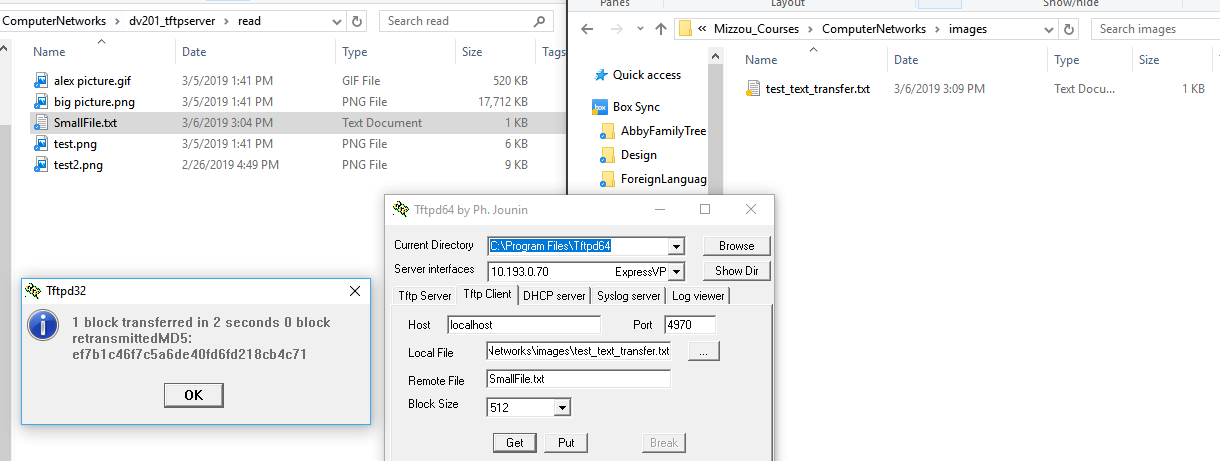
DV201 (Software Engineering) Assignment 3 (TFTP Server)

By: Alex and Fabian

# Problem 1 (TFTP Server):

**Read Request from Client with file less than 512 Bytes (Screenshot):**

In the following test the windows explorer window shows the contents of the “read” directory for the tftp server. This directory contains all the available files that the client user can get. The explorer window on the right shows the local directory for the client user where the resultant image was stored. The TFTP client at the bottom shows that we will connect to localhost on port 4970 and in this case get the file SmallFile.txt. Since the file is less that 512 bytes it only requires 1 block to transfer. When the “Get” button is pressed the transfer begins and the popup message indicates that the 1 block has been successfully transferred and the file on the local filesystem is shown on the left (test\_text\_transfer.txt)



**Socket and sendSocket:**

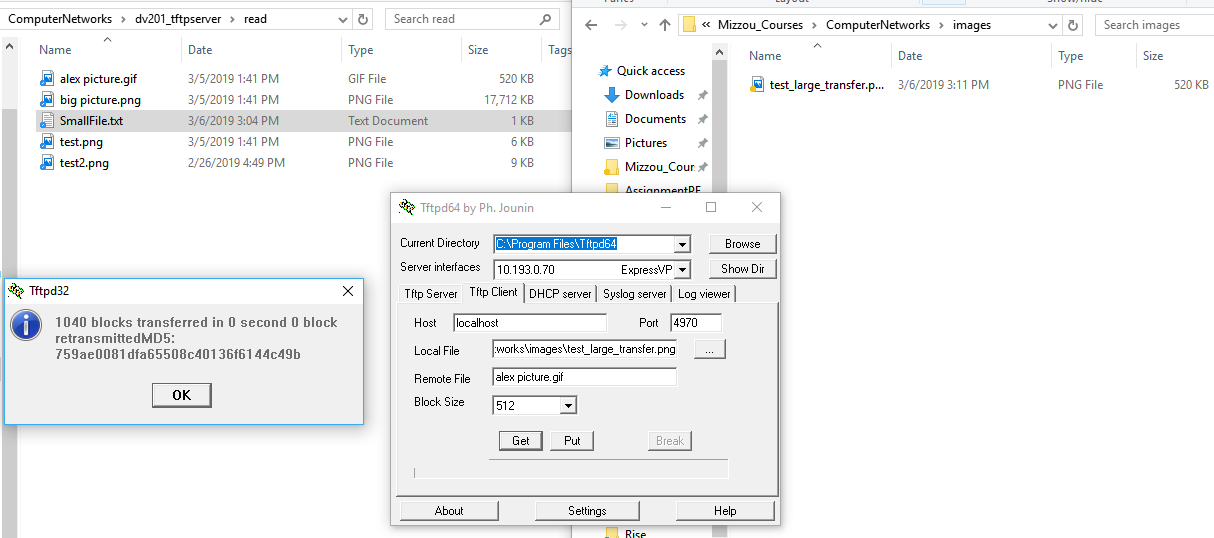
The reason that the server uses socket and sendSocket is that socket is used to start the server on a known port so incoming connections can be accepted. sendSocket is then used when a connection is accepted to open a new port so that the file/data can be transferred from the server to the client. This is done so that the known port the server runs on can be open to accept new incoming requests while the server is handling/transmitting files to clients on other ports. sendSocket picks any available port while socket uses a well-known port so that users know which port to access the server on.

