



**Linnéuniversitetet**

Kalmar Våxjö

## Report

# Assignment 3

*IDV701*



*Authors:*

Melika Moayer  
Martim Oliveira

*Semester:*

Spring 2024

*Emails:*

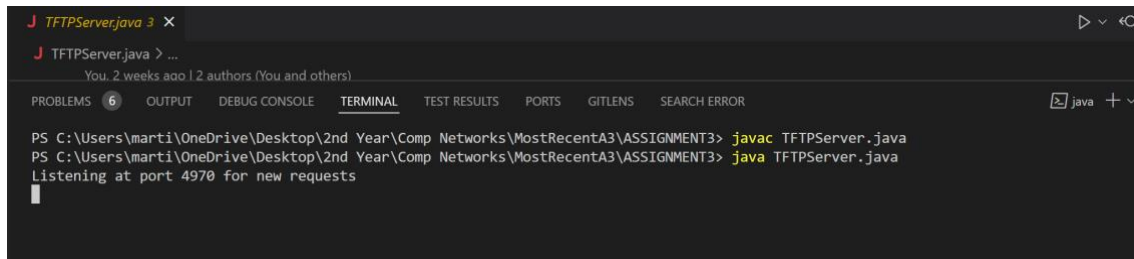
mm224ys@student.lnu.se  
mo223tz@student.lnu.se

Table of Contents

**1 Problem 1 ..... I**  
    1.1 Discussion.....II  
**2 Problem 2 (VG) ..... III**  
    2.1 Discussion.....III  
**3 Problem 3 (VG) ..... V**  
    3.1 Discussion..... V  
**4 Contribution of each group member ..... VI**  
**References:..... VI**

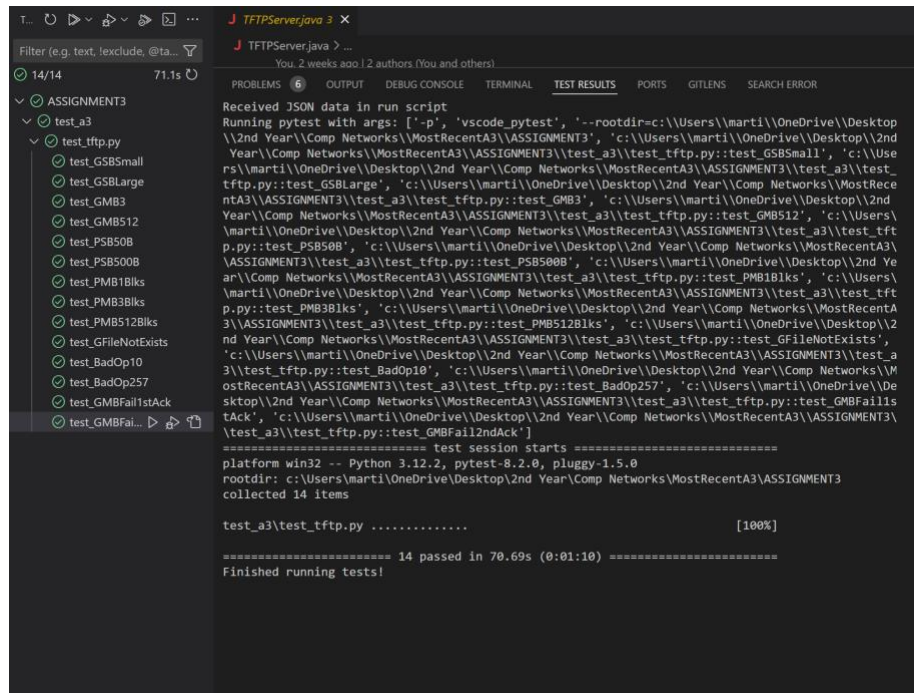
# 1 Problem 1

Screenshots:



```
J TFTPServer.java 3 X
J TFTPServer.java > ...
You, 2 weeks ago | 2 authors (You and others)
PROBLEMS 6 OUTPUT DEBUG CONSOLE TERMINAL TEST RESULTS PORTS GITLENS SEARCH ERROR
PS C:\Users\marti\OneDrive\Desktop\2nd Year\Comp Networks\MostRecentA3\ASSIGNMENT3> javac TFTPServer.java
PS C:\Users\marti\OneDrive\Desktop\2nd Year\Comp Networks\MostRecentA3\ASSIGNMENT3> java TFTPServer.java
Listening at port 4970 for new requests
```

