

计算机网络课程期中项目实验报告

一、基本信息

小组成员：

年级	专业（方向）	学号	姓名	分工
2016	软件工程（计算机应用）	16340102	赖成枫	基于 UDP 的可靠传输 服务器支持多用户传输的实现
2016	软件工程（计算机应用）	16340008	蔡梓珩	Python 的文件读写 服务器和客户端的 UDP 传输 类似 TCP 的流控制和阻塞控制实现

实验环境：

操作系统：Windows 10

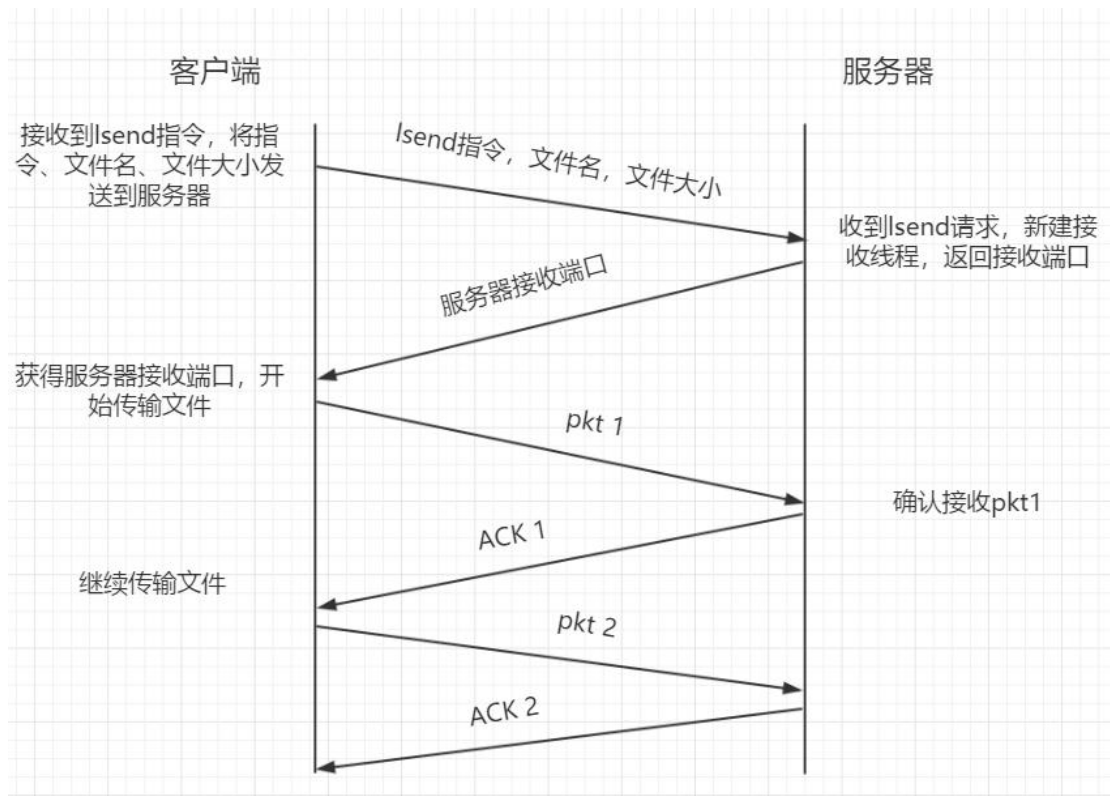
语言：Python 3.7

编译器：Sublime、VS Code

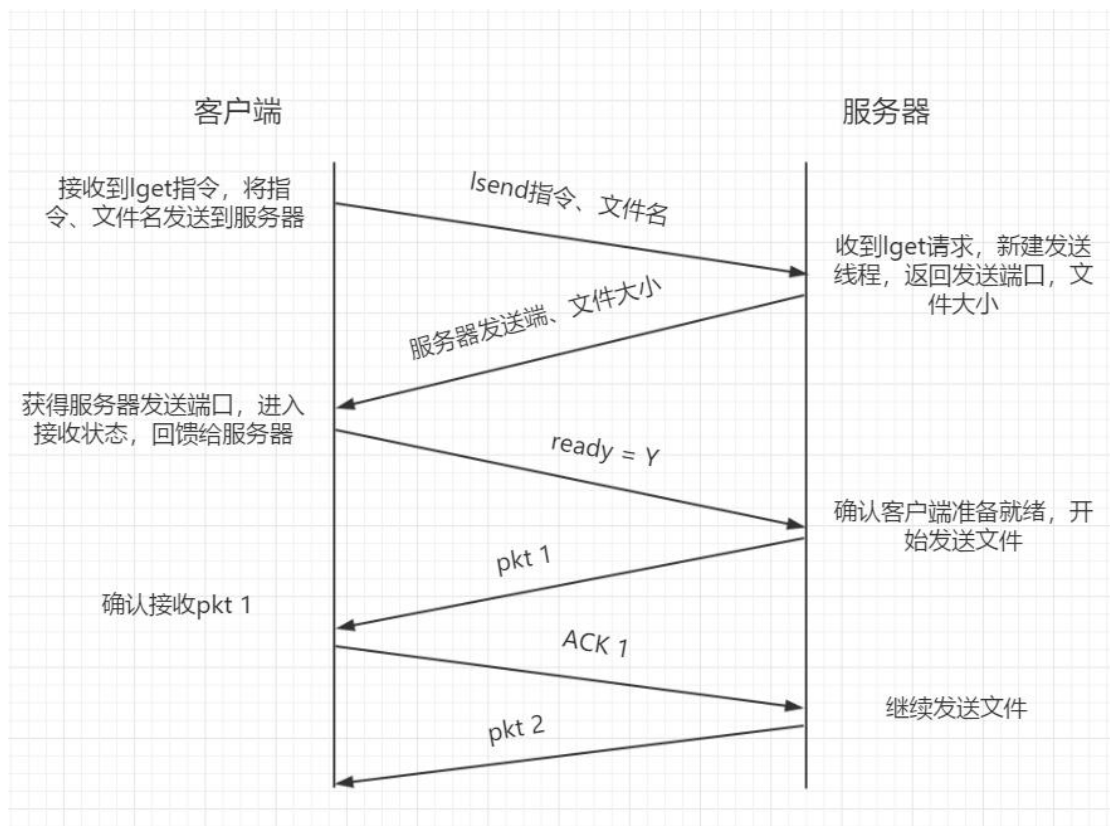
二、LFTP 实现要求

1. 客户端可以通过 `lsend myserver mylargefile` 命令将本地文件上传到服务器
2. 客户端可以通过 `lget myserver mylargefile` 命令从服务器下载文件到本地
3. 基于 UDP 的传输方式实现可靠传输
4. 实现传输时的流控制和阻塞控制
5. 服务器允许多用户同时上传和下载文件

三、发送模型

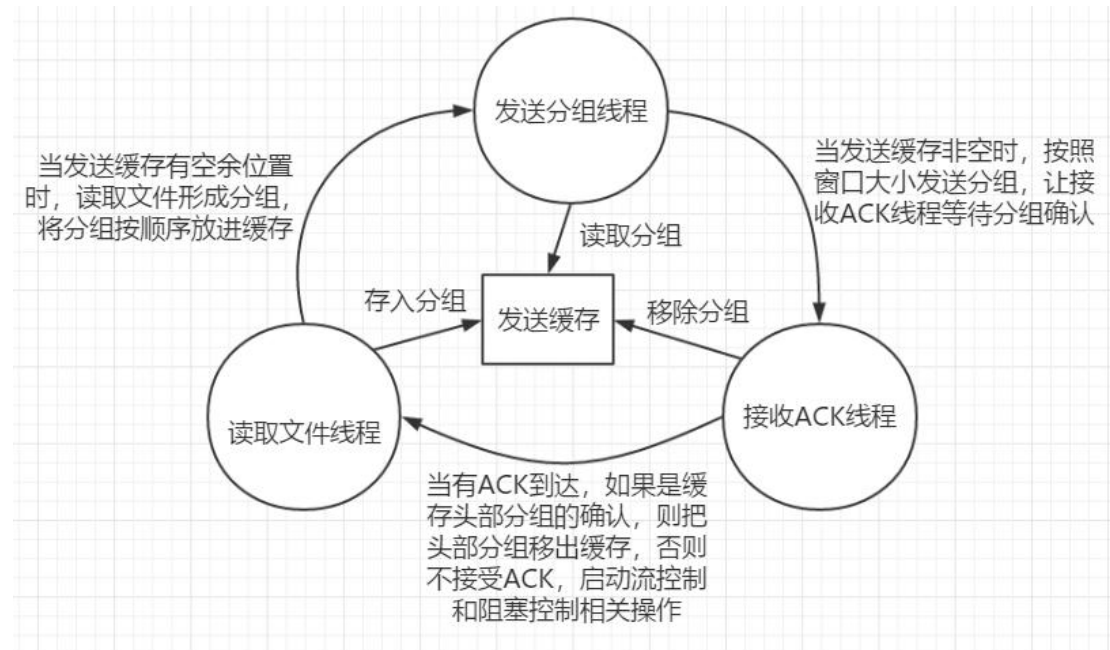


四、下载模型



五、发送方代码运行逻辑

当客户端或者服务器确认自己是发送方时，启动发送分组线程作为主线程，读取文件线程和接受 ACK 线程为子线程，3 个线程对缓存、窗口大小等全局变量进行操作。在更改全局变量完成发送逻辑时需要利用线程锁的机制，避免读写冲突。



