Report

Assignment 3
- TFTP Server

10 Mar, 2017

Semester: Spring 2017
Course: Computer Networks
Authors: Sarpreet Singh Buttar
Emil Mattsson

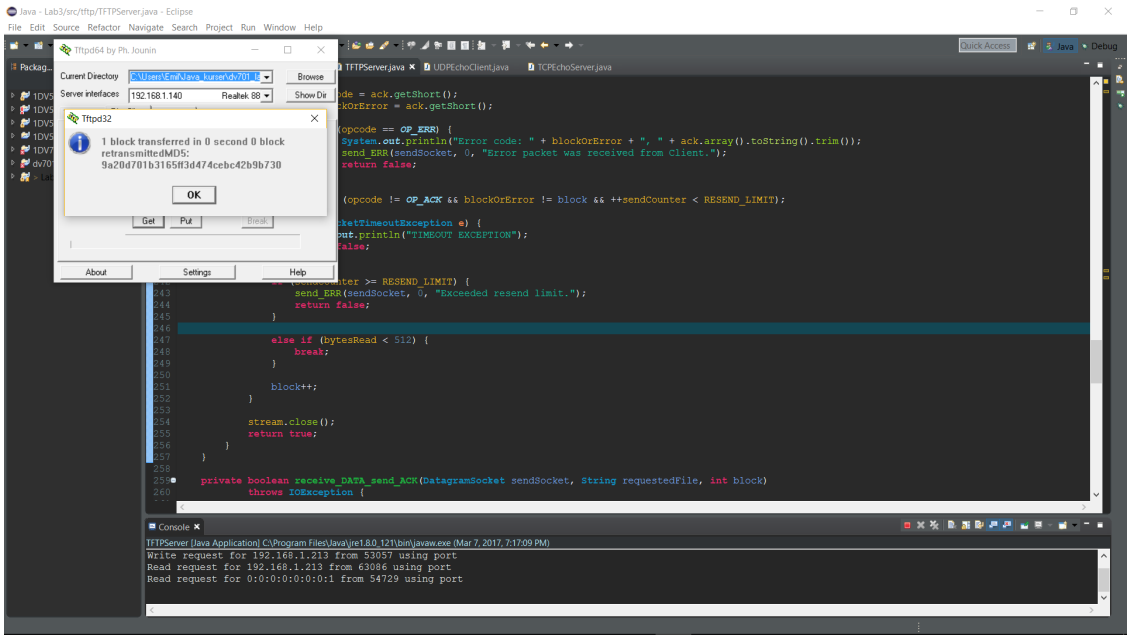
Summary of work done

Work by both

We have done all the work together. We give 50% to each other.

Problem 1

Read request less than 512 bytes



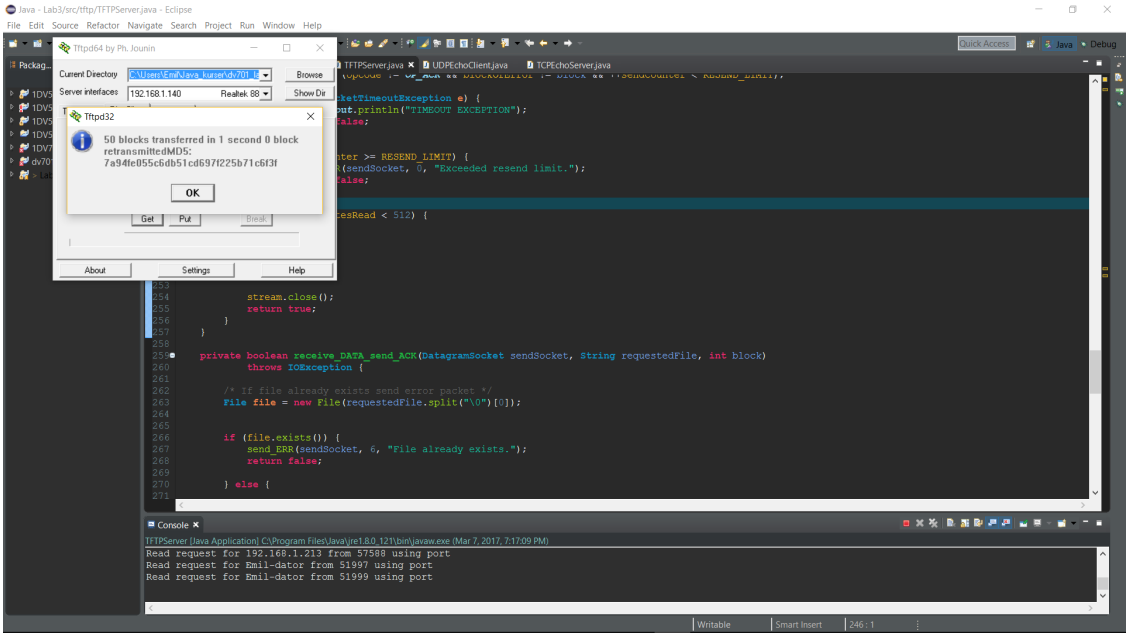
The picture above shows the result of a read request from TFTP Client to the TFTP Server. The requested file was less than 512 bytes.

