10.173.63.31	120.48.30.29	FTP	75Request: LIST -a
2611.039364	120.48.30.29	10.173.63.31	FTP-DATA	307FTP Data: 241 bytes (PASV) (LIST -a)
2911.039601	120.48.30.29	10.173.63.31	FTP	105Response: 150 Here comes the directory
3311.067364	120.48.30.29	10.173.63.31	FTP	90Response: 226 Directory send OK.
3624.688224	10.173.63.31	120.48.30.29	FTP	72Request: PASV
3724.713491	120.48.30.29	10.173.63.31	FTP	117Response: 227 Entering Passive Mode (12
4224.757572	10.173.63.31	120.48.30.29	FTP	74Request: LIST /
4324.801581	120.48.30.29	10.173.63.31	FTP	105Response: 150 Here comes the directory
4524.805525	120.48.30.29	10.173.63.31	FTP-DATA	1139FTP Data: 1073 bytes (PASV) (LIST /)
4724.807644	120.48.30.29	10.173.63.31	FTP-DATA	441FTP Data: 375 bytes (PASV) (LIST /)
4924.827488	120.48.30.29	10.173.63.31	FTP	90Response: 226 Directory send OK.

图 8-46 连接服务器时调用 dir 命令截获的报文

(11) get & recv & restart

get 和 recv 是两条名字不同，但代码完全一样、功能也均为从服务器下载单个文件的命令，故本节中仅对 get 命令进行测试。

如图 8-47 所示，当客户端未连接服务器时，调用 get 命令会提示错误。

```

ftp> get
Not connected.
ftp>

```

图 8-47 未连接服务器时调用 get 命令

get 可接受 1-2 个参数，输入参数过少时将在给予一次提示后报错，输入参数过多时将仅使用前两个参数。第一个参数指明了目标文件，第二个参数指明了将目标文件下载到本地时使用的文件名。第二个参数的缺省值为第一个参数。

在服务器上创建如图 8-48 所示的 a.txt 和 b.txt 文件（本地无同名文件）。在客户端执行“get a.txt”后，本地出现了如图 8-48 (a) 所示的 a.txt。再执行“get b.txt a.txt”后，本地 a.txt 的内容变成了如图 8-48 (b) 所示的。最后尝试下载文件夹，执行“get /home a.txt”，发现执行失败，本地 a.txt 的内容也未发生变化。图 8-49 通过终端和报文展示了上述过程。



aaa
aa

(a) a.txt



bbb
bb

(b) b.txt

图 8-48 服务器上的文件

```

ftp> get a.txt
local: a.txt remote: a.txt
---> PORT 10,173,63,31,169,141
200 PORT command successful. Consider using PASV.
---> RETR a.txt
150 Opening BINARY mode data connection for a.txt (7 bytes).
226 Transfer complete.
7 bytes received in 0.00 secs (24.5897 kB/s)
ftp> get b.txt a.txt
local: a.txt remote: b.txt
---> PORT 10,173,63,31,195,121
200 PORT command successful. Consider using PASV.
---> RETR b.txt
150 Opening BINARY mode data connection for b.txt (7 bytes).
226 Transfer complete.
7 bytes received in 0.00 secs (23.9019 kB/s)
ftp> get /home a.txt
local: a.txt remote: /home
---> PORT 10,173,63,31,148,65
200 PORT command successful. Consider using PASV.
---> RETR /home
550 Failed to open file.
ftp> []

```

(a) 终端显示的信息

10.000000	10.173.63.31	120.48.30.29	FTP	93Request: PORT 10,173,63,31,169,141
20.028650	120.48.30.29	10.173.63.31	FTP	117Response: 200 PORT command successful.
40.028862	10.173.63.31	120.48.30.29	FTP	78Request: RETR a.txt
80.082469	120.48.30.29	10.173.63.31	FTP-DATA	73FTP Data: 7 bytes (PORT) (RETR a.txt)
110.083718	120.48.30.29	10.173.63.31	FTP	128Response: 150 Opening BINARY mode data
150.112328	120.48.30.29	10.173.63.31	FTP	90Response: 226 Transfer complete.
176.934141	10.173.63.31	120.48.30.29	FTP	93Request: PORT 10,173,63,31,195,121
186.961970	120.48.30.29	10.173.63.31	FTP	117Response: 200 PORT command successful.
206.962165	10.173.63.31	120.48.30.29	FTP	78Request: RETR b.txt
247.014921	120.48.30.29	10.173.63.31	FTP-DATA	73FTP Data: 7 bytes (PORT) (RETR b.txt)
277.016720	120.48.30.29	10.173.63.31	FTP	128Response: 150 Opening BINARY mode data
307.042716	120.48.30.29	10.173.63.31	FTP	90Response: 226 Transfer complete.
3311.100004	10.173.63.31	120.48.30.29	FTP	92Request: PORT 10,173,63,31,148,65
3411.128006	120.48.30.29	10.173.63.31	FTP	117Response: 200 PORT command successful.
3611.128193	10.173.63.31	120.48.30.29	FTP	78Request: RETR /home
3711.156444	120.48.30.29	10.173.63.31	FTP	92Response: 550 Failed to open file.

(b) 截获的报文

图 8-49 连接服务器时调用 get 命令

restart 命令用于设置断点值，配合 get、put 命令可实现断点续传。

“PORT a,b,c,d,e,f\r\n”

在上述设定下，执行“get a.txt”后再执行“restart 3”和“get b.txt a.txt”，发现本地 a.txt 的内容变为图 8-50 所示的样子。具体而言，a.txt 的前三个字符保持不变，后面的字符才变为 b.txt 的内容，这说明断点续传（下载）功能能够正常运作。另外，断点续传（下载）要求本地必须存在名为 get 第二个参数的文件，否

则即使服务器正确传输了文件数据，本地文件也不会发生变化，执行“restart 3”和“get b.txt notexist.txt”命令后本地未出现 notexist.txt 说明了这一点。



图 8-50 断点续传（下载）后的 a.txt

图 8-51 通过终端和报文展示了上述断点续传（下载）过程。

```
ftp> restart 3
restarting at 3. execute get, put or append to initiate transfer
ftp> get b.txt a.txt
local: a.txt remote: b.txt
---> PASV
227 Entering Passive Mode (120,48,30,29,202,227).
---> REST 3
350 Restart position accepted (3).
---> RETR b.txt
150 Opening BINARY mode data connection for b.txt (7 bytes).
226 Transfer complete.
4 bytes received in 0.00 secs (2.0701 kB/s)
ftp> restart 3
restarting at 3. execute get, put or append to initiate transfer
ftp> get b.txt notexist.txt
local: notexist.txt remote: b.txt
---> PASV
227 Entering Passive Mode (120,48,30,29,226,77).
---> REST 3
350 Restart position accepted (3).
---> RETR b.txt
150 Opening BINARY mode data connection for b.txt (7 bytes).
226 Transfer complete.
4 bytes received in 0.00 secs (3.5969 kB/s)
ftp> 
```

(a) 终端显示的信息

2510.930489	10.173.63.31	120.48.30.29	FTP	74Request: REST 3
2610.972336	120.48.30.29	10.173.63.31	FTP	102Response: 350 Restart position accept
2810.972547	10.173.63.31	120.48.30.29	FTP	78Request: RETR b.txt
2911.014621	120.48.30.29	10.173.63.31	FTP	128Response: 150 Opening BINARY mode dat
3111.016276	120.48.30.29	10.173.63.31	FTP-DATA	70FTP Data: 4 bytes (PASV) (REST 3)
3511.058932	120.48.30.29	10.173.63.31	FTP	90Response: 226 Transfer complete.
6752.906597	10.173.63.31	120.48.30.29	FTP	72Request: PASV
6852.952801	120.48.30.29	10.173.63.31	FTP	116Response: 227 Entering Passive Mode (
7352.976861	10.173.63.31	120.48.30.29	FTP	74Request: REST 3
7453.019155	120.48.30.29	10.173.63.31	FTP	102Response: 350 Restart position accept
7653.019360	10.173.63.31	120.48.30.29	FTP	78Request: RETR b.txt
7753.065399	120.48.30.29	10.173.63.31	FTP	128Response: 150 Opening BINARY mode dat
7853.066542	120.48.30.29	10.173.63.31	FTP-DATA	70FTP Data: 4 bytes (PASV) (REST 3)
8453.150764	120.48.30.29	10.173.63.31	FTP	90Response: 226 Transfer complete.

(b) 截获的报文

图 8-51 断点续传（下载）

(12) help

如图 8-52 所示，help 命令用于展示所有客户端支持的命令。

```
ftp> help
Commands may be abbreviated.
Commands are:
!
append      ascii      binary      bye
cd          close      debug       delete     dir
disconnect   get        help        lcd        literal
ls          mdelete   mdir       mget      mkdir
mls         mput      newer      open      passive
prompt      put        pwd        quit      quote
recv        reget     remotehelp rename    restart
rmdir      send      size        status    system
type        user

To get the usage of each command, please visit https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/ftp
ftp> 
```

图 8-52 help 命令

(13) lcd

lcd 接受 0-1 个参数，输入参数过多时报错。如图 8-53 所示，不输入参数时，lcd 将列出客户端当前工作路径；否则 lcd 将改变客户端当前工作路径。与 cd 命令类似，切换到不存在目录的命令将无法正确执行。

```
ftp> lcd
Local directory now /home/hayate/exp8
ftp> lcd /home
Local directory now /home
ftp> lcd nonexistent
local: nonexistent: No such directory
ftp> lcd hayate/exp8 useless useless
usage: lcd local-directory
ftp> 
```

图 8-53 lcd 命令

(14) literal & quote & Server 对多用户的计数

literal 和 quote 是两条名字不同，但代码完全一样、功能也均为给服务器单个 FTP 命令的命令，故本节中仅对 quote 命令进行测试。

如图 8-54 所示，当客户端未连接服务器时，调用 quote 命令会提示错误。

```
ftp> quote  
Not connected.  
ftp> 
```

图 8-54 未连接服务器时调用 quote 命令

quote 可接受至少一个参数。如图 8-55 所示，输入参数过少时，quote 会在给予一次提示后报错；否则 quote 会用空格连接所有参数向服务器发送出去。

```
ftp> quote  
(command line to send)  
usage: quote line-to-send  
ftp> quote USER abc cd d  
---> USER abc cd d  
530 Can't change to another user.  
ftp> 
```

图 8-55 使用不同的参数个数调用 quote 命令

下面将利用“quote STAT”命令测试服务器对用户连接个数的计算情况。如图 8-56 所示，在第一个客户端进程连上服务器后执行“quote STAT”命令，服务器返回了“你刚建立好连接时，总共有一个客户端与服务器建立了连接”。保持第一个客户端的连接，在第二个客户端进程连上服务器后执行“quote STAT”命令，服务器返回了“你刚建立好连接时，总共有两个客户端与服务器建立了连接”。使用 quit 命令断开第一个客户端与服务器的连接，在第三个客户端进程连上服务器后执行“quote STAT”命令，服务器返回了“你刚建立好连接时，总共有两个客户端与服务器建立了连接”。这说明服务器能够通过软中断正确判断当前有多少个客户端与其建立了连接。上述过程产生的报文如图 8-57 所示。

```
ftp> quote STAT  
---> STAT  
211-FTP server status:  
Connected to ::ffff:111.20.226.11  
Logged in as ubuntu  
TYPE: binary  
No session bandwidth limit  
Session timeout in seconds is 300  
Control connection is plain text  
Data connections will be plain text  
At session startup, client count was 1  
ylxFTP 1.0.0 - secure, fast, stable  
211 End of status
```

(a) 第一个客户端

```

ftp> quote STAT
---> STAT
211-FTP server status:
    Connected to ::ffff:111.20.226.11
    Logged in as ubuntu
    TYPE: binary
    No session bandwidth limit
    Session timeout in seconds is 300
    Control connection is plain text
    Data connections will be plain text
    At session startup, client count was 2
    ylxFTP 1.0.0 - secure, fast, stable
211 End of status

```

(b) 第二个客户端

```

ftp> quote STAT
---> STAT
211-FTP server status:
    Connected to ::ffff:111.20.226.11
    Logged in as ubuntu
    TYPE: binary
    No session bandwidth limit
    Session timeout in seconds is 300
    Control connection is plain text
    Data connections will be plain text
    At session startup, client count was 2
    ylxFTP 1.0.0 - secure, fast, stable
211 End of status
ftp>

