实 验 报 告

1. 实验名称

FTP程序

1. 实验原理

文件传输协议（File Transfer Protocol，FTP）是用于在网络上进行文件传输的一套标准协议，它工作在 OSI 模型的第七层， TCP 模型的第四层， 即应用层， 使用 TCP 传输而不是 UDP， 客户在和服务器建立连接前要经过一个“三次握手”的过程， 保证客户与服务器之间的连接是可靠的， 而且是面向连接， 为数据传输提供可靠保证。 [1]

FTP允许用户以文件操作的方式（如文件的增、删、改、查、传送等）与另一主机相互通信。然而， 用户并不真正登录到自己想要存取的计算机上面而成为完全用户， 可用FTP程序访问远程资源， 实现用户往返传输文件、目录管理以及访问电子邮件等等， 即使双方计算机可能配有不同的操作系统和文件存储方式。

详情见参考文档。

1. 实验目的

掌握FTP程序的相关知识和原理（主动模式、被动模式）。

1. 实验内容
   1. 基本功能

A.用户和口令的认证；

B.目录的各种操作（进入、创建、列表等）；

C.文件上传和下载。

* 1. 高级功能

PASSIVE模式

断点续传

1. 实验实现
2. 人员分工

队友：客户端代码编写、调试、测试

我：客户端代码测试，服务器代码编写、调试，实验报告

1. 实验设计
2. 协议

FTP 采用 Internet 标准文件传输协议 FTP 的用户界面， 向用户提供了一组用来管理计算机之间文件传输的应用程序。

FTP 是基于客户———服务器（C/S）模型而设计的，在客户端与 FTP 服务器之间建立两个连接。

开发任何基于 FTP 的客户端软件都必须遵循 FTP 的工作原理，FTP 的独特的优势同时也是与其它客户服务器程序最大的不同点就在于它在两台通信的主机之间使用了两条 TCP 连接，一条是数据连接，用于数据传送；另一条是控制连接，用于传送控制信息（命令和响应），这种将命令和数据分开传送的思想大大提高了 FTP 的效率，而其它客户服务器应用程序一般只有一条 TCP 连接。图 1 给出了 FTP 的基本模型。客户有三个构件：用户接口、客户控制进程和客户数据传送进程。服务器有两个构件：服务器控制进程和服务器数据传送进程。在整个交互的 FTP 会话中，控制连接始终是处于连接状态的，数据连接则在每一次文件传送时先打开后关闭。

FTP客户端发起FTP会话，与FTP服务器建立相应的连接。FTP会话期间要建立控制信息进程与数据进程两个连接。控制连接不能完成传输数据的任务，只能用来传送FTP执行的内部命令以及命令的响应等控制信息；数据连接是服务器与客户端之间传输文件的连接，是全双工的，允许同时进行双向数据传输。当数据传输完成后，数据连接会撤消，再回到FTP会话状态，直到控制连接被撤消，并退出会话为止。

FTP支持两种模式：Standard （PORT方式，主动方式），Passive （PASV，被动方式）。

Port模式

FTP 客户端首先和服务器的TCP 21端口建立连接，用来发送命令，客户端需要接收数据的时候在这个通道上发送PORT命令。PORT命令包含了客户端用什么端口接收数据。在传送数据的时候，服务器端通过自己的TCP 20端口连接至客户端的指定端口发送数据。FTP server必须和客户端建立一个新的连接用来传送数据。

Passive模式

建立控制通道和Standard模式类似，但建立连接后发送Pasv命令。服务器收到Pasv命令后，打开一个临时端口（端口号大于1023小于65535）并且通知客户端在这个端口上传送数据的请求，客户端连接FTP服务器此端口，然后FTP服务器将通过这个端口传送数据。

很多防火墙在设置的时候都是不允许接受外部发起的连接的，所以许多位于防火墙后或内网的FTP服务器不支持PASV模式，因为客户端无法穿过防火墙打开FTP服务器的高端端口；而许多内网的客户端不能用PORT模式登陆FTP服务器，因为从服务器的TCP 20无法和内部网络的客户端建立一个新的连接，造成无法工作。

默认状态下，FTP 站点允许匿名访问，FTP 服务器接受对该资源的所有请求，并且不提示用户输入用户名或密码。

基本目前所有ip均支持ftp协议。

1. UI设计

本次实验采用命令行方法控制。

1. 框架结构

【说清楚客户端程序和服务器端程序的框架结构。特别是服务器程序：说明支持多个用户（连接）的方法，如多进程，多线程，多路复用等；如果存在进程或线程之间共享数据的，还要注意保护加锁等。说明错误处理的方法。】