读取文件线程检测到已经读取完全部文件分组时即可结束线程，即使有部分分组仍未确认，但它们已经读取到发送缓存中，读取文件线程的结束并不会造成影响。

接收 ACK 线程接收到所有分组确认后便可以结束自身线程，此时所有分组已经发送和确认，发送分组线程结束，完成整个文件发送的过程。

1. 发送分组线程

```
1. def sendPacket(self, senderSocket, dst, filePath, fileSize):
2.     #socket 设置超时限制
3.     senderSocket.settimeout(4)
4.     #启动 getACK、文件读取线程
5.     getACKFunc = threading.Thread(target = self.getACK, args = (senderSocket, fileSize,))
6.     readFile = threading.Thread(target = self.readFile, args = (filePath, fileSize,))
7.     getACKFunc.start()
8.     readFile.start()
9.     #开始传输文件
10.    while (not self.finish):
11.        if (self.mutex.acquire(5)):
12.            # LastPacketSent - LastPacketAked <= min {rwnd, cwnd}
13.            if self.newSeqnum - self.base > self.rwnd or self.newSeqnum - self.base > self.cwnd:
14.                self.mutex.release()
15.                continue
16.            # 发送窗口有等待发送的 packet
17.            if len(self.sendList) != 0 and self.newSeqnum - self.base < len(self.sendList):
18.                # 获得窗口队列可发送而未发送的 packet
19.                packet = self.sendList[self.newSeqnum - self.base]
```

```

20.             # 发送 packet
21.             print ("Sending packet " + str(self.newSeqnum))
22.             senderSocket.sendto(packet, dst)
23.             self.newSeqnum = self.newSeqnum + 1
24.             self.mutex.release()
25.             #控制发送间隔
26.             time.sleep(self.waitingTime)
27.             #文件传输结束
28.             senderSocket.close()

```

2. 读取文件线程

```

1.     def readFile(self, filePath, fileSize):
2.         with open(filePath, 'rb') as file:
3.             info = os.lstat(filePath)
4.             fileSize = info.st_size
5.             stream = file.read()
6.             #开始读取文件
7.             while True:
8.                 if (self.mutex.acquire(5)):
9.                     if len(self.sendList) < self.windowSize:
10.                        packet = b''
11.                        for i in range(self.fileIter, self.fileIter + self.fSize):
12.                            if (i == fileSize):
13.                                break
14.                            packetpacket = packet + bytes((stream[i],))
15.                            self.fileIter = self.fileIter + 1
16.                        if (packet != b''):
17.                            #packet 添加 header
18.                            ...
19.                            self.sendList.append(packet) #将报文添加进发送窗口
20.                            self.seqInPacket += 1
21.                            i += 1
22.                        else:
23.                            self.mutex.release()
24.                            break
25.                            self.mutex.release()
26.                    else:
27.                        # 发送窗口已满
28.                        self.mutex.release()
29.                        time.sleep(self.waitingTime)
30.                else:
31.                    time.sleep(self.waitingTime)
32.            file.close()

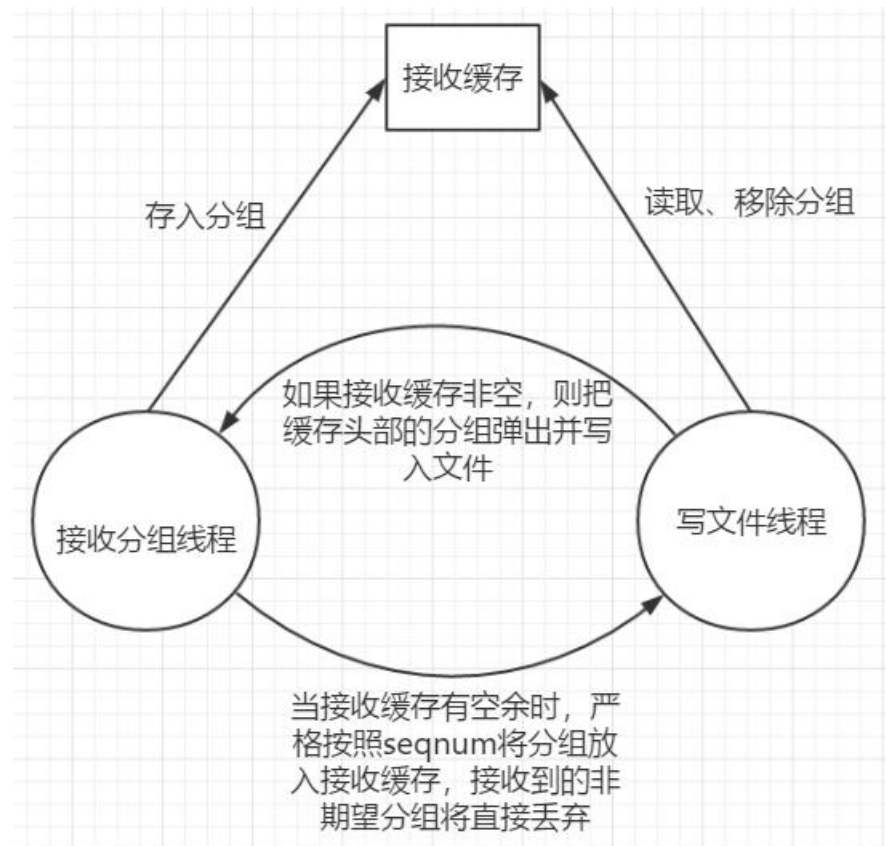
```

3. 接收 ACK 线程

```
1.  def getACK(self, senderSocket, fileSize):
2.      # 发送窗口有等待确认的序号
3.      seqTemp = 0
4.      overtime = 0 # 同一序列 seqTemp 报文重发次数
5.      redundancyACK = 0 # 冗余 ACK
6.      redundancyTimes = 0 # 冗余次数
7.      while True:
8.          if (self.newSeqnum - self.base) != 0:
9.              try:
10.                 header, addr = senderSocket.recvfrom(self.bufsize)
11.             except BaseException as e:
12.                 #超时
13.                 print(e)
14.                 #超时时发送方将重传所有已发送而未确认的分组
15.                 ...
16.             else:
17.                 #没超时
18.                 if (self.mutex.acquire(5)):
19.                     ACK, rwnd = struct.unpack("II", header)
20.                     # 假如收到的是发送窗的 base 分组, 则更新窗口队列
21.                     if (ACK < self.base):
22.                         break;
23.                     if (ACK == self.base):
24.                         redundancyTimes = 0
25.                         self.base += 1
26.                         self.sendList.pop(0) # 更新窗口队列, 将 base 分组弹出
27.                         ...
28.                         #接收到全部的 ACK
29.                         if (self.fileIter == fileSize and len(self.sendList) == 0):
30.                             self.finish = True
31.                             self.mutex.release()
32.                             break
33.                     else:
34.                         #未按序接收确认时发送方将重传所有已发送而未确认的分组
35.                         ...
36.                         self.mutex.release()
37.                 if overtime >= 10:
38.                     break
39.                 #接收 ACK 超时过多的情况处理, 关闭文件发送
40.                 if (not self.finish):
41.                     self.finish = True
```

六、接收方代码运行逻辑

为了实现流控制，文件接收方也需要设置一个缓存窗口，将接受分组和写文件的操作分成 2 个独立线程。



每当接收分组线程接收到 1 个正确的分组时，回馈的 ACK 的同时也会把接收缓存剩余的窗口大小（`rwnd`）返回，发送方根据这个返回的 `rwnd` 进行流控制的相关操作。当写文件线程把最后的分组写入文件，接收缓存被清空，宣布整个接收文件的过程结束，2 个线程不再接收分组和写文件。

