



Linnéuniversitetet
Kalmar Västjör

Report

Assignment 3
IDV701



Linnéuniversitetet

Kalmar Våxjö



Authors:

Eldaras Zutautas

Tim Persson

Semester: Spring 2024

Emails:

ez222eq@student.lnu.se

tp222nu@student.lnu.se

Contents

1 Problem 1	1
2 VG Problem 2	3
3 VG Problem 3	4
4 VG Problem 4	6

1 Problem 1

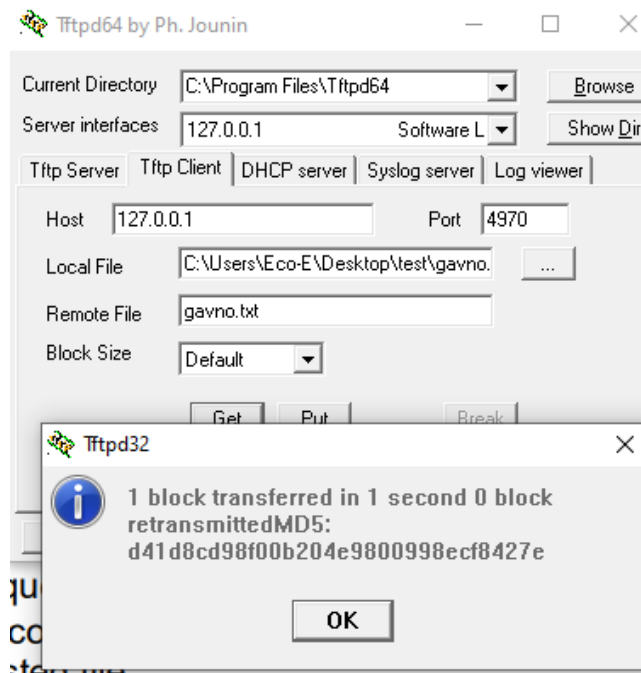


Figure 1. Sending GET request through Tftpd64 software.

```
segment3_b948fc2e\bin' 'src.TFTPServer'  
Listening at port 4970 for new requests  
Read request from 127.0.0.1 using port 56093
```

Figure 2. Read request (Console output).

3. As we can see in Figure 1 and 2, the file transfer to local file was successful, in Figure 1 we can see ACK.

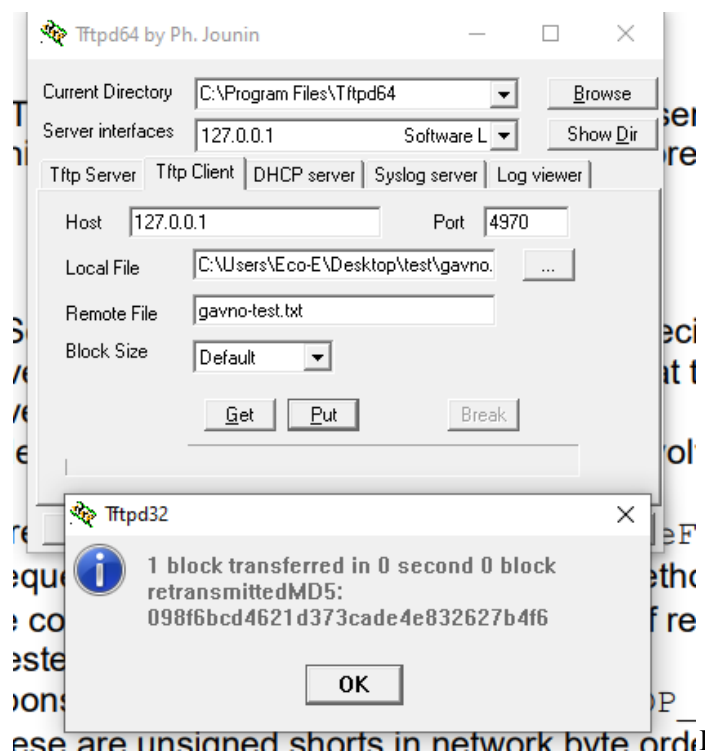


Figure 3. Requesting a PUT request.