客户端程序：将客户端分为用户交互部分与底层实现部分，底层实现部分分为直接支持用户交互部分代码交互的顶层方法（如ls函数、mkdir函数等）与支持顶层方法的底层方法（如send\_cmd函数与get\_data\_socket函数）。客户端部分使用socket库、os库与time库实现。其具体实现方法见详细代码。

服务器程序：整体结构与客户端类似。将服务器设为循环接受命令，使用多线程处理多客户端问题，使用信号量与自旋锁处理读-写问题。由于若全部使用套接字实现代码量过大，服务器使用socket库、pyftpdlib库、os库、time库实现。Pyftpdlib库详情可参考<https://pyftpdlib.readthedocs.io/en/latest/tutorial.html>

1. 环境配置：

主机系统环境&云服务器系统环境：Ubuntu22.04

由于windows与ubuntu直接使用20/21端口调试执行会影响系统，并需绕过防火墙，故本次实验端口使用2120/2121端口。

由于本人windows上安全软件拦截陌生ip信息，故本次实验使用ubuntu。

附：不要用openEular。

语言：python3.10

云服务器：华为弹性云服务器

云服务器安全组配置：端口全部开放

编写/调试工具：vscode&vim

需安装库：pyftpdlib

1. 关键代码的描述

本次实验完全为自主编写。

客户端编写者：

服务器端编写者：

1. FTP客户端程序：FTPclient。

全部代码见附加文件FTPclient(1).py

底层代码-基础功能实现

命令发送函数：

def send\_cmd(sock, cmd):

    sock.sendall(f"{cmd}\r\n".encode())

    time.sleep(0.2)

    response = sock.recv(4096).decode()

    return response

其中sock为所用命令套接字，encode为字符串编码函数，将字符串编码为UTF-8编码。

Sleep方法调用是由于若在发送命令后立即接收消息会导致无法接收消息从而报错，故需延迟0.2秒。

Sock的Recv方法会接收消息，decode为将UTF-8编码解码为字符串。

def get\_data\_socket(sock, use\_passive):

    if use\_passive:

        send\_cmd(sock, "PASV")

        sock.sendall(f"PASV\r\n".encode())

        response = sock.recv(4096).decode()

        ip\_and\_port = response.split("(")[1].split(")")[0].split(",")

        ip = ".".join(ip\_and\_port[:4])

        port = int(ip\_and\_port[4]) \* 256 + int(ip\_and\_port[5])

        data\_sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

        data\_sock.connect((ip, port))

    else:

        data\_sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

        data\_sock.bind(("0.0.0.0", 0))

        data\_sock.listen(1)

        ip = socket.gethostbyname(socket.gethostname())

        ip\_and\_port = ",".join(ip.split(".") + [str(data\_sock.getsockname()[1] // 256), str(data\_sock.getsockname()[1] % 256)])

        send\_cmd(sock, f"PORT {ip\_and\_port}")

        data\_sock, \_ = data\_sock.accept()

    return data\_sock

get\_data\_socket函数使用被动模式或主动模式与服务器建立新的数据连接。如果use\_Passive参数为True，则使用被动模式，否则使用主动模式。在被动模式下，服务器向客户端提供IP地址和端口号，客户端使用这些值连接到服务器。在活动模式下，客户端向服务器提供IP地址和端口号，服务器使用这些值连接到客户端。

def ftp\_connect(host, username, password):

    sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

    sock.connect((host, 2121))

    print(sock.recv(4096).decode())

    send\_cmd(sock, f"USER {username}")

    repos=send\_cmd(sock, f"PASS {password}")

    print(repos)

    return sock

主函数ftp\_connect连接到具有给定主机、用户名和密码的ftp服务器，并返回可用于发送ftp命令的套接字对象。

def list\_directory(sock, use\_passive):

    data\_sock = get\_data\_socket(sock, use\_passive)

    send\_cmd(sock, "LIST")

    print(data\_sock.recv(4096).decode())

data\_sock.close()