Figure 1. Running the program



```
T... 14/14 71.1s
Filter (e.g. text, !exclude, @ta...)
ASSIGNMENT3
test_a3
test_tftp.py
test_gs5Small
test_gs5Large
test_gmb3
test_gmb512
test_p5b500b
test_pmb1blks
test_pmb3blks
test_pmb512blks
test_gfileNotExists
test_badOp10
test_badOp257
test_gmbFail1stAck
test_gmbFail2ndAck
Received JSON data in run script
Running pytest with args: ['-p', 'vscode_pytest', '--rootdir=c:\\Users\\marti\\OneDrive\\Desktop\\2nd Year\\Comp Networks\\MostRecentA3\\ASSIGNMENT3', 'c:\\Users\\marti\\OneDrive\\Desktop\\2nd Year\\Comp Networks\\MostRecentA3\\ASSIGNMENT3\\test_a3\\test_tftp.py::test_gs5Small', 'c:\\Users\\marti\\OneDrive\\Desktop\\2nd Year\\Comp Networks\\MostRecentA3\\ASSIGNMENT3\\test_a3\\test_tftp.py::test_gs5Large', 'c:\\Users\\marti\\OneDrive\\Desktop\\2nd Year\\Comp Networks\\MostRecentA3\\ASSIGNMENT3\\test_a3\\test_tftp.py::test_gmb3', 'c:\\Users\\marti\\OneDrive\\Desktop\\2nd Year\\Comp Networks\\MostRecentA3\\ASSIGNMENT3\\test_a3\\test_tftp.py::test_gmb512', 'c:\\Users\\marti\\OneDrive\\Desktop\\2nd Year\\Comp Networks\\MostRecentA3\\ASSIGNMENT3\\test_a3\\test_tftp.py::test_p5b500b', 'c:\\Users\\marti\\OneDrive\\Desktop\\2nd Year\\Comp Networks\\MostRecentA3\\ASSIGNMENT3\\test_a3\\test_tftp.py::test_pmb1blks', 'c:\\Users\\marti\\OneDrive\\Desktop\\2nd Year\\Comp Networks\\MostRecentA3\\ASSIGNMENT3\\test_a3\\test_tftp.py::test_pmb3blks', 'c:\\Users\\marti\\OneDrive\\Desktop\\2nd Year\\Comp Networks\\MostRecentA3\\ASSIGNMENT3\\test_a3\\test_tftp.py::test_pmb512blks', 'c:\\Users\\marti\\OneDrive\\Desktop\\2nd Year\\Comp Networks\\MostRecentA3\\ASSIGNMENT3\\test_a3\\test_tftp.py::test_gfileNotExists', 'c:\\Users\\marti\\OneDrive\\Desktop\\2nd Year\\Comp Networks\\MostRecentA3\\ASSIGNMENT3\\test_a3\\test_tftp.py::test_badOp10', 'c:\\Users\\marti\\OneDrive\\Desktop\\2nd Year\\Comp Networks\\MostRecentA3\\ASSIGNMENT3\\test_a3\\test_tftp.py::test_badOp257', 'c:\\Users\\marti\\OneDrive\\Desktop\\2nd Year\\Comp Networks\\MostRecentA3\\ASSIGNMENT3\\test_a3\\test_tftp.py::test_gmbFail1stAck', 'c:\\Users\\marti\\OneDrive\\Desktop\\2nd Year\\Comp Networks\\MostRecentA3\\ASSIGNMENT3\\test_a3\\test_tftp.py::test_gmbFail2ndAck']
===== test session starts =====
platform win32 -- Python 3.12.2, pytest-8.2.0, pluggy-1.5.0
rootdir: c:\\Users\\marti\\OneDrive\\Desktop\\2nd Year\\Comp Networks\\MostRecentA3\\ASSIGNMENT3
collected 14 items

test_a3\\test_tftp.py ..... [100%]

===== 14 passed in 70.69s (0:01:10) =====
Finished running tests!
```