```

(c) 第三个客户端

图 8-56 测试服务器对客户端连接的计数

40.086939	120.48.30.29	10.173.63.31	FTP	86Response: 220 (ylxFTP 1.0.0)
61.946147	10.173.63.31	120.48.30.29	FTP	79Request: USER ubuntu
81.990018	120.48.30.29	10.173.63.31	FTP	100Response: 331 Please specify the pass
105.833958	10.173.63.31	120.48.30.29	FTP	80Request: PASS ylx0324
125.878075	120.48.30.29	10.173.63.31	FTP	89Response: 230 Login successful.
145.878182	10.173.63.31	120.48.30.29	FTP	72Request: SYST
165.922326	120.48.30.29	10.173.63.31	FTP	85Response: 215 UNIX Type: L8
185.922450	10.173.63.31	120.48.30.29	FTP	74Request: TYPE I
195.966732	120.48.30.29	10.173.63.31	FTP	97Response: 200 Switching to Binary mod
2111.796293	10.173.63.31	120.48.30.29	FTP	72Request: STAT
2211.837843	120.48.30.29	10.173.63.31	FTP	426Response: 211-FTP server status:
4637.536061	120.48.30.29	10.173.63.31	FTP	86Response: 220 (ylxFTP 1.0.0)
5239.468434	10.173.63.31	120.48.30.29	FTP	79Request: USER ubuntu
5439.497543	120.48.30.29	10.173.63.31	FTP	100Response: 331 Please specify the pass
5641.975399	10.173.63.31	120.48.30.29	FTP	80Request: PASS ylx0324
5842.004902	120.48.30.29	10.173.63.31	FTP	89Response: 230 Login successful.
6042.005002	10.173.63.31	120.48.30.29	FTP	72Request: SYST

(a) 第一部分

```

6042.005002 10.173.63.31 120.48.30.29 FTP      72Request: SYST
6242.034973 120.48.30.29 10.173.63.31 FTP      85Response: 215 UNIX Type: L8
6442.035081 10.173.63.31 120.48.30.29 FTP      74Request: TYPE I
6542.064600 120.48.30.29 10.173.63.31 FTP      97Response: 200 Switching to Binary mod
6750.187312 10.173.63.31 120.48.30.29 FTP      72Request: STAT
6850.216635 120.48.30.29 10.173.63.31 FTP      426Response: 211-FTP server status:
7054.646411 10.173.63.31 120.48.30.29 FTP      72Request: QUIT
7154.688312 120.48.30.29 10.173.63.31 FTP      80Response: 221 Goodbye.
8257.360207 120.48.30.29 10.173.63.31 FTP      86Response: 220 (ylxFTP 1.0.0)
8459.661395 10.173.63.31 120.48.30.29 FTP      79Request: USER ubuntu
8659.688795 120.48.30.29 10.173.63.31 FTP      100Response: 331 Please specify the pass-
10262.856183 10.173.63.31 120.48.30.29 FTP      80Request: PASS ylx0324
10462.882683 120.48.30.29 10.173.63.31 FTP      89Response: 230 Login successful.
10662.882845 10.173.63.31 120.48.30.29 FTP      72Request: SYST
10862.908688 120.48.30.29 10.173.63.31 FTP      85Response: 215 UNIX Type: L8
11062.908847 10.173.63.31 120.48.30.29 FTP      74Request: TYPE I
11162.934362 120.48.30.29 10.173.63.31 FTP      97Response: 200 Switching to Binary mod
14170.154486 10.173.63.31 120.48.30.29 FTP      72Request: STAT
14270.180440 120.48.30.29 10.173.63.31 FTP      426Response: 211-FTP server status:

```

(b) 第二部分

图 8-57 测试服务器对客户端连接的计数产生的报文

(15) ls

ls 命令用于列出服务器上目录或文件的简略信息。

如图 8-58 所示，当客户端未连接服务器时，调用 ls 命令会提示错误。

```

ftp> ls
Not connected.
ftp> []

```

图 8-58 未连接服务器时调用 ls 命令

ls 可接受 0-2 个参数，输入参数过多时将仅使用前两个参数。第一个参数指明了目标目录或文件，使用“-”表示服务器当前工作目录；第二个参数指明了保存详细信息的位置，使用“-”表示将信息输出在屏幕上。两参数的缺省值均为“-”。

图 8-59 展示了各种调用 ls 命令的方式，图 8-60 所示的报文展示了 ls 命令的执行过程。

```
ftp> ls
---> PORT 10,173,63,31,210,151
200 PORT command successful. Consider using PASV.
---> NLST
150 Here comes the directory listing.
b
a
226 Directory send OK.
ftp> ls -la -
---> PORT 10,173,63,31,136,75
200 PORT command successful. Consider using PASV.
---> NLST -la
150 Here comes the directory listing.
drwxr-xr-x    2 root      root          4096 Apr 18 14:18 b
drwxr-xr-x    3 root      root          4096 Apr 18 14:45 .
drwxr-xr-x    9 root      root          4096 Apr 18 16:48 ..
-rw-r--r--    1 root      root          22 Apr 18 14:45 a
226 Directory send OK.
ftp> ls / lsres.txt
output to localfile: lsres.txt? y
---> PORT 10,173,63,31,159,171
200 PORT command successful. Consider using PASV.
---> NLST /
150 Here comes the directory listing.
226 Directory send OK.
ftp> 
```

(a) 调用 ls 命令

```
/mnt
/run
/dev
/sys
/srv
/proc
/libx32
/snap
/boot
/media
/tmp
/lost+found
/lib
/lib32
/var
/lib64
/usr
/home
/etc
/root
/bin
/opt
/sbin
```

(b) 调用 ls 命令后生成的 lsres.txt

图 8-59 连接服务器时调用 dir 命令

```
10.000000 10.173.63.31 120.48.30.29 FTP      93Request: PORT 10,173,63,31,210,151
20.025410 120.48.30.29 10.173.63.31 FTP      117Response: 200 PORT command successful.
40.025614 10.173.63.31 120.48.30.29 FTP      72Request: NLST
120.077787 120.48.30.29 10.173.63.31 FTP      105Response: 150 Here comes the directory
150.145500 120.48.30.29 10.173.63.31 FTP      90Response: 226 Directory send OK.
193.087985 10.173.63.31 120.48.30.29 FTP      92Request: PORT 10,173,63,31,136,75
203.113925 120.48.30.29 10.173.63.31 FTP      117Response: 200 PORT command successful.
223.114081 10.173.63.31 120.48.30.29 FTP      76Request: NLST -la
253.167554 120.48.30.29 10.173.63.31 FTP      105Response: 150 Here comes the directory
333.195450 120.48.30.29 10.173.63.31 FTP      90Response: 226 Directory send OK.
3816.121138 10.173.63.31 120.48.30.29 FTP      93Request: PORT 10,173,63,31,159,171
3916.147636 120.48.30.29 10.173.63.31 FTP      117Response: 200 PORT command successful.
4116.147897 10.173.63.31 120.48.30.29 FTP      74Request: NLST /
4416.201624 120.48.30.29 10.173.63.31 FTP      105Response: 150 Here comes the directory
4716.203943 120.48.30.29 10.173.63.31 FTP-DATA 226FTP Data: 160 bytes (PORT) (NLST /)
5116.227610 120.48.30.29 10.173.63.31 FTP      90Response: 226 Directory send OK.
```

图 8-60 连接服务器时调用 ls 命令截获的报文

(16) mdelete & prompt

mdelete 功能为删除服务器上的多个文件，prompt 则用于切换多文件操作时是否显示提示。

如图 8-61 所示，当客户端未连接服务器时，调用 mdelete 命令会提示错误。

```
ftp> mdelete a
Not connected.
ftp> 
```

图 8-61 未连接服务器时调用 mdelete 命令

如图 8-62 所示，mdelete 可接受至少一个参数，输入参数过少时将在给予一次提示后报错。

```
ftp> mdelete
(remote-files)
usage: mdelete remote-files
ftp> 
```

图 8-62 异常调用 mdelete 命令

在服务器工作目录下创建名为 a 和 c 的文件，以及名为 b 的文件夹。如图 8-63 所示，分别在提示模式打开和关闭的情况下执行命令“mdelete a b c d”，发现两种模式仅对于客户端而言有差别，客户端与服务器的交互过程（截获的报文）是一样的。

```

ftp> prompt
Interactive mode: on;
ftp> mdelete a b c d
---> PASV
227 Entering Passive Mode (120,48,30,29,145,245).
---> NLST a
---> PASV
227 Entering Passive Mode (120,48,30,29,155,225).
---> NLST b
---> PASV
227 Entering Passive Mode (120,48,30,29,129,187).
---> NLST c
---> PASV
227 Entering Passive Mode (120,48,30,29,179,239).
---> NLST d
mdelete a? y
---> DELE a
250 Delete operation successful.
mdelete c? y
---> DELE c
250 Delete operation successful.

```

(a) 开启提示模式

10.000000	10.173.63.31	120.48.30.29	FTP	72 Request: PASV
20.042714	120.48.30.29	10.173.63.31	FTP	117 Response: 227 Entering Passive Mode (120,48,30,29,145,245).
70.070166	10.173.63.31	120.48.30.29	FTP	74 Request: NLST a
80.112592	120.48.30.29	10.173.63.31	FTP-DATA	69 FTP Data: 3 bytes (PASV) (NLST a)
110.112920	120.48.30.29	10.173.63.31	FTP	105 Response: 150 Here comes the directory listing.
150.198672	120.48.30.29	10.173.63.31	FTP	90 Response: 226 Directory send OK.
170.198879	10.173.63.31	120.48.30.29	FTP	72 Request: PASV
180.242661	120.48.30.29	10.173.63.31	FTP	117 Response: 227 Entering Passive Mode (120,48,30,29,179,239).
230.273431	10.173.63.31	120.48.30.29	FTP	74 Request: NLST b
240.316587	120.48.30.29	10.173.63.31	FTP	105 Response: 150 Here comes the directory listing.
290.358678	120.48.30.29	10.173.63.31	FTP	90 Response: 226 Directory send OK.
310.358883	10.173.63.31	120.48.30.29	FTP	72 Request: PASV
320.400659	120.48.30.29	10.173.63.31	FTP	117 Response: 227 Entering Passive Mode (120,48,30,29,129,187).
360.428726	10.173.63.31	120.48.30.29	FTP	74 Request: NLST c
370.470727	120.48.30.29	10.173.63.31	FTP	105 Response: 150 Here comes the directory listing.
380.472622	120.48.30.29	10.173.63.31	FTP-DATA	69 FTP Data: 3 bytes (PASV) (NLST c)
440.554660	120.48.30.29	10.173.63.31	FTP	90 Response: 226 Directory send OK.
460.554857	10.173.63.31	120.48.30.29	FTP	72 Request: PASV
470.596634	120.48.30.29	10.173.63.31	FTP	117 Response: 227 Entering Passive Mode (120,48,30,29,179,239).
520.624848	10.173.63.31	120.48.30.29	FTP	74 Request: NLST d
550.668558	120.48.30.29	10.173.63.31	FTP	105 Response: 150 Here comes the directory listing.
590.710754	120.48.30.29	10.173.63.31	FTP	90 Response: 226 Directory send OK.
611.934098	10.173.63.31	120.48.30.29	FTP	74 Request: DELE a
621.976622	120.48.30.29	10.173.63.31	FTP	100 Response: 250 Delete operation successful.
642.679070	10.173.63.31	120.48.30.29	FTP	74 Request: DELE c
652.720716	120.48.30.29	10.173.63.31	FTP	100 Response: 250 Delete operation successful.

