Linnéuniversitetet
Kalmar Växjö

Assignment 3: TFTP Server

1DV701 Computer Networks
- Introduction-

Ruth Dirnfeld & Alexandra Bjäremo
rd222dv@student.lnu.se
ab223zm@student.lnu.se
Faculty of Technology
Department of Computer Science

Table of contents

Faculty of Technology

Department of Computer Science

Introduction

This assignment presents a TFTP server implementation, which handles only one transfer mode (octet). The TFTP server has been implemented according to the RFC1350 document and it is based on the provided starter code.

Summary of work

This project represents the collaborative work of Ruth Dirnfeld and Alexandra Bjäremo. We have both participated in the planning and the development of the TFTPServer and of the report.

Ruth Dirnfeld has worked on implementing some of the missing methods from the provided code for the TFTPServer, performed manual tests which have resulted in the screenshots found in this document, as well as writing some of the report. Moreover, Ruth has been the group leader, ensuring the workflow and keeping up the morale. Her participation represents 55% of the workload.

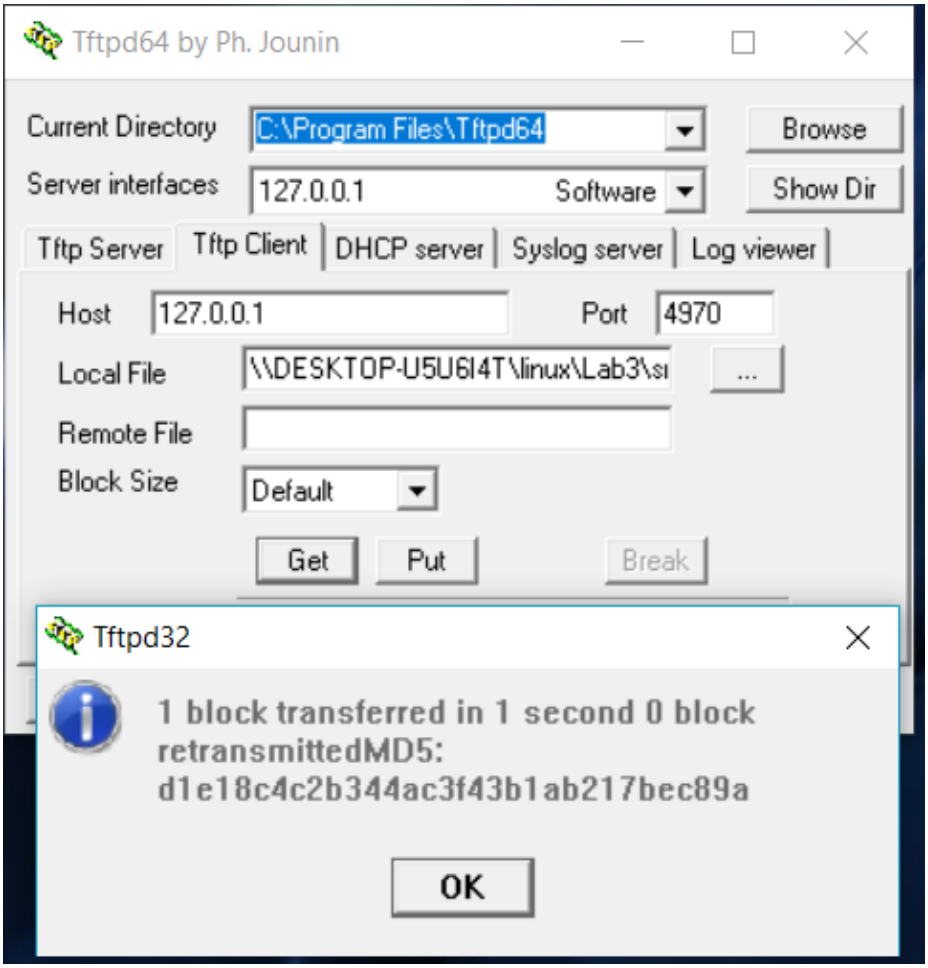
Alexandra Bjäremo has worked on implementing some of the missing methods from the provided code for the TFTPServer, performed manual tests which have resulted in the screenshots found in this document, as well as writing some of the report. Her participation represents 45% of the workload.

Problem 1

For this part of the assignment we have implemented a TFTP server that handles only one transfer mode (octet).