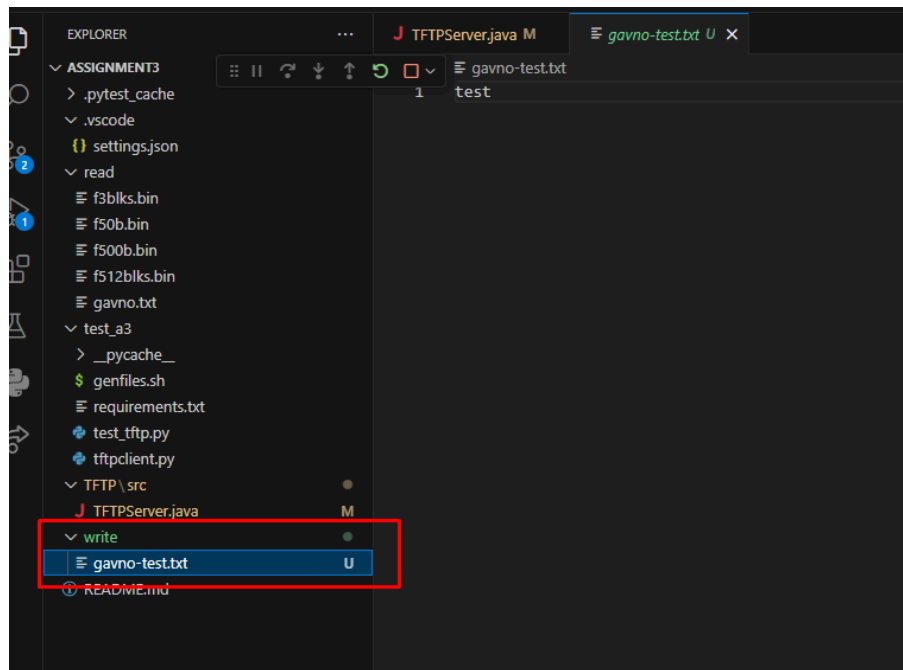


Figure 4. After the Put request, the file appears in the write folder.

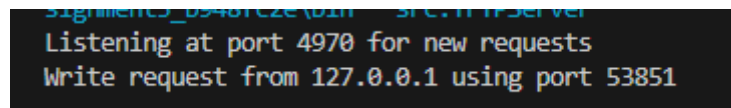


Figure 5. Write request (Console output).

4. As we can see in Figure 3, 4 and 5. The request for a test file “gavno-text.txt” was successful, once we put a request for it, it was written to the write folder, with a custom test name and has all the contents intact.

5. The separation of the two sockets allows for a more controlled and modular approach in handling client requests. This design facilitates concurrent processing of multiple client requests by using separate threads and sockets, enabling better organization and isolation of communication channels with each client.

Socket - responsible for listening to incoming clients requests. It is bound to a specific local port “TFTPPORT” and is used to receive datagrams from clients.

- Its primary purpose is to establish a connection with clients, determine the type of request (read or write), and then delegate the handling of the request to a separate thread.

SendSocket - similar to socket, but it's made dynamically with a port number of 0, that means that the operating system assigns an available port.

- It is usually used for sending responses back to the client that made the initial request. This socket is created for each client request within a new thread. This helps in organizing and managing the bidirectional communication with clients.

6. We began from reading the TFTP specification documents, which outlined the rules and structure of the protocol. This helped me understand the requirements and functionality expected from the server. Some of the difficulties have been encountered, so we had to look for the reference online, in order to enhance our understanding and

overcome these issues. For the Java code, we were following the TFTP specification documents, and trying to employ it into code, had to implement parsing requests, handling read and write requests, managing data transfer and other methods. In order to test it, we had to use a trivial file transfer protocol software called “Tftpd64”, with it we could test the GET requests and PUT requests. Also a good help was “Wireshark”, with it we could see how the UDP packets were exchanged between server and client.

2 VG Problem 2

We did not do the GET request. Only PUT request.