Figure 2. All the tests passing

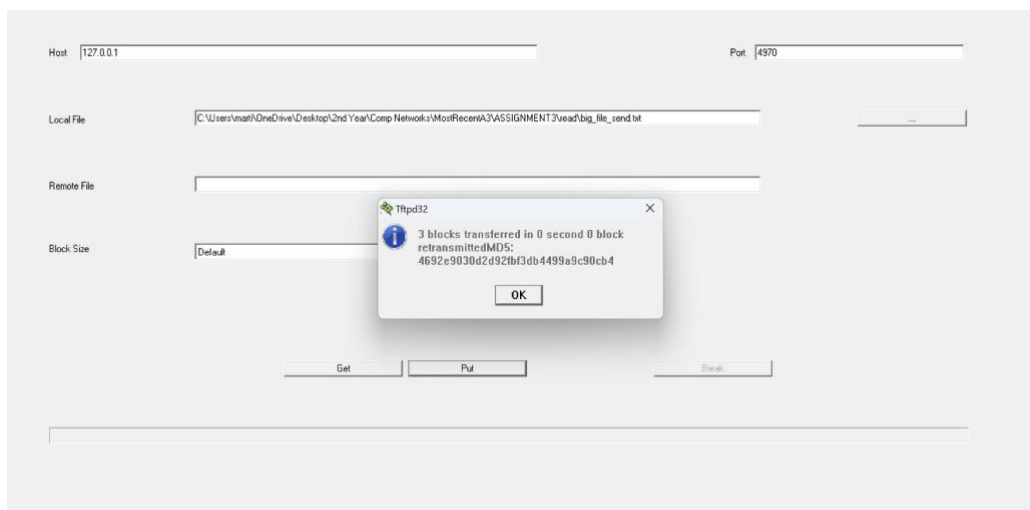


Figure 3. TFTP Server Reading Requests

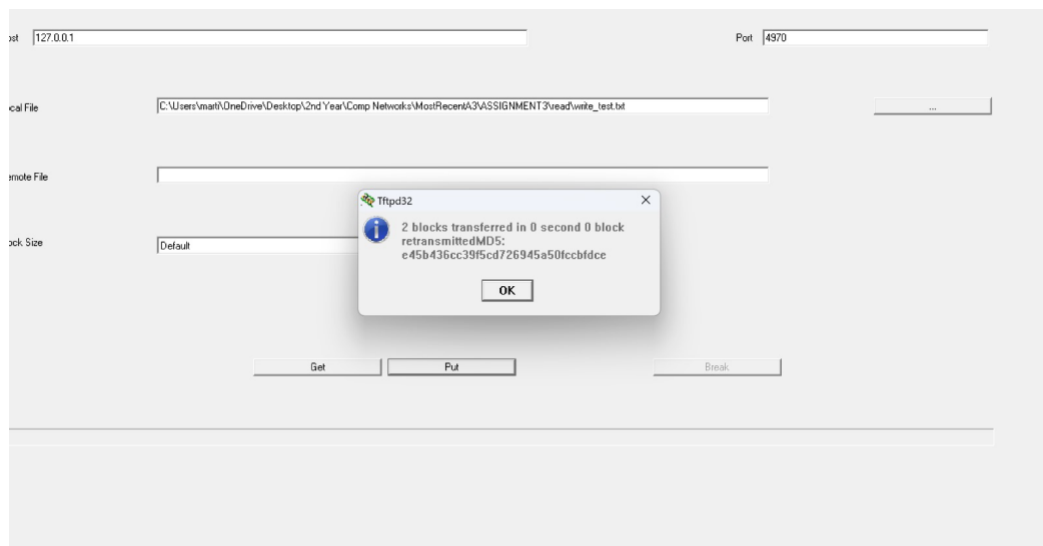


Figure 4. TFTP Server Writing Requests

## 1.1 Discussion

The first problem is an implementation of a Trivial File Transfer Protocol (TFTP) server in Java. TFTP is a simple protocol used for transferring files between a client and a server. The server listens for requests from clients, handles read and write operations, and sends or receives data packets accordingly.