# Problem 2:

**Multiple Read Requests from Client with files greater than 512 Bytes (Screenshot):**

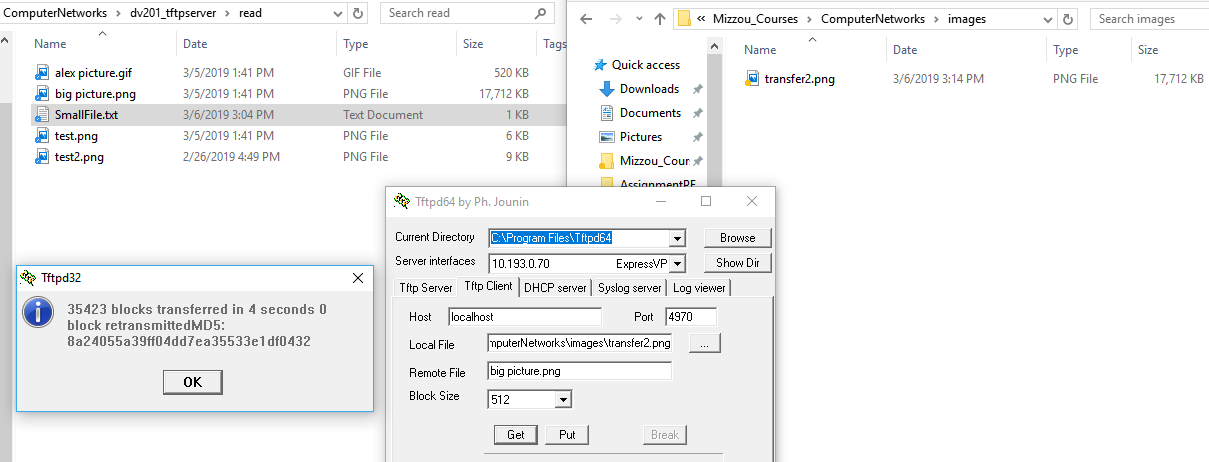
**Large Transfer 1:**

This test was performed in the same way as the test for problem 1 but this time with larger files. As you can see in the confirmation screenshot, for the file “alex picture.gif”, which is 520 KB in size, there were 1040 blocks transferred. The completed transfer to the local system is shown in the right explorer window. You can see that the size of the file on the server and on the local filesystem is the same.