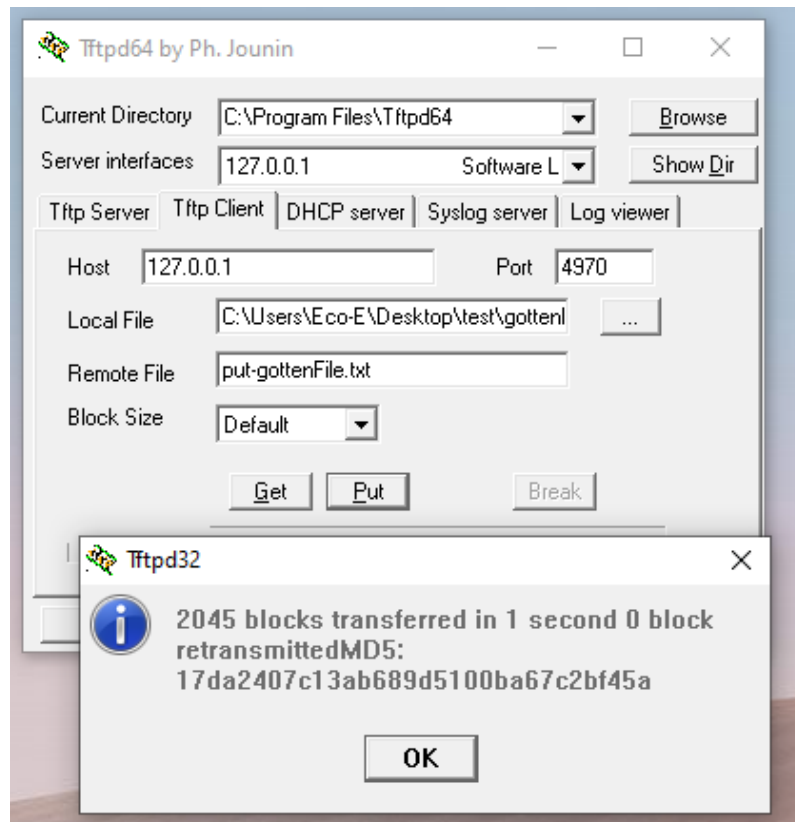


Figure 6. Put a request of a big file.

Writing is done with a couple of steps consisting of

Step 1: In the write request we see the op code setting setting the mode to write, also the filename of which we want to write then the format, octet referring to binary.

Step 2: After we get an ACK from the server that our client's request was successful, if it was unsuccessful we would send an error instead which uses OP-Code OP_ERR.

Step 3: Then we send the data in a data request using OP-Code OP_DATA, we get a ACK back because it's successful.

Step 4: We continue this until the OP_DATA request contains less than 512 bytes. When this occurs we know it's the end of transfer and we send back one last ACK after which the connection is done.

Read Request

25	5.854541077	127.0.0.1	127.0.0.1	TFTP	60 Read Request, File: f500b.bin, Transfer type: octet
30	5.855998900	127.0.0.1	127.0.0.1	TFTP	546 Data Packet, Block: 1 (last)
31	5.856183404	127.0.0.1	127.0.0.1	TFTP	46 Acknowledgement, Block: 1