### Code explanation:

The TFTPServer class consists of methods and fields necessary for handling TFTP requests and managing file transfer operations.

The *main* method initializes the server by creating an instance of the TFTPServer class and starting it. Then the *start* method sets up a DatagramSocket to listen for incoming requests on the specified port. It enters a loop to continuously handle client requests by spawning a new thread for each request. Then the *ReceiveFrom* Method reads the first block of data from the received DatagramPacket, which contains the request for an action (read or write), and returns the client's socket address. Afterwards *ParseRQ* Method parses the request packet to determine the type of request (read or write) and the requested file name. It extracts the opcode and the file name from the packet data. *HandleRQ* Method handles read and write requests based on the opcode extracted from the request packet. For read requests, it reads the requested file from the server's directory and sends it to the client in blocks of data. For write requests, it receives data packets from the client and writes them to the specified file on the server. As acknowledgments are quite important, *SendDataReceiveACK* Method sends data packets to the client and waits for an acknowledgment (ACK) packet. It retries sending the data packet in case of timeouts or missing acknowledgments. And on the other hand *ReceiveDataSendACK* Method receives data packets from the client and sends acknowledgment packets (ACK) back to the client. It handles timeouts and retransmissions to ensure reliable data transfer. *SendERR* Method sends an error packet to the client in case of invalid requests or other error conditions during file transfer operations. And finally *Ack* Method and *AckPck* Method handle the acknowledgment packets sent and received during data transfer, ensuring proper acknowledgment of data blocks.

### Explanation of Socket and SendSocket:

The code uses two DatagramSocket objects: 'socket' and 'sendSocket'. The 'socket' is used for receiving incoming packets from clients, while the 'sendSocket' is used for sending packets back to clients. The reason for using two separate sockets is to ensure that the server can handle multiple client requests concurrently. By creating a new 'sendSocket' for each client request within a separate thread, the server can maintain separate communication channels with each client without blocking while waiting for acknowledgments or handling timeouts.

Note:

Figure.1, indicates the code successfully passing the provided test file.

## 2 Problem 2 (VG)

Screenshots:

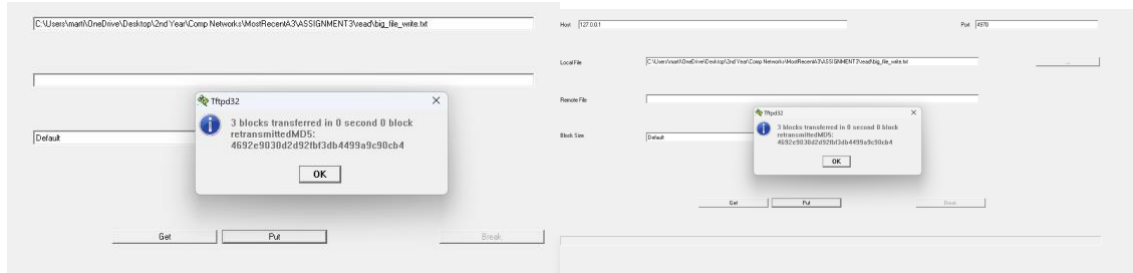


Figure 5. TFTP Server Read and Write Requests for a file larger than 512 bytes

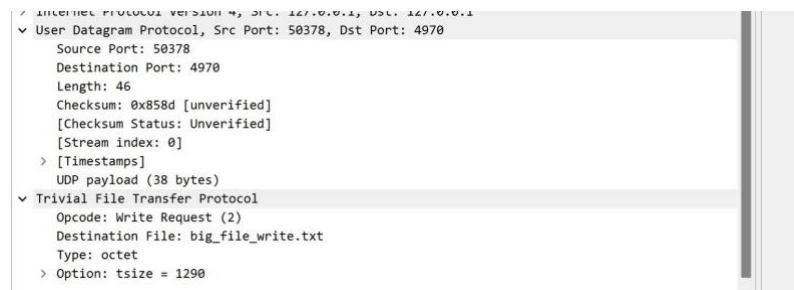


Figure 6. Screenshots of sending multiple large files demonstrating server's capability to handle files larger than 512 bytes.

### 2.1 Discussion

#### Support for Files Larger than 512 Bytes:

The TFTPServer has been extended to support files larger than 512 bytes.

- *Changes Made:*

- The `DATA\_SIZE` constant was set to 512 bytes to maintain compatibility with the original TFTP packet size.
- The `send\_DATA\_receive\_ACK` and `receive\_DATA\_send\_ACK` methods were adjusted to handle larger files by sending and receiving multiple data packets and ACKs.

#### Timeout and Retransmission Functionality:

Implemented timeout and retransmission functionality for read requests.

- *Implemented Changes:*

- A timeout mechanism was added to the `send\_DATA\_receive\_ACK` method.
- If no acknowledgment is received within a specified time (5 seconds), the server retransmits the previous packet.
- Implemented checks to ensure that retransmission is done only when necessary, i.e., when the acknowledgment for the wrong packet is received (incorrect block number).
- Ensured the server does not enter an endless retransmission loop by limiting the number of retries to 5.

#### Code explanation:

*files larger than 512 bytes:*

The method (DATA\_SIZE) reads data from the file in chunks of up to 512 bytes.  
It constructs and sends data packets (OP\_DAT) to the receiver.  
Implements a loop for handling retries and timeouts to ensure reliable transmission.  
Sends final ACK packet when the end of the file is reached.  
In receive\_DATA\_send\_ACK it listens for incoming data packets (OP\_DAT).  
Writes received data to a file and sends acknowledgments (OP\_ACK).  
Closes the file when the end of the file is detected (packet smaller than 512 bytes).

*Timeout Implementation:*

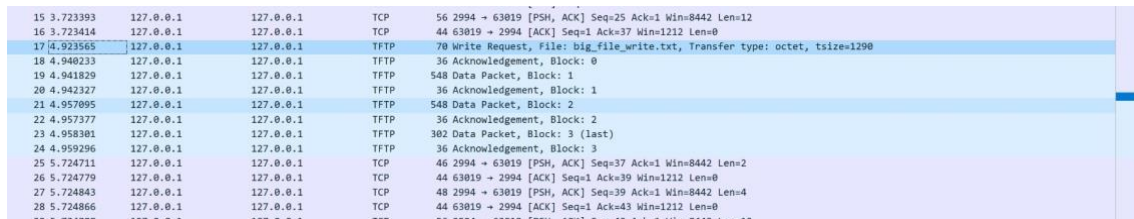
socketSend.setSoTimeout(5000); sets a timeout of 5 seconds for waiting for an ACK packet.  
If no ACK is received within this time, a SocketTimeoutException is caught, and the server retries sending the data packet.  
retryCount keeps track of the number of retries, and if it reaches 5, the server sends an error packet (OP\_ERR) with an appropriate message.

*Retransmission Handling:*

If the server receives an ACK packet with the wrong block number (ackBlock != block), it considers it an error and sends an error packet.  
After a timeout or receiving an incorrect ACK, the server retransmits the previous data packet.

## 3 Problem 3 (VG)

Screenshot:

A screenshot of a Wireshark network packet capture. The table shows 13 packets. Packets 15, 16, 25, 26, 27, and 28 are TCP segments. Packets 17, 18, 19, 20, 21, 22, 23, and 24 are TFTP packets. The TFTP packets include a Write Request (17), Acknowledgements (18, 19, 20, 21, 22, 23, 24), and Data Packets (19, 20, 21, 22, 23, 24). The TCP segments show a sequence of push and acknowledgment flags, indicating a data transfer from port 2994 to port 63019.