(b) 开启提示模式截获的报文

```

ftp> prompt
Interactive mode: off;
ftp> mdelete a b c d
---> PASV
227 Entering Passive Mode (120,48,30,29,142,249).
---> NLST a
---> PASV
227 Entering Passive Mode (120,48,30,29,218,145).
---> NLST b
---> PASV
227 Entering Passive Mode (120,48,30,29,157,161).
---> NLST c
---> PASV
227 Entering Passive Mode (120,48,30,29,150,231).
---> NLST d
---> DELE a
250 Delete operation successful.
---> DELE c
250 Delete operation successful.
ftp> []

```

(c) 关闭提示模式

7016.881298	10.173.63.31	120.48.30.29	FTP	72Request: PASV
7116.927037	120.48.30.29	10.173.63.31	FTP	117Response: 227 Entering Passive Mode (120,48,30,29,
7616.951114	10.173.63.31	120.48.30.29	FTP	74Request: NLST a
7716.992902	120.48.30.29	10.173.63.31	FTP	105Response: 150 Here comes the directory listing.
7816.993103	120.48.30.29	10.173.63.31	FTP-DATA	69FTP Data: 3 bytes (PASV) (NLST a)
8417.078855	120.48.30.29	10.173.63.31	FTP	90Response: 226 Directory send OK.
8617.079068	10.173.63.31	120.48.30.29	FTP	72Request: PASV
8717.123963	120.48.30.29	10.173.63.31	FTP	117Response: 227 Entering Passive Mode (120,48,30,29,
9217.176073	10.173.63.31	120.48.30.29	FTP	74Request: NLST b
9317.224869	120.48.30.29	10.173.63.31	FTP	105Response: 150 Here comes the directory listing.
9717.268168	120.48.30.29	10.173.63.31	FTP	90Response: 226 Directory send OK.
9917.268378	10.173.63.31	120.48.30.29	FTP	72Request: PASV
10117.311781	120.48.30.29	10.173.63.31	FTP	117Response: 227 Entering Passive Mode (120,48,30,29,
11548.790013	10.173.63.31	120.48.30.29	FTP	74Request: NLST c
11648.849142	120.48.30.29	10.173.63.31	FTP-DATA	69FTP Data: 3 bytes (PASV) (NLST c)
11948.851388	120.48.30.29	10.173.63.31	FTP	105Response: 150 Here comes the directory listing.
12348.893049	120.48.30.29	10.173.63.31	FTP	90Response: 226 Directory send OK.
12548.893168	10.173.63.31	120.48.30.29	FTP	72Request: PASV
12648.937040	120.48.30.29	10.173.63.31	FTP	117Response: 227 Entering Passive Mode (120,48,30,29,
13048.961137	10.173.63.31	120.48.30.29	FTP	74Request: NLST d
13149.003023	120.48.30.29	10.173.63.31	FTP	105Response: 150 Here comes the directory listing.
13649.085398	120.48.30.29	10.173.63.31	FTP	90Response: 226 Directory send OK.
13849.085596	10.173.63.31	120.48.30.29	FTP	74Request: DELE a
13949.128055	120.48.30.29	10.173.63.31	FTP	100Response: 250 Delete operation successful.
14149.128241	10.173.63.31	120.48.30.29	FTP	74Request: DELE c
14249.171206	120.48.30.29	10.173.63.31	FTP	100Response: 250 Delete operation successful.

(d) 关闭提示模式截获的报文

图 8-63 正常调用 mdelete 命令

(17) mdir & mls

mdir 和 mls 均可用来获取多个服务器目录和文件的信息，只不过前者获取到的是详细信息，后者获取到的是简略信息。

如图 8-64 所示，当客户端未连接服务器时，调用 mdir 和 mls 命令会提示错误。

```
ftp> mdir  
Not connected.  
ftp> mls  
Not connected.  
ftp> 
```

图 8-64 未连接服务器时调用 mdir 和 mls 命令

mdir 和 mls 均接受至少两个参数，参数过少会在给予一次提示后报错。它们将最后一个参数视为输出信息的位置，同样使用“-”表示输出在屏幕上；前面的参数则代表了目标目录或文件，使用“-”表示服务器当前工作目录。

图 8-65 展示了调用 mdir 和 mls 的结果，事实上 mdir (mls)命令就是多个 dir (ls)命令的组合体。

```
ftp> mdir - -a notexist /home -  
---> PORT 10,173,63,31,179,245  
200 PORT command successful. Consider using PASV.  
---> LIST  
150 Here comes the directory listing.  
-rw-r--r--    1 root      root          9 Apr 18 19:11 c  
drwxr-xr-x    2 root      root        4096 Apr 18 19:11 b  
-rw-r--r--    1 root      root         23 Apr 18 19:11 a  
226 Directory send OK.  
---> PORT 10,173,63,31,235,103  
200 PORT command successful. Consider using PASV.  
---> LIST -a  
150 Here comes the directory listing.  
-rw-r--r--    1 root      root          9 Apr 18 19:11 c  
drwxr-xr-x    2 root      root        4096 Apr 18 19:11 b  
drwxr-xr-x    3 root      root        4096 Apr 18 19:11 .  
drwxr-xr-x   11 root     root        4096 Apr 18 19:11 ..  
-rw-r--r--    1 root      root         23 Apr 18 19:11 a  
226 Directory send OK.  
---> PORT 10,173,63,31,144,81  
200 PORT command successful. Consider using PASV.  
---> LIST notexist  
150 Here comes the directory listing.  
226 Directory send OK.  
---> PORT 10,173,63,31,167,15  
200 PORT command successful. Consider using PASV.  
---> LIST /home  
150 Here comes the directory listing.  
drwxr-xr-x   11 root     root        4096 Apr 18 19:11 ubuntu  
226 Directory send OK.
```

(a) mdir 命令

10.000000	10.173.63.31	120.48.30.29	FTP	93Request: PORT 10,173,63,31,179,245
20.027982	120.48.30.29	10.173.63.31	FTP	117Response: 200 PORT command successful. Co
40.028180	10.173.63.31	120.48.30.29	FTP	72Request: LIST
70.081981	120.48.30.29	10.173.63.31	FTP	105Response: 150 Here comes the directory li
150.110055	120.48.30.29	10.173.63.31	FTP	90Response: 226 Directory send OK.
170.110323	10.173.63.31	120.48.30.29	FTP	93Request: PORT 10,173,63,31,235,103
180.140139	120.48.30.29	10.173.63.31	FTP	117Response: 200 PORT command successful. Co
200.140348	10.173.63.31	120.48.30.29	FTP	75Request: LIST -a
230.193056	120.48.30.29	10.173.63.31	FTP	105Response: 150 Here comes the directory li
250.195237	120.48.30.29	10.173.63.31	FTP-DATA	367FTP Data: 301 bytes (PORT) (LIST -a)
310.259074	120.48.30.29	10.173.63.31	FTP	90Response: 226 Directory send OK.
330.259369	10.173.63.31	120.48.30.29	FTP	92Request: PORT 10,173,63,31,144,81
340.286997	120.48.30.29	10.173.63.31	FTP	117Response: 200 PORT command successful. Co
360.287222	10.173.63.31	120.48.30.29	FTP	81Request: LIST notexist
400.338976	120.48.30.29	10.173.63.31	FTP	105Response: 150 Here comes the directory li
470.365352	120.48.30.29	10.173.63.31	FTP	90Response: 226 Directory send OK.
490.365612	10.173.63.31	120.48.30.29	FTP	92Request: PORT 10,173,63,31,167,15
500.395058	120.48.30.29	10.173.63.31	FTP	117Response: 200 PORT command successful. Co
510.395223	10.173.63.31	120.48.30.29	FTP	78Request: LIST /home
540.449065	120.48.30.29	10.173.63.31	FTP	105Response: 150 Here comes the directory li
620.517398	120.48.30.29	10.173.63.31	FTP	90Response: 226 Directory send OK.

(b) 调用 mdir 命令截获的报文

```

ftp> mls - -a notexist /home -
---> PORT 10,173,63,31,204,151
200 PORT command successful. Consider using PASV.
---> NLST
150 Here comes the directory listing.
c
b
a
226 Directory send OK.
---> PORT 10,173,63,31,204,105
200 PORT command successful. Consider using PASV.
---> NLST -a
150 Here comes the directory listing.
c
b
.
..
a
226 Directory send OK.
---> PORT 10,173,63,31,210,191
200 PORT command successful. Consider using PASV.
---> NLST notexist
150 Here comes the directory listing.
226 Directory send OK.
---> PORT 10,173,63,31,166,195
200 PORT command successful. Consider using PASV.
---> NLST /home
150 Here comes the directory listing.
/home/ubuntu
226 Directory send OK.

```

(c) mls 命令

```

6943.946145 10.173.63.31 120.48.30.29 FTP      93Request: PORT 10,173,63,31,204,151
7044.011584 120.48.30.29 10.173.63.31 FTP      117Response: 200 PORT command successful. Co
7244.011791 10.173.63.31 120.48.30.29 FTP      72Request: NLST
7544.077761 120.48.30.29 10.173.63.31 FTP      105Response: 150 Here comes the directory li
7844.079938 120.48.30.29 10.173.63.31 FTP-DATA  75FTP Data: 9 bytes (PORT) (NLST)
8244.117770 120.48.30.29 10.173.63.31 FTP      90Response: 226 Directory send OK.
8444.118017 10.173.63.31 120.48.30.29 FTP      93Request: PORT 10,173,63,31,204,105
8644.147420 120.48.30.29 10.173.63.31 FTP      117Response: 200 PORT command successful. Co
8844.147661 10.173.63.31 120.48.30.29 FTP      75Request: NLST -a
9244.225800 120.48.30.29 10.173.63.31 FTP      105Response: 150 Here comes the directory li
9444.239699 120.48.30.29 10.173.63.31 FTP-DATA  82FTP Data: 16 bytes (PORT) (NLST -a)
10044.303536 120.48.30.29 10.173.63.31 FTP      90Response: 226 Directory send OK.
10244.303839 10.173.63.31 120.48.30.29 FTP      93Request: PORT 10,173,63,31,210,191
10444.333403 120.48.30.29 10.173.63.31 FTP      117Response: 200 PORT command successful. Co
10644.333594 10.173.63.31 120.48.30.29 FTP      81Request: NLST notexist
11044.387461 120.48.30.29 10.173.63.31 FTP      105Response: 150 Here comes the directory li
11544.415453 120.48.30.29 10.173.63.31 FTP      90Response: 226 Directory send OK.
11744.415752 10.173.63.31 120.48.30.29 FTP      93Request: PORT 10,173,63,31,166,195
12044.445494 120.48.30.29 10.173.63.31 FTP      117Response: 200 PORT command successful. Co
12144.445671 10.173.63.31 120.48.30.29 FTP      78Request: NLST /home
12544.497993 120.48.30.29 10.173.63.31 FTP-DATA  80FTP Data: 14 bytes (PORT) (NLST /home)
12844.498959 120.48.30.29 10.173.63.31 FTP      105Response: 150 Here comes the directory li
13244.569526 120.48.30.29 10.173.63.31 FTP      90Response: 226 Directory send OK.

```

(d) 调用 mls 命令截获的报文

图 8-65 正常调用 mdir 和 mls 命令

(18) mget

mget 用于下载服务器上的多个文件。

如图 8-66 所示，当客户端未连接服务器时，调用 mget 命令会提示错误。

```

ftp> mget a b c
Not connected.
ftp> []

```

图 8-66 未连接服务器时调用 mget 命令

如图 8-67 所示，mget 至少接受一个参数，参数过少会在给予一次提示后报错。

```

ftp> mget
(remote-files)
usage: mget remote-files
ftp> []

```

图 8-67 异常调用 mget 命令

同样在服务器工作目录下创建名为 a 和 c 的文件，以及名为 b 的文件夹，在本地创建一个名为 a 的不同文件。如图 8-68 所示，设置断点值后执行“mget a b c d”，仅有 a 和 c 被下载到了本地，且断点值仅影响到第一个文件 a。

```

ftp> mget a b c d
---> PASV
227 Entering Passive Mode (120,48,30,29,175,153).
---> NLST a
---> PASV
227 Entering Passive Mode (120,48,30,29,144,45).
---> NLST b
---> PASV
227 Entering Passive Mode (120,48,30,29,204,35).
---> NLST c
---> PASV
227 Entering Passive Mode (120,48,30,29,169,37).
---> NLST d
mget a? y
local: a remote: a
---> PASV
227 Entering Passive Mode (120,48,30,29,172,201).
---> REST 2
350 Restart position accepted (2).
---> RETR a
150 Opening BINARY mode data connection for a (23 bytes).
226 Transfer complete.
21 bytes received in 0.00 secs (188.1451 kB/s)
mget c? y
local: c remote: c
---> PASV
227 Entering Passive Mode (120,48,30,29,177,227).
---> RETR c
150 Opening BINARY mode data connection for c (9 bytes).
226 Transfer complete.
9 bytes received in 0.00 secs (36.4691 kB/s)
ftp> []