底层代码-命令实现

list\_directory函数向服务器发送“list”命令，以获取当前目录的内容列表，并将响应打印到控制台。

def create\_directory(ctrl\_sock, dir\_name):

    response = send\_cmd(ctrl\_sock, f"MKD {dir\_name}")

    print(response)

create\_directory函数向服务器发送“MKD”命令，以创建一个具有给定名称的新目录。

def get\_directory(ctrl\_sock):

    re=send\_cmd(ctrl\_sock, "PWD")

    return re.strip().split('"')[1]

get\_directory函数向服务器发送“PWD”命令以获取当前工作目录，并返回带有目录路径的响应。

def change\_directory(ctrl\_sock, path):

    response = send\_cmd(ctrl\_sock, f"CWD {path}")

    print(response)

change\_directory函数向服务器发送“CWD”命令，将当前工作目录更改为给定的路径。

def delete\_file(ctrl\_sock, filename):

    response = send\_cmd(ctrl\_sock, f"DELE {filename}")

    print(response)

delete\_file函数向服务器发送“DELE”命令，以删除具有给定名称的文件。

def delete\_directory(ctrl\_sock, dir\_name):

    response = send\_cmd(ctrl\_sock, f"RMD {dir\_name}")

    print(response)

delete\_directory函数向服务器发送“RMD”命令，以删除具有给定名称的目录。

def download\_file(sock, remote\_file, local\_file, use\_passive):

    send\_cmd(sock, "TYPE I")

    tmp\_file = local\_file + '.tmp'

    bytes\_transferred = 0

    local\_size = 0

    if os.path.exists(tmp\_file):

        bytes\_transferred = os.path.getsize(tmp\_file)

    if os.path.exists(local\_file):

        local\_size = os.path.getsize(local\_file)

    ft=os.path.exists(tmp\_file)

    if local\_size==0 and (not ft):

        data\_sock = get\_data\_socket(sock, use\_passive)

        re=send\_cmd(sock, f"RETR {remote\_file}")

        if re.startswith("550"):

            print("No such file or directory in remote files.\n")

            data\_sock.close()

            return

        with open(tmp\_file, "ab") as file:

            while True:

                data = data\_sock.recv(4096)

                if not data:

                    break

                file.write(data)

        if os.path.exists(local\_file):

           os.remove(local\_file)

        os.rename(tmp\_file, local\_file)

        print(f"finished\n")

    elif local\_size > 0 and (not ft):

        print(f"This file is not blank,do you want to resume? y/n")

        while True:

            user\_input = input(f" > ").strip().lower()

            cmd\_parts=user\_input.split()

            if len(cmd\_parts) == 0:

                continue

            cmd=cmd\_parts[0]

            if cmd=="n" or cmd== "N":

                break

            elif cmd=="y"or cmd=="Y":

                data\_sock = get\_data\_socket(sock, use\_passive)

                re=send\_cmd(sock, f"RETR {remote\_file}")

                if re.startswith("550"):

                    print("No such file or directory in remote files.\n")

                    data\_sock.close()

                    return

                with open(tmp\_file, "ab") as file:

                    while True:

                        data = data\_sock.recv(4096)

                        if not data:

                            break

                        file.write(data)

                os.remove(local\_file)

                os.rename(tmp\_file, local\_file)

                print(f"finished\n")

                break

            else:

                continue

    elif ft:

        print(f"Resuming upload from {bytes\_transferred} bytes")

        send\_cmd(sock, f"REST {bytes\_transferred}")

        data\_sock = get\_data\_socket(sock, use\_passive)

        re=send\_cmd(sock, f"RETR {remote\_file}")

        if re.startswith("550"):

            print("No such file or directory in remote files.\n")

            data\_sock.close()

            return

        with open(tmp\_file, "ab") as file:

            while True:

                data = data\_sock.recv(4096)

                if not data:

                    break

                file.write(data)

        os.rename(tmp\_file, local\_file)

        print(f"finished\n")

    data\_sock.close()

download\_file函数将文件从服务器下载到客户端的本地文件系统。它首先向服务器发送“TYPE I”命令，将传输模式设置为二进制。然后，它检查本地文件是否存在以及是否为空。如果本地文件为空或不存在，则会打开与服务器的新数据连接，并发送“RETR”命令来检索远程文件。它从数据连接中读取数据，并将其写入客户端本地文件系统上的临时文件。一旦传输完成，它会将临时文件重命名为最终目标文件。如果本地文件存在且不为空，则会提示用户是继续传输还是开始新的传输。如果用户选择继续，它会向服务器发送“REST”命令，以指定已传输的字节数，打开数据连接，并从停止的位置恢复传输。如果用户选择开始新的传输，它会删除本地文件并从头开始传输。