**Large Transfer 2:**

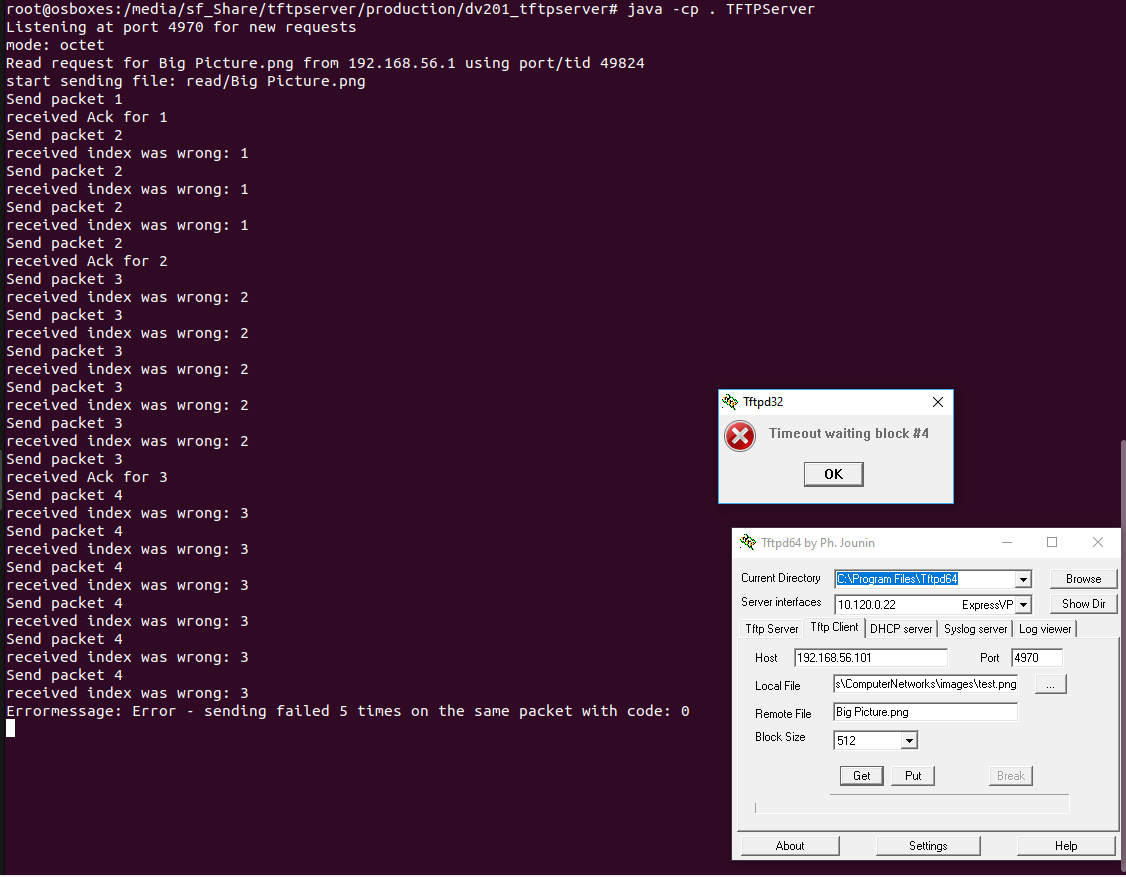
As you can see in the confirmation screenshot, for the file “big picture.png”, which is 17,712 KB in size, there were 35423 blocks transferred. This transfer took 4 seconds because of how large the image was but the server was still capable of handling the transfer. The completed transfer to the local system is shown in the right explorer window. You can see that the size of the file on the server and on the local filesystem is the same.



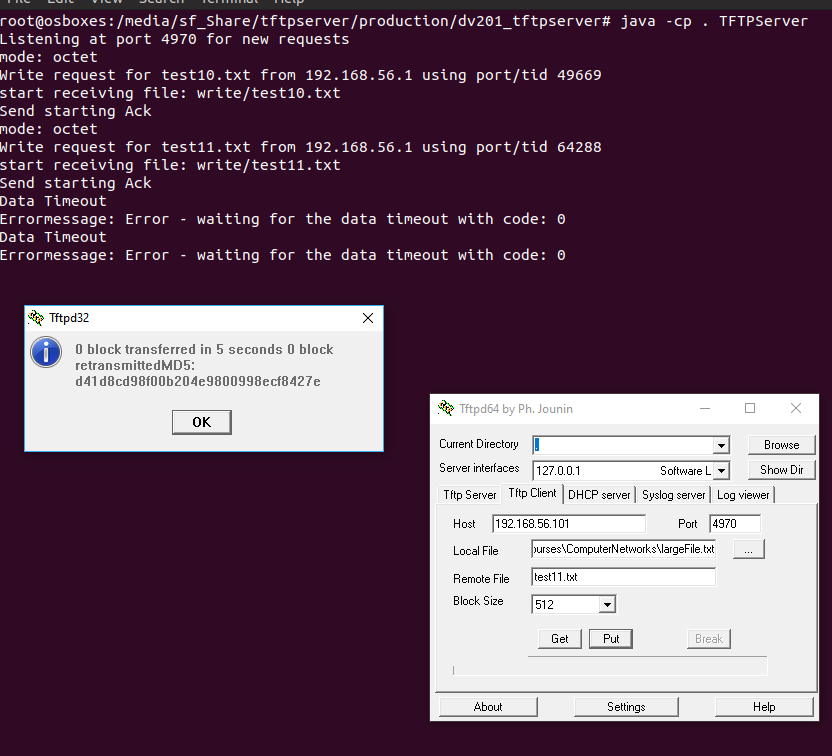
Timeouts and Retransmissions:

### Regulated Delay (Endless Retransmission):

To test the timeouts and retransmissions we added an artificial delay to all requests coming into the server using the following UNIX command ( tc qdisc add dev enp0s8 root netem delay 10000ms). This command adds a 10 second delay to all inbound communication on the selected network adapter. Since there is such a long delay to receive the first acknowledgment the server does not send the next data packet and the client attempts to retransmit the acknowledgment. Thetftp program is set to retransmit data and acknowledgments to 6 times if it doesn’t receive the next package. Because of the specification in the assignment (“If an acknowledgment of the wrong packet arrives (check the block number), you should also retransmit.”) the server will retransmit the next data package several times. This means that it will not receive the ACK (acknowledgement) that the request was successful from the client until at least 10 seconds after the packet was sent. The server will attempt to retransmit a packet 5 times before it will not longer transmit anything more and the client application has a 3 second timeout in its settings. The screenshot below shows the terminal of the VM where the server is running. There is also the local TFTP client in the bottom right corner. The server tries to retransmit a packet if it does not receive the correct ACK. Since there is a delay in the network the ACK packets are all very late which causes many retransmissions until eventually on block 4 the timeout is too long and the server stops trying to retransmit and sends a timeout error.

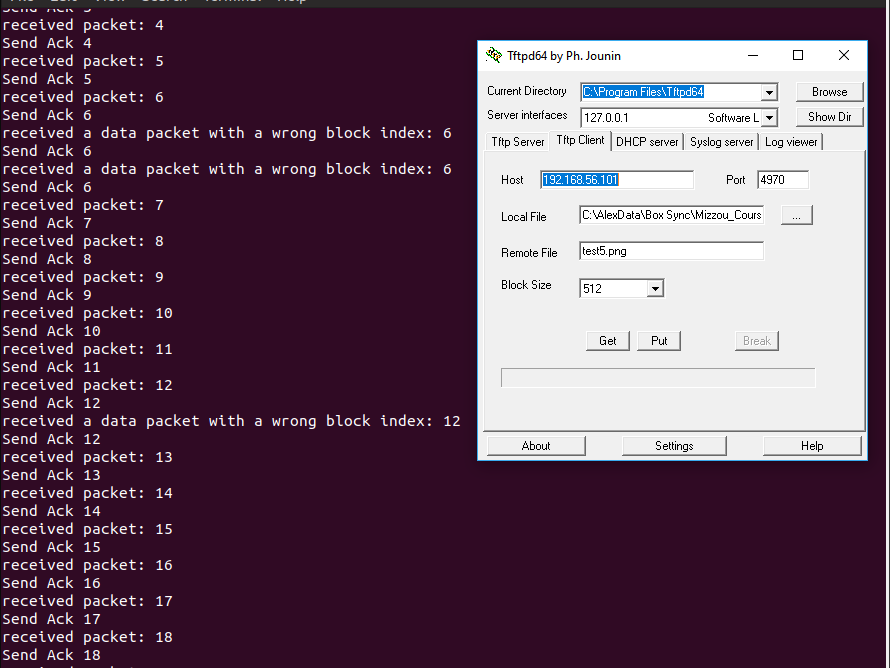


### Receiving Data Timeout:



### Sporadic Packet Loss:

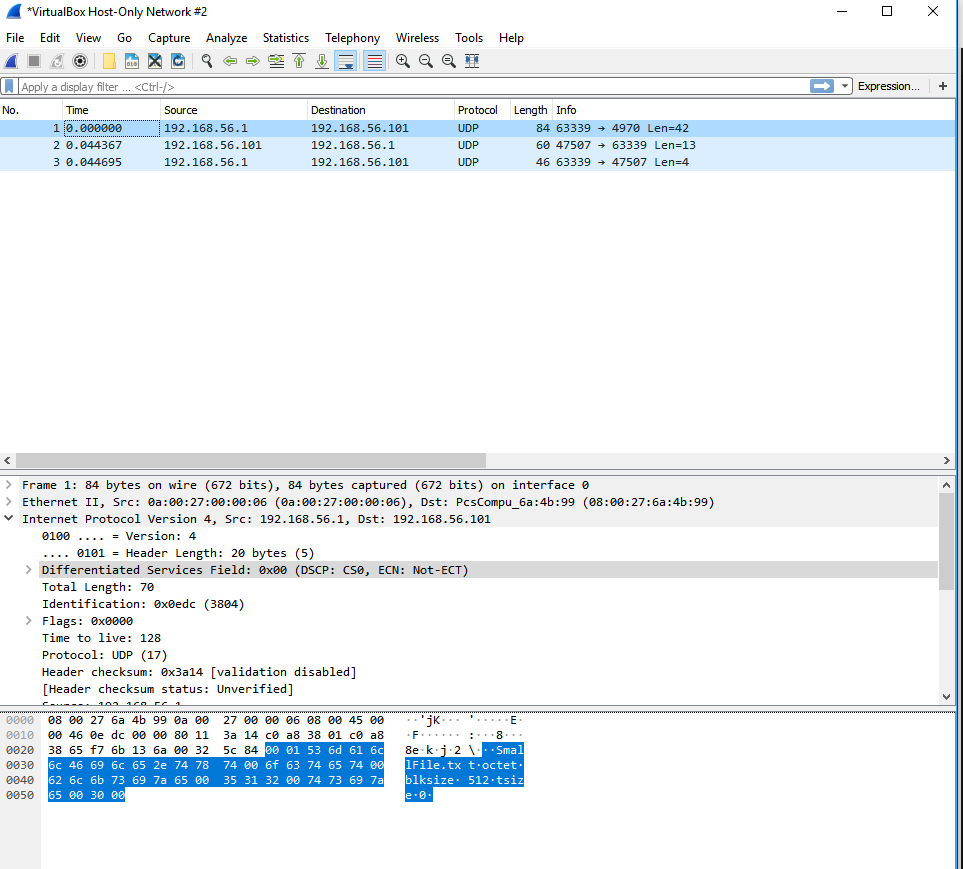
To test the timeouts and retransmissions we added sporadic packet loss to incoming requests into the server using the following UNIX command ( tc qdisc add dev enp0s8 root netem loss 10%). This command drops 10% of packets on the selected network adapter. This means that somewhere in the transfer packets will be lost randomly. The screenshot below shows the terminal of the VM where the server is running. The client is waiting for an ACK block from the server (where it says “received a data packet with a wrong block index”) but the ACK block is lost because of the packet loss. The server will resend the ACK message again and then the client will continue transmitting data. This shows our server is able to handle sporadic packet loss.



## VG-Task 1:

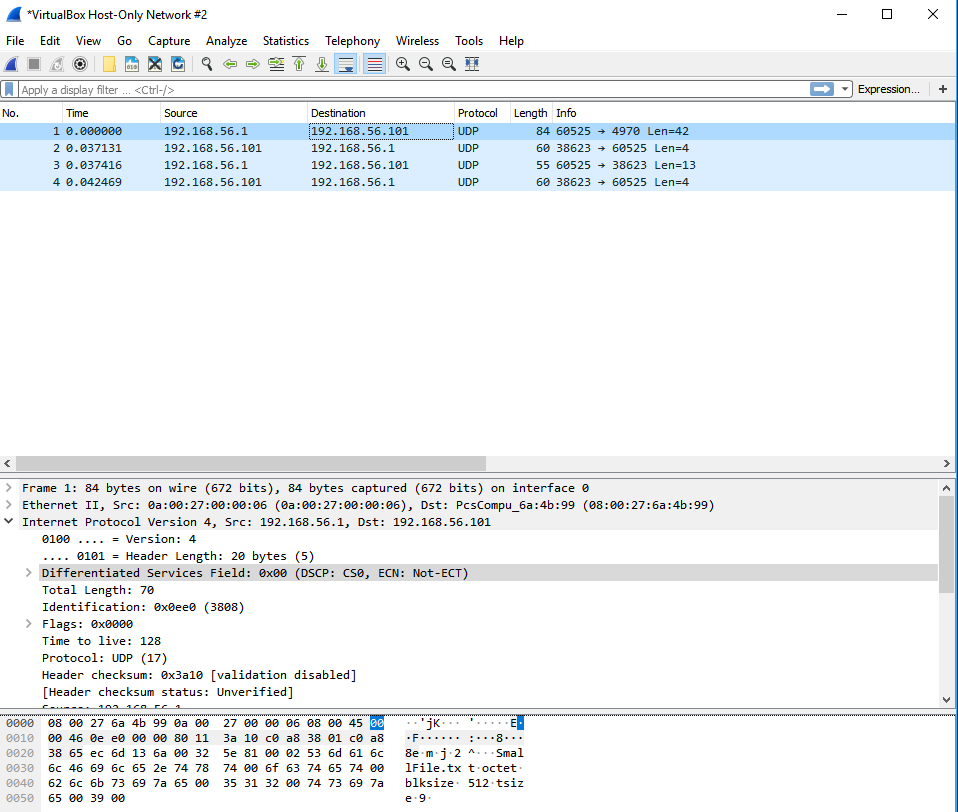
### Read Request:

For this test a read request was preformed to the server running on a virtual machine with the ip address 192.168.56.101 The client (my computer) is issuing a read request to the server in line 1 of the Wireshark screenshot. This request includes the file name, a 0-Byte, and stream mode (octet in this case). After this we can see additional information that is not part of the RFC. This is using the UDP protocol so there is no handshake like TCP. Shown with the red circle below, the read request uses OP Code 1 to tell the TFTP server that the request is a read request. Then in line 2 the server sends the binary contents of the file to the client. In line 3 the client sends an Acknowledgment (ACK) message (OP Code 4) that they have received the data successfully.



### Write Request:

For this test a write request was performed to the server running on a virtual machine with the ip address 192.168.56.101 The client (my computer) is issuing a write request to the server in line 1 of the Wireshark screenshot. This request includes the file name, and stream mode (octet in this case) and block size (not part of RFC). This is using the UDP protocol so there is no handshake like TCP. Shown with the red circle below, the write request uses OP Code 2 to tell the TFTP server that the request is a write request. In line 2, the server sends an Acknowledgment (ACK) message to the client that it has received the request. Then in line 3, the client transmits the binary data of the file to the server. In line 4, the server sends an ACK message to the client that it has received and written the data to the specified file.



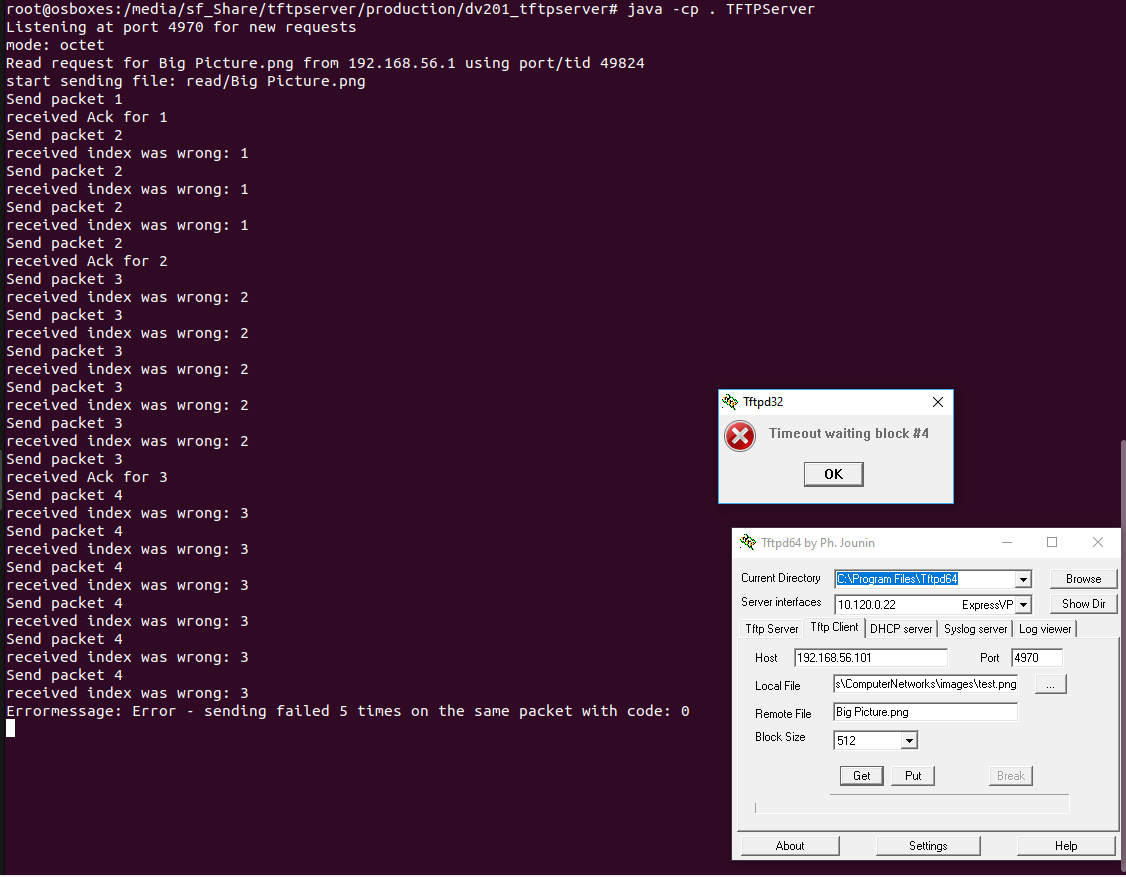
# Problem 3 (Error Handling):