1. 接受分组线程

```
1. def receivePacket(self, receiveSocket, fileName, fileSize):
2.     self.fileSize = fileSize
3.     ACK = 0
4.     bufsize = 1024
5.     receiveSocket.settimeout(4)
6.     timeoutCount = 0
7.     writeFile = threading.Thread(target = self.writeFile, args = (fileName,))
8.     writeFile.start()
9.     while(True):
10.         if timeoutCount >= 10:
11.             break
12.         if (self.fileIter + 1 == fileSize):
13.             if self.finish:
14.                 break
15.             time.sleep(self.waitingTime)
```

```

16.         continue
17.         # 接收缓存使用量 小于 接收缓存大小 (单位: 分组)
18.         if len(self.receiveBuffer) < self.receiveBufferSize:
19.             try:
20.                 data, addr = receiveSocket.recvfrom(bufsize)
21.                 self.addrTemp = addr
22.             except BaseException as e:
23.                 print (e)
24.                 timeoutCounttimeoutCount = timeoutCount + 1
25.                 time.sleep(self.waitingTime)
26.             else:
27.                 #header 长度 8 个字节
28.                 header = data[0:8]
29.                 datadata = data[8:]
30.                 seqnum, flIter = struct.unpack("II", header)
31.                 #如果是期望收到的 packet, 则写入缓存
32.                 if (seqnum == ACK):
33.                     self.fileIter = flIter
34.                     print("Packet accepted, ACK: " + str(ACK))
35.                     self.mutex.acquire()
36.                     self.receiveBuffer.append(data)
37.                     self.mutex.release()
38.                     rwnd = self.receiveBufferSize - len(self.receiveBuffer)
39.                     feedback = struct.pack("II", ACK, rwnd)
40.                     receiveSocket.sendto(feedback, addr)
41.                     ACK += 1
42.                 else:
43.                     rwnd = self.receiveBufferSize - len(self.receiveBuffer)
44.                     feedback = struct.pack("II", ACK, rwnd)
45.                     receiveSocket.sendto(feedback, addr)
46.                 # 接收窗口已满
47.             else:
48.                 while len(self.receiveBuffer) >= self.receiveBufferSize:
49.                     # 发送拒收信息
50.                     rwnd = 0
51.                     feedback = struct.pack("II", ACK, rwnd)
52.                     receiveSocket.sendto(feedback, self.addrTemp)
53.                     time.sleep(self.waitingTime)
54.                 # 接收窗口重新有空余, 发送接收信息
55.                 rwnd = self.receiveBufferSize - len(self.receiveBuffer)
56.                 feedback = struct.pack("II", ACK, rwnd)
57.                 receiveSocket.sendto(feedback, self.addrTemp)

```

2. 写文件线程

```

1. def writeFile(self, fileName):
2.     #创建文件
3.     with open(fileName, "ab") as file:
4.         i = 0
5.         while True:
6.             if len(self.receiveBuffer) != 0:
7.                 file.write(self.receiveBuffer[0])
8.                 print("Write " + str(i) + " packet.")
9.                 i += 1
10.                self.mutex.acquire()
11.                self.receiveBuffer.pop(0)
12.                if self.fileIter + 1 == self.fileSize and len(self.receiveBuffer) == 0:
13.                    self.finish = True
14.                    self.mutex.release()
15.                    break
16.                self.mutex.release()
17.            else:
18.                time.sleep(self.waitingTime)
19.        file.close()

```

七、流控制

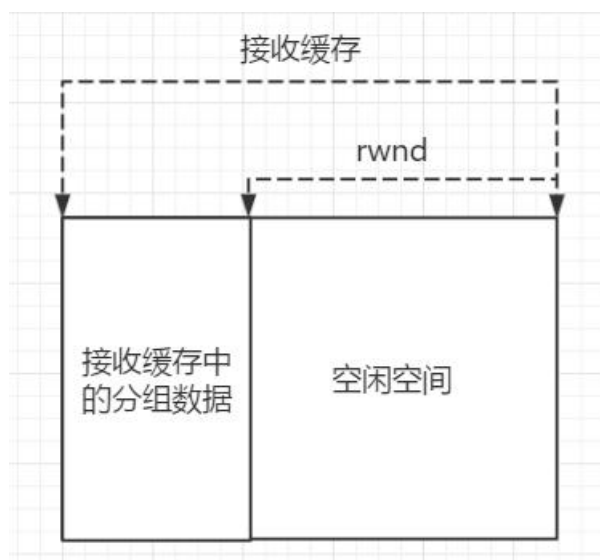
想要实现类似 TCP 的流控制，需要在发送方和接收方都设置一个缓存进行分组的存放。接收方接收到分组时把分组放入缓存，回馈接收的分组号和 `rwnd` 给发送方，发送方根据 `rwnd` 更新自身的变量。满足以下条件时才能继续发送分组，以此达到流控制的目的：

$$\text{LastPacketSent} - \text{LastPacketAked} \leq \text{rwnd}$$

其中，`LastPacketSent` 为最后发送的分组号，`LastPacketAked` 为最后确认的分组号。

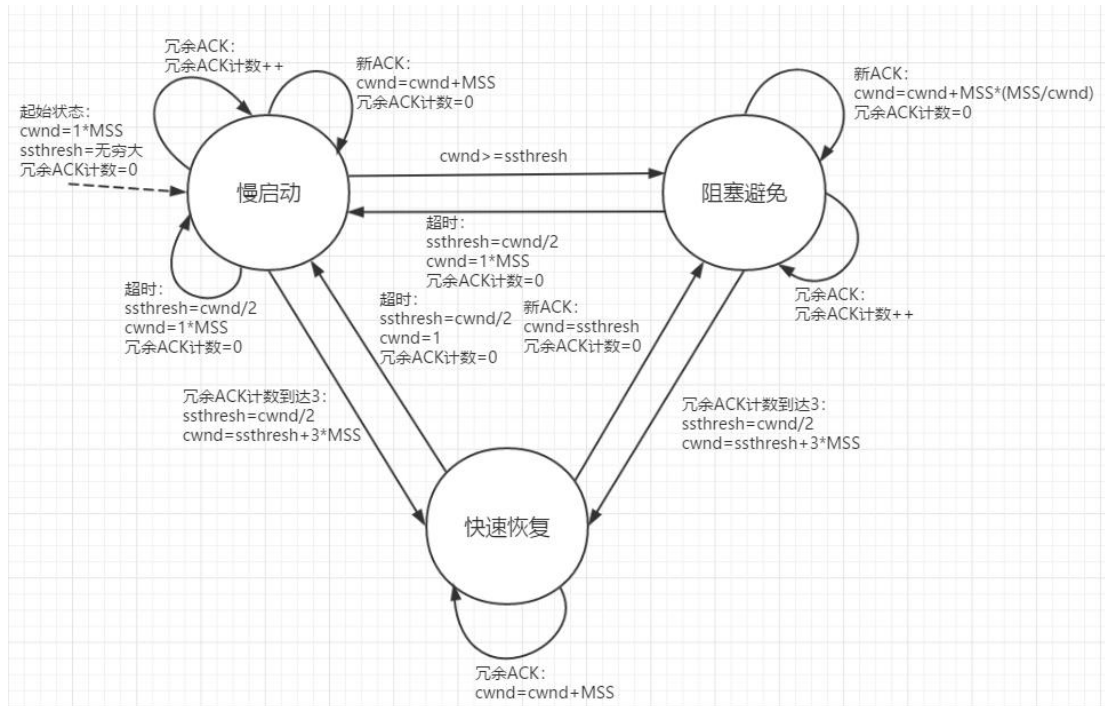
当引入阻塞控制时，条件变为：

$$\text{LastPacketSent} - \text{LastPacketAked} \leq \min \{ \text{rwnd}, \text{cwnd} \}$$



八、阻塞控制

LFTP 的阻塞控制仿照 TCP 实现，发送方一开始处于慢启动状态，根据 $cwnd$ 、 $ssthresh$ 和冗余 ACK 计数三个变量进行状态变换，最终实现发送窗口的大小控制。



九、多用户

服务器用 23333 端口接收客户端请求，每当接收到一个请求，服务器则会分配一个端口给一个新的子线程，这个子线程负责与客户端完成文件传输或者下载的交互，主线程继续监听 23333 端口，等待另一个新请求的到来。

1. 服务器

```

1. while True:
2.     request, addr = udpServer.recvfrom(bufsize)
3.     cmd, fname, fileSize = request.split(b'\n', 2)
4.     cmd = str(cmd, encoding = 'utf-8')
5.     fname = str(fname, encoding = 'utf-8')
6.     fileSize = int(str(fileSize, encoding='utf-8'))
7.     clientCount += 1
8.     newSocket = socket(AF_INET,SOCK_DGRAM)
9.     newAddr = (host, port + clientCount)
10.    newSocket.bind(newAddr)
11.    #把新线程的端口告诉客户端
12.    data = bytes(str(port + clientCount), encoding = 'utf-8')
13.    udpServer.sendto(data, addr)
14.    if (cmd == "send"):
15.        #创建接收文件子线程
16.        sendThread = threading.Thread(target = send, args = (newSocket, fname, fileSize,))
17.        sendThread.start()
18.    else:
19.        #创建发送文件子线程

```

```
20.         getThread = threading.Thread(target = get, args = (udpServer, newSocket, addr, flname,))
21.         getThread.start()
```

