Network Application Development Project A
Simple File Transfer Service

Develop a pair of client-server programs that communicate via Python stream sockets or any programming language (such as Java) and simulate partially the file transfer protocol (FTP). The main purpose of these client/server programs is to give the client the ability to download files from the server directory to the client directory and upload files from the client directory to the server directory. We should be able to transfer any file type such as txt, doc, jpg.

FTP contains two types of protocol user datagram protocol (UDP) and transmission control protocol (TCP). You should implement both protocols for the project.

A typical FTP interaction between a client and a server is as follows:

- When starting, client program connects to the server.
- If connection is successful, the client program can transfer files to the server using the put command and can retrieve files from the server using the get command. The client can also change the names of the files in the server's machine.
- The client program reads the user command, parses it, forms a message request according to the type of command, and then sends this request to the server.
- The server is expected to receive the request, parse it and execute the command. Then the server is expected to reply with an appropriate response message. The client should handle the incoming response message.
- When the client quits, it closes the communication with the server. The server should keep listening passively for new connection requests from other clients.

II. Protocol Description

The protocol that describes the interaction between the server and the client is a typical request-response protocol.

II.1 User Commands

The user utilizes the commands shown in the table below to interface with the client in order to instruct the client program to request a service from the server. The commands: put, get, change, help, and bye. The following table shows a description of each instruction at the client side.

	User Commands	**Description**
1.	put *filename*	This command instructs the client to send a *put request* to the server in order to transfer a file from the client machine to the server machine (see the section below for the format of the *put request*) Example: put file.txt
2.	get *filename*	This command instructs the client to send a *get request* to the server in order to retrieve a file from the server machine to the client machine. (see the section below for the format of the *get request*) Example: get file.txt
3.	summary *filename*	This command instructs the client to send a *summary request* to the server in order to generate the statistical summary of a specific file on the server side (The file contains just numbers). The summary is the **maximum, minimum, and average** of the numbers. The server will reply by generating a file containing the **maximum, minimum, and average** and sending it to the client. Summary: get file.txt
4.	change *OldFileName NewFileName*	This command instructs the client to send a *change request* to the server to rename a file at the server machine. (see the section below for the format of the *change request*)

		Example: change oldfile.doc newfile.doc
5.	help	This command instructs the client to send a *help request* to the server to get a list of the commands that the server support. (see the section below for the format of the *help request*) Example: help
6.	bye	This command instructs the client to break the connection with the server and exit.

II.2 Format of the Request Messages

The request message that the client sends to the server has the following format:

Operation code (opcode) / filename length	**Remaining bytes**
1 byte (3bits for opcode / 5 bits for filename length)	• 0 bytes in case of help request. • Variable numbers of bytes in case of put, get, and change request.

The request's 3-bit *opcode* encodes the type of the instruction requested; the table below describes how this information is encoded:

Opcode	**Instruction**	**Request Format**
000	put filename	<table><tr><td>Opcode</td><td>Filename Length (FL)</td><td>*FileName*</td><td>File Size (FS)</td></tr><tr><td>$b_7b_6b_5$</td><td>$b_4b_3b_2b_1b_0$</td><td>FL bytes</td><td>4Bytes</td></tr><tr><td colspan="2">Byte 1</td><td>From Byte 2 to Byte FL+1</td><td>From Byte FL+2 to Byte FL+5</td></tr></table> <table><tr><td>***File Data***</td></tr><tr><td>**FS Bytes**</td></tr></table> **Opcode:** last 3 bits (of the first byte) specify the request operation code. For *put* the binary value of these bits is 000 **Filename Length (FL):** first 5bits (of the first byte) specify the length of the name of the file. The name of the file should not exceed 31 characters including the end of string character. For example, if the filename is *example.doc* then FL value is 12. **File Name:** *FL* bytes specify the name of the transferred file. **File Size (FS): 4** bytes that specify the size of the file. **File Data:** FS bytes containing the file itself. It can be sent as chunks of data.
001	get filename	<table><tr><td>Opcode</td><td>Filename Length (FL)</td><td>*FileName*</td></tr><tr><td>$b_7b_6b_5$</td><td>$b_4b_3b_2b_1b_0$</td><td>FL bytes</td></tr><tr><td colspan="2">Byte 1</td><td>From Byte 2 to Byte FL+1</td></tr></table> **Opcode:** last 3 bits (of the first byte) specify the request operation code. For *get* the binary value of these bits is 001 **Filename Length (FL):** first 5bits (of the first byte) specify the length of the name of the file. The name of the file should not exceed 31 characters including the end of string character. **File Name:** *FL* bytes specify the name of the retrieved file.