We began the implementation by first downloading the TFTPServer starter code. Afterwards, we decided upon making the server able to handle a read request. This was realized by implementing the receiveFrom() method and the ParseRQ() method, which parses the request. With these two methods done, the TFTP server was able to receive a request and parse it. For the testing of this functionality we have to use the Tftpd64 application. During this test the console was printing out the opcode (1 for read requests), the requested file's name and transfer mode: octet. Next, we implemented the send_DATA_receive_ACK() and receive_DATA_send_ACK() methods, which made the transfers possible.

The screenshot below illustrates a text read request for a file under 512 bytes.

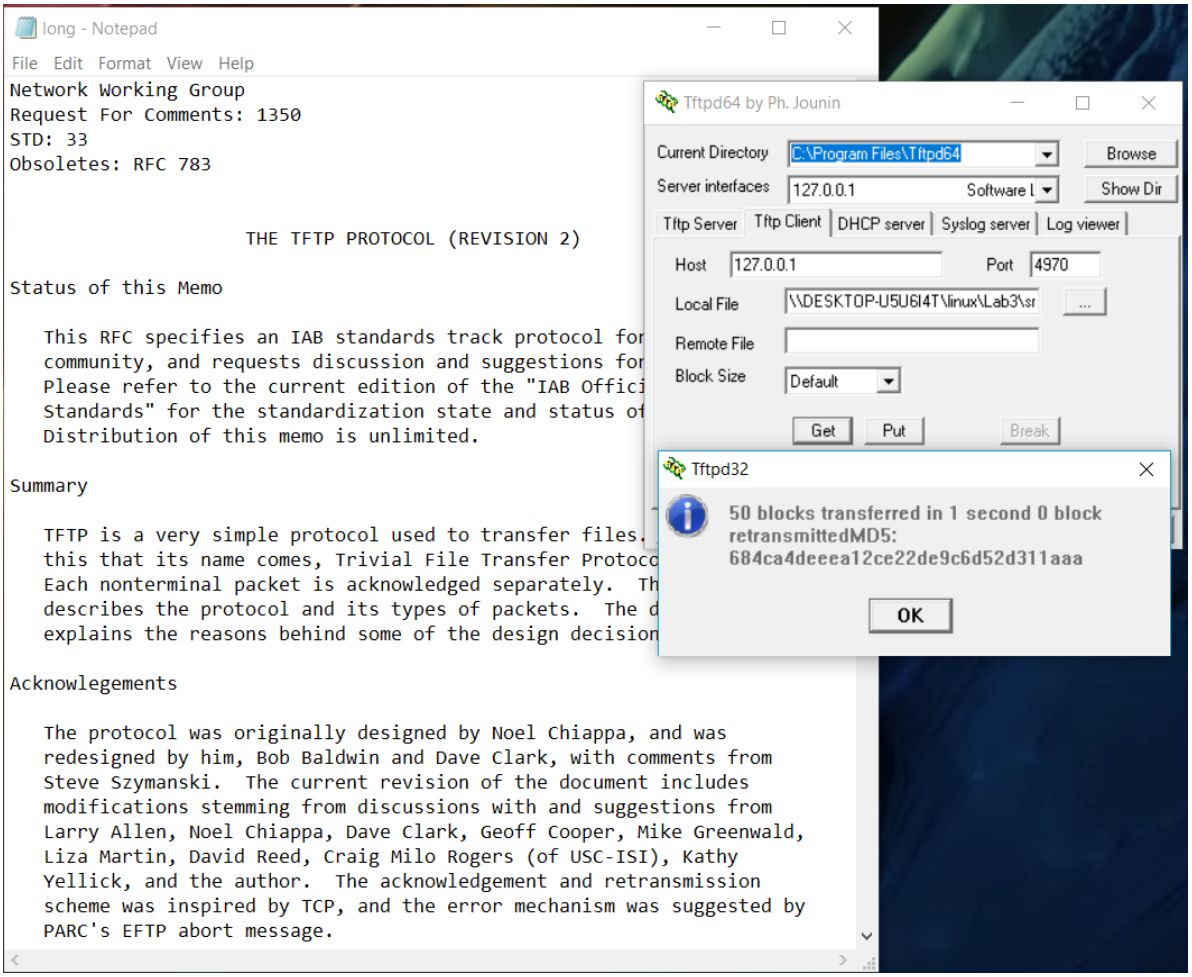


The TFTPServer uses both socket and sendSocket because it is meant to use one socket for listening to incoming requests from clients and another for the connection with these clients. When receiving a request, a connection will be establish and because we are not using multithreading when handling the client request, we must then create a connection on the other socket. Leaving the first one available so it can listen to possible incoming client requests.