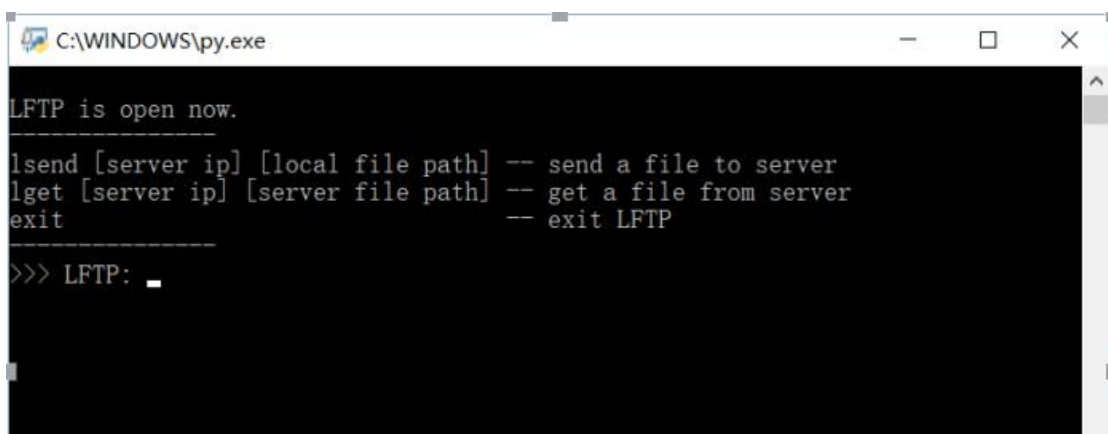
2. 客户端

```
1.     #握手需要的基本信息
2.     cmd = bytes("send", encoding = 'utf-8')
3.     filename = self.filepath.split('/')[ -1].split('\\')[ -1] # split the path to get the filename
4.     filename = bytes(filename, encoding = 'utf-8')
5.     flSize = bytes(str(fileSize), encoding='utf-8')
6.     request = cmd + b'\n' + filename + b'\n' + flSize
7.
8.     # send the command and filepath to server
9.     udpClient = socket(AF_INET, SOCK_DGRAM)
10.    udpClient.sendto(request, dst)
11.
12.    #等待服务器新线程分配的端口地址
13.    print("Waiting for addr")
14.    newPort, addr = udpClient.recvfrom(self.bufsize)
15.    newPort = int(str(newPort, encoding = 'utf-8'))
16.    newAddr = (self.host, newPort)
```

十、传输测试

作业要求文件传输不仅仅在校园网内进行，所以我们租用了一个服务器，在上面运行服务端代码，个人电脑运行客户端代码。为了方便用户使用 LFTP，我们还实现了一个简单的 UI 窗口，客户端启动 window.py 即可输入指令，UI 窗口有基本的指令错误提示。

1. UI 窗口



```
C:\WINDOWS\py.exe

LFTP is open now.
-----
lsend [server ip] [local file path] -- send a file to server
lget [server ip] [server file path] -- get a file from server
exit                                -- exit LFTP
-----
>>> LFTP: _
```

```
C:\WINDOWS\py.exe

LFTP is open now.
-----
lsend [server ip] [local file path] -- send a file to server
lget [server ip] [server file path] -- get a file from server
exit -- exit LFTP
-----

>>> LFTP: get 134.175.103.254 D:\lena.jpg 错误指令
Command get is invalid, please check your input.

LFTP is open now.
-----
lsend [server ip] [local file path] -- send a file to server
lget [server ip] [server file path] -- get a file from server
exit -- exit LFTP
-----

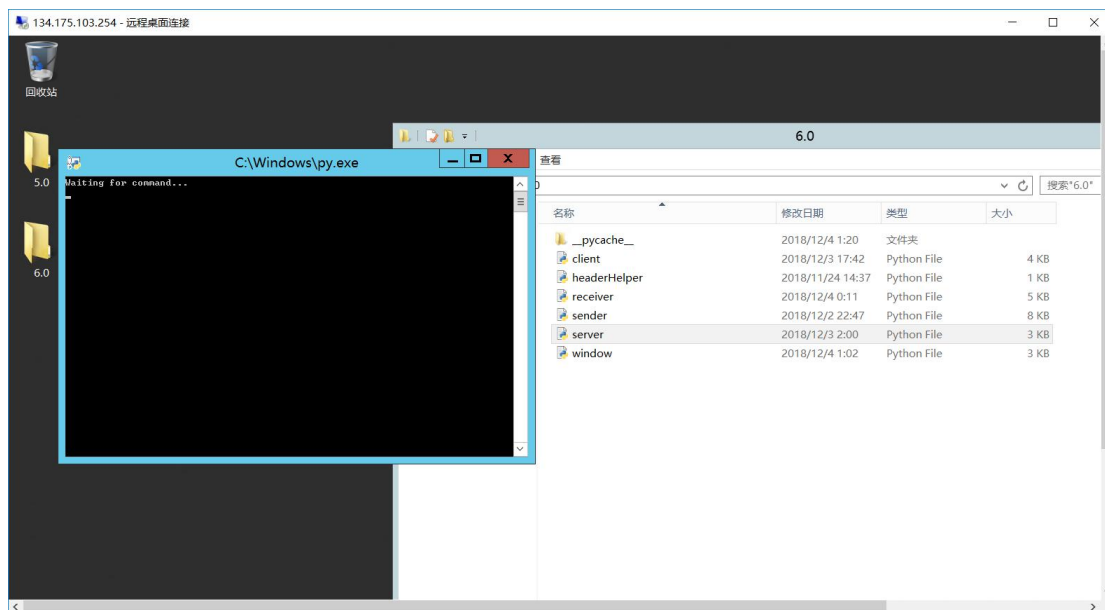
>>> LFTP: lget 172.18.34.162 D:\lena.jpg 错误IP地址
IP address 172.18.34.162 is invalid, please check your input.

LFTP is open now.
-----
lsend [server ip] [local file path] -- send a file to server
lget [server ip] [server file path] -- get a file from server
exit -- exit LFTP
-----

>>> LFTP: _
```

2. Lsend 指令测试

(1) 租用服务器运行服务端代码，等待客户端请求。



(2) 客户端发送 Lsend 指令。

```
LFTP is open now.
-----
lsend [server ip] [local file path] -- send a file to server
lget [server ip] [server file path] -- get a file from server
exit -- exit LFTP
-----

>>> LFTP: lsend 134.175.103.254 D:\lena.jpg _
```

(3) 文件发送中。

服务器：

```
C:\Windows\py.exe

Write 41 packet.
Received packet 42
Packet accepted, ACK: 42
rwnd: 19
Received packet 43
Packet accepted, ACK: 43
rwnd: 18
Write 42 packet.
Write 43 packet.
Received packet 44
Packet accepted, ACK: 44
rwnd: 19
Received packet 45
Packet accepted, ACK: 45
rwnd: 18
Write 44 packet.
Write 45 packet.
Received packet 46
Packet accepted, ACK: 46
rwnd: 19
Received packet 47
Packet accepted, ACK: 47
rwnd: 18
Received packet 48
Packet accepted, ACK: 48
rwnd: 17
Received packet 49
Packet accepted, ACK: 49
rwnd: 16
Write 46 packet.
Write 47 packet.
Write 48 packet.
Write 49 packet.
Received packet 50
Packet accepted, ACK: 50
rwnd: 19
Received packet 51
Packet accepted, ACK: 51
```

```
C:\Windows\py.exe

rwnd: 18
Write 106 packet.
Write 107 packet.
Received packet 108
Packet accepted, ACK: 108
rwnd: 19
Received packet 109
Packet accepted, ACK: 109
rwnd: 18
Received packet 110
Packet accepted, ACK: 110
rwnd: 17
Received packet 111
Packet accepted, ACK: 111
rwnd: 16
Received packet 112
Packet accepted, ACK: 112
rwnd: 15
Received packet 113
Packet accepted, ACK: 113
rwnd: 14
Received packet 114
Packet accepted, ACK: 114
rwnd: 13
Write 108 packet.
Write 109 packet.
Write 110 packet.
Write 111 packet.
Write 112 packet.
Write 113 packet.
Write 114 packet.
Received packet 115
Packet accepted, ACK: 115
rwnd: 19
Write 115 packet.
-----
Receive file successfully.
```

客户端：

```
C:\WINDOWS\py.exe

LFTP is open now.
-----
lsend [server ip] [local file path] -- send a file to server
lget [server ip] [server file path] -- get a file from server
exit                                -- exit LFTP
-----

>>> LFTP: lsend 134.175.103.254 D:\lena.jpg
Waiting for addr
Now begin to send the file.
filePath is D:\lena.jpg, fileSize is 23037
appending packet
base:0 seq:0 len:1
appending packet
base:0 seq:0 len:2
appending packet
base:0 seq:0 len:3
Sending packet 0
base:0 seq:1 len:3
appending packet
base:0 seq:1 len:4
appending packet
base:0 seq:1 len:5
appending packet
base:0 seq:1 len:6
appending packet
```