<table>
<tr><td>010</td><td>Change
oldFilename
newFilename</td><td>
<table>
<tr><td>Opcode</td><td>oldFilename Length (OFL)</td><td>oldFilename</td><td>newFilename Length (NFL)</td><td>newFilename</td></tr>
<tr><td>$b_7b_6b_5$</td><td>$b_4b_3b_2b_1b_0$</td><td>OFL bytes</td><td>1 Byte</td><td>NFL Bytes</td></tr>
<tr><td colspan="2">Byte 1</td><td>From Byte 2 to Byte OFL+1</td><td>Byte OFL+2</td><td>From Byte OFL+3 to Byte OFL+NFL+3</td></tr>
</table>
Opcode: last 3 bits (of the first byte) specify the request operation code. For change the binary value of these bits is 010
oldFilename Length (OFL): first 5bits (of the first byte) specify the length of the old name of the file. The name of the file should not exceed 31 characters including the end of string character.
Old File Name: OFL bytes specify the name of the file to be changed.
newFilename Length (NFL): 1 byte specifies the length of the new name of the file. The name of the file should not exceed 31 characters including the end of string character.
newFilename: : NFL bytes specify the new name of the file.
</td></tr>
<tr><td>011</td><td>Summary
filename</td><td>
<table>
<tr><td>Opcode</td><td>Filename Length (FL)</td><td>FileName</td></tr>
<tr><td>$b_7b_6b_5$</td><td>$b_4b_3b_2b_1b_0$</td><td>FL bytes</td></tr>
<tr><td colspan="2">Byte 1</td><td>From Byte 2 to Byte FL+1</td></tr>
</table>
Opcode: last 3 bits (of the first byte) specify the request operation code. For get the binary value of these bits is 011
Filename Length (FL): first 5bits (of the first byte) specify the length of the name of the file. The name of the file should not exceed 31 characters including the end of string character.
File Name: FL bytes specify the name of the file that you should get the maximum, minimum, and average for its numbers.
</td></tr>
<tr><td>100</td><td>Help</td><td>
<table>
<tr><td>Opcode</td><td>unused</td></tr>
<tr><td>$b_7b_6b_5$</td><td>$b_4b_3b_2b_1b_0$</td></tr>
<tr><td colspan="2">Byte 1</td></tr>
</table>
Opcode: last 3 bits (of the first byte) specify the request operation code. For change the binary value of these bits is 100
</td></tr>
</table>

II.3 Format of the Response Messages

The response message that the server sends back to the client has the following form:

Response code (res-code)	**Data**
3 bits	*n* bytes in case of response for correct get and correct help request. 0 bytes otherwise.

The table below describes the format of the possible response messages:

<table>
<tr><th>Res-code</th><th>Mnemonic</th><th>Response Message Format</th></tr>
<tr><td>000</td><td>Response for correct put request and correct change request</td><td>
<table>
<tr><td>res-code</td><td>unused</td></tr>
<tr><td>$b_7b_6b_5$</td><td>$b_4b_3b_2b_1b_0$</td></tr>
<tr><td colspan="2">Byte 1</td></tr>
</table>
Res-code: last 3 bits (of the first byte) specify the response message code. To acknowledge correct put and change requests, the binary value of these bits is 000.
</td></tr>
</table>