Socket and sendSocket

The reason for having two sockets is that the first socket is to establish a connection and the other socket is for sending the packets between Client and Server.

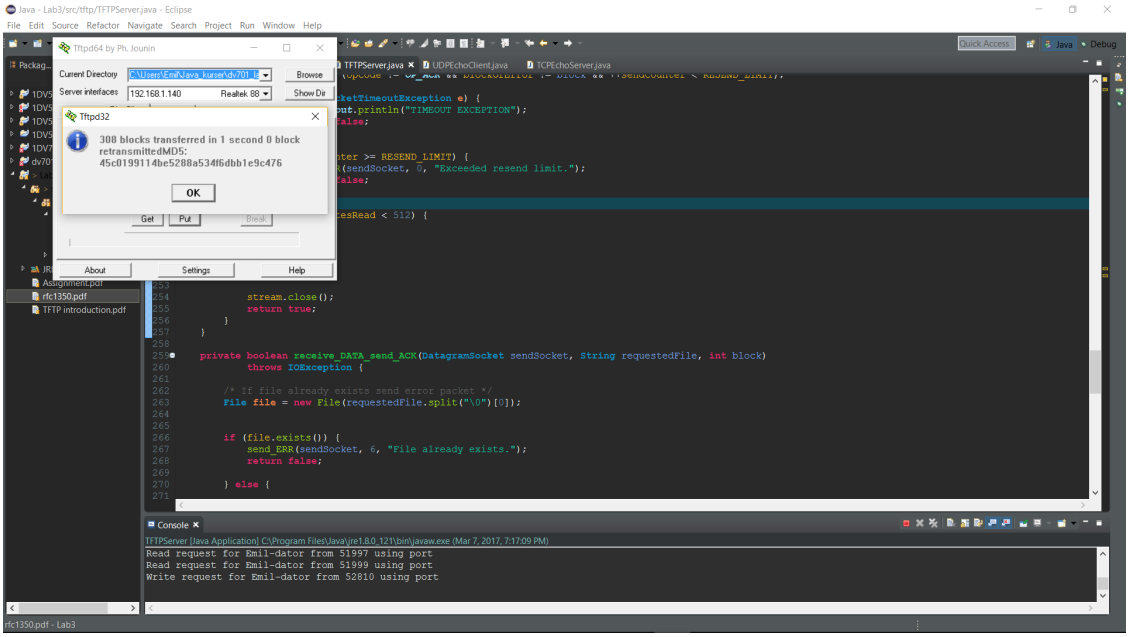
Problem 2

Read request more than 512 bytes



The picture above shows a read request from TFTP Client to the TFTP Server. The requested file was larger than 512 bytes, which can be seen in the TFTP Client because there are more than 1 block of data.