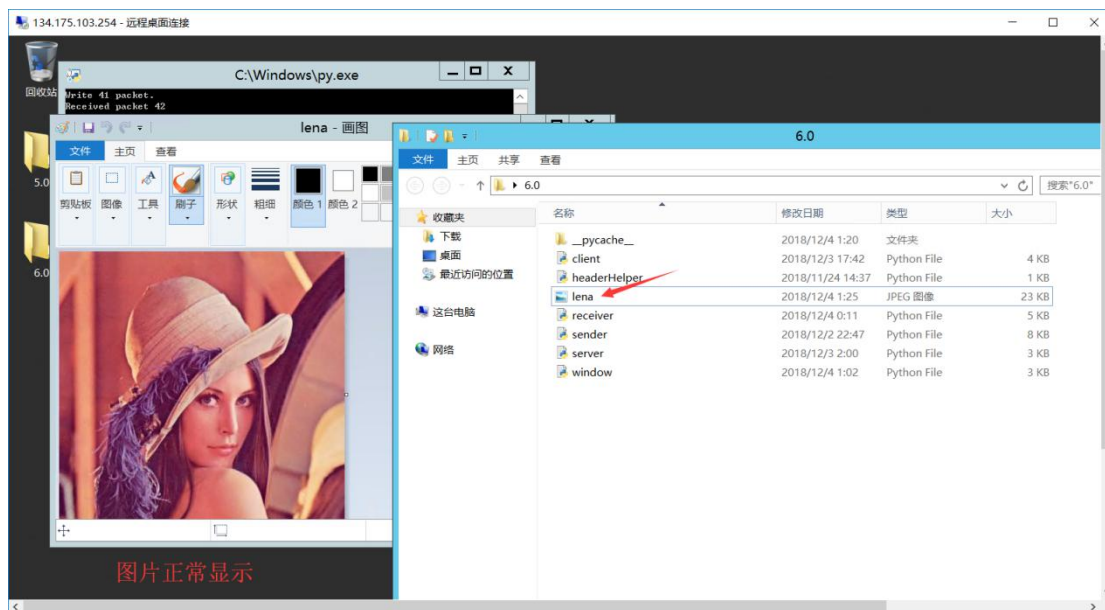
```
C:\WINDOWS\py.exe

base:111 seq:114 len:5
Receive ACK 111
base:112 seq:114 len:4
Sending packet 114
base:112 seq:115 len:4
Receive ACK 112
base:113 seq:115 len:3
Sending packet 115
base:113 seq:116 len:3
Receive ACK 113
base:114 seq:116 len:2
Receive ACK 114
base:115 seq:116 len:1
Receive ACK 115
base:116 seq:116 len:0
-----
Finish receiving ACK.
Finish sending the file.

LFTP is open now.
-----
lsend [server ip] [local file path] -- send a file to server
lget [server ip] [server file path] -- get a file from server
exit                                -- exit LFTP
-----

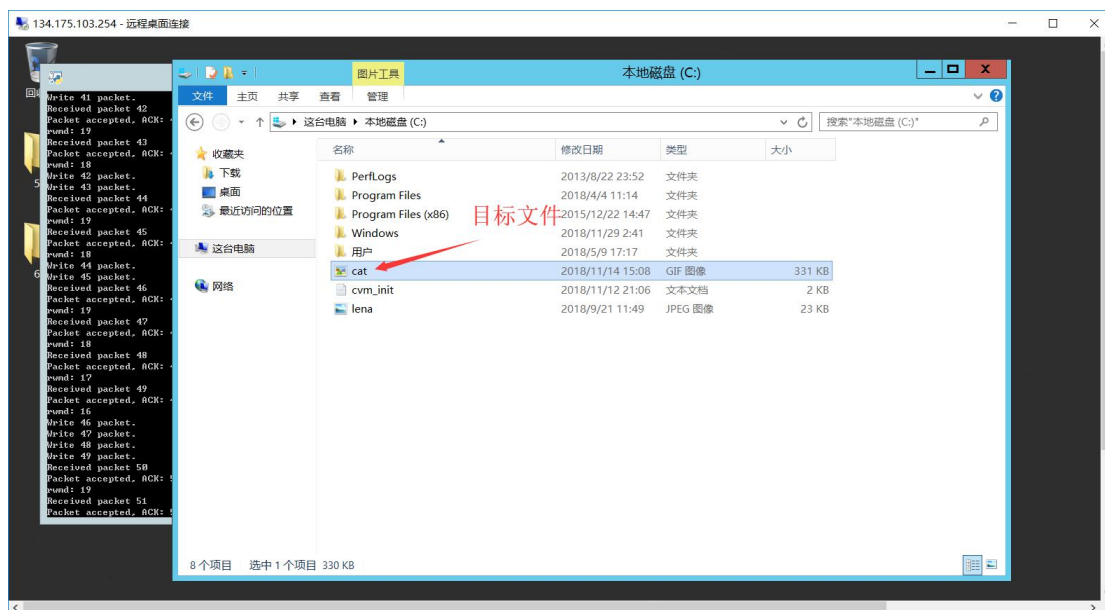
>>> LFTP: _
```