def upload\_file(sock, local\_file, remote\_file, use\_passive):

    send\_cmd(sock, "TYPE I")

    try:

        re=send\_cmd(sock, f"SIZE {remote\_file}")

    except Exception as re:

        remote\_size = 0

    else:

        if re.startswith("550"):

            remote\_size=0

        else:

            remote\_size = int(str(re).split()[-1])

    tmp\_file = remote\_file + '.tmp'

    bytes\_transferred = 0

    ft=False

    try:

         ret=send\_cmd(sock, f"SIZE {tmp\_file}")

    except Exception as re:

        bytes\_transferred = 0

    else:

        if ret.startswith("550"):

            bytes\_transferred=0

        else:

            bytes\_transferred = int(str(re).split()[-1])

            ft=True

    if (not os.path.exists(local\_file)):

        print("No such file or directory.\n")

        data\_sock.close()

        return

    if remote\_size==0 and (not ft):

        data\_sock = get\_data\_socket(sock, use\_passive)

        send\_cmd(sock, f"STOR {tmp\_file}")

        with open(local\_file, "rb") as file:

            while True:

                data = file.read(4096)

                if not data:

                    break

                data\_sock.sendall(data)

        if not re.startswith("550"):

            send\_cmd(sock, f"DELE {remote\_file}")

        send\_cmd(sock, "RNFR {}".format(tmp\_file))

        send\_cmd(sock, "RNTO {}".format(remote\_file))

        print(f"finished\n")

    elif remote\_size > 0 and (not ft):

        print(f"This file is not blank,do you want to resume? y/n")

        while True:

            user\_input = input(f" > ").strip().lower()

            cmd\_parts=user\_input.split()

            if len(cmd\_parts) == 0:

                continue

            cmd=cmd\_parts[0]

            if cmd=="n" or cmd== "N":

                break

            elif cmd=="y"or cmd=="Y":

                data\_sock = get\_data\_socket(sock, use\_passive)

                send\_cmd(sock, f"STOR {tmp\_file}")

                with open(local\_file, "rb") as file:

                    while True:

                        data = file.read(4096)

                        if not data:

                            break

                    data\_sock.sendall(data)

                send\_cmd(sock, f"DELE {remote\_file}")

                send\_cmd(sock, "RNFR {}".format(tmp\_file))

                send\_cmd(sock, "RNTO {}".format(remote\_file))

                print(f"finished\n")

                break

    elif ft:

        print(f"Resuming upload from {bytes\_transferred} bytes")

        send\_cmd(sock, f"REST {bytes\_transferred}")

        data\_sock = get\_data\_socket(sock, use\_passive)

        send\_cmd(sock, f"STOR {tmp\_file}")

        with open(local\_file, "rb") as file:

            while True:

                data = file.read(4096)

                if not data:

                    break

                data\_sock.sendall(data)

        send\_cmd(sock, "RNFR {}".format(tmp\_file))

        send\_cmd(sock, "RNTO {}".format(remote\_file))

        print(f"finished\n")

    data\_sock.close()

upload\_file函数将文件从客户端的本地文件系统上载到服务器。它首先向服务器发送“TYPE I”命令，将传输模式设置为二进制。然后，它检查远程文件是否存在以及是否为空。如果远程文件为空或不存在，则会打开与服务器的新数据连接，并发送“STOR”命令来存储本地文件。它从本地文件中读取数据并将其写入数据连接。一旦传输完成，它将关闭数据连接。如果远程文件存在且不为空，则会提示用户是继续传输还是开始新的传输。如果用户选择继续，它会向服务器发送“REST”命令，以指定已传输的字节数，打开数据连接，并从停止的位置恢复传输。如果用户选择开始新的传输，它会删除本地文件并从头开始传输。

def print\_help():

    print("Commands:")

    print("  help          - Show this help")

    print("  passive       - Use passive mode")

    print("  active        - Use active mode")

    print("  upload [path] [name] - Upload a file to the server")

    print("  mkdir [name]  - Create a directory on the server")

    print("  ls            - List the contents of the current directory")

    print("  cd [path]     - Change the current directory")

    print("  rm [file]     - Delete a file")

    print("  rmdir [dir]   - Delete a directory")

    print("  download [remote\_file] [local\_file] - Download a file")

    print("  quit          - Disconnect and exit")

print\_help函数输出全部的命令使用方法。

上层代码-用户交互模块

def main():

    print("----------------welcome to FTP client!---------------\n")

    host = input("Enter the host: ").strip()

    username = input("Enter your username: ").strip()

    password = input("Enter your password: ").strip()