In the following screenshots the terminal window is the server running on a virtual machine

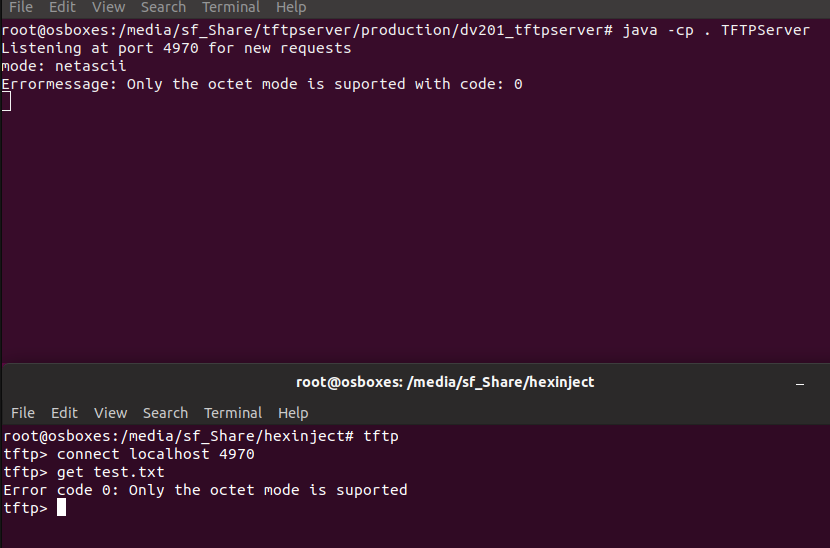
## Code 0:

According to the RFC 1350 spec the Code 0 Error code indicates an undefined error and you should check the error message if there is any. Our server can send a Code 0 for a few conditions. In the first screenshot below the server attempted to retransmit the same packet 5 times without receiving an acknowledgement and after not receiving an ACK message the 5th time it sent the Code 0 error message. In the 2nd screenshot the client program (bottom terminal) tries to get a file using a mode other than the octet mode and the server (top terminal) returns a code 0 error. Our program will also send a code 0 on the following conditions:

* Sending or Receiving a file larger than 32 MB
* Unknown Errors/Failures
* Timeout waiting Errors
* Errors receiving incoming data

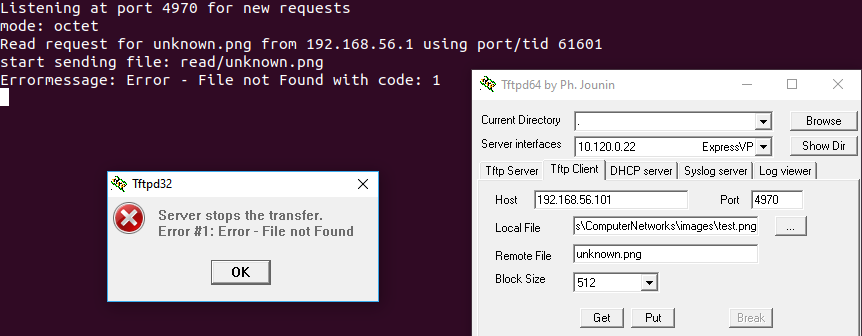
1.

2.



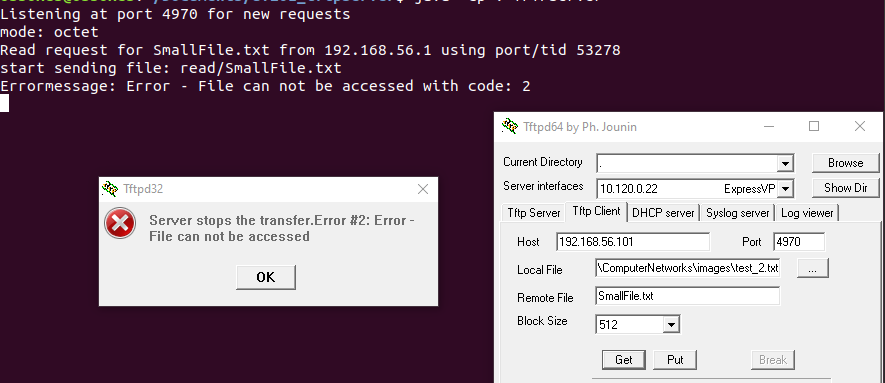
## Code 1:

According to the RFC 1350 spec the Code 1 Error code indicates that the File is not Found. Our server will send this error code if the client user requests a file that does not exist on the server. For this test the client user requested “unknown.png”. The server could not find the file and send the Code 1: File not found error to the client.