(4) 文件发送结束，可以在服务器看到接收到的 lena.jpg 图片，并且可以正常打开。

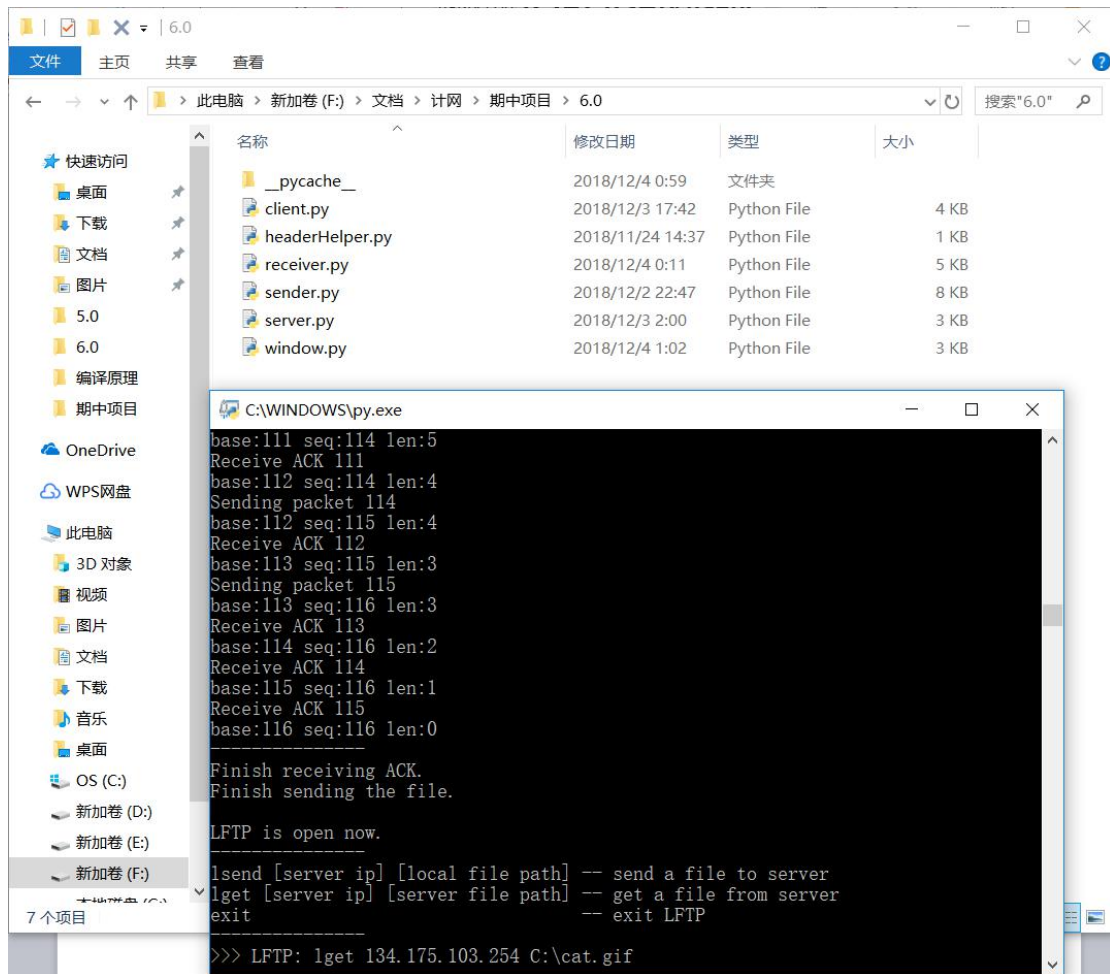


3. lget 指令测试

(1) 服务器存在目标文件 C:\cat.gif



(2) 客户端通过 lget 指令请求下载 C:\cat.gif, 此时下载文件目录中并没有 cat.gif



(3) 文件发送中。

服务器:

```
C:\Windows\py.exe

Write 112 packet.
Write 113 packet.
Write 114 packet.
Received packet 115
Packet accepted, ACK: 115
rwnd: 19
Write 115 packet.

-----
Receive file successfully.
Now begin to send the file.
filePath is C:\cat.gif, fileSize is 338122
appending packet
base:0 seq:0 len:1
appending packet
base:0 seq:0 len:2
appending packet
base:0 seq:0 len:3
appending packet
base:0 seq:0 len:4
appending packet
base:0 seq:0 len:5
appending packet
base:0 seq:0 len:6
appending packet
base:0 seq:0 len:7
appending packet
base:0 seq:0 len:8
appending packet
base:0 seq:0 len:9
appending packet
base:0 seq:0 len:10
appending packet
base:0 seq:0 len:11
appending packet
base:0 seq:0 len:12
appending packet
base:0 seq:0 len:13
appending packet
```

```
C:\Windows\py.exe

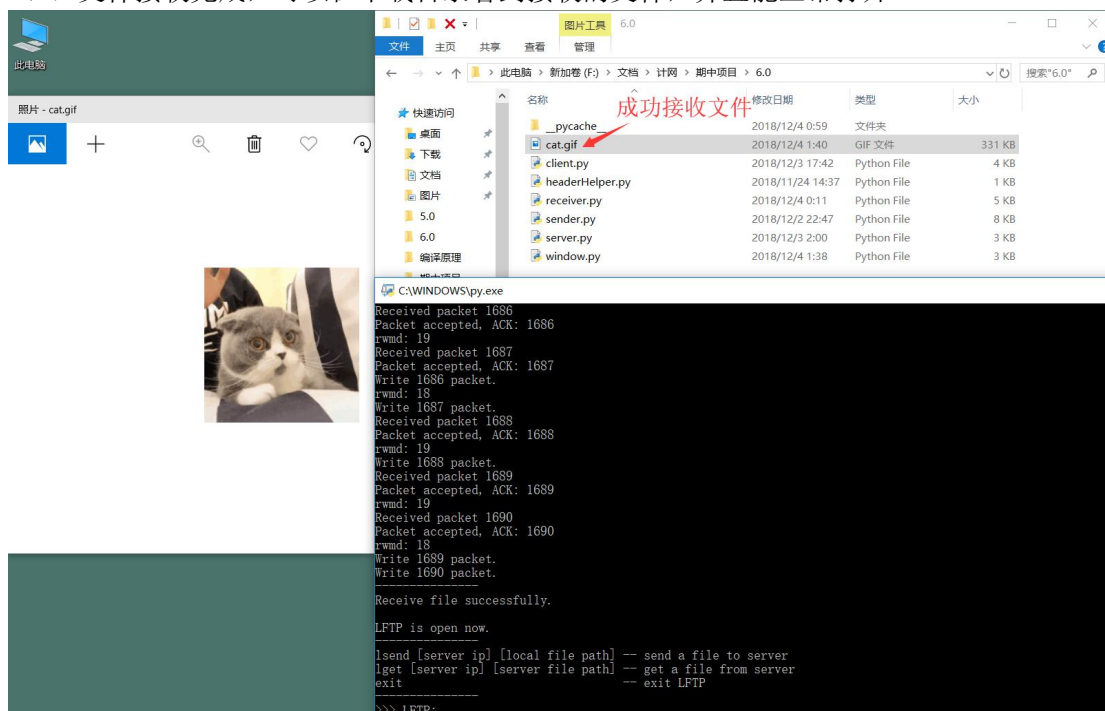
base:266 seq:266 len:19
appending packet
base:266 seq:266 len:20
Sending packet 266
base:266 seq:267 len:20
Receive ACK 266
base:267 seq:267 len:19
Sending packet 267
base:267 seq:268 len:19
appending packet
base:267 seq:268 len:20
Receive ACK 267
base:268 seq:268 len:19
appending packet
base:268 seq:268 len:20
Sending packet 268
base:268 seq:269 len:20
Receive ACK 268
base:269 seq:269 len:19
Sending packet 269
base:269 seq:270 len:19
appending packet
base:269 seq:270 len:20
Receive ACK 269
base:270 seq:270 len:19
Sending packet 270
base:270 seq:271 len:19
appending packet
base:270 seq:271 len:20
Receive ACK 270
base:271 seq:271 len:19
appending packet
base:271 seq:271 len:20
Sending packet 271
base:271 seq:272 len:20
Receive ACK 271
base:272 seq:272 len:19
```

客户端:


```
C:\WINDOWS\py.exe
Receive ACK 113
base:114 seq:116 len:2
Receive ACK 114
base:115 seq:116 len:1
Receive ACK 115
base:116 seq:116 len:0
-----
Finish receiving ACK.
Finish sending the file.

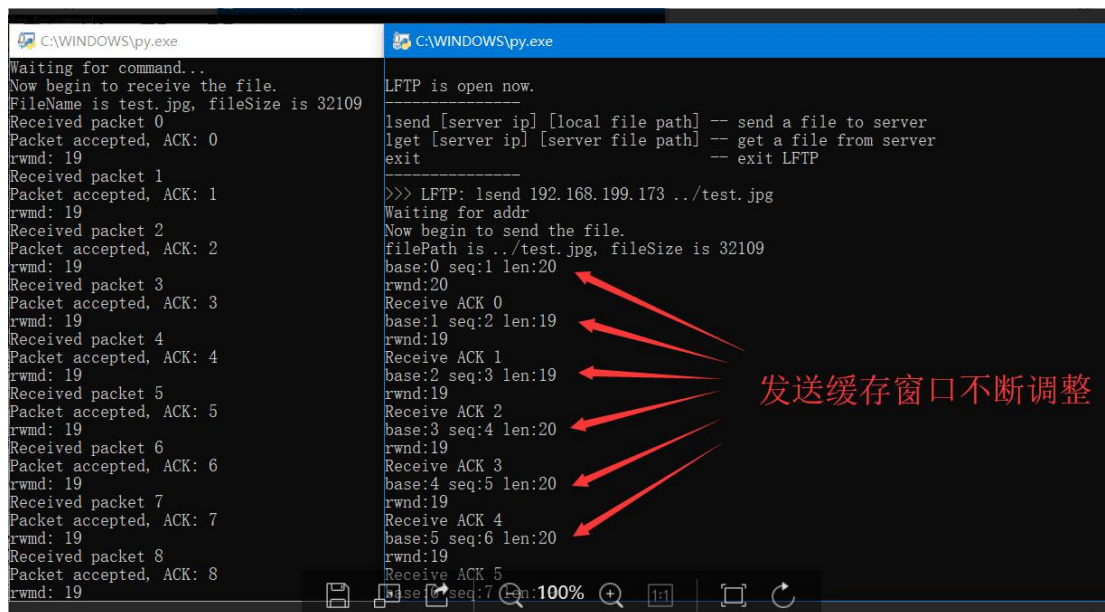
LFTP is open now.
-----
lsend [server ip] [local file path] -- send a file to server
lget [server ip] [server file path] -- get a file from server
exit -- exit LFTP
-----
>>> LFTP: lget 134.175.103.254 C:\cat.gif
Waiting for addr
('134.175.103.254', 23339)
Now begin to receive the file.
FileName is cat.gif, fileSize is 338122
Received packet 0
Packet accepted, ACK: 0
rwnd: 19
Write 0 packet.
Received packet 1
```

(4) 文件接收完成，可以在下载目录看到接收的文件，并且能正常打开。



4. 流控制测试

流控制机制的作用是匹配发送方和接收方的缓存读写速度,接收方每次返回确认分组都携带接收缓存窗口长度,发送方根据这个接收缓存窗口长度调整自己的发送缓存窗口长度。在截图中可以明显地看到发送缓存窗口长度不断地在调整,流控制机制在正确运作。



```
C:\WINDOWS\py.exe
Waiting for command...
Now begin to receive the file.
FileName is test.jpg, fileSize is 32109
Received packet 0
Packet accepted, ACK: 0
rwnd: 19
Received packet 1
Packet accepted, ACK: 1
rwnd: 19
Received packet 2
Packet accepted, ACK: 2
rwnd: 19
Received packet 3
Packet accepted, ACK: 3
rwnd: 19
Received packet 4
Packet accepted, ACK: 4
rwnd: 19
Received packet 5
Packet accepted, ACK: 5
rwnd: 19
Received packet 6
Packet accepted, ACK: 6
rwnd: 19
Received packet 7
Packet accepted, ACK: 7
rwnd: 19
Received packet 8
Packet accepted, ACK: 8
rwnd: 19

C:\WINDOWS\py.exe
LFTP is open now.
-----
lsend [server ip] [local file path] -- send a file to server
lget [server ip] [server file path] -- get a file from server
exit
-----
>>> LFTP: lsend 192.168.199.173 ../test.jpg
Waiting for addr
Now begin to send the file.
filePath is ../test.jpg, fileSize is 32109
base:0 seq:1 len:20
rwnd:20
Receive ACK 0
base:1 seq:2 len:19
rwnd:19
Receive ACK 1
base:2 seq:3 len:19
rwnd:19
Receive ACK 2
base:3 seq:4 len:20
rwnd:19
Receive ACK 3
base:4 seq:5 len:20
rwnd:19
Receive ACK 4
base:5 seq:6 len:20
rwnd:19
Receive ACK 5
base:6 seq:7 len:20
rwnd:19
```