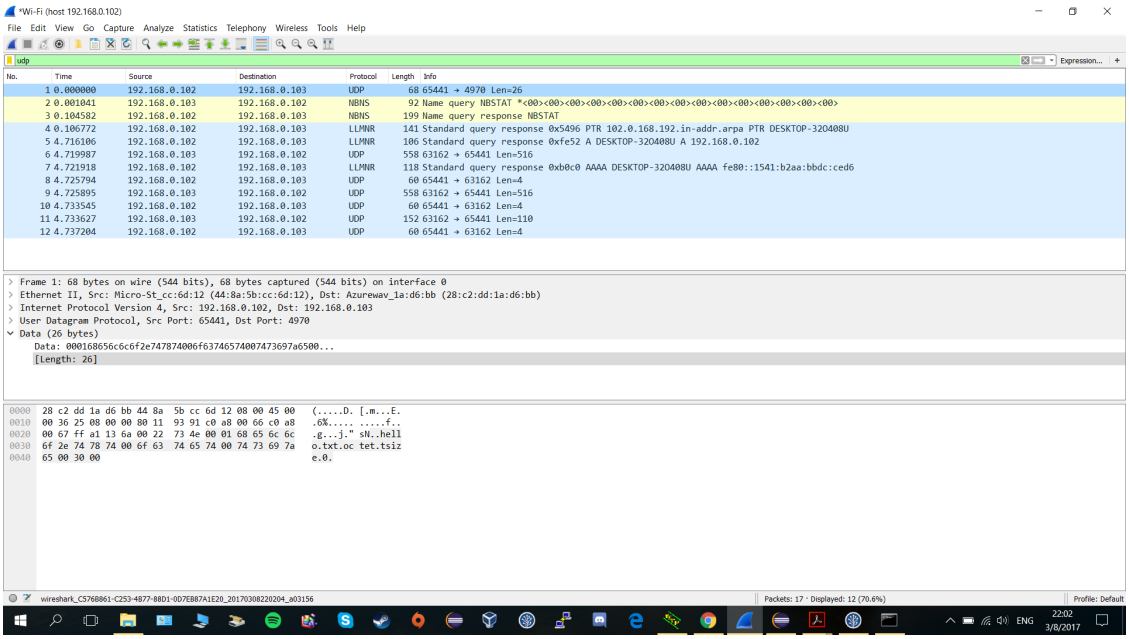
Write request



The screenshot above shows a successful Write request from the TFTP Client to the TFTP Server. Since the image sent was quite large the transfer required 308 blocks of data.

VG Task 1

Wireshark analysis



The screenshot above is showing a Read request from my desktop to our TFTP Server on my laptop. The requested file was hello.txt file and a bit over 1000 bytes large. Since the TFTP protocol is built on top of UDP all the TFTP packets are seen as UDP packets.

Analysis line by line

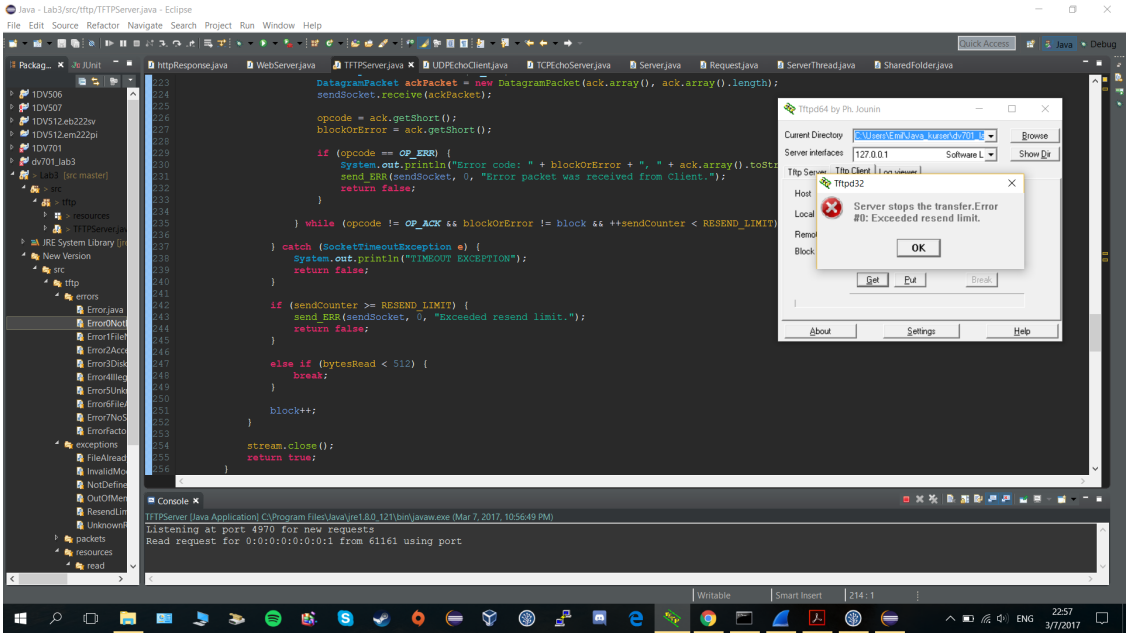
1. First line is the Read request from my desktop to the TFTP Server on my laptop and that it is 26 bytes large. In the data one can see the requested: file "hello.txt" and the mode: octet.
2. The next two lines is the NBNS protocol which is a protocol for translating human-readable names to IP adresses like DNS.
3. The next two lines is LLMNR protocol which is built on top of DNS packet format which allows hosts on the same network to perform name resolution.
4. Here is where the first DATA packet is sent from the TFTP Server to my TFTP Client on my desktop. Although the UDP packet length is 558 the DATA packet is 516 as can be seen in the info. Of these 516 bytes only 512 is the actual data of the requested file, the rest is 2 bytes for opcode and 2 for block number.
5. Another LLMNR protocol.
6. Next is an ACK packet from the desktop to the server for the first DATA packet. Here we can see that the ACK packet has the length 4, 2 for opcode and 2 for the block number.
7. Next is the next data packet of the requested file and since its not the last DATA packet its larger than 515 bytes.
8. ACK packet for the last DATA packet.
9. This is the last DATA packet because it's less than 516 bytes. If the DATA packet is received the TFTP Client will know it's the end of the data of the requested file. If the server don't get an ACK packet after a while it will retransmit the DATA packet a number of times.
10. Here is the last ACK packet for the received DATA packet and now the Client will terminate the connection.

