Computer Networks Lab 3

Assignment Requirements:

- In this lab I had to build a TFTP server that had to support Read and Write requests (RRQ&WRQ).
- The files had to be read and written to a certain directory.
- Initially the server should listen to port 69 for any requests then decide on another port to continue the communication.
- Then depending on the opcodes the server should be able to reply to the clients requests.
- The server should also support following errors:

Value	Meaning
0	Not defined, see error message (if any).
1	File not found.
2	Access violation.
3	Disk full or allocation exceeded.
4	Illegal TFTP operation.
5	Unknown transfer ID.
6	File already exists.

My Implementation

- First of all the server listens to port 69 in the beginning.
- When a package comes a new thread is created with a new socket.
- This socket chooses a random port in my computer to communicate with the other client.
- First of all I check if the package is a RRQ or a WRQ if it is anything else except those 2 I send an Illegal TFTP operation error.
- Then depending on the request types I do different operations.
- If it is a RRQ I send data and receive acknowledgment.
- If it is a WRQ I receive data and send out acknowledgements.

Send Data Receive Acknowledgements

- I first check if the file requested by the client exists on the drive I have allocated.
- If it doesn’t I send a File Not Found error package.
- Else I start to read the file at max 512 bytes a time.
- If I have read less then 512 bytes that means that, that is the last package I am going to send the client and end the connection but if I have read 512 bytes then that means there will be more packages to come.
- In case I read 512 bytes of data, I put that data into a DATA package.
- I send that package to the client and wait for an acknowledgment.
- If the client doesn’t send an ACK package in 600 milliseconds I send the package again, I do that for 7 times and on the 7th time if I still don’t get an ACK package for that DATA package I end the connection.
- When I receive a package from that socket I first make sure that the package is from the initial client. If it isn’t I send an Unknown Transfer ID error to the sender of the package.
- If the sender is correct I check that the package is in fact an ACK package.
- If it is an ACK package I check that the ACK is for the wright DATA package I have sent by comparing the ACK block number they have sent the block number which I am keeping a track of.
- If that also is true I send the next block of data.

Receive Data Send Acknowledgements

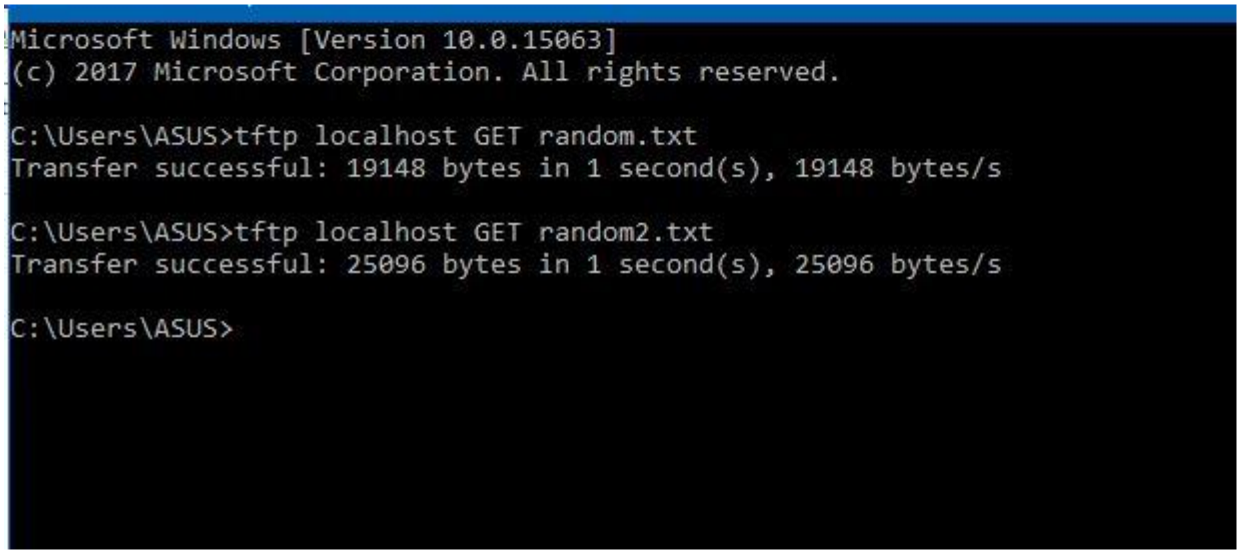
- I first check if the file requested by the client to be written already exists on the drive I have allocated.
- If it does I send a File Already Exists error package and end the connection.
- If it doesn’t I send an ACK package with block number 0 to let the client know that they can send the data.
- I send that package to the client and wait for an acknowledgment.
- If the client doesn’t send an ACK package in 600 milliseconds I send the package again, I do that for 7 times and on the 7th time if I still don’t get an ACK package for that DATA package I end the connection.

- When I receive a package from that socket I first make sure that the package is from the initial client. If it isn't I send an Unknown Transfer ID error to the sender of the package.
- If the sender is correct I check that the package is in fact a DATA package.
- If it is a DATA package I see if it is the DATA package I was expecting by looking at the block number.
- If it also is the right DATA package I check if I have enough space to store that data in my drive, If I don't I send a Disk Full error to the client and end the connection.
- Else I send an ACK package for that DATA package and continue listening for the next DATA package.
- This goes on until I get a DATA package which has less than 512 bytes of data, if it does I end the connection.

Details

- 2 sockets are used during this project.
- The first socket uses port 69 for any requests from the clients.
- The second request is used to communicate with the client which has sent the initial request to the first socket.
- The second socket uses a random port number and this socket is closed after the communication has ended.

Handling Large Files



```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\ASUS>tftp localhost GET random.txt
Transfer successful: 19148 bytes in 1 second(s), 19148 bytes/s

C:\Users\ASUS>tftp localhost GET random2.txt
Transfer successful: 25096 bytes in 1 second(s), 25096 bytes/s

C:\Users\ASUS>
```

I have read the data from the files using FileInputStream into a byte array of size at max 512 bytes. During this time I also have kept count of the amount of bytes I read at each iteration, where the last one was going to be less the 512 bytes. I always copied the contents of the initial array to secondary array with the size of the amount of data read by the input stream so the final package would have be smaller than packages beforehand. If the package sent was smaller than 512 I would end the retransmission.

Time Outs

I set SOTime out to the socket used for communication for 600 milliseconds. Then I created a while loop. In the while loop I sent the package first then I created a try – catch. In the try section I listened the port for a reply and of the reply didn't come within 600 milli second it would throw an error which would activate my catch which would increase the amount of tries then get back to the while loop which means me sending the package again and listening to the port for a reply, in the catch statement I set an if statement which broke the loop if I had sent the package 7 times already.