```

(a) 终端内容

10.000000	10.173.63.31	120.48.30.29	FTP	72 Request: PASV
20.027434	120.48.30.29	10.173.63.31	FTP	117 Response: 227 Entering Passive Mode (120,48,30,29,175,153).
70.057587	10.173.63.31	120.48.30.29	FTP	74 Request: NLST a
80.105917	120.48.30.29	10.173.63.31	FTP	105 Response: 150 Here comes the directory listing.
100.106132	120.48.30.29	10.173.63.31	FTP-DATA	69 FTP Data: 3 bytes (PASV) (NLST a)
140.131667	120.48.30.29	10.173.63.31	FTP	99 Response: 226 Directory send OK.
160.131764	10.173.63.31	120.48.30.29	FTP	72 Request: PASV
180.151886	120.48.30.29	10.173.63.31	FTP	116 Response: 227 Entering Passive Mode (120,48,30,29,144,45).
230.181454	10.173.63.31	120.48.30.29	FTP	74 Request: NLST b
260.207595	120.48.30.29	10.173.63.31	FTP	105 Response: 150 Here comes the directory listing.
300.277869	120.48.30.29	10.173.63.31	FTP	99 Response: 226 Directory send OK.
320.278059	10.173.63.31	120.48.30.29	FTP	72 Request: PASV
330.307436	120.48.30.29	10.173.63.31	FTP	116 Response: 227 Entering Passive Mode (120,48,30,29,204,35).
380.333523	10.173.63.31	120.48.30.29	FTP	74 Request: NLST c
390.358543	120.48.30.29	10.173.63.31	FTP-DATA	69 FTP Data: 3 bytes (PASV) (NLST c)
420.359471	120.48.30.29	10.173.63.31	FTP	105 Response: 150 Here comes the directory listing.
460.385521	120.48.30.29	10.173.63.31	FTP	99 Response: 226 Directory send OK.
480.385717	10.173.63.31	120.48.30.29	FTP	72 Request: PASV
490.411266	120.48.30.29	10.173.63.31	FTP	116 Response: 227 Entering Passive Mode (120,48,30,29,169,37).
530.437556	10.173.63.31	120.48.30.29	FTP	74 Request: NLST d
550.465321	120.48.30.29	10.173.63.31	FTP	105 Response: 150 Here comes the directory listing.
590.533519	120.48.30.29	10.173.63.31	FTP	99 Response: 226 Directory send OK.
611.699730	10.173.63.31	120.48.30.29	FTP	72 Request: PASV
621.725365	120.48.30.29	10.173.63.31	FTP	117 Response: 227 Entering Passive Mode (120,48,30,29,172,201).
671.755462	10.173.63.31	120.48.30.29	FTP	74 Request: REST 2
681.783142	120.48.30.29	10.173.63.31	FTP	102 Response: 350 Restart position accepted (2).
701.783328	10.173.63.31	120.48.30.29	FTP	74 Request: RETR a
711.809231	120.48.30.29	10.173.63.31	FTP	125 Response: 150 Opening BINARY mode data connection for a (23 bytes)
731.809445	120.48.30.29	10.173.63.31	FTP-DATA	87 FTP Data: 21 bytes (PASV) (REST 2)
771.833213	120.48.30.29	10.173.63.31	FTP	99 Response: 226 Transfer complete.
802.912765	10.173.63.31	120.48.30.29	FTP	72 Request: PASV
812.939056	120.48.30.29	10.173.63.31	FTP	117 Response: 227 Entering Passive Mode (120,48,30,29,177,227).
862.965587	10.173.63.31	120.48.30.29	FTP	74 Request: RETR c
892.991429	120.48.30.29	10.173.63.31	FTP	124 Response: 150 Opening BINARY mode data connection for c (9 bytes).
943.061161	120.48.30.29	10.173.63.31	FTP	99 Response: 226 Transfer complete.

(b) 截获的报文

图 8-68 正常调用 mget 命令

(19) mkdir

mkdir 用于在服务器上创建目录。

如图 8-69 所示，当客户端未连接服务器时，调用 mkdir 命令会提示错误。

```
ftp> mkdir  
Not connected.
```

图 8-69 未连接服务器时调用 mkdir 命令

如图 8-70 所示，mkdir 仅接受一个参数，参数过少会在给予一次提示后报错，参数过多时仅采用第一个参数。

```
ftp> mkdir  
(directory-name)  
usage: mkdir directory-name  
ftp> mkdir e f g  
---> MKD e  
257 "/home/ubuntu/mget/e" create  
ftp> 
```

图 8-70 异常调用 mkdir 命令

进入服务器目录“/home/ubuntu/mkdir”，该目录下初始有名为 a 和 c 的文件，以及名为 b 的文件夹。如图 8-71 所示，尝试创建名为 b、c、d 文件，仅有最后一个操作被成功执行，说明 mkdir 指定的目录名不能与当前目录下已存在的文件名和目录名相同。

```

ftp> dir
---> PORT 10,173,63,31,205,59
200 PORT command successful. Consider using PASV.
---> LIST
150 Here comes the directory listing.
-rw-r--r--    1 root      root           9 Apr 19 14:53 c
drwxr-xr-x    2 root      root          4096 Apr 19 14:53 b
-rw-r--r--    1 root      root          23 Apr 19 14:53 a
226 Directory send OK.
ftp> mkdir b
---> MKD b
550 Create directory operation failed.
ftp> mkdir c
---> MKD c
550 Create directory operation failed.
ftp> mkdir d
---> MKD d
257 "/home/ubuntu/mkdir/d" create
ftp> 

```

(a) 终端内容

10.000000	10.173.63.31	120.48.30.29	FTP	92 Request: PORT 10,173,63,31,205,59
20.028034	120.48.30.29	10.173.63.31	FTP	117 Response: 200 PORT command successful. Consider using
40.028211	10.173.63.31	120.48.30.29	FTP	72 Request: LIST
80.104039	120.48.30.29	10.173.63.31	FTP	105 Response: 150 Here comes the directory listing.
110.122353	120.48.30.29	10.173.63.31	FTP-DATA	246 FTP Data: 180 bytes (PORT) (LIST)
150.130004	120.48.30.29	10.173.63.31	FTP	90 Response: 226 Directory send OK.
182.243997	10.173.63.31	120.48.30.29	FTP	73 Request: MKD b
202.272371	120.48.30.29	10.173.63.31	FTP	106 Response: 550 Create directory operation failed.
224.563878	10.173.63.31	120.48.30.29	FTP	73 Request: MKD c
244.592358	120.48.30.29	10.173.63.31	FTP	106 Response: 550 Create directory operation failed.
267.327900	10.173.63.31	120.48.30.29	FTP	73 Request: MKD d
287.356582	120.48.30.29	10.173.63.31	FTP	101 Response: 257 "/home/ubuntu/mkdir/d" create

(b) 截获的报文

图 8-71 正常调用 mkdir 命令

(20) mput

mput 用于向服务器上传多个文件。

如图 8-72 所示，当客户端未连接服务器时，调用 mput 命令会提示错误。

```

ftp> mput a b c
Not connected.
ftp> 

```

图 8-72 未连接服务器时调用 mput 命令

如图 8-73 所示，mput 至少接受一个参数，参数过少会在给予一次提示后报错。

```

ftp> mput
(local-files)
usage: mput local-files
ftp> 

```

图 8-73 异常调用 mput 命令

在本地创建名为 **a** 和 **c** 的文件，以及名为 **b** 的文件夹，在服务器工作目录下创建名为 **a** 的不同文件。如图 8-74 所示，设置断点值后执行“**mput a b c d**”，仅有 **a** 和 **c** 被上传到了服务器，且断点值仅影响到第一个文件 **a**。

```

ftp> restart 5
restarting at 5. execute get, put or append to initiate transfer
ftp> mput a b c d
mput a? y
local: a remote: a
---> PASV
227 Entering Passive Mode (120,48,30,29,144,203).
---> REST 5
350 Restart position accepted (5).
---> STOR a
150 Ok to send data.
226 Transfer complete.
5 bytes sent in 0.00 secs (41.3798 kB/s)
mput b? y
local: b remote: b
local: b: not a plain file.
mput c? y
local: c remote: c
---> PASV
227 Entering Passive Mode (120,48,30,29,167,185).
---> STOR c
150 Ok to send data.
226 Transfer complete.
9 bytes sent in 0.00 secs (90.6089 kB/s)
mput d? y
local: d remote: d
local: d: No such file or directory
ftp> []

```

(a) 终端内容

10.000000	10.173.63.31	120.48.30.29	FTP	72 Request: PASV
20.043377	120.48.30.29	10.173.63.31	FTP	117 Response: 227 Entering Passive Mode (120,48,30,29,144
70.073145	10.173.63.31	120.48.30.29	FTP	74 Request: REST 5
80.116971	120.48.30.29	10.173.63.31	FTP	102 Response: 350 Restart position accepted (5).
90.117136	10.173.63.31	120.48.30.29	FTP	74 Request: STOR a
100.161027	120.48.30.29	10.173.63.31	FTP	88 Response: 150 Ok to send data.
110.161221	10.173.63.31	120.48.30.29	FTP-DATA	71 FTP Data: 5 bytes (PASV) (REST 5)
180.249130	120.48.30.29	10.173.63.31	FTP	90 Response: 226 Transfer complete.
202.324060	10.173.63.31	120.48.30.29	FTP	72 Request: PASV
212.439588	120.48.30.29	10.173.63.31	FTP	117 Response: 227 Entering Passive Mode (120,48,30,29,167
262.489193	10.173.63.31	120.48.30.29	FTP	74 Request: STOR c
272.553725	120.48.30.29	10.173.63.31	FTP	88 Response: 150 Ok to send data.
292.553941	10.173.63.31	120.48.30.29	FTP-DATA	75 FTP Data: 9 bytes (PASV) (STOR c)
342.621721	120.48.30.29	10.173.63.31	FTP	90 Response: 226 Transfer complete.

(b) 截获的报文

图 8-74 正常调用 **mput** 命令

(21) newer

newer 用于从服务器更新文件。

如图 8-75 所示，当客户端未连接服务器时，调用 **newer** 命令会提示错误。

```
ftp> newer  
Not connected.  
ftp> [ ]
```

图 8-75 未连接服务器时调用 newer 命令

如图 8-76 所示，newer 仅接受 1-2 个参数，参数过少会在给予一次提示后报错，参数过多则只取前两个参数。

```
ftp> newer  
(remote-file)  
usage: newer remote-file [ local-file ]  
ftp> newer z y x w  
local: y remote: z  
---> PORT 10,173,63,31,156,89  
200 PORT command successful. Consider using PASV.  
---> RETR z  
550 Failed to open file.  
ftp> [ ]
```

图 8-76 异常调用 newer 命令

在服务器创建名为 a 的文件（本地不存在同名文件）。如图 8-77 所示，执行“newer a”后 a 被下载到了本地，再执行一次“newer a”发现不会传输文件数据。此时再在服务器创建名为 b 的文件，执行“newer b a”，发现本地 a 文件的内容变成了服务器上 b 文件的内容。

```

ftp> newer a
local: a remote: a
---> PORT 10,173,63,31,146,183
200 PORT command successful. Consider using PASV.
---> RETR a
150 Opening BINARY mode data connection for a (25 bytes).
226 Transfer complete.
25 bytes received in 0.00 secs (76.7738 kB/s)
ftp> newer a
---> MDTM a
213 20220419154758
Local file "a" is newer than remote file "a"
ftp> newer b a
---> MDTM b
213 20220419154841
local: a remote: b
---> PORT 10,173,63,31,220,17
200 PORT command successful. Consider using PASV.
---> RETR b
150 Opening BINARY mode data connection for b (23 bytes).
226 Transfer complete.
23 bytes received in 0.00 secs (77.9894 kB/s)
ftp> []

```

(a) 终端内容

10.000000	10.173.63.31	120.48.30.29	FTP	93 Request: PORT 10,173,63,31,146,183
20.062367	120.48.30.29	10.173.63.31	FTP	117 Response: 200 PORT command successful. Consider using
40.062570	10.173.63.31	120.48.30.29	FTP	74 Request: RETR a
80.134618	120.48.30.29	10.173.63.31	FTP-DATA	91 FTP Data: 25 bytes (PORT) (RETR a)
110.150397	120.48.30.29	10.173.63.31	FTP	125 Response: 150 Opening BINARY mode data connection for
150.210454	120.48.30.29	10.173.63.31	FTP	96 Response: 226 Transfer complete.
174.512677	10.173.63.31	120.48.30.29	FTP	74 Request: MDTM a
184.572464	120.48.30.29	10.173.63.31	FTP	86 Response: 213 20220419154758
13419.888929	10.173.63.31	120.48.30.29	FTP	74 Request: MDTM b
13519.950801	120.48.30.29	10.173.63.31	FTP	86 Response: 213 20220419154841
13719.951128	10.173.63.31	120.48.30.29	FTP	92 Request: PORT 10,173,63,31,220,17
13820.012679	120.48.30.29	10.173.63.31	FTP	117 Response: 200 PORT command successful. Consider using
14020.012888	10.173.63.31	120.48.30.29	FTP	74 Request: RETR b
14420.087372	120.48.30.29	10.173.63.31	FTP-DATA	89 FTP Data: 23 bytes (PORT) (RETR b)
14720.102669	120.48.30.29	10.173.63.31	FTP	125 Response: 150 Opening BINARY mode data connection for
15120.206701	120.48.30.29	10.173.63.31	FTP	90 Response: 226 Transfer complete.