Difference between Read and Write request

Read Request	**Write Request**
get is the keyword for making this request	*put* is the keyword for making this request
Its opcode is 1	Its opcode is 2
After accepting it, server creates a DATA packet of requested file and sends it to client	After accepting it, server send ACK packet with same block number to client
After sending the initial packet, server waits for ACK packet from client	After sending the initial packet, server waits for DATA packet from client
Once the packet is received, it only send a new packet (block number will increase by 1) if last sent packet was 512 bytes.	Once the packet is received, server sends ACK packet to client with same block number as received DATA packet have. If last packet received packet was 512 bytes, process repeats again.

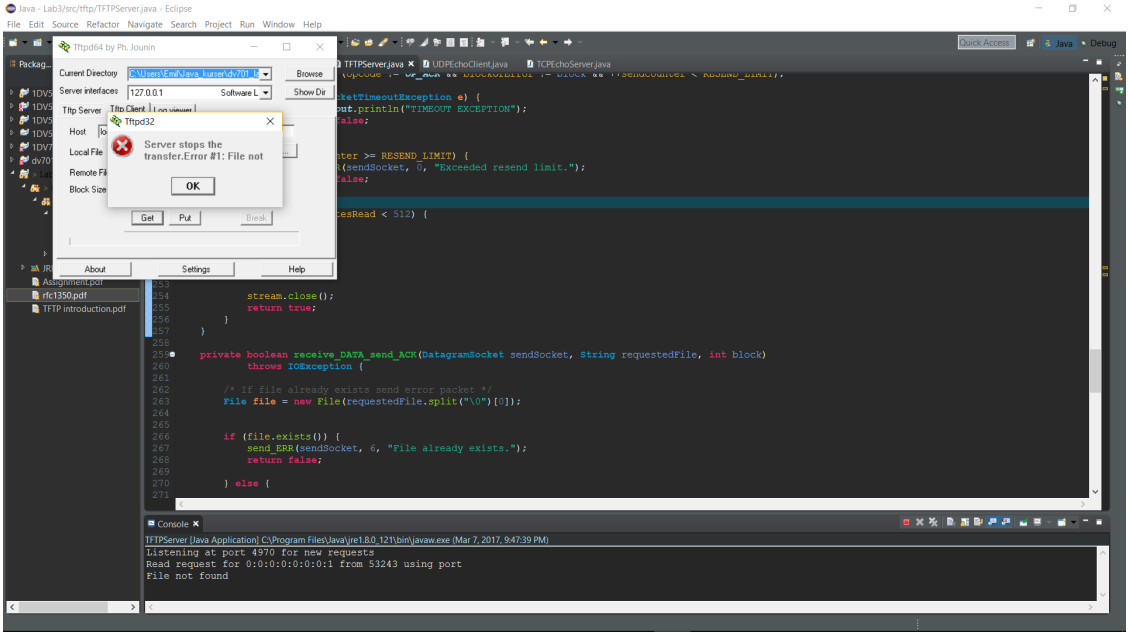
Problem 3

Error code 0 (Not defined, see error message (if any))