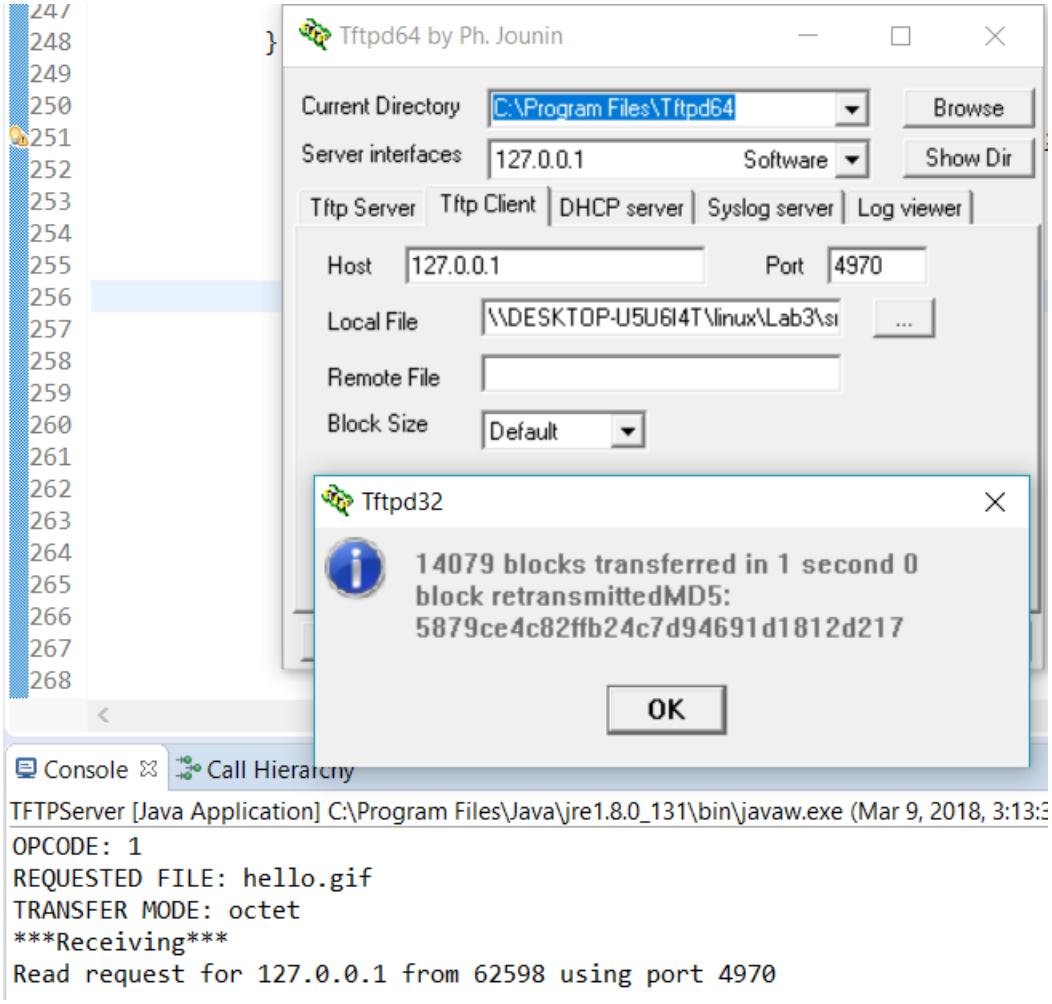
Problem 2

For this part of the assignment we have implemented the possibility for the server to handle larger files, no longer being limited to files smaller than 512 bytes.

The screenshot below illustrates a text read request for a file over 512 bytes in size.



The screenshot below illustrates a .gif read request for a file over 512 bytes in size.