(b) 截获的报文

图 8-77 正常调用 newer 命令

(22) open

open 用于建立与服务器的连接。

如图 8-78 所示，当客户端已连接到服务器时，调用 open 命令会提示错误。

```

ftp> open 120.48.30.116
Already connected to 120.48.30.29, use close first.
ftp> []

```

图 8-78 已连接服务器时调用 open 命令

如图 8-79 所示，open 仅接受一个参数，参数过少会在给予一次提示后报错，参数过多则直接报错。

```

ftp> open
(to)
usage: open host-name
ftp> open
(to) 1.1.1.1 2.2.2.2
usage: open host-name
ftp> open 1.1.1.1 2.2.2.2
usage: open host-name
ftp> []

```

图 8-79 异常调用 open 命令

如图 8-80 所示，正常调用 open 即可使客户端连上服务器，并自动开始登录过程。

```

ftp> open 120.48.30.29
Connected to 120.48.30.29.
220 (ylxFTP 1.0.0)
Name (120.48.30.29:hayate): []

```

(a) 终端内容

10.000000	10.173.63.31	120.48.30.29	TCP	7435864 → 21 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_
20.023402	120.48.30.29	10.173.63.31	TCP	7421 → 35864 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS
30.023457	10.173.63.31	120.48.30.29	TCP	6635864 → 21 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=22
40.047295	120.48.30.29	10.173.63.31	FTP	86 Response: 220 (ylxFTP 1.0.0)
50.047330	10.173.63.31	120.48.30.29	TCP	6635864 → 21 [ACK] Seq=1 Ack=21 Win=64256 Len=0 TSval=2

(b) 截获的报文

图 8-80 正常调用 open 命令

(23) put & send

put 和 send 是两条名字不同，但代码完全一样、功能也均为向服务器上传单个文件的命令，故本节中仅对 put 命令进行测试。

如图 8-81 所示，当客户端未连接服务器时，调用 put 命令会提示错误。

```

ftp> put
Not connected.
ftp> []

```

图 8-81 未连接服务器时调用 put 命令

如图 8-82 所示，put 可接受 1-2 个参数，输入参数过少时将在给予一次提示后报错，输入参数过多时将仅使用前两个参数。第一个参数指明了目标文件，第二个参数指明了将目标文件上传到服务器时使用的文件名。第二个参数的缺省值为第一个参数。

```
ftp> put  
(local-file)  
usage: put local-file [ remote-file ]  
ftp> put z y x w  
local: z remote: y  
local: z: No such file or directory  
ftp> 
```

图 8-82 异常调用 put 命令

在本地创建如图 8-83 所示的 a.txt 和 b.txt 文件（服务器工作目录无同名文件）。在客户端执行“put a.txt”后，服务器上出现了如图 8-83(a)所示的 a.txt。再执行“put b.txt a.txt”后，服务器上 a.txt 的内容变成了如图 8-83(b)所示的。最后尝试上传文件夹，执行“put /home a.txt”，发现执行失败，服务器上 a.txt 的内容也未发生变化。图 8-84 通过终端和报文展示了上述过程。

```
aaa  
aa
```

(a) a.txt

```
bbb  
bb
```

(b) b.txt

图 8-83 本地文件

```

ftp> put a.txt
local: a.txt remote: a.txt
---> PASV
227 Entering Passive Mode (120,48,30,29,192,127).
---> STOR a.txt
150 Ok to send data.
226 Transfer complete.
7 bytes sent in 0.00 secs (51.0145 kB/s)
ftp> put b.txt a.txt
local: b.txt remote: a.txt
---> PASV
227 Entering Passive Mode (120,48,30,29,196,161).
---> STOR a.txt
150 Ok to send data.
226 Transfer complete.
7 bytes sent in 0.00 secs (71.9572 kB/s)
ftp> put /home a.txt
local: /home remote: a.txt
local: /home: not a plain file.
ftp> []

```

(a) 终端显示的信息

32.541218	10.173.63.31	120.48.30.29	FTP	72 Request: PASV
42.567996	120.48.30.29	10.173.63.31	FTP	117 Response: 227 Entering Passive Mode (120,48,30,29,192,127)
92.594398	10.173.63.31	120.48.30.29	FTP	78 Request: STOR a.txt
102.624057	120.48.30.29	10.173.63.31	FTP	88 Response: 150 OK to send data.
112.624281	10.173.63.31	120.48.30.29	FTP-DATA	73 FTP Data: 7 bytes (PASV) (STOR a.txt)
172.692117	120.48.30.29	10.173.63.31	FTP	99 Response: 226 Transfer complete.
198.894440	10.173.63.31	120.48.30.29	FTP	72 Request: PASV
208.922137	120.48.30.29	10.173.63.31	FTP	117 Response: 227 Entering Passive Mode (120,48,30,29,196,161)
258.948214	10.173.63.31	120.48.30.29	FTP	78 Request: STOR a.txt
268.974083	120.48.30.29	10.173.63.31	FTP	88 Response: 150 OK to send data.
288.974295	10.173.63.31	120.48.30.29	FTP-DATA	73 FTP Data: 7 bytes (PASV) (STOR a.txt)
309.002106	120.48.30.29	10.173.63.31	FTP	99 Response: 226 Transfer complete.

(b) 截获的报文

图 8-84 连接服务器时调用 putt 命令

接下来测试断点续传（上传）功能。分别在本地和服务器上创建如图 8-85 所示的 whole.txt 和 part.txt，按照图 8-86 的过程进行断点续传（上传），完成后发现服务器上 part.txt 的内容变得与本地 whole.txt 的内容一样了。

```

abcdefg
hijklmn
opqrstuvwxyz

```

(a) 本地的 whole.txt

```

abcdefg
hijklmn

```

(b) 服务器上的 part.txt

图 8-85 断点续传（上传）相关文件

```
ftp> size part.txt
---> SIZE part.txt
213 16
ftp> restart 16
restarting at 16. execute get, put or append to initiate transfer
ftp> put whole.txt part.txt
local: whole.txt remote: part.txt
---> PASV
227 Entering Passive Mode (120,48,30,29,194,165).
---> REST 16
350 Restart position accepted (16).
---> STOR part.txt
150 Ok to send data.
226 Transfer complete.
14 bytes sent in 0.00 secs (123.1700 kB/s)
ftp> [ ]
```

(a) 终端显示的信息

10.000000	10.173.63.31	120.48.30.29	FTP	81Request: SIZE part.txt
20.025720	120.48.30.29	10.173.63.31	FTP	74Response: 213 16
412.338015	10.173.63.31	120.48.30.29	FTP	72Request: PASV
512.364603	120.48.30.29	10.173.63.31	FTP	117Response: 227 Entering Passive Mode (120,48,30,29,194,165)
1012.390730	10.173.63.31	120.48.30.29	FTP	75Request: REST 16
1112.414603	120.48.30.29	10.173.63.31	FTP	103Response: 350 Restart position accepted (16).
1212.414791	10.173.63.31	120.48.30.29	FTP	81Request: STOR part.txt
1312.440653	120.48.30.29	10.173.63.31	FTP	88Response: 150 Ok to send data.
1412.440848	10.173.63.31	120.48.30.29	FTP-DATA	80FTP Data: 14 bytes (PASV) (REST 16)
2012.508662	120.48.30.29	10.173.63.31	FTP	90Response: 226 Transfer complete.

(b) 截获的报文

图 8-86 断点续传（上传）

(24) reget

reget 能够实现断点续传（下载）。除了能根据本地文件大小自动设置断点值外，reget 与 restart 和 get 的组合并无太大区别。

如图 8-87 所示，当客户端未连接服务器时，调用 reget 命令会提示错误。

```
ftp> reget
Not connected.
ftp> [ ]
```

图 8-87 未连接服务器时调用 reget 命令

如图 8-88 所示，reget 可接受 1-2 个参数，输入参数过少时将在给予一次提示后报错，输入参数过多时将仅使用前两个参数。第一个参数指明了目标文件，第二个参数指明了将目标文件下载到本地时使用的文件名。第二个参数的缺省值为第一个参数。此外，图 8-88 还体现了当本地不存在名为第二个参数的文件时，

reget 命令无法正确执行。

```
ftp> reget  
(remote-file)  
usage: reget remote-file [ local-file ]  
ftp> reget p q r s  
local: q: No such file or directory
```

图 8-88 异常调用 reget 命令

分别在服务器和本地创建如图 8-89 所示的 whole.txt 和 part.txt，执行“reget part.txt whole.txt”后发现本地 part.txt 的内容变得与服务器 whole.txt 的内容一样了。图 8-90 展现了上述过程。

```
abcdefg  
hijklmn  
opqrstuvwxyz
```

(a) 服务器的 whole.txt

```
abcdefg  
hijklmn
```

(b) 本地的 part.txt

图 8-89 reget 相关文件

```
ftp> reget part.txt whole.txt  
local: whole.txt remote: part.txt  
---> PORT 10,173,63,31,226,93  
200 PORT command successful. Consider using PASV.  
---> REST 16  
350 Restart position accepted (16).  
---> RETR part.txt  
150 Opening BINARY mode data connection for part.txt (30 bytes).  
226 Transfer complete.  
14 bytes received in 0.00 secs (99.0716 kB/s)  
ftp> []
```

(a) 终端显示的信息

10.000000	10.173.63.31	120.48.30.29	FTP	92 Request: PORT 10,173,63,31,226,93
20.028587	120.48.30.29	10.173.63.31	FTP	117 Response: 200 PORT command successful. Consider using
40.028772	10.173.63.31	120.48.30.29	FTP	75 Request: REST 16
50.054767	120.48.30.29	10.173.63.31	FTP	103 Response: 350 Restart position accepted (16).
60.054937	10.173.63.31	120.48.30.29	FTP	81 Request: RETR part.txt
90.106393	120.48.30.29	10.173.63.31	FTP-DATA	80 FTP Data: 14 bytes (PORT) (REST 16)
140.107865	120.48.30.29	10.173.63.31	FTP	132 Response: 150 Opening BINARY mode data connection for
180.176023	120.48.30.29	10.173.63.31	FTP	90 Response: 226 Transfer complete.

(b) 截获的报文

图 8-90 正确执行 reget 命令

(25) remotehelp

remotehelp 用于获取 FTP 底层命令相关信息, 图 8-91 展示了它的使用方法。

```
ftp> remotehelp
Not connected.
ftp> [ ]
```

(a) 未连接服务器时调用 remotehelp

```
ftp> remotehelp
---> HELP
214-The following commands are recognized.
ABOR ACCT ALLO APPE CDUP CWD  DELE FEAT HELP LIST MDTM MKD  NLST NOOP
OPTS PASS PASV PORT PWD  QUIT REIN REST RETR RMD  RNFR RNTO SIZE STAT
STOR STOU SYST TYPE USER XCUP XCWD XMKD XPWD XRMD
214 Help OK.
ftp> [ ]
```

(b) 连接服务器后调用 remotehelp

2920.098288	10.173.63.31	120.48.30.29	FTP	72 Request: HELP
3020.124510	120.48.30.29	10.173.63.31	FTP	320 Response: 214-The following commands are recognized.