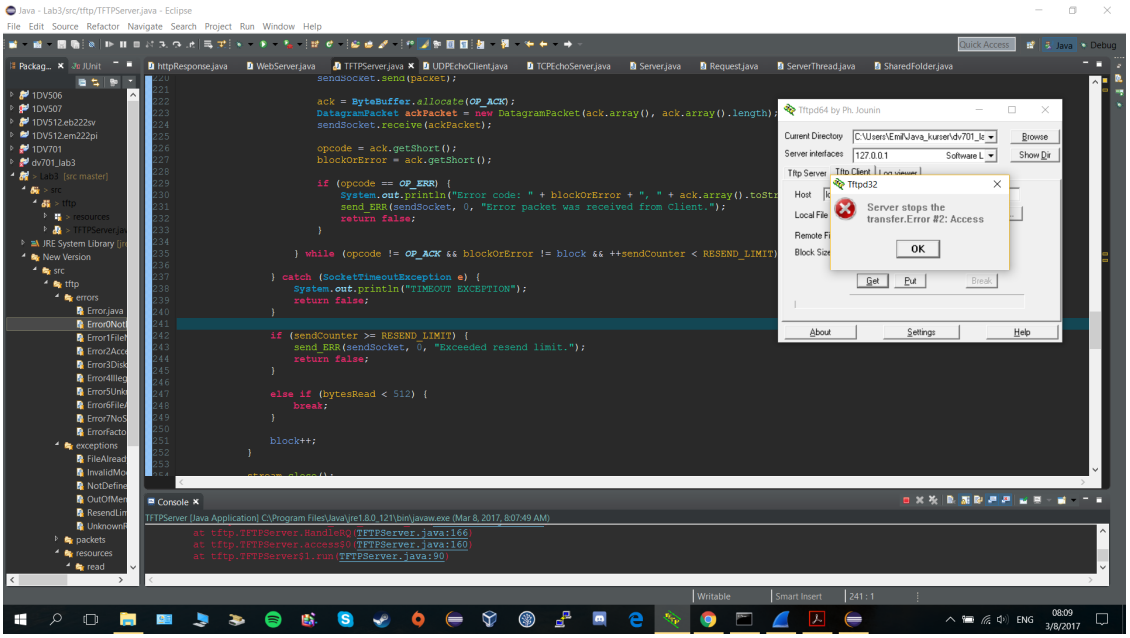
The screenshot above shows an error message from the Server to the Client, the error code received is error #0 "Not defined, see error message". In our case we use it when the Server has retransmitted a DATA packet a number of times.

Error code 1 (File not found)



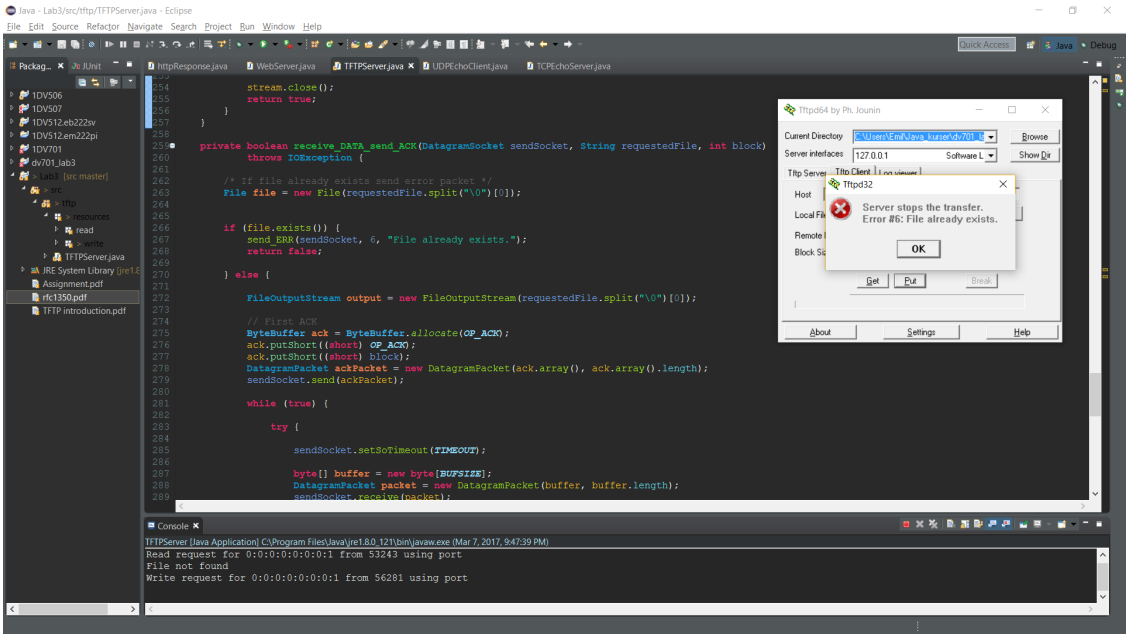
The screenshot above shows when the TFTP Client request a file that doesn't exist. The Client receives an error packet with error code #1: "File not found". Note that for some reason the entire "File not found" sentence can't be seen.