5. 阻塞控制测试

为了测试阻塞控制,在接收方添加随机数代码,当随机的数字小于 0.6 时不接收分组(即使它是期望收到的分组),即每个期望分组只有 40%的概率被接收方接收,以此造成阻塞假象,达到测试目的。

```
#如果是期望收到的packet, 则写入缓存
if (seqnum == ACK):
    rand = random.random()
    if (rand < 0.6):
        print("随机触发丢包, 检验cwnd")
        continue
    self.fileIter = flIter
    print("Packet accepted, ACK: " + str(ACK))
    self.mutex.acquire()
    self.receiveBuffer.append(data)
    self.mutex.release()

    rwnd = self.receiveBufferSize - len(self.receiveBuffer)
    # print("rwnd: " + str(rwnd))
    feedback = struct.pack("II", ACK, rwnd)
    receiveSocket.sendto(feedback, addr)
    ACK += 1
```

(1) 由于连续 3 次冗余 ACK, 触发了阻塞控制, 状态由慢启动变为快速恢复, ssthresh 变为 cwnd/2, cwnd 变为 ssthresh+3。

```
C:\WINDOWS\py.exe
Packet accepted, ACK: 7
Received packet 8
Packet accepted, ACK: 8
随机触发丢包, 检验cwnd
Received packet 10
Received packet 9
随机触发丢包, 检验cwnd
Received packet 10
Received packet 9
随机触发丢包, 检验cwnd
Received packet 10
Received packet 9
随机触发丢包, 检验cwnd
Received packet 10
Received packet 9
随机触发丢包, 检验cwnd
Received packet 10
Received packet 9
随机触发丢包, 检验cwnd
Received packet 10
Received packet 9
Packet accepted, ACK: 9
Received packet 10
随机触发丢包, 检验cwnd
Received packet 11
Received packet 10

选择C:\WINDOWS\py.exe
[fault]Receive ACK 8
base:9 seq:10 len:20
cwnd:10
base:9 seq:11 len:20
cwnd:10
[fault]Receive ACK 8
base:9 seq:10 len:20
cwnd:10
base:9 seq:11 len:20
cwnd:10
[fault]Receive ACK 8
base:9 seq:10 len:20
cwnd:10
base:9 seq:11 len:20
cwnd:10
[fault]Receive ACK 8
base:9 seq:10 len:20
cwnd:8
base:9 seq:11 len:20
cwnd:8
[fault]Receive ACK 8
base:9 seq:10 len:20
cwnd:7
base:9 seq:11 len:20
cwnd:7
[fault]Receive ACK 8
base:9 seq:10 len:20
cwnd:6
base:9 seq:11 len:20
cwnd:6
```

第1次ACK冗余
第2次ACK冗余
第3次ACK冗余
因为ACK冗余到达3次, 阈值变为cwnd一半, cwnd变为阈值+3

(2) 接收方尝试接收分组 40, 触发了随机丢包, 但是因为这是最后一个分组, 所以没有后续报文造成的冗余回送 ACK, 于是触发了超时, 发送方没接收到确认 ACK 认定超时, 不管此时阻塞控制处于哪个状态, 都转变为慢启动状态, cwnd 重置为 1。

```
C:\WINDOWS\py.exe
随机触发丢包, 检验cwnd
Received packet 36
Received packet 35
Packet accepted, ACK: 35
Received packet 36
Packet accepted, ACK: 36
Received packet 37
随机触发丢包, 检验cwnd
Received packet 38
Received packet 37
Packet accepted, ACK: 37
Packet accepted, ACK: 38
Received packet 39
随机触发丢包, 检验cwnd
Received packet 40
Received packet 39
随机触发丢包, 检验cwnd
Received packet 40
Received packet 39
Packet accepted, ACK: 39
Received packet 40
随机触发丢包, 检验cwnd
timed out
Receive packet timeout happens.
Received packet 40
Packet accepted, ACK: 40
Receive file successfully.

C:\WINDOWS\py.exe
Receive ACK 38
base:39 seq:41 len:2
cwnd:17
[fault]Receive ACK 38
base:39 seq:40 len:2
cwnd:17
base:39 seq:41 len:2
cwnd:17
[fault]Receive ACK 38
base:39 seq:40 len:2
cwnd:17
Receive ACK 39
base:40 seq:41 len:1
cwnd:18
timed out
Receive ACK timeout happens.
base:40 seq:41 len:1
cwnd:1
Receive ACK 40
Finish sending the file.
Finish receiving ACK.
LFTP is open now.
lsend [server ip] [local file path] -- send a file to server
lget [server ip] [server file path] -- get a file from server
exit
-- exit LFTP
```

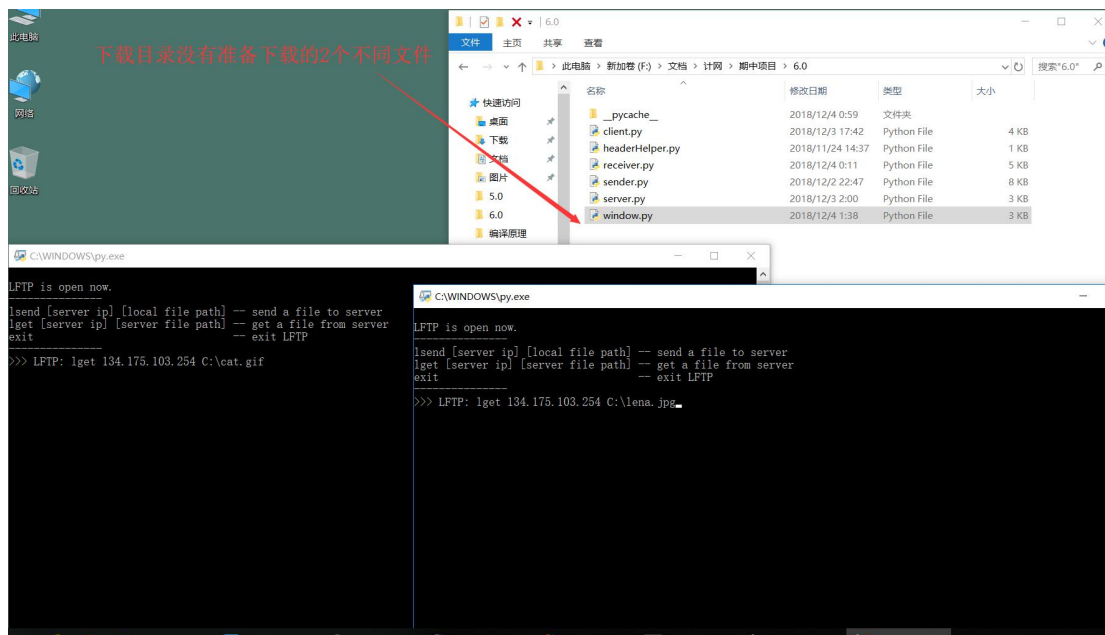
超时事件发生
变为慢启动状态, cwnd=1

(3) 经过冗余 ACK 和超时事件的测试, 阻塞控制机制都如预想中的那样运作, 测试成功!