<table>
<tr><th>Res-code</th><th>Mnemonic</th><th>Response Message Format</th></tr>
<tr><td>001</td><td>Response for correct get request</td><td>
<table>
<tr><td>Res-code</td><td>Filename Length (FL)</td><td>FileName</td><td>File Size (FS)</td></tr>
<tr><td>$b_7b_6b_5$</td><td>$b_4b_3b_2b_1b_0$</td><td>FL bytes</td><td>4 Bytes</td></tr>
<tr><td colspan="2">Byte 1</td><td>From Byte 2 to Byte FL+1</td><td>From Byte FL+2 to Byte FL+5</td></tr>
</table>
<table>
<tr><td>File Data</td></tr>
<tr><td>FS Bytes</td></tr>
</table>
Res-code: last 3 bits (of the first byte) specify the response message code. To acknowledge a get request when everything is correct, the binary value of these bits is 001
Filename Length (FL): first 5bits (of the first byte) specify the length of the name of the retrieved file. The name of the file should not exceed 31 characters including the end of string character.
File Name: FL bytes specify the name of the retrieved file.
File Size (FS): 4 bytes that specify the size of the file.
File Data: FS bytes containing the file itself. It can be sent as chunks of data.
</td></tr>
<tr><td>010</td><td>Statistical Summary</td><td>
<table>
<tr><td>Res-code</td><td>Filename Length (FL)</td><td>FileName</td><td>File Size (FS)</td></tr>
<tr><td>$b_7b_6b_5$</td><td>$b_4b_3b_2b_1b_0$</td><td>FL bytes</td><td>4 Bytes</td></tr>
<tr><td colspan="2">Byte 1</td><td>From Byte 2 to Byte FL+1</td><td>From Byte FL+2 to Byte FL+5</td></tr>
</table>
Res-code: last 3 bits (of the first byte) specify the response message code. To acknowledge a summary request when everything is correct, the binary value of these bits is 010
Filename Length (FL): first 5bits (of the first byte) specify the length of the name of the generated summary file. The name of the file should not exceed 31 characters including the end of string character.
File Name: FL bytes specify the name of the summary file.
File Size (FS): 4 bytes that specify the size of the file.
File Data: FS bytes containing the file itself. It can be sent as chunks of data
</td></tr>
<tr><td>011</td><td>Error-File Not Found</td><td>
<table>
<tr><td>res-code</td><td>unused</td></tr>
<tr><td>$b_7b_6b_5$</td><td>$b_4b_3b_2b_1b_0$</td></tr>
<tr><td colspan="2">Byte 1</td></tr>
</table>
Res-code: last 3 bits (of the first byte) specify the response message code. To acknowledge get request when the requested file is not found, the binary value of these bits is 011.
</td></tr>
<tr><td>100</td><td>Error-Unknown request</td><td>
<table>
<tr><td>res-code</td><td>unused</td></tr>
<tr><td>$b_7b_6b_5$</td><td>$b_4b_3b_2b_1b_0$</td></tr>
<tr><td colspan="2">Byte 1</td></tr>
</table>
Res-code: last 3 bits (of the first byte) specify the response message code. When the received request is not supported by the server, the binary value of these bits is 100.
</td></tr>
<tr><td>101</td><td>Response for unsuccessful change</td><td>
<table>
<tr><td>res-code</td><td>unused</td></tr>
<tr><td>$b_7b_6b_5$</td><td>$b_4b_3b_2b_1b_0$</td></tr>
<tr><td colspan="2">Byte 1</td></tr>
</table>
Res-code: last 3 bits (of the first byte) specify the response message code. When the received change request fails, the binary value of these bits is 110.
</td></tr>
</table>

110	Help-response	Res-code: $b_7b_6b_5$ (Byte 1)	length: $b_4b_3b_2b_1b_0$ (Byte 1)	Help Data: length bytes (From Byte 2 to Byte length+1)

Res-code: last 3 bits (of the first byte) specify the response message code. To acknowledge a *help* request, the binary value of these bits is 110.
Length : first 5bits (of the first byte) specify the length of the help data. It should not exceed 31 characters including the end of string character.
HelpData: *length* bytes contain a list of the supported commands.