Error code 2 (Access violation)



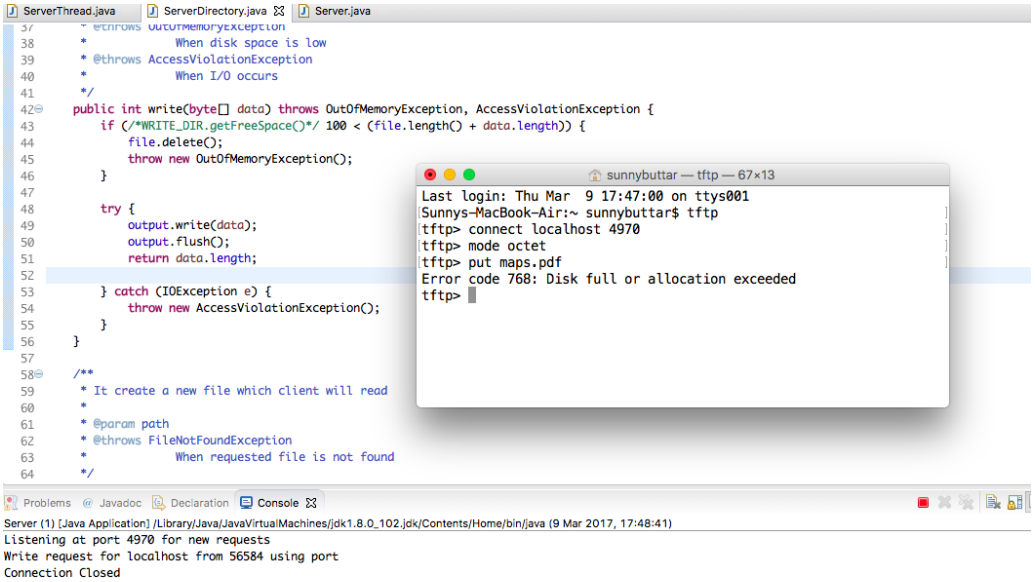
The screenshot above shows that the TFTP Client has received error code #2 "Access violation" from the TFTP Server. We implemented this on my windows laptop, so error code #2 is triggered by a generic IOException.

Error code 6 (File already exists)



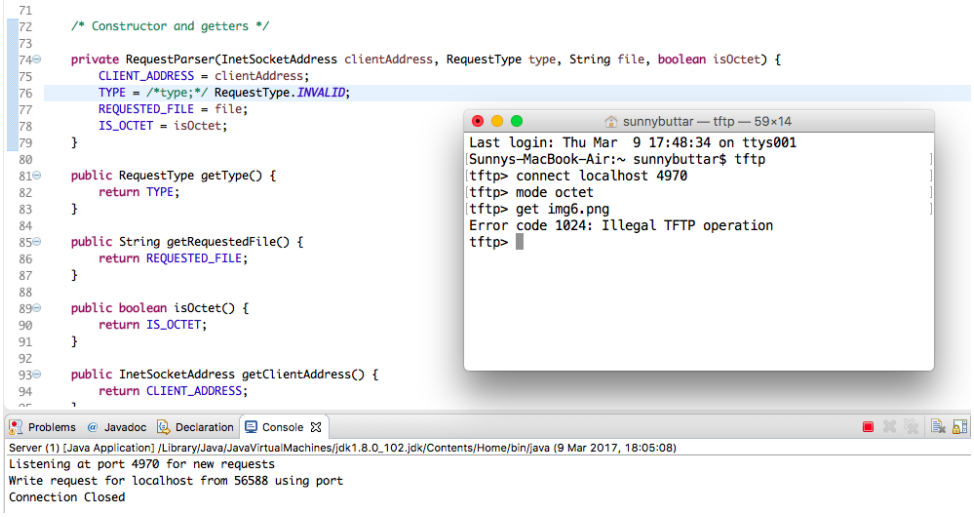
The screenshot above shows that the client tried to write a file that already exists on the server and receiving an error packet with error code #6 "File already exists".