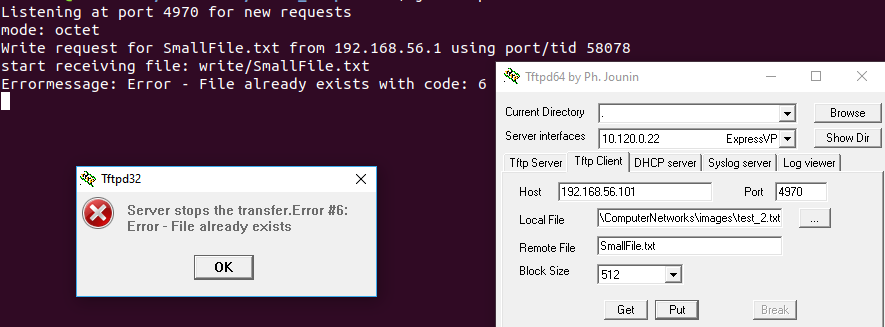
## Code 2:

According to the RFC 1350 spec the Code 1 Error code indicates that there is an access violation. For this test I changed the permissions of the file “SmallFile.txt” using “chmod a-r” which makes it so that no one user can read the file. This means when the server attempts to access the file it will cause an access violation and send a Code 2 error as shown by the error message window.



## Code 6:

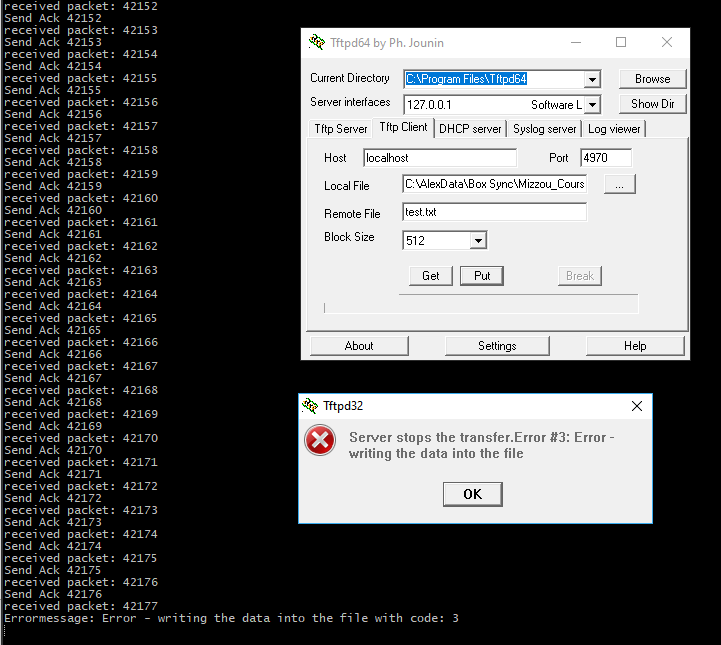
According to the RFC 1350 spec the Code 1 Error code indicates that the File Already Exists. Our server will send this error code if the file that the client user is trying to “Put” on the server already exists. In this case the file “SmallFile.txt” already exists on the server so the server will not allow it to be overwritten and the server sends a code 6 error message.



## VG-task 2:

### Code 3:

According to the RFC 1350 spec the Code 3 Error code indicates that the disk is full or the allocation is exceeded. Our server will send this code if there is a problem writing the file to the disk (in this case there is no more space on the disk). This was tested by generating a dummy file that nearly filled a USB drive. Then the transfer was attempted, and the server returned the code 3 error when there was no more space left on the disk.



### Code 4:

According to the RFC 1350 spec the Code 4 Error code indicates an illegal TFTP operation. Our server will return this error code if the client uses an invalid OP code (not read or write) to start a transmission. It will also return this error code if it receives a packet that should be a data packet and the OP code is not for a data packet.

### Code 5:

According to the RFC 1350 spec the Code 5 Error code indicates an unknown transfer id. On our server after we send data we check to see if the port of the received Acknowledgment packet for the sent data matches the transfer ID of the sent packet. If it does not we return a code 5 error to the client.