(c) 截获的报文

图 8-91 remotehelp 命令

(26) rename

rename 命令用于重命名服务器上的文件。

如图 8-92 所示, 当客户端未连接服务器时, 调用 rename 命令会提示错误。

```
ftp> rename
Not connected.
ftp> [ ]
```

图 8-92 未连接服务器时调用 rename 命令

如图 8-93 所示, rename 接受且仅接受两个参数, 输入参数过少时将在给予一次提示后报错, 输入参数过多时将仅使用前两个参数。第一个参数指明了旧文件名, 第二个参数指明了新文件名。

```

ftp> rename
(from-name)
rename from-name to-name
ftp> rename
(from-name) a
(to-name) b
---> RNFR a
350 Ready for RNTO.
---> RNTO b
250 Rename successful.
ftp> rename
(from-name) b a
---> RNFR b
350 Ready for RNTO.
---> RNTO a
250 Rename successful.
ftp> rename b a c d
---> RNFR b
550 RNFR command failed.
ftp> []

```

(a) 终端显示的信息

155.468049	10.173.63.31	120.48.30.29	FTP	74 Request: RNFR a
165.494151	120.48.30.29	10.173.63.31	FTP	87 Response: 350 Ready for RNTO.
185.494380	10.173.63.31	120.48.30.29	FTP	74 Request: RNTO b
195.519961	120.48.30.29	10.173.63.31	FTP	90 Response: 250 Rename successful.
2114.098059	10.173.63.31	120.48.30.29	FTP	74 Request: RNFR b
2214.126387	120.48.30.29	10.173.63.31	FTP	87 Response: 350 Ready for RNTO.
2414.126557	10.173.63.31	120.48.30.29	FTP	74 Request: RNTO a
2514.152135	120.48.30.29	10.173.63.31	FTP	90 Response: 250 Rename successful.
3124.769915	10.173.63.31	120.48.30.29	FTP	74 Request: RNFR b
3224.796129	120.48.30.29	10.173.63.31	FTP	92 Response: 550 RNFR command failed.

(b) 截获的报文

图 8-93 rename 命令

(27) size

size 命令用于查询服务器上某文件的大小。

如图 8-94 所示，当客户端未连接服务器时，调用 size 命令会提示错误。

```

ftp> rename
Not connected.
ftp> []

```

图 8-94 未连接服务器时调用 size 命令

如图 8-95 所示，size 接受且仅接受一个参数，输入参数过少时将在给予一次提示后报错，输入参数过多时将仅使用第一个参数。

```
ftp> size  
(filename)  
usage: size filename  
ftp> size  
(filename) a  
---> SIZE a  
213 25  
ftp> size notexist a b  
---> SIZE notexist  
550 Could not get file size.  
ftp> []
```

(a) 终端显示的信息

33.016540	10.173.63.31	120.48.30.29	FTP	74 Request: SIZE a
43.044112	120.48.30.29	10.173.63.31	FTP	74 Response: 213 25
88.774623	10.173.63.31	120.48.30.29	FTP	81 Request: SIZE notexist
98.801993	120.48.30.29	10.173.63.31	FTP	96 Response: 550 Could not get file size.

(b) 截获的报文

图 8-95 size 命令

(28) status

status 命令用于查看当前客户端的状态，图 8-96 展示了它的使用方法。

```
ftp> status  
Connected to 120.48.30.29.  
Type: binary; Debugging: on;  
Passive mode: off;  
Interactive mode: on;  
ftp> close  
---> QUIT  
221 Goodbye.  
ftp> status  
Not connected.  
Type: binary; Debugging: on;  
Passive mode: off;  
Interactive mode: on;  
ftp> []
```

图 8-96 status 命令

(29) system

system 命令用于获取服务器的系统环境，图 8-97 展示了它的使用方法。

```
ftp> system  
Not connected.  
ftp> []
```

(a) 未连接服务器时调用 system

```
ftp> system  
---> SYST  
215 UNIX Type: L8  
ftp> []
```

(b) 连接服务器后调用 system

30.680143	10.173.63.31	120.48.30.29	FTP	72 Request: SYST
40.705946	120.48.30.29	10.173.63.31	FTP	85 Response: 215 UNIX Type: L8

(c) 截获的报文

图 8-97 system 命令

(30) type

type 命令用于获取显示/改变传输模式，图 8-98 展示了它的使用方法。

```
ftp> type ascii  
Not connected.  
ftp> []
```

(a) 未连接服务器时调用 type

```
ftp> type  
Using binary mode to transfer files.  
ftp> type ascii  
---> TYPE A  
200 Switching to ASCII mode.  
ftp> type binary  
---> TYPE I  
200 Switching to Binary mode.  
ftp> type aa  
aa: Unknown mode  
ftp> type a b c  
Usage: type [ ascii | binary ]  
ftp> []
```

(b) 连接服务器后调用 type

115.292070	10.173.63.31	120.48.30.29	FTP	74 Request: TYPE A
125.318055	120.48.30.29	10.173.63.31	FTP	96 Response: 200 Switching to ASCII mode.
1611.512043	10.173.63.31	120.48.30.29	FTP	74 Request: TYPE I
1711.536079	120.48.30.29	10.173.63.31	FTP	97 Response: 200 Switching to Binary mode.

(c) 截获的报文

图 8-98 type 命令

(31) user

user 用于向服务器提出登录请求。

如图 8-99 所示，当客户端未连接服务器时，调用 user 命令会提示错误。

```

ftp> user ubuntu
Not connected.
ftp> []

```

图 8-99 未连接服务器时调用 user 命令

如图 8-100 所示，user 接受且 1-3 个参数，未输入参数时将在给予一次提示后报错，输入参数过多时将直接报错。

```

ftp> user
(username)
usage: user username [password] [account]
ftp> user
(username) ubuntu
---> USER ubuntu
331 Please specify the password.
Password:
---> PASS XXXX
230 Login successful.
---> SYST
215 UNIX Type: L8
---> TYPE I
200 Switching to Binary mode.
ftp> user another_user password acc bbb ww
usage: user username [password] [account]
ftp> user another_user password
---> USER another_user
530 Can't change to another user.
Login failed.

```

(a) 终端显示的信息

10.000000	10.173.63.31	120.48.30.29	FTP	79 Request: USER ubuntu
30.027966	120.48.30.29	10.173.63.31	FTP	100 Response: 331 Please specify the password.
54.203755	10.173.63.31	120.48.30.29	FTP	80 Request: PASS ylx0324
64.231508	120.48.30.29	10.173.63.31	FTP	89 Response: 230 Login successful.
84.231689	10.173.63.31	120.48.30.29	FTP	72 Request: SYST
94.259430	120.48.30.29	10.173.63.31	FTP	85 Response: 215 UNIX Type: L8
114.259593	10.173.63.31	120.48.30.29	FTP	74 Request: TYPE I
124.287517	120.48.30.29	10.173.63.31	FTP	97 Response: 200 Switching to Binary mode.
3354.163127	10.173.63.31	120.48.30.29	FTP	85 Request: USER another_user
3454.190741	120.48.30.29	10.173.63.31	FTP	101 Response: 530 Can't change to another user.

(b) 截获的报文

图 8-100 user 命令

(三) 带 NAT 穿透的客户端与带 NAT 穿透的服务器连接

本项测试由 PC 运行服务器程序，ESC1 运行客户端程序，ESC2 运行 TURN 服务器程序。

在 ESC2 上开启报文截获，并令 ESC1 执行一些常见的命令。由图 8-101 可

知，NAT 穿透模块正常工作，ESC2 截获的报文也体现了它在 FTP 客户端和服务器之间承担了报文转发工作。

```
root@instance-617qxkqm:/home/ubuntu# ./cl eth0 111.20.226.11
Connected to 111.20.226.11.
220 (ylxFTP 2.0.0)
Name (120.48.30.116:root): ubuntuPC
331 Please specify the password.
Password:
230 Login successful.
215 UNIX Type: L8
200 Switching to Binary mode.
ftp> debug
Debugging on.
ftp> passive
NAT traversal service doesn't support active mode.
Passive mode: on;
ftp> dir /home
---> TYPE A
200 Switching to ASCII mode.
---> PASV
250 38693
---> LIST /home
150 Here comes the directory listing.
drwxr-x---    18 hayate    hayate        4096 Apr 20 09:12 hayate
226 Directory send OK.
ftp> quote STAT
---> STAT
211-FTP server status:
    NAT Server ::ffff:120.48.30.116
    Logged in as ubuntuPC
    TYPE: ascii
    No session bandwidth limit
    Session timeout in seconds is 300
    Control connection is plain text
    Data connections will be plain text
    At session startup, client count was 1
    ylxFTP 2.0.0 - Kuro-chan Daisuki.
211 End of status
ftp> pwd
---> PWD
257 "/home/hayate/exp8" is the current directory
ftp> quit
---> QUIT
221 Goodbye.
```

(a) ESC1 终端显示的信息

1139.877505	120.48.30.29	192.168.16.4	TCP	6647808 → 55531 [ACK] Seq=1 Ack=21 Win=64256 Len=0
11812.749071	120.48.30.29	192.168.16.4	TCP	8147808 → 55531 [PSH, ACK] Seq=1 Ack=21 Win=64256 Len=0
11912.749102	192.168.16.4	120.48.30.29	TCP	6655531 → 47808 [ACK] Seq=21 Ack=16 Win=65280 Len=0
12012.749167	192.168.16.4	111.20.226.11	TCP	8153583 → 14168 [PSH, ACK] Seq=1 Ack=21 Win=65152 Len=0
12112.773089	111.20.226.11	192.168.16.4	TCP	6614168 → 53583 [ACK] Seq=21 Ack=16 Win=64256 Len=0
12212.773201	111.20.226.11	192.168.16.4	TCP	10014168 → 53583 [PSH, ACK] Seq=21 Ack=16 Win=64256
12312.773206	192.168.16.4	111.20.226.11	TCP	6653583 → 14168 [ACK] Seq=16 Ack=55 Win=65152 Len=0
12412.773238	192.168.16.4	120.48.30.29	TCP	10095531 → 47808 [PSH, ACK] Seq=21 Ack=16 Win=65280
12512.773323	120.48.30.29	192.168.16.4	TCP	6647808 → 55531 [ACK] Seq=16 Ack=55 Win=64256 Len=0
12814.784374	120.48.30.29	192.168.16.4	TCP	8047808 → 55531 [PSH, ACK] Seq=16 Ack=55 Win=64256
12914.784403	192.168.16.4	120.48.30.29	TCP	6655531 → 47808 [ACK] Seq=55 Ack=30 Win=65280 Len=0
13014.784439	192.168.16.4	111.20.226.11	TCP	80953583 → 14168 [PSH, ACK] Seq=16 Ack=55 Win=65152
13114.807183	111.20.226.11	192.168.16.4	TCP	6614168 → 53583 [ACK] Seq=55 Ack=30 Win=64256 Len=0
13214.811119	111.20.226.11	192.168.16.4	TCP	8914168 → 53583 [PSH, ACK] Seq=55 Ack=30 Win=64256
13314.811124	192.168.16.4	111.20.226.11	TCP	6653583 → 14168 [ACK] Seq=30 Ack=78 Win=65152 Len=0
13414.811146	192.168.16.4	120.48.30.29	TCP	8995531 → 47808 [PSH, ACK] Seq=55 Ack=30 Win=65280
13514.811238	120.48.30.29	192.168.16.4	TCP	6647808 → 55531 [ACK] Seq=30 Ack=78 Win=64256 Len=0
13614.811291	120.48.30.29	192.168.16.4	TCP	7247808 → 55531 [PSH, ACK] Seq=30 Ack=78 Win=64256
13714.811293	192.168.16.4	120.48.30.29	TCP	6655531 → 47808 [ACK] Seq=78 Ack=36 Win=65280 Len=0
13814.811308	192.168.16.4	111.20.226.11	TCP	7253583 → 14168 [PSH, ACK] Seq=30 Ack=78 Win=65152
13914.835119	111.20.226.11	192.168.16.4	TCP	6614168 → 53583 [ACK] Seq=78 Ack=36 Win=64256 Len=0
14014.835194	111.20.226.11	192.168.16.4	TCP	8514168 → 53583 [PSH, ACK] Seq=78 Ack=36 Win=64256
14114.835198	192.168.16.4	111.20.226.11	TCP	6653583 → 14168 [ACK] Seq=36 Ack=97 Win=65152 Len=0