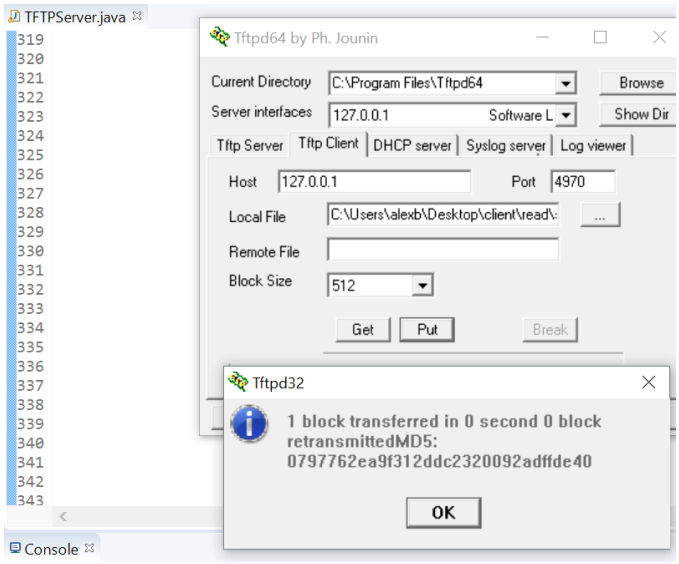


To ensure that all packets are being received, we have implemented a timer, that starts when the packet has been sent and has a predetermined duration. When a packet has been received an acknowledgment should arrive. If that does not happen before the time expires, then the packet should be retransmited and the timer restarted.
Another problem that could appear when transferring packets is the arrival of an acknowledgment of the wrong packet. To avoid this situation we are checking the block number of the packets upon arrival.

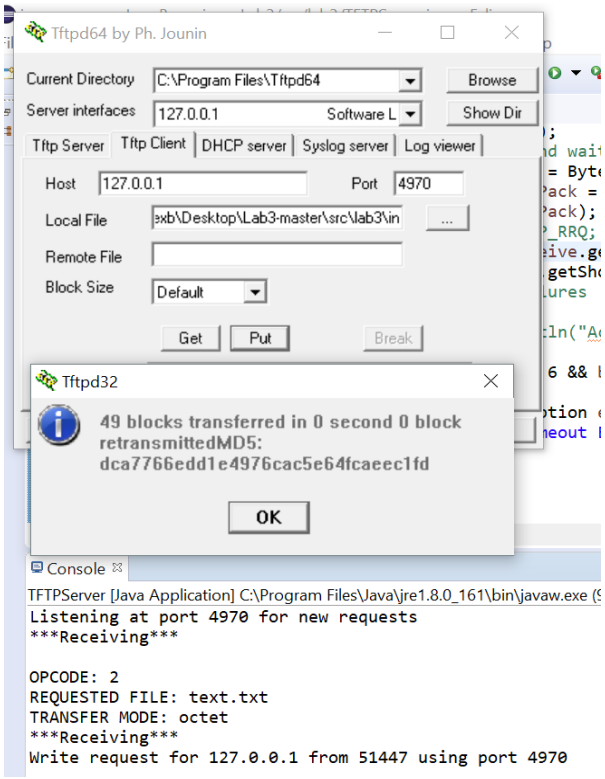
Following the same logic applied for solving the read request we have implemented the methods necessary to make the server able to handle write requests.

The results can be seen in the following screenshots:

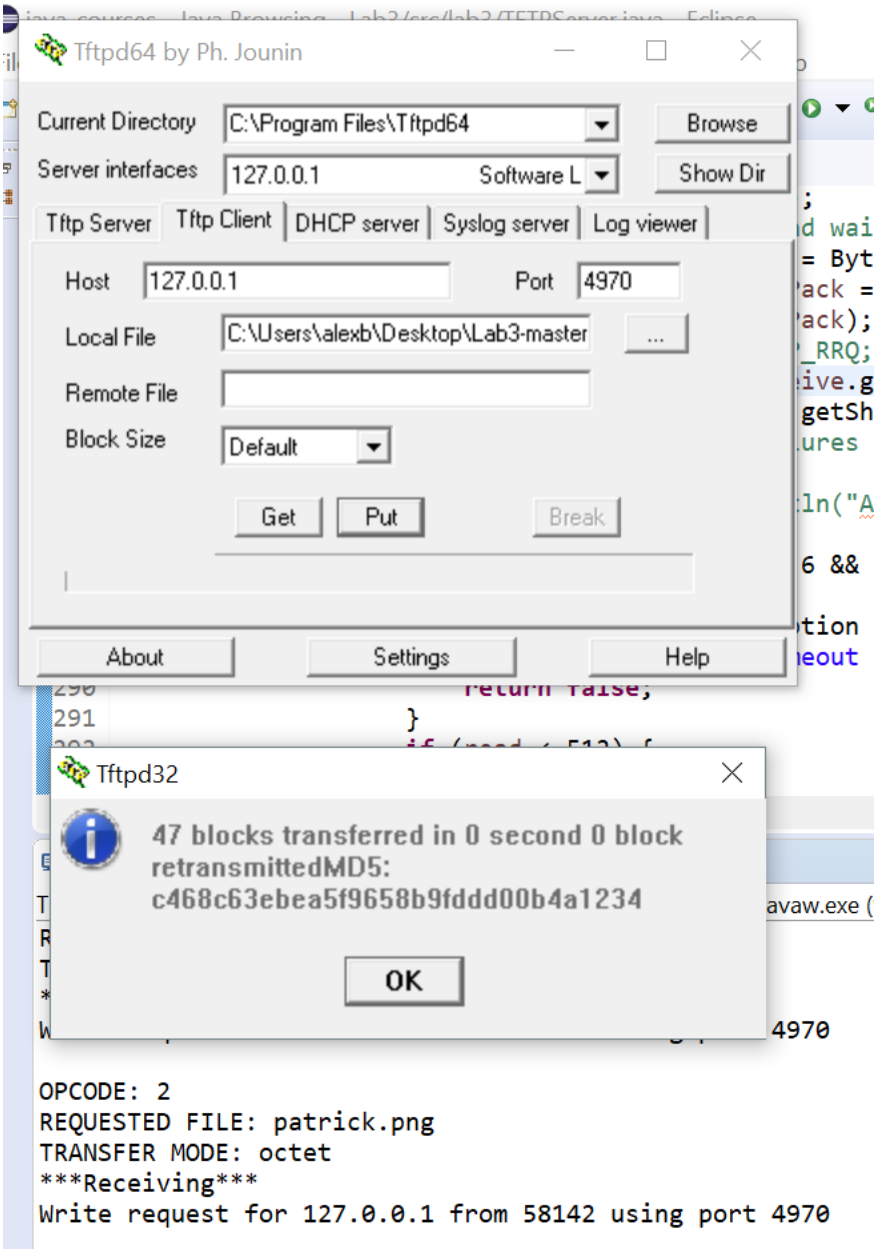
- Write request for a txt file smaller than 512 bytes



- Write request for a txt file larger than 512 bytes.