III. Client/Server Description

1. The server program listens to the port specified as command line argument 1, accepts *request messages* from a client programs, decodes these messages, performs the required computation, and returns the result of the computation in a *response message.*
2. When starting the client program, the user provides the server **IP address** and **port number** as command line arguments 1 and 2
3. Both client and server programs should support another command line argument as a debug flag that turns on/off printing of the messages sent and received. 0 means Debug mode is OFF 1 means Debug mode is ON.
4. The client program establishes a connection with the server and then waits for the user to enter a command. The program checks the command and forms the appropriate request message and sends it to the server. The server performs the needed computation and responds back to the client. The client handles the response messages from the server.
5. Open the Wireshark while running the client-server program, and capture the packet that is sent from server to the client. To recognize the packets that are being transferred from the server program, check the source and destination addresses. The source and destination address will be 127.0.0.1 as you are using localhost and answer the following questions:

   For TCP:

   - What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection between the client and server? What is in the segment that identifies as a SYN segment?
   - What are the sequence numbers of the first two segments in the TCP connection? At what time was each segment sent? When was the ACK for each segment received? Given the difference between when each TCP segment was sent, and when its acknowledgment was received, what is the RTT value for each of the two segments? Build the round-trip time graph.
   - What is the length of each of the first six TCP segments?
   - What is the minimum amount of available buffer space advertised at the received for the entire trace? Does the lack of receiver buffer space ever throttle the sender?
   - Are there any retransmitted segments in the trace file? What did you check for (in the trace) in order to answer this question?
   - How much data does the receiver typically acknowledge in an ACK? Can you identify cases where the receiver is ACKing in every other received segment.
   - What is the throughput (bytes transferred per unit of time) for the TCP connection? Explain how you calculated this value
   - Use the *Time-Sequence-Graph (Stevens*) plotting tool to view the sequence number versus time plot of segments being sent from the client to the server.

   For UDP:

   - Select one UDP packet from your trace. From this packet, determine how many fields there are in the UDP header. Name these fields.
   - By consulting the displayed information in Wireshark's packet content field for this packet, determine the length (in bytes) of each of the UDP header fields
   - The value in the Length field is the length of what? Verify your claim with your captured UDP packet.

- What is the maximum number of bytes that can be included in a UDP payload?
- What is the protocol number for UDP? Give your answer in both hexadecimal and decimal notation.
- Examine a pair of UDP packets in which your host sends the first UDP packet and the second UDP packet is a reply to this first UDP packet. (Hint: for a second packet to be sent in response to a first packet, the sender of the first packet should be the destination of the second packet). Describe the relationship between the port numbers in the two packets.

6. The client accepts input lines that consist of one request. User commands are:
    - **put** *filename*
    - **get** *filename*
    - **summary** *filename*
    - **change** *oldfilename newfilename*
    - **help**
    - **bye**

```
myftp>Press 1 for TCP, Press 2 for UDP myftp>
Provide IP address and Port number
myftp> Session has been established. myftp>help
Commands are: bye change get help put summary
myftp>get file1.txt
file1.txt has been downloaded successfully.
myftp>put file2.txt
file2.txt has been uploaded successfully.
myftp>change file2.txt file3.txt
file2.txt has been changed into file3.txt.
myftp> get summary.txt
summary.txt has been downloaded successfully.
myftp>bye
Session is terminated.
$
```

Example (at the client side

IV. Submission

You shall submit a design document, the source files of your two programs:

- The design document should reflect your high-level implementation design of the client and server programs. For example, it can describe function prototypes and algorithms for each program and also document the experiment you have done with Wireshark and take screenshots and provide RTT and time sequence graph.
- For the code, you shall submit a single archive file (a zipped file) named yourID-coen-366-prj.zip via Moodle. It must contain the files that make up your programming assignment.

  You should also include a plain text file, README, that describes how to run and test your programs.

  The archive should also include a tests directory that contains at least two test files.

Make sure that all of your files start with a block of comments that gives your names, student-ids, user-ids, and the purpose of the file. Include in this block comment a statement asserting that you are the sole author of the file.

Any detected copying will not be tolerated and necessary measures will be taken.