    #host = "127.0.0.1"

    #username = "user"

    #password = "12345"

    use\_passive = True # Set to False for active mode

    ctrl\_sock = ftp\_connect(host, username, password)

    print("-----------------------------------------------------\n")

    while True:

        current\_directory = get\_directory(ctrl\_sock)

        user\_input = input(f"{current\_directory} > ").strip().lower()

        cmd\_parts = user\_input.split()

        if len(cmd\_parts) == 0:

            continue

        cmd = cmd\_parts[0]

        if cmd == "help":

            print\_help()

        elif cmd == "passive":

            use\_passive = True

            print("Switched to passive mode.")

        elif cmd == "active":

            use\_passive = False

            print("Switched to active mode.")

        elif cmd == "upload":

            if len(cmd\_parts) != 3:

                print("Usage: upload [local\_file] [remote\_file]")

            else:

                file\_path = cmd\_parts[1]

                if not os.path.isfile(file\_path):

                    print("File not found.")

                else:

                    remote\_file = cmd\_parts[2]

                    local\_file = cmd\_parts[1]

                    upload\_file(ctrl\_sock, local\_file, remote\_file, use\_passive)

        elif cmd == "mkdir":

            if len(cmd\_parts) != 2:

                print("Usage: mkdir [name]")

            else:

                dir\_name = cmd\_parts[1]

                create\_directory(ctrl\_sock, dir\_name)

        elif cmd == "ls":

            list\_directory(ctrl\_sock, use\_passive)

        elif cmd == "cd":

            if len(cmd\_parts) != 2:

                print("Usage: cd [path]")

            else:

                path = cmd\_parts[1]

                change\_directory(ctrl\_sock, path)

        elif cmd == "rm":

            if len(cmd\_parts) != 2:

                print("Usage: rm [file]")

            else:

                filename = cmd\_parts[1]

                delete\_file(ctrl\_sock, filename)

        elif cmd == "rmdir":

            if len(cmd\_parts) != 2:

                print("Usage: rmdir [dir]")

            else:

                dir\_name = cmd\_parts[1]

                delete\_directory(ctrl\_sock, dir\_name)

        elif cmd == "download":

            if len(cmd\_parts) != 3:

                print("Usage: download [remote\_file] [local\_file]")

            else:

                remote\_file = cmd\_parts[1]

                local\_file = cmd\_parts[2]

                download\_file(ctrl\_sock, remote\_file, local\_file, use\_passive)

        elif cmd == "quit":

            print("Disconnecting...")

            send\_cmd(ctrl\_sock, "QUIT")

            ctrl\_sock.close()

            break

        else:

            print("Unknown command. Type 'help' for available commands.")

主函数首先根据用户的输入来链接服务器并登录账号，之后根据上文中help的命令使用方法来对命令进行识别。若识别出相应的指令，则调用对应的函数进行处理。

服务器代码：

import os

from pyftpdlib.authorizers import DummyAuthorizer

from pyftpdlib.handlers import FTPHandler

from pyftpdlib.servers import FTPServer

def main():

    authorizer = DummyAuthorizer()

    authorizer.add\_user('user', '12345', 'd:\\file', perm='elradfmwMT')

    authorizer.add\_anonymous('d:\\file')

    handler = FTPHandler

    handler.authorizer = authorizer

    handler.permit\_foreign\_addresses(True)

    handler.banner = "pyftpdlib based ftpd ready."

    address = ('', 2121)

    server = FTPServer(address, handler)

    server.max\_cons = 256

    server.max\_cons\_per\_ip = 5

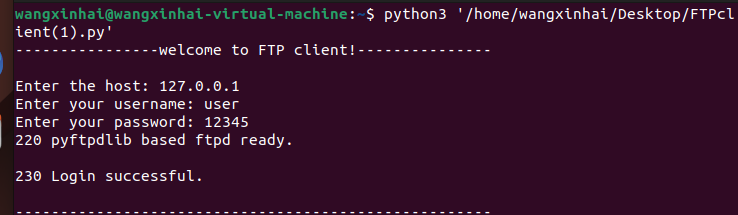
    server.serve\_forever()

if \_\_name\_\_ == '\_\_main\_\_':

    main()