- Write request for a png file that is larger than 512 bytes

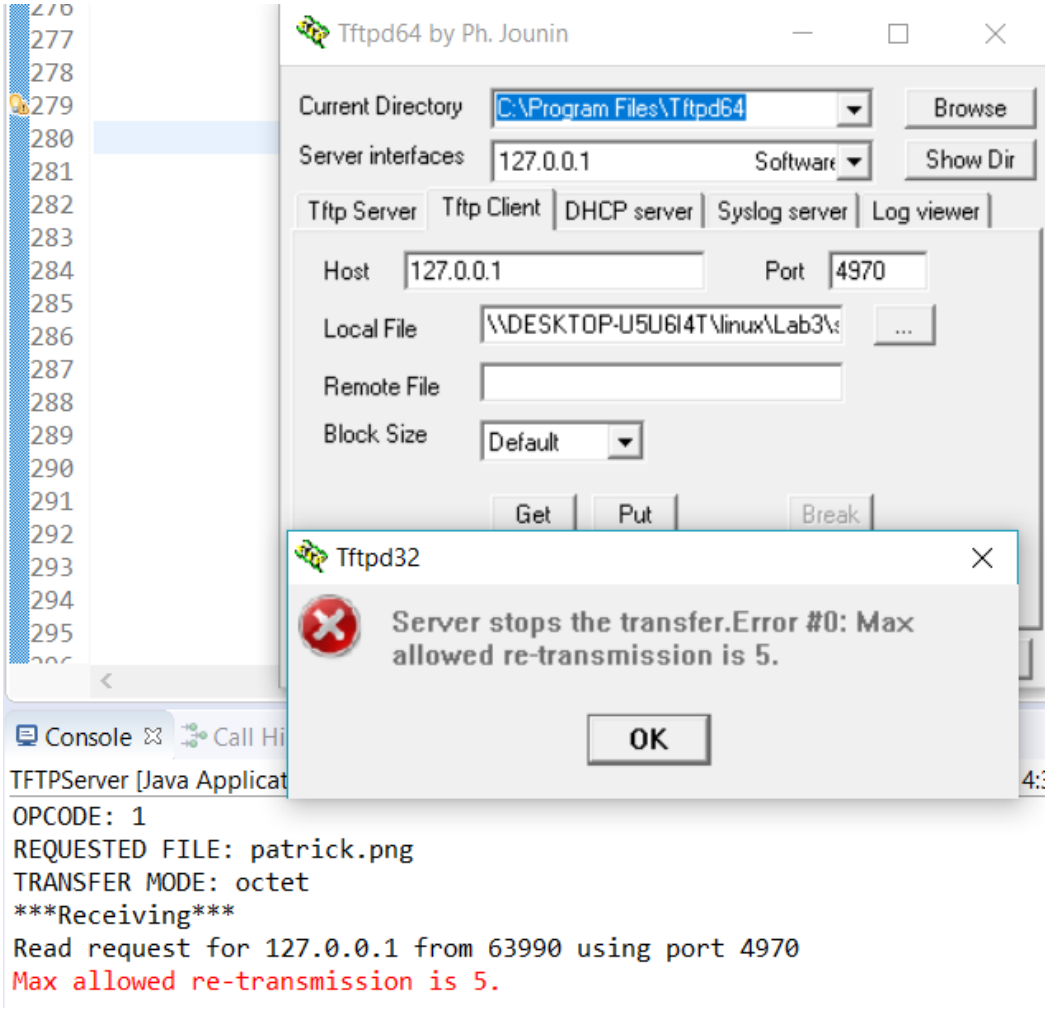


Problem 3

For this part of the assignment the method send_ERR has been implemented. The method handles the error codes 0 (Not defined, see error message (if any)), 1 (File not found), 6 (File already exists).

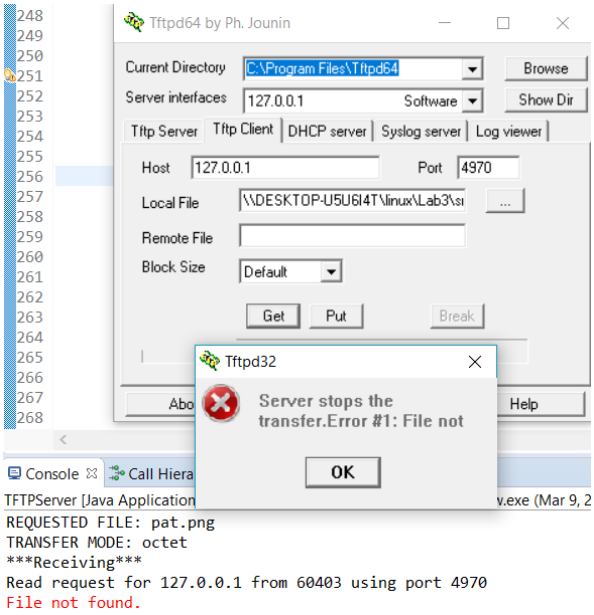
The following screenshots illustrate:

- Error Code #0 - Not defined (in our example: Max allowed retransmission is 5)

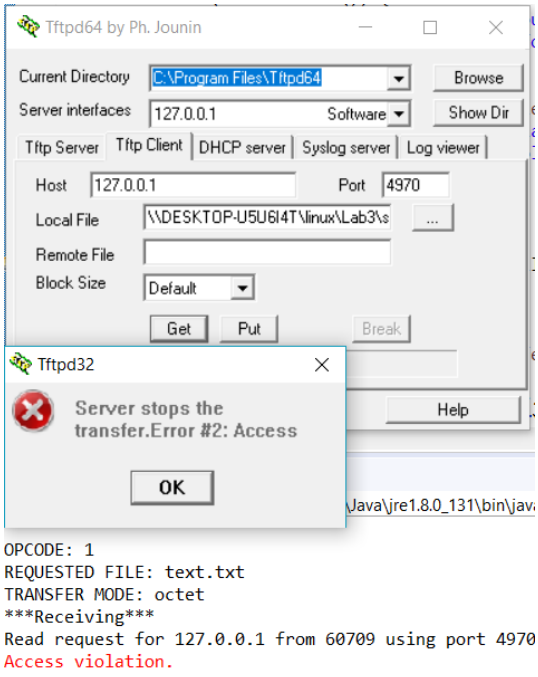


In order to produce this error we have changed the Opcode or block number in the code, so that we force retransmission. For example, when changing the opcode to OP_RRQ, it will start to retransmit the data packet, because it did not receive an ACK. After the limit of 5 retransmissions the Error window will be produced.

- Error Code #1 - File Not Found (when the requested file does not exist)



- Error Code #2 - Access Violation (achieved by throwing a generic IOException)



- Error Code #6 - File already exists (when attempting to make a PUT request for a file that already exists in the “write” directory)