(b) ESC2 截获的报文

图 8-101 NAT 穿透

七、实验结论

(一) 聊天程序

本网络聊天程序程序成功在 C/S 模式基础上实现了客户端之间消息通信的基本架构，能够满足用户的群聊和一对聊天需求，也能够进行文件的传输。

(二) FTP 程序

本 FTP 程序成功在 FTP 协议的基础上实现了主机之间的数据传输，能够满足用户的日常使用需求。

八、总结及心得体会

我实现该网络聊天程序的兴趣受到 SDN 课程第一次作业的激发，因为刚接触 socket 模块，很多东西都不了解，在查阅相关资料时阅览到很多大佬利用 socket 模块实现很多有意思的东西，我对此网络聊天程序产生了兴趣。然而到真正实现的过程时，我才发现自己很菜，利用 python 进行导包和对导入包的使用出现了大大小小的错误，因为自己了解到 python 对很多功能的实现都有一定封装好的库，可以直接使用，但是自己对 python 的使用不是很熟练，就在实现过程中看过很多博客文章来学习对应知识。本次实验给我带来很多收获，对我 python 程序设计能力具有了一定程度的促进提高，也让我掌握 socket 编程方面的相关内容，即便自己对这些内容的记忆不是很牢靠，但是我学会了具体的使用方法，让我能够在短时间学习知识及强化使用，在这个过程中我和同学之间交流了很多，

同学们也给我提了许多建议。

—— 苏贵林

最后一个实验的工作量十分庞大。为了完成本次实验，我先花了好长时间去学习 `Socket` 模块的使用方法，之后又费了很大功夫编写代码。但是，本次实验带给我的收获也是丰厚的。我不仅掌握了 `Socket` 模块的相关知识，编写、调试复杂程序的能力也有了长足的进步，更重要的是我学会了面对一无所知的任务时该如何行动。

—— 袁龙骥

附件

(一) 聊天程序

1. 源码文件

```
//utils 工具程序

from Crypto.Cipher import AES
from Crypto import Random
import struct
import json

max_buff_size = 1024
key = b'fdj27pFJ992FkHQb'      # 秘钥

def encrypt(data):
    # 生成长度等于 AES 块大小的不可重复的密钥向量
    code = Random.new().read(AES.block_size)
    # 使用 key 和 code 初始化 AES 对象，使用 MODE_CFB 模式
    cipher = AES.new(key, AES.MODE_CFB, code)
    # 加密的明文长度必须为 16 的倍数，如果长度不为 16 的倍数，则需要补足
    # 为 16 的倍数
```

```

# 将 code 密钥向量加到加密的密文开头，一起传输
return code + cipher.encrypt(data)

def decrypt(data):
    return AES.new(key, AES.MODE_CFB, data[:16]).decrypt(data[16:])

def pack(data):
    return struct.pack('>H', len(data)) + data #在数据报头封装数据包的数据大小

def send(socket, data_dict):
    socket.send(pack(encrypt(json.dumps(data_dict).encode('utf-8'))))
    # json.dumps() 函数是将一个 Python 数据类型列表进行 json 格式的编码
    # (json.dumps()函数是将字典转化为字符串)
    # json.loads()函数是将 json 格式数据转换为字典 (json.loads()函数是将字符串转
    # 化为字典)

def recv(socket):
    data = b"
    surplus = struct.unpack('>H', socket.recv(2))[0]#首先按无符号短整型解析接收
    到的 socket 消息的前两个字节
    socket.settimeout(5)#进入非阻塞状态来进行接收大量数据
    while surplus:#当前剩余未传输完的数据大小
        recv_data = socket.recv(max_buff_size if surplus > max_buff_size else
        surplus)
        data += recv_data
        surplus -= len(recv_data)
    socket.settimeout(None)#继续进入阻塞状态，以保证服务器与所连接对象都
    能进行通信并传输数据

```

```
    return json.loads(decrypt(data))#对数据进行加密

//服务器端程序
import socketserver
import pickle
import time

import utils

users = None
history = None

def load_users():
    try:
        return pickle.load(open('users.dat', 'rb'))
    except:
        return {}

def register(usr, pwd):
    if usr not in users.keys():
        users[usr] = pwd
        save_users()
        return True
    else:
        return False

def validate(usr, pwd):
    if usr in users.keys() and users[usr] == pwd:
        return True
    return False
```

```

def save_users():
    pickle.dump(users, open('users.dat', 'wb'))

def load_history():
    try:
        return pickle.load(open('history.dat', 'rb'))
    except:
        return {}

def get_key(u1, u2):
    return (u1, u2) if (u2, u1) not in history.keys() else (u2, u1)

def append_history(sender, receiver, msg):
    if receiver == ":":
        key = ("")
    else:
        key = get_key(sender, receiver)
    if key not in history.keys():
        history[key] = []
    history[key].append((sender, time.strftime("%m 月 %d 日 %H:%M",
                                              time.localtime(time.time())), msg))
    save_history()

def get_history(sender, receiver):
    if receiver == ":":
        key = ("")
    else:
        key = get_key(sender, receiver)
    return history[key] if key in history.keys() else []

```

```

def save_history():
    pickle.dump(history, open('history.dat', 'wb'))


class Handler(socketserver.BaseRequestHandler):
    clients = {}

    def setup(self):#准备请求处理， 默认什么都不做， 进行初始化，在
        StreamRequestHandler 中会创建类似的对象来读/写 socket
        self.user =
        self.file_peer =
        self.authed = False #授权标记

    def handle(self):#处理请求， 解析传入的请求， 处理数据，并发送响应， 用于
        通信循环， 处理 socket 请求
        while True:
            data = utils.recv(self.request)#self.request 为客户端和服务端的连接
            对象， 用于发送数据， 接收数据
            if not self.authed:
                self.user = data['user']
                if data['cmd'] == 'login':#登录
                    if validate(data['user'], data['pwd']):
                        utils.send(self.request, {'response': 'ok'})
                        self.authed = True
                    for user in Handler.clients.keys():#进一步进行用户信
                        息验证
                        utils.send(Handler.clients[user].request, {'type':
                            'peer_joined', 'peer': self.user})
                    Handler.clients[self.user] = self
            else:

```

```

        utils.send(self.request, {'response': 'fail', 'reason': '账号
或密码错误！'})

    elif data['cmd'] == 'register':#注册
        if register(data['user'], data['pwd']):
            utils.send(self.request, {'response': 'ok'})
        else:
            utils.send(self.request, {'response': 'fail', 'reason': '账号
已存在！'})

    else:
        if data['cmd'] == 'get_users':#获取用户
            users = []
            for user in Handler.clients.keys():
                if user != self.user:
                    users.append(user)
            utils.send(self.request, {'type': 'get_users', 'data': users})

        elif data['cmd'] == 'get_history':
            utils.send(self.request, {'type': 'get_history', 'peer': data['peer'],
            'data': get_history(self.user, data['peer'])})

        elif data['cmd'] == 'chat' and data['peer'] != ":":
            utils.send(Handler.clients[data['peer']].request, {'type': 'msg',
            'peer': self.user, 'msg': data['msg']})
            append_history(self.user, data['peer'], data['msg'])

        elif data['cmd'] == 'chat' and data['peer'] == ":":
            for user in Handler.clients.keys():
                if user != self.user:
                    utils.send(Handler.clients[user].request, {'type':
                    'broadcast', 'peer': self.user, 'msg': data['msg']})
                    append_history(self.user, ", data['msg'])"

        elif data['cmd'] == 'file_request':
            Handler.clients[data['peer']].file_peer = self.user

```

```

        utils.send(Handler.clients[data['peer']].request,      {'type':
'file_request', 'peer': self.user, 'filename': data['filename'], 'size': data['size'], 'md5':
data['md5']})

        elif data['cmd'] == 'file_deny' and data['peer'] == self.file_peer:
            self.file_peer = ""
            utils.send(Handler.clients[data['peer']].request,      {'type':
'file_deny', 'peer': self.user})

        elif data['cmd'] == 'file_accept' and data['peer'] == self.file_peer:
            self.file_peer = ""
            utils.send(Handler.clients[data['peer']].request,      {'type':
'file_accept', 'ip': self.client_address[0]})

        elif data['cmd'] == 'close':
            self.finish()

```

```

def finish(self):#环境清理，如果 setup 产生异常，则不会执行 finish()

    if self.authed:
        self.authed = False
        if self.user in Handler.clients.keys():
            del Handler.clients[self.user]
        for user in Handler.clients.keys():
            utils.send(Handler.clients[user].request, {'type': 'peer_left', 'peer':
self.user})

    if __name__ == '__main__':
        users = load_users()#加载在线用户
        history = load_history()#加载聊天记录
        #循环地从半连接池中取出链接请求与其建立双向链接，拿到链接对象进行
        #通信循环，从而实现并发响应
        app = socketserver.ThreadingTCPServer(('0.0.0.0', 8888), Handler)

```

#0.0.0.0 表示不确定所有地址，相当于 socket 连接只绑定端口，让路由表决
定传到哪个 ip

```
app.serve_forever()

app_file = socketserver.ThreadingTCPServer(('0.0.0.0', 8889), Handler)

app_file.serve_forever()

//客户端程序

import tkinter.filedialog

import tkinter.messagebox

import tkinter as tk

import threading

import hashlib

import socket

import time

import sys

import os

import utils

class Login_win:

    def show(self):
        self.win.mainloop()

    def destroy(self):
        self.win.destroy()#退出窗口

    def __init__(self):
        self.win = tk.Tk()

        self.user = tk.StringVar()

        self.pwd = tk.StringVar()
```

```
# 加载图片  
self.canvas = tkinter.Canvas(width=320, height=240, bg=None)  
self.image_file = tkinter.PhotoImage(file="1.png")  
self.image      = self.canvas.create_image(0,      0,      anchor='nw',  
image=self.image_file)  
self.canvas.pack()  
  
self.win.geometry("320x240")  
self.win.title("登录 v1.0.0")  
self.win.resizable(width=False, height=False)  
  
self.label1 = tk.Label(self.win)  
self.label1.place(relx=0.055, rely=0.1, height=31, width=70)  
self.label1.configure(text=' 账 号 ',bg='lightskyblue', fg='white',  
font=('Arial', 12), width=1, height=1)  
  
self.entry_user = tk.Entry(self.win)  
self.entry_user.place(relx=0.28, rely=0.11, height=31, relwidth=0.554)  
self.entry_user.configure(textvariable=self.user)  
  
self.label2 = tk.Label(self.win)  
self.label2.place(relx=0.055, rely=0.27, height=31, width=70)  
self.label2.configure(text=' 密 码 ',bg='lightskyblue', fg='white',  
font=('Arial', 12), width=1, height=1)  
  
self.entry_pwd = tk.Entry(self.win)  
self.entry_pwd.place(relx=0.28, rely=0.28, height=31, relwidth=0.554)  
self.entry_pwd.configure(show="*")  
self.entry_pwd.configure(textvariable=self.pwd)
```

```
    self.btn_login = tk.Button(self.win)
    self.btn_login.place(relx=0.13, rely=0.6, height=32, width=88)
    self.btn_login.configure(text='登录',bg='Tan',font=('Arial', 12))

    self.btn_reg = tk.Button(self.win)
    self.btn_reg.place(relx=0.6, rely=0.6, height=32, width=88)
    self.btn_reg.configure(text='注册',bg='Tan',font=('Arial', 12))

class Main_win:
    closed_fun = None

    def show(self):
        self.win.mainloop()

    def destroy(self):
        try:
            self.closed_fun()
        except:
            pass
        self.win.destroy()

    def __init__(self):
        self.win = tk.Tk()
        self.win.protocol('WM_DELETE_WINDOW', self.destroy)
        self.win.geometry("480x320")

        self.win.title("聊天室")
        self.win.resizable(width=False,height=False)
```

```
self.win.configure(background='pink')

self.msg = tk.StringVar()
self.name = tk.StringVar()

self.user_list = tk.Listbox(self.win)
self.user_list.place(relx=0.75, rely=0.15, relheight=0.72, relwidth=0.23)

self.label1 = tk.Label(self.win)
self.label1.place(relx=0.75, rely=0.075, height=21, width=101)
self.label1.configure(text='在线用户列表')

self.history = tk.Text(self.win)
self.history.place(relx=0.02, rely=0.24, relheight=0.63, relwidth=0.696)
self.history.configure(state='disabled')

self.entry_msg = tk.Entry(self.win)
self.entry_msg.place(relx=0.02, rely=0.9, height=24, relwidth=0.59)
self.entry_msg.configure(textvariable=self.msg)

self.btn_send = tk.Button(self.win)
self.btn_send.place(relx=0.62, rely=0.89, height=28, width=45)
self.btn_send.configure(text='发送')

self.btn_file = tk.Button(self.win)
self.btn_file.place(relx=0.752, rely=0.89, height=28, width=108)
self.btn_file.configure(text='发送文件')
self.btn_file.configure(state='disabled')
```

```
    self.label2 = tk.Label(self.win)
    self.label2.place(relx=0.24, rely=0.0, height=57, width=140)
    self.label2.configure(textvariable=self.name)
```

```
login_win = None
main_win = None
my_socket = None
user_name = ""
current_session = "#当前会话"
users = {}
filename = ""
filename_short = ""
file_transfer_pending = False
```

```
server_ip = "119.3.66.178"
server_port = "8888"
```

```
def close_socket():
    utils.send(my_socket, {'cmd': 'close'})
    my_socket.shutdown(2) #关闭单方向的通信管道,2 代表禁止下次的数据
    读取和写入
    my_socket.close()
```

```
def on_btn_login_clicked():
    global my_socket, user_name, login_win, main_win
    my_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #创
    建 socket 连接对象
    my_socket.settimeout(5)
```

```

if login_win.user.get() != " and login_win.pwd != "#当用户名框和密码框
不为空

    my_socket.connect((server_ip, int(server_port)))
    utils.send(my_socket, {'cmd': 'login', 'user': login_win.user.get(), 'pwd':
hashlib.sha1(login_win.pwd.get().encode('utf-8')).hexdigest()})

    server_response = utils.recv(my_socket)
    if server_response['response'] == 'ok':

        user_name = login_win.user.get()
        login_win.destroy()
        main_win = Main_win()
        main_win.closed_fun = on_closed
        main_win.name.set('Hi!\n%s' % user_name)
        main_win.btn_file.configure(command=on_btn_file_clicked)
        main_win.btn_send.configure(command=on_btn_send_clicked)
        main_win.user_list.bind('<<ListboxSelect>>', on_session_select)
        utils.send(my_socket, {'cmd': 'get_users'})
        utils.send(my_socket, {'cmd': 'get_history', 'peer': ''})
        t = threading.Thread(target=recv_async, args=())#创建接收进程
        t.setDaemon(True)#将主线程设置为子线程 t 的守护线程，子线
程 t 会随着主线程的退出而退出

        t.start()
        main_win.show()

    elif server_response['response'] == 'fail':
        tkinter.messagebox.showerror('警告', '登录失败：' +
server_response['reason'])

        close_socket()

    else:
        tkinter.messagebox.showerror('警告', '账号和密码不能为空！')