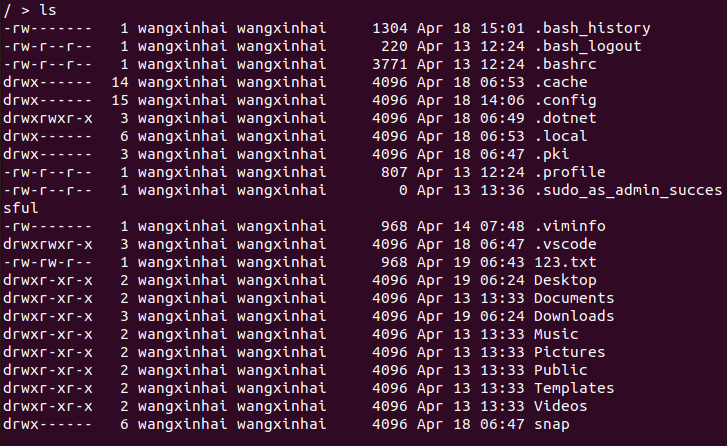
1. 测试及结果分析
2. 基础功能测试

测试过程、说明、结果及分析。

(1).用户登录：

成功。

由于服务器与客户端代码均在本人虚拟机上运行，故ip地址使用127.0.0.1，即回送地址。

(2).列表

成功。目录使用主机中的//home目录。

(3).创建文件夹

成功。

(4).进入文件夹

成功。

(5).上传文件

成功。

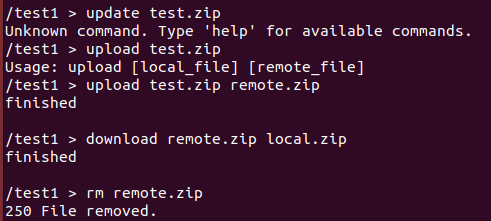
可以看到在test1文件夹中已经有了remote压缩文件，并且解压出内容与test文件一致。