Error code 3 (Disk full or allocation exceeded)



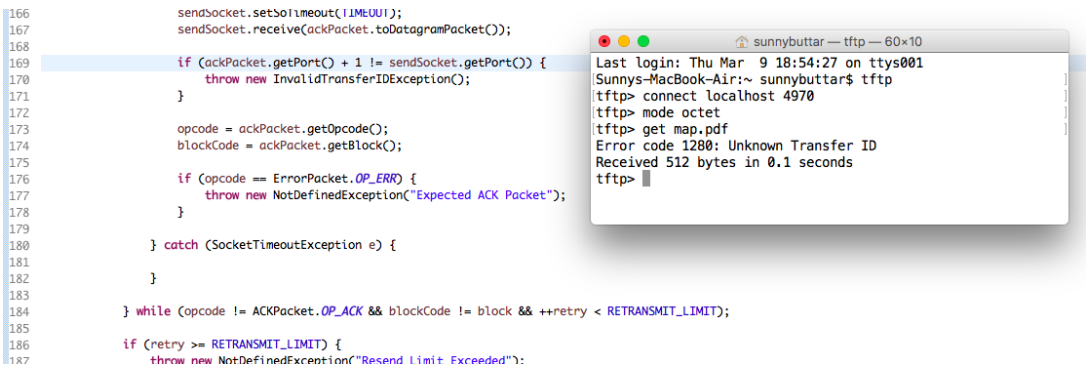
The screenshot above shows that the client tried to write a file but there is not enough space on the disk and server replied with error code 3. For creating this error, we have manually set the disk size which is visible in the image. *Note:* TFTP on mac does not shows the correct error code as windows do.

Error code 4 (Illegal TFTP operation)



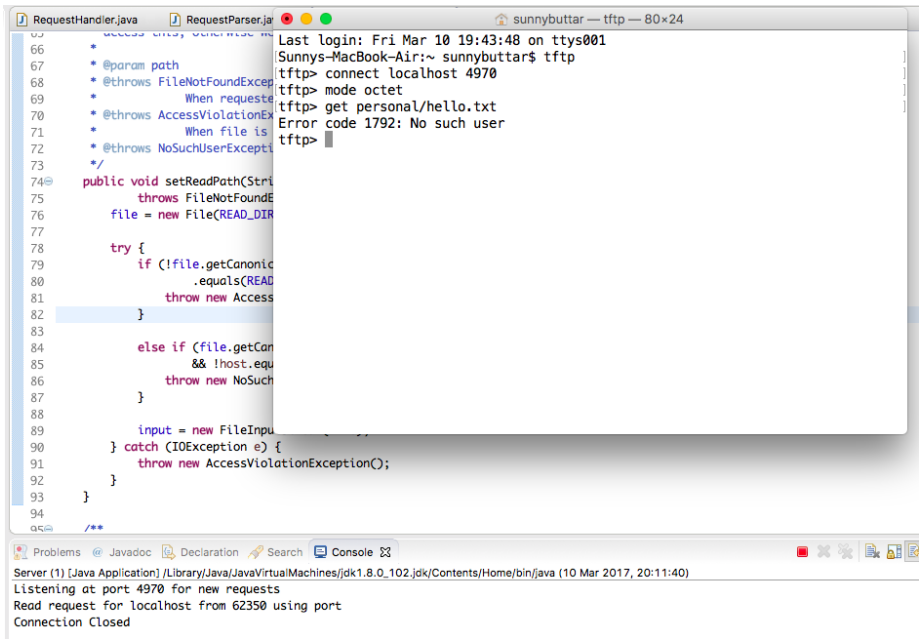
The screenshot above shows the error code 4. We have found that on mac it is not possible to send a request other than *get* or *put*, so again we have manually changed the request type in order to produce this response.

Error code 5 (Unknown transfer ID)



The screenshot above shows that received packet is not form same client as expected and therefore server respond with error code 5. We have manually changed the port number in order to produce this error.

Error code 7 (No such user)



The screenshot above shows that the client tried to access *personal* folder and server respond with error code 7. We show this error if client IP address is not same as we expected for accessing the folder.

NOTE: Error code 2 and 7 follow the windows OS, which means we are catching the generic IOExeptions for producing these errors.