No.	Time	Source	Destination	Protocol	Length	Info
15	3.723393	127.0.0.1	127.0.0.1	TCP	56	2994 → 63019 [PSH, ACK] Seq=25 Ack=1 Win=8442 Len=12
16	3.723414	127.0.0.1	127.0.0.1	TCP	44	63019 → 2994 [ACK] Seq=1 Ack=37 Win=1212 Len=0
17	4.923565	127.0.0.1	127.0.0.1	TFTP	70	Write Request, File: big_file_write.txt, Transfer type: octet, tsize=1290
18	4.940233	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 0
19	4.941829	127.0.0.1	127.0.0.1	TFTP	548	Data Packet, Block: 1
20	4.942327	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 1
21	4.957895	127.0.0.1	127.0.0.1	TFTP	548	Data Packet, Block: 2
22	4.957377	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 2
23	4.958301	127.0.0.1	127.0.0.1	TFTP	302	Data Packet, Block: 3 (Last)
24	4.959296	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 3
25	5.724711	127.0.0.1	127.0.0.1	TCP	46	2994 → 63019 [PSH, ACK] Seq=37 Ack=1 Win=8442 Len=2
26	5.724779	127.0.0.1	127.0.0.1	TCP	44	63019 → 2994 [ACK] Seq=1 Ack=39 Win=1212 Len=0
27	5.724843	127.0.0.1	127.0.0.1	TCP	48	2994 → 63019 [PSH, ACK] Seq=39 Ack=1 Win=8442 Len=4
28	5.724866	127.0.0.1	127.0.0.1	TCP	44	63019 → 2994 [ACK] Seq=1 Ack=43 Win=1212 Len=0

Figure 7. Wireshark capture during a read request

### 3.1 Discussion

#### line-by-line analysis of the Wireshark capture during a read request:

- Frame 15:  
TCP segment with the Push and Acknowledgment flags set. It contains data from port 2994 to port 63019. The sequence number is 25, and the acknowledgment number is 1. Window size is 8442 bytes, and the payload length is 12 bytes.
- Frame 16:  
TCP acknowledgment from port 63019 to port 2994. Acknowledgment number is 37, indicating the successful reception of data. Window size is 1212 bytes.
- Frame 17:  
TFTP Write Request. The file being requested is "big\_file\_write.txt" with transfer type "octet" and tsize (total file size) is 1290 bytes.
- Frame 18:  
TFTP Acknowledgment for Block 0. It acknowledges the successful receipt of the write request.
- Frames 19-24:  
Data packets and acknowledgments for blocks 1, 2, and 3. Each data packet contains 512 bytes of data, except the last one which contains 302 bytes. After receiving each data packet, the client sends an acknowledgment.
- Frames 25-28:  
TCP segments with Push and Acknowledgment flags set, indicating the transfer of remaining data. The sequence numbers and acknowledgment numbers increment accordingly.

#### Difference between Read and Write Requests:

- Read Request: In a read request, the client requests a file from the server. The server responds by sending data packets containing the contents of the requested file, and the client acknowledges each packet received.
- Write Request: In a write request, the client sends a file to the server. The server responds with acknowledgment packets for each data block received from the client.

## 4 Contribution of each group member

Martim Oliveira: 50%, Melika Moayer: 50%

Like the previous assignment, We mostly had Zoom meetings and worked on the code together, trying to come up with solutions to solve the problems. Finally, we both wrote the report after holding relative discussions.

## References:

1. IETF, <https://www.ietf.org/rfc/rfc1350.txt> (accessed April 28, 2024).
2. “Java socket programming - simple client server program,” YouTube, <https://www.youtube.com/watch?v=-xKgxqG411c> (accessed April 26, 2024).
3. “What is TFTP?,” Tutorialspoint, <https://www.tutorialspoint.com/what-is-tftp> (accessed May 9, 2024).