| Report |
| --- |

Assignment 3

*1DV701*

| *Authors:*  Henry Shafer  Elias Michalsky  *Semester:* Spring 2024  *Emails:*  hs223nr@student.lnuse  em223qw@student.lnu.se |
| --- |

[**1 Problem 1 2**](#_xiykoxdvna3p)

[1.1 Preconception 2](#_gsfwqmvxe9z)

[1.1.1 Starter Code 3](#_mc78gz80f5lm)

[1.1.2 RFC1350 3](#_dul5l5hej7mu)

[1.1.3 Python Tests 3](#_zi5t4gglrotp)

[1.2 Handling a Single Read Request 3](#_5weq0dq0wg6x)

[1.2a. Implement receiveFrom() method. 3](#_yrp72py27cdm)

[1.2b. Implement ParseRQ() method. 3](#_x7xd61qam7ee)

[1.2c. Open the requested file. 3](#_sis725qmawu0)

[1.2d. Create the response packet with the opcode for data ( OP\_DATA ) and a block 3](#_s003oo2tcpox)

[number (1). 3](#_gk1jazy0oepn)

[1.2e. Read a maximum of 512 bytes from the file, add these to the packet and send it 4](#_1qd3vtevrxy1)

[to the client. 4](#_740emdhsq4lt)

[1.2f. Receive client response with an acknowledgement ( ACK ) of the first package. 4](#_xjmp8jbqw1a6)

[1.1 Discussion 5](#_c9p2n166qsa6)

[**2 Problem 2 5**](#_21j6eih8y8rl)

[2.1 Discussion 5](#_sigqunhnzwq9)

[2.2 VG 1 5](#_97176oil95ok)

[2.2.1 Discussion 5](#_rn2btfsujz9q)

[**3 Problem 3 5**](#_87pbvnwolwin)

[3.1 Discussion 5](#_ukeq1owzjpqz)

[3.2 VG 2 6](#_epjmyzya1nld)

[3.2.1 Discussion 6](#_7i073v9uqki)

# **1 Problem 1**

Implementing a TFTP server according to RFC1350 with octet mode compatibility.

## 1.1 Preconception

Upon downloading and reading the starter code, RFC1350, and the tests, we made the following assumptions about the task.

### 1.1.1 Starter Code

Provides a framework for UDP socket communication and TFTP requests.

### 1.1.2 RFC1350

Reading the RFC 1350 gave valuable insight into key elements such as opcode definitions, packet formats, error handling, and transfer modes (octet mode in particular).

### 1.1.3 Python Tests

The tests give insights into how the server is expected to perform, they also outline critical functionalities of the server such as handling read/write requests, parsing request packets, sending data packets, and handling acknowledgments.

## 1.2 Handling a Single Read Request

Steps with notes:

### 1.2a. Implement receiveFrom() method.

The method listens on a predefined port for TFTP packet transmissions, upon arrival the server processes the packet according to desired TFTP protocol.

### 1.2b. Implement ParseRQ() method.

This method does the parsing according to TFTP protocol RFC1350. It extracts relevant information from the incoming request packet, such as the requested filename and transfer mode.

### 1.2c. Open the requested file.

Once the request has been parsed successfully, the server must access the requested file from storage; this includes validating the existence of the file and ensuring that appropriate permissions are in place for reading it and handling any errors that could arise from this.

### 1.2d. Create the response packet with the opcode for data ( OP\_DATA ) and a block number (1).

After opening the file, a response packet is constructed using the desired opcode and block number.

### 1.2e. Read a maximum of 512 bytes from the file, add these to the packet and send it

### to the client.

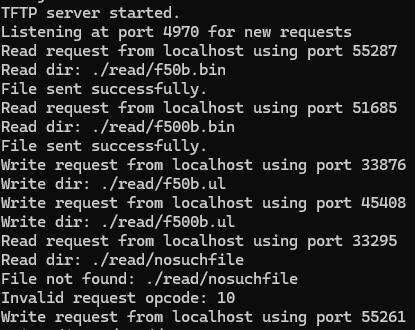
The server must now start sending packets, using the constructed response from step d and appending up to 512 bytes from the requested file to the response. This packet is then sent over the established UDP connection.

### 1.2f. Receive client response with an acknowledgement ( ACK ) of the first package.

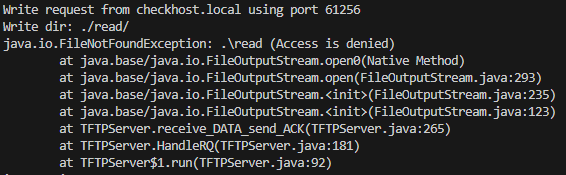
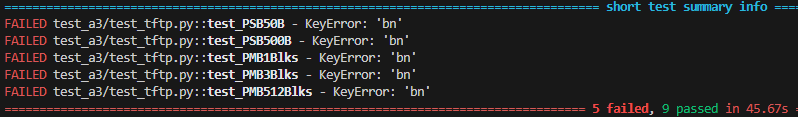
After receiving the first packet, the client is expected to send back an acknowledgement (ACK) packet saying they received the first packet. The server is expected to wait and handle this packet upon receival.

### 1.3. Make a Request to Read

Once you have successfully completed the steps, make a read request from the client (request to read a file that is shorter than 512 bytes) and check that everything works properly. Include a screenshot of this in your report.



### 1.4. Request to write

A file that is shorter than 512 bytes and check that everything works properly. Include a screenshot of this in your report.

## 1.5. Why socket, and sendSocket?

While both sockets establish a connection to the client it is essential to separate their responsibilities. Socket is responsible for establishing the initial connection to the client and receiving requests while the sendSocket is for sending packets. It is important to keep the two separated to ensure reliability, scalability, and robustness.

## 1.6. Solving Process

We worked in the order of the steps to some degree but as concepts from the server's implementation started to make more sense similar parts of each unfinished method were finished in unison. Finally we had something that could be tested, then we tested and tediously corrected all the failed tests.

## **1.1 Discussion**

While working on this assignment we ran into a lot of trouble with our computers not having the same results, which made testing and splitting the workload difficult. For instance, while testing one of us could have two tests passing while the other had all of them pass except one. Having only failed one test on the one machine, we decided to primarily work through that computer, however we did not manage to implement the write functionality. While trying to implement the write, we got invalid block numbers, Keyerror: ‘bn’ and access errors. We can’t seem to get them to line up with the tests.