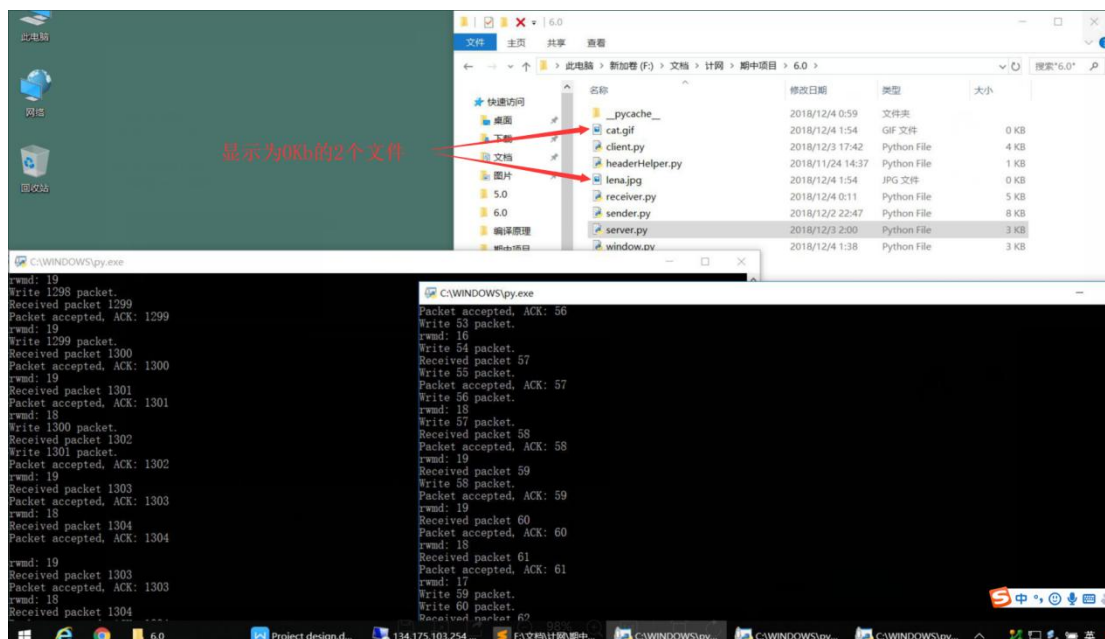
6. 多用户测试

为了验证服务器支持多用户同时下载或者传输文件, 我们在一台主机上运行两个客户端, 同时下载不同的文件, 最后在下载目录中查看文件是否成功下载并且能够正常查看。

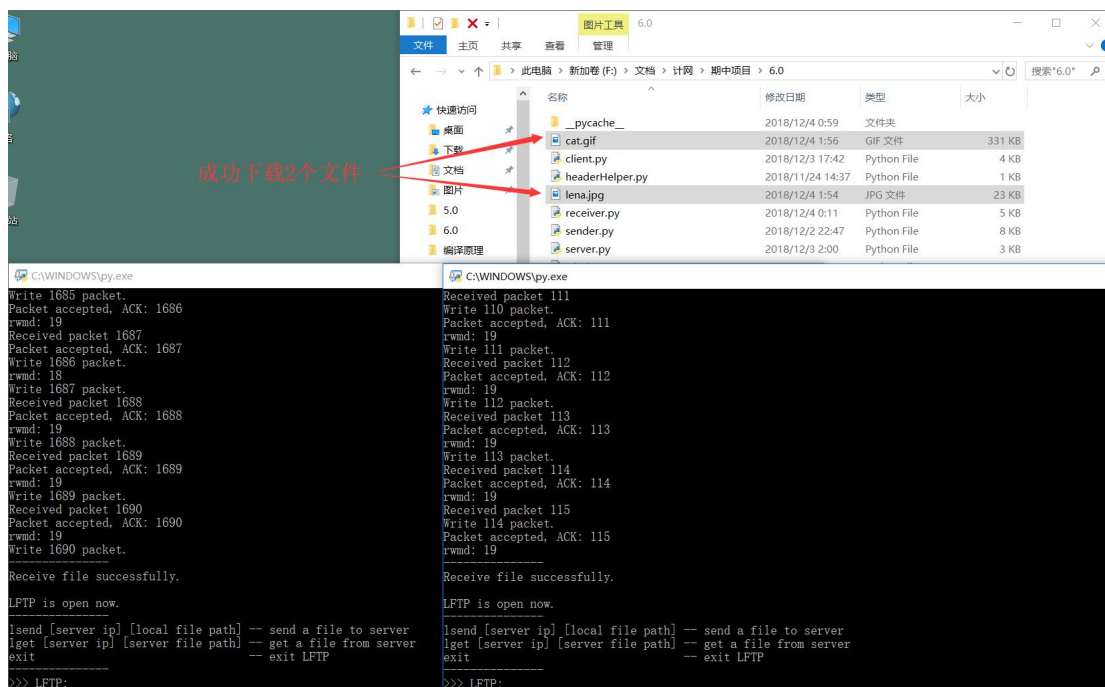
(1) 下载前



(2) 运行下载文件的指令，可以看到 2 个命令行窗口同时在下载文件并打印信息。此时下载目录已经创建了 2 个正在下载的文件，但因为下载未完成，大小显示为 0Kb。



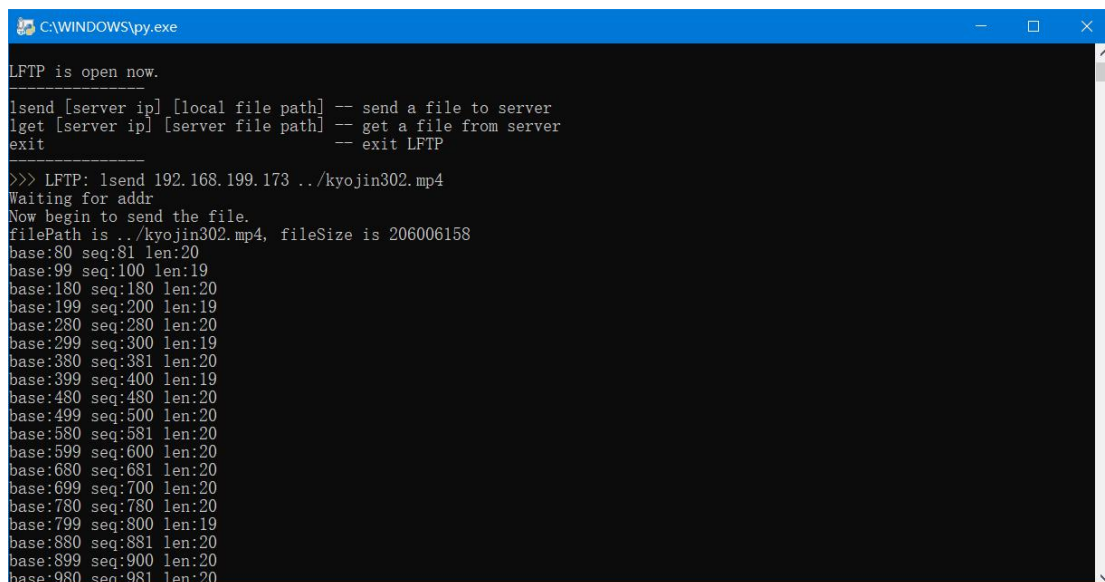
(3) 下载完成，2 个命令行窗口显示下载完成，并且可以在下载目录看到 2 个正常大小下载文件，测试成功！



7. 大文件传输测试

由于网速的限制，即使是同一台主机传输文件，预估计的最快速度也只有 50Kb/s，传输 1G 大文件的耗时大概需要 8 小时。因此我们选择了一个大小约为 200M 的 mp4 文件作为测试文件。

(1) 为了提高传输速度，大文件传输测试没有选择租用的外部服务器，而是选择了校园网内的 2 台主机作为服务器跟客户端。通过 lsend 指令向服务器请求传递 mp4 文件，可以看到文件的大小大约为 2 亿 Bytes，每个分组的大小为 800KB，计算出总共需要发送 257500 个分组。



(2) 文件传输结束，服务器成功接收了 257500 个分组。

```
C:\WINDOWS\py.exe
Packet accepted, ACK: 255000
Packet accepted, ACK: 255100
Packet accepted, ACK: 255200
Packet accepted, ACK: 255300
Packet accepted, ACK: 255400
Packet accepted, ACK: 255500
Packet accepted, ACK: 255600
Packet accepted, ACK: 255700
Packet accepted, ACK: 255800
Packet accepted, ACK: 255900
Packet accepted, ACK: 256000
Packet accepted, ACK: 256100
Packet accepted, ACK: 256200
Packet accepted, ACK: 256300
Packet accepted, ACK: 256400
Packet accepted, ACK: 256500
Packet accepted, ACK: 256600
Packet accepted, ACK: 256700
Packet accepted, ACK: 256800
Packet accepted, ACK: 256900
Packet accepted, ACK: 257000
Packet accepted, ACK: 257100
Packet accepted, ACK: 257200
Packet accepted, ACK: 257300
Packet accepted, ACK: 257400
Packet accepted, ACK: 257500
Receive file successfully.
```

(4) 查看本地文件，可以看到 mp4 文件从创建到修改总共历时 1 小时 23 分钟，也就是整个文件传输过程的耗时，平均传输速度只有 41Kb/s。打开 mp4 文件视频能正常播放，大文件传输测试成功！

mp4 创建、修改日期：

大小	196 MB
创建日期	2018/12/4 0:56
修改日期	2018/12/4 2:19

平均传输速度：

$$196 \times (2^{10})^{2/83/60}$$

41 269.2562249

播放 mp4 文件：