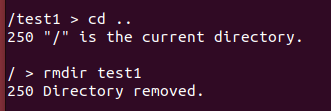
(6)下载文件

成功。

可以看到remote.zip解压内容与local.zip一致。

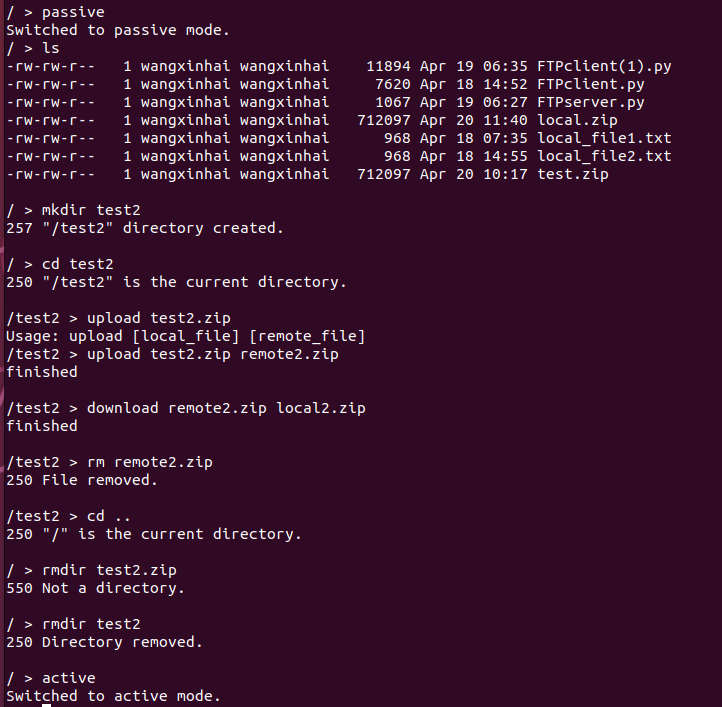
(7)删除文件

成功。可以看到在test1文件夹中文件已被删除。

(8)删除文件夹。

成功。可看到test1文件夹已被删除。

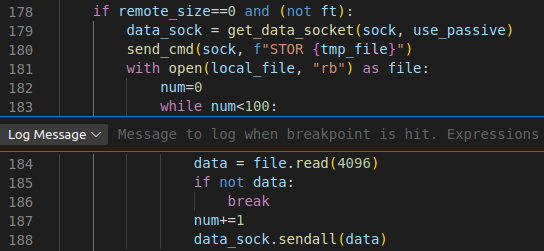
测试2 被动模式测试

由于指令与上述指令一致，故不赘述指令意义与结果。

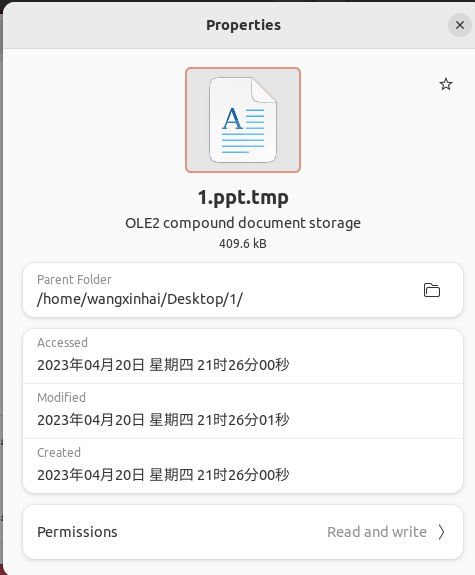
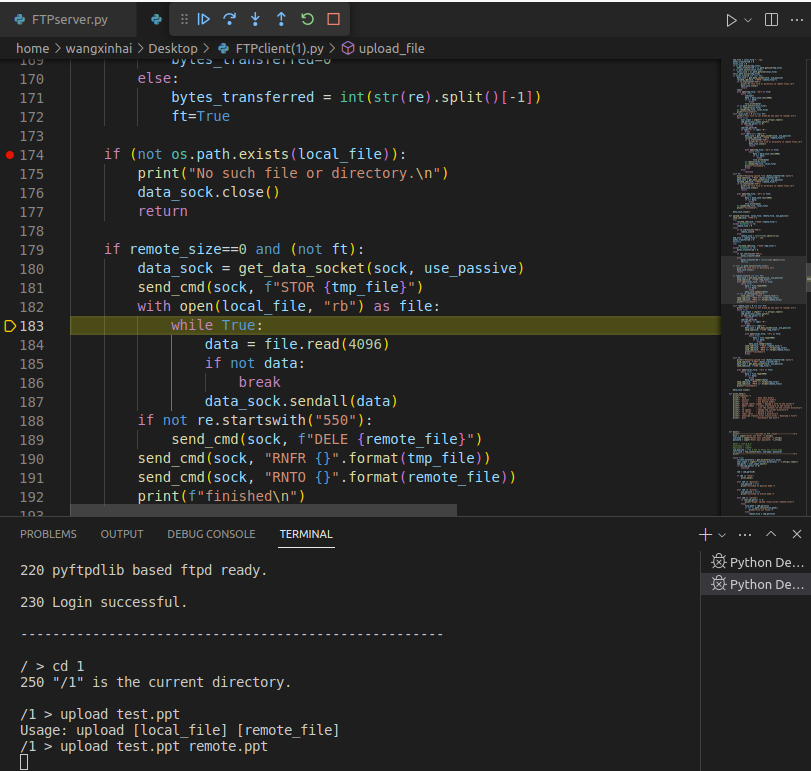
可以看到所有命令均能正确执行。

测试3 断点续传测试

本次使用方法为在源代码中设置断点传输一个3M的ppt，当传输100个包时中断，并重新传输。

上传代码：（在上传完100个后手动退出）

调试见下页

在传输100个包后，停止运行。临时文件如下图。

使用断点续传上传完文件后，发现可以成功打开。具体见验收。

1. 实验结论

本次实验中，我们完成了FTP服务器与客户端的搭设与测试，并实现了高级功能中的被动模式、被动模式NAT穿透与断点续传，并实现了FTP服务器与客户端的绝大部分功能。

1. 总结及心得体会

FTP（文件传输协议）是一种标准通信协议，用于通过互联网在计算机之间传输文件。它被广泛用于将文件从服务器上传和下载到客户端，反之亦然。在我的实验中，我和我的队友编写了FTP服务器代码和FTP客户端代码，以在两台计算机之间传输文件。

编写FTP代码很有挑战性，因为它涉及到理解复杂的网络编程以及FTP协议所需的各种命令和响应。然而，通过仔细的计划和对细节的关注，我和我的队友成功地编写和实现了实验所需的代码。