```

```

def on_btn_reg_clicked():
    global my_socket, login_win
    my_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    my_socket.settimeout(5)
    if login_win.user.get() != "" and login_win.pwd.get() != "":
        my_socket.connect((server_ip, int(server_port)))
        utils.send(my_socket, {'cmd': 'register', 'user': login_win.user.get(),
        'pwd': hashlib.sha1(login_win.pwd.get().encode('utf-8')).hexdigest()})
        server_response = utils.recv(my_socket)
        if server_response['response'] == 'ok':
            tkinter.messagebox.showinfo('注意', '注册成功！')
        elif server_response['response'] == 'fail':
            tkinter.messagebox.showerror('警告', '注册失败：' +
server_response['reason'])
        else:
            tkinter.messagebox.showerror('警告', '账号和密码不能为空！')
        close_socket()

```

```

def recv_async():#异步接收
    global my_socket, users, main_win, current_session, file_transfer_pending,
filename_short, filename
    while True:
        data = utils.recv(my_socket)
        if data['type'] == 'get_users':
            users = {}
            for user in [""] + data['data']:
                users[user] = False
            refresh_user_list()
        elif data['type'] == 'get_history':

```

```

if data['peer'] == current_session:
    main_win.history['state'] = 'normal'
    main_win.history.delete('1.0', 'end')
    main_win.history['state'] = 'disabled'
for entry in data['data']:
    append_history(entry[0], entry[1], entry[2])
elif data['type'] == 'peer_joined':
    users[data['peer']] = False
    refresh_user_list()
elif data['type'] == 'peer_left':
    if data['peer'] in users.keys():
        del users[data['peer']]
    if data['peer'] == current_session:
        current_session =
        main_win.btn_file.configure(state='disabled')
        main_win.name.set('%s -> global' % user_name)
        users[""] = False
        utils.send(my_socket, {'cmd': 'get_history', 'peer': ""})
        refresh_user_list()
    elif data['type'] == 'msg':
        if data['peer'] == current_session:
            append_history(data['peer'], time.strftime("%m 月 %d 日 %H:%M", time.localtime(time.time())), data['msg'])
        else:
            users[data['peer']] = True
            refresh_user_list()
    elif data['type'] == 'broadcast':
        if current_session == "":
            append_history(data['peer'], time.strftime("%m 月 %d", time.localtime()))

```

```

日 %H:%M', time.localtime(time.time()), data['msg'])

else:

    users[""] = True

    refresh_user_list()

elif data['type'] == 'file_request':

    if tkinter.messagebox.askyesno('注意', "%s 想要发文件给你\\文
件名: %s\\n 大小: %s\\n 接收?" % (data['peer'], data['filename'], data['size'])):

        utils.send(my_socket, {'cmd': 'file_accept', 'peer': data['peer']})

    try:

        total_bytes = 0

        addr = ('0.0.0.0', 8889)

        server = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)

        server.bind(addr)

        server.listen(5)

        client_socket, addr = server.accept()#client_socket 表示
客户端与服务器端进行通信的通道连接

        starttime = time.time()

        with open(data['filename'], "wb") as f:#以二进制格式打
开一个文件只用于写入

        while True:

            fdata = client_socket.recv(1024)

            total_bytes += len(fdata)

            if not fdata:

                break

            f.write(fdata)

        f.close()

        client_socket.close()

        server.close()

        endtime = time.time()

```

```

        received_md5 = get_file_md5(data['filename'])

        if received_md5 == data['md5']:
            tkinter.messagebox.showinfo('注意', '文件接收成功！')

            main_win.history['state'] = 'normal'
            main_win.history.insert('end', 'Received %s bytes\nfrom %s in %s seconds\n' % (
                total_bytes, data['peer'], format(endtime - starttime,
                '.2f)), 'hint')

            main_win.history.see('end')
            main_win.history['state'] = 'disabled'

        except:
            pass

        else:
            utils.send(my_socket, {'cmd': 'file_deny', 'peer': data['peer']})

        elif data['type'] == 'file_deny':
            main_win.btn_file.configure(text='发送文件')
            if current_session == '':
                main_win.btn_file.configure(state='disabled')
            else:
                main_win.btn_file.configure(state='normal')
            tkinter.messagebox.showinfo('警告', '对方拒绝接收！')

        elif data['type'] == 'file_accept':
            try:
                total_bytes = 0
                starttime = time.time()
                # data['ip']='127.0.0.1'
                addr = ('0.0.0.0', 8889)
                client = socket.socket(socket.AF_INET,
                socket.SOCK_STREAM)

```

```

client.connect(addr)

with open(filename, 'rb') as f:

    while True:

        fdata = f.read(1024)

        if not fdata:

            break

        total_bytes += len(fdata)

        client.send(fdata)

    f.close()

    client.close()

    endtime = time.time()

    main_win.history['state'] = 'normal'

    main_win.history.insert('end', 'Sent %s bytes in %s

seconds\n\n' % (
        total_bytes, format(endtime - starttime, '.2f')), 'hint')

    main_win.history.see('end')

    main_win.history['state'] = 'disabled'

finally:

    filename = ""

    filename_short = ""

    file_transfer_pending = False #文件传输是否挂起

main_win.btn_file.configure(text='发送文件')

if current_session == ":":

    main_win.btn_file.configure(state='disabled')

else:

    main_win.btn_file.configure(state='normal')

tkinter.messagebox.showinfo('注意', '文件发送成功！')

```

```

def refresh_user_list():

    main_win.user_list.delete(0, 'end')

```

```

for user in users.keys():
    name = 'Public 聊天室' if user == " else user
    if users[user]:
        name += ' (*)'
    main_win.user_list.insert('end', name)

def append_history(sender, time, msg):
    main_win.history['state'] = 'normal'
    main_win.history.insert('end', '%s - %s\n' % (sender, time))
    main_win.history.insert('end', msg + '\n\n', 'text')
    main_win.history.see('end')
    main_win.history['state'] = 'disabled'

def on_btn_file_clicked():
    global my_socket, main_win, filename, filename_short,
    file_transfer_pending
    try:
        filename = tkinter.filedialog.askopenfilename()
        if filename == ":
            return
        filename_short = ""
        if len(filename.split('/')) < len(filename.split('\\')):
            filename_short = filename.split('\\')[-1]
        else:
            filename_short = filename.split('/')[-1]
        size = os.path.getsize(filename)
        count = 0
        while not 1 < size < 1024 and count < 6:
            size /= 1024

```

```
    count += 1

    size = str(format(size, '.2f')) + ['B', 'KB', 'MB', 'GB', 'TB', 'PB'][count]

    md5_checksum = get_file_md5(filename)

    utils.send(my_socket, {'cmd': 'file_request', 'peer': current_session,
'filename': filename_short, 'size': size, 'md5': md5_checksum})

    main_win.btn_file.configure(text='等待中...')

    main_win.btn_file.configure(state='disabled')

    file_transfer_pending = True

except:

    sys.exit(1)
```

```
def on_btn_send_clicked():

    global my_socket, user_name, current_session, main_win

    if main_win.msg.get() != "":

        utils.send(my_socket, {'cmd': 'chat', 'peer': current_session, 'msg':
main_win.msg.get()})

        append_history(user_name, time.strftime('%m 月 %d 日 %H:%M',
time.localtime(time.time())), main_win.msg.get())

        main_win.msg.set("")

    else:

        tkinter.messagebox.showinfo('警告', '消息不能为空！')
```

```
def on_session_select(event):

    global current_session, main_win, user_name, users, file_transfer_pending

    w = event.widget

    changed = False

    if len(w.curselection()) != 0:

        index = int(w.curselection()[0])

        if index != 0:
```

```

        if current_session != w.get(index).rstrip('(*)'):#删除字符串末尾
的指定字符(*)

        changed = True

        current_session = w.get(index).rstrip('(*)')

        if not file_transfer_pending:

            main_win.btn_file.configure(state='normal')

            main_win.name.set('%s -> %s' % (user_name,
current_session))

            users[current_session] = False

            refresh_user_list()

        elif index == 0:

            if current_session != ":

                changed = True

                current_session = ""

                main_win.btn_file.configure(state='disabled')

                main_win.name.set('%s -> global' % user_name)

                users[""] = False

                refresh_user_list()

            if changed:

                utils.send(my_socket, {'cmd': 'get_history', 'peer': current_session})

def on_closed():

    close_socket()

def get_file_md5(file_path):

    md5obj = hashlib.md5()

    maxbuf = 8192

    f = open(file_path, 'rb')

```

```

while True:
    buf = f.read(maxbuf)
    if not buf:
        break
    md5obj.update(buf)
f.close()
hash = md5obj.hexdigest()
return str(hash).upper()

if __name__ == '__main__':
    login_win = Login_win()
    login_win.btn_login.configure(command=on_btn_login_clicked)
    login_win.btn_reg.configure(command=on_btn_reg_clicked)
    login_win.show()

```

2. 相关文档

<https://pan.baidu.com/s/1LwCAOvCLAy8mySboiVqGQ?pwd=tgbp>

3. 参考链接

<https://www.cnblogs.com/caesar-id/p/12094529.html>
<https://www.cnblogs.com/liuzhongkun/p/15831133.html>
<https://www.cnblogs.com/baby12138/p/10414981.html>
<https://www.cnblogs.com/zhongguiyao/p/11049436.html>
<https://www.cnblogs.com/xuliuzai/p/15506254.html>
https://blog.csdn.net/m0_37382341/article/details/101173124

(二) FTP 程序

1. 源码文件

//不含 NAT 穿透功能的客户端程序

#pragma GCC optimize(2)

```
    return 0;  
}  
  
}
```

2. 相关文档

RFC 765: <http://www.faqs.org/rfcs/rfc765.html>

RFC 959: <http://www.faqs.org/rfcs/rfc959.html>

3. 参考链接

<https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/ftp>

<https://www.cnblogs.com/kefeiGame/p/7246942.html>

<https://blog.csdn.net/yongaini10/article/details/55095784>

<https://www.cnblogs.com/echo579/articles/6802120.html>

https://blog.csdn.net/qq_36553387/article/details/119190783