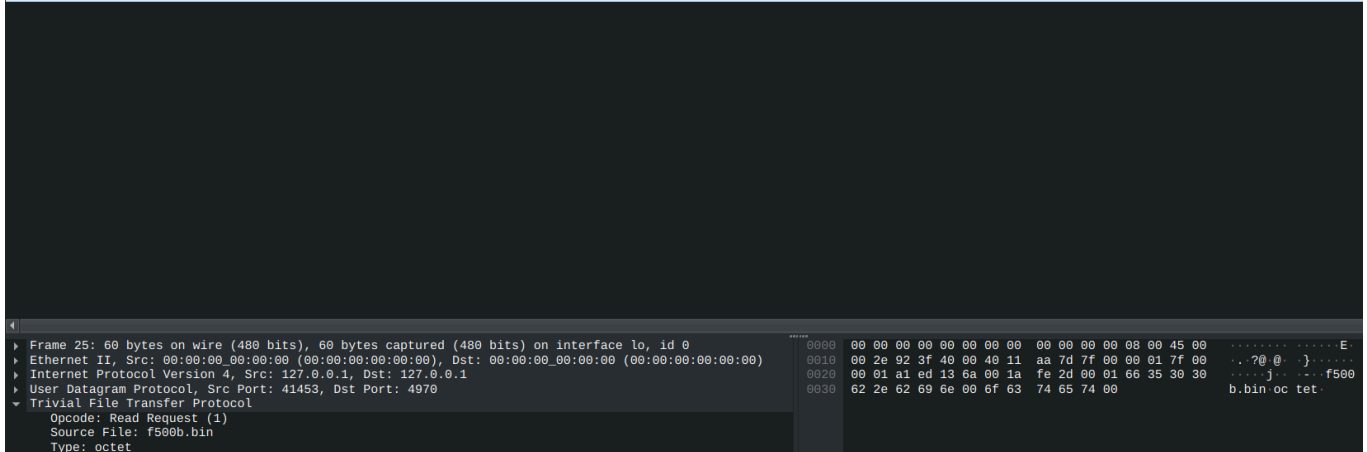


Figure 9. Wireshark application read request.

Step 1: The first step is we request to read a file, we use OP-Code OP_RRQ for read requests also making sure to provide file name and transfer format, octet referring to binary.

Step 2: We get back a data packet from the server containing the data, we send back acknowledgement to the server letting it know we received the data.

Step 3: because the data payload is less than 512 bytes for the data response we know it to be the last last packet in the exchange. If this was not the case we would continue the loop of ACK and DATA requests.

4 VG Problem 4

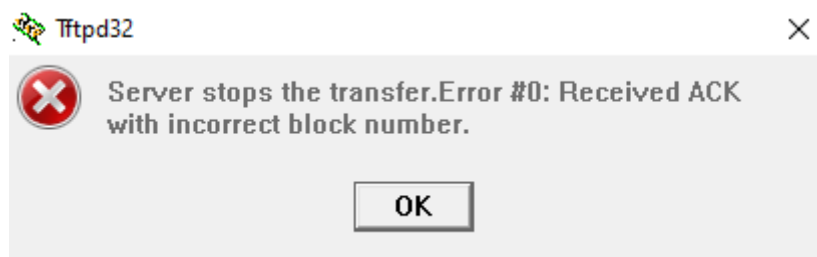


Figure 10. Error handling with opcode 0 (Special message).

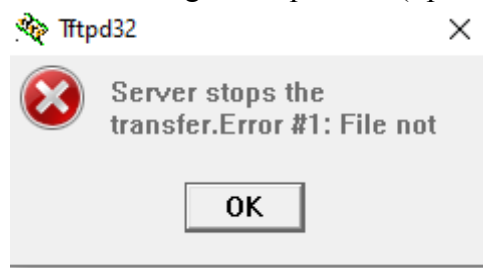


Figure 11. Error handling with opcode 1 (File not found).



Figure 12. Error handling with opcode 2 (Access violation).

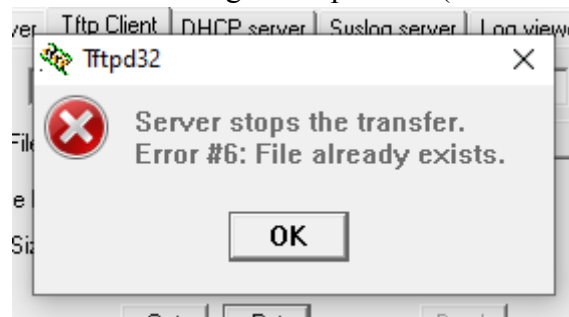


Figure 13. Error handling with opcode 6 (File already exists).

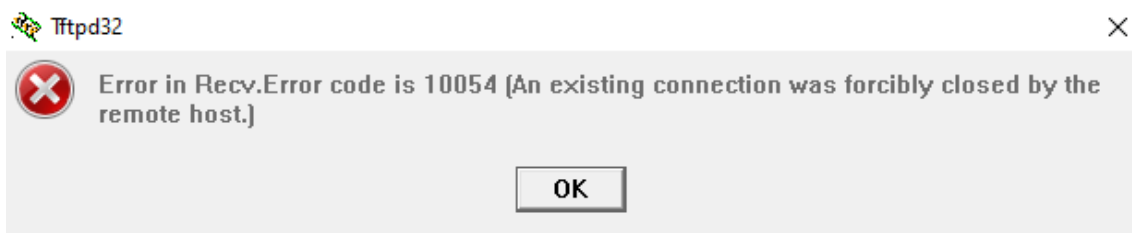


Figure 14. Error message when the server wants to close connection.