该过程的第一步是设计和实现服务器端代码，该代码将接收并响应来自客户端的请求。这包括定义必要的命令，如LIST（列出服务器上的文件）、GET（从服务器检索文件）和PUT（将文件上传到服务器），并实现相应的代码来处理这些请求。

接下来，我和我的搭档处理客户端代码，该代码将启动与服务器的连接，并发送适当的命令来传输文件。这包括编写代码来处理用户输入，例如指定要传输的文件和传输方向（上传或下载），以及建立与服务器的连接并发送适当的FTP命令。

在整个过程中，我和我的队友遇到了许多挑战，例如确保代码正确同步以及处理文件传输过程中可能发生的错误。然而，通过协同工作并仔细测试代码的每个组件，我们克服了这些挑战，并成功地在客户端和服务器之间传输文件。

总的来说，我们在FTP协议实验方面获得的经验是网络编程和软件开发方面的宝贵学习经验。通过这个项目，我和我的队友能够获得广泛使用的协议的实践经验，并发展编程、测试和调试方面的重要技能。

附件

1. 源码文件

见上方

2.相关文档

<https://www.rfc-editor.org/rfc/rfc959>

<https://baike.baidu.com/item/Ftp/13839>

<https://www.rfc-editor.org/rfc/rfc114>

<https://baike.baidu.com/item/nat/320024>

<https://baike.baidu.com/item/IPV4/422599>

<https://zhuanlan.zhihu.com/p/428372105>

3.参考资料（链接）

<https://zhuanlan.zhihu.com/p/260139078>

<http://c.biancheng.net/view/4305.html>

<https://docs.python.org/3/library/socket.html>

<https://www.runoob.com/python/python-socket.html>

<https://docs.python.org/3/library/ftplib.html>

<https://pyftpdlib.readthedocs.io/en/latest/tutorial.html>

<https://docs.python.org/3/library/os.html>

<https://zhuanlan.zhihu.com/p/150835193>

<https://www.geeksforgeeks.org/python-time-module/>

<https://docs.python.org/3/library/time.html>

<https://www.runoob.com/python/python-strings.html>

<https://docs.python.org/3/library/string.html>

<https://blog.csdn.net/weixin_39639550/article/details/111792489>

<https://blog.csdn.net/qq_53164182/article/details/128619507>

<https://linux.cn/article-15185-1.html>

<https://www.runoob.com/python/file-methods.html>

<https://docs.python.org/3/library/filesys.html>

<https://www.runoob.com/python/python-functions.html>

<https://blog.csdn.net/Mart1nn/article/details/81077234>

<https://docs.python.org/3/howto/sockets.html>

<https://geek-docs.com/python/python-tutorial/python-ftp.html>

<https://blog.csdn.net/netlt/article/details/120906402>

<https://www.cnblogs.com/gc2770/p/14918622.html>

<https://www.cnblogs.com/niulang/p/13203739.html>

<https://blog.csdn.net/weixin_42359464/article/details/127598227>

<https://blog.csdn.net/jiesunliu3215/article/details/111603465>

<https://juejin.cn/s/python%20ftp%E6%96%AD%E7%82%B9%E7%BB%AD%E4%BC%A0>

<https://www.huaweicloud.com/>

<https://blog.csdn.net/qq_16944261/article/details/117738762>

<https://zhuanlan.zhihu.com/p/165614066>

<https://info.support.huawei.com/info-finder/encyclopedia/zh/NAT.html>

<https://www.iplocation.net/find-ip-address>

<https://blog.csdn.net/LINZAI508/article/details/111039825>

<https://blog.csdn.net/zzwwhhpp/article/details/113077747>

<https://support.microsoft.com/zh-cn/windows/%E6%89%93%E5%BC%80%E6%88%96%E5%85%B3%E9%97%AD-microsoft-defender-%E9%98%B2%E7%81%AB%E5%A2%99-ec0844f7-aebd-0583-67fe-601ecf5d774f>

<https://blog.csdn.net/weixin_42332985/article/details/106259719>

<https://blog.csdn.net/qq_51212018/article/details/110350950>

<https://blog.csdn.net/frighting_ing/article/details/122688413>

<https://www.runoob.com/python/python-func-input.html>

<https://blog.csdn.net/pear86743/article/details/8812362>

<https://blog.csdn.net/qy_getforward/article/details/111469678>

<https://www.runoob.com/python/python-multithreading.html>

<https://docs.python.org/3/library/threading.html>