Linnéuniversitetet
Kalmar Växjö

Report

Assignment 3

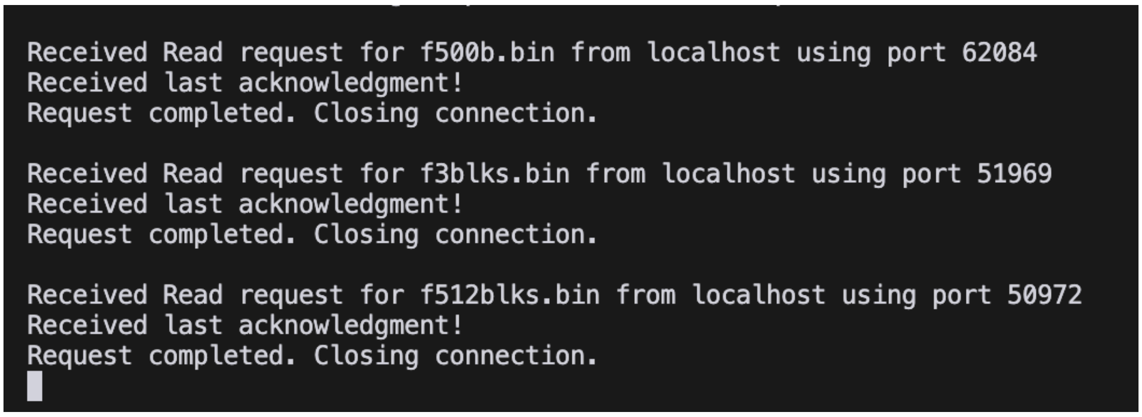
1DV701

Linnéuniversitetet
Kalmar Växjö

Author: Aleksas Ladinskis,
Rihards Okmanis
Semester: Spring 2024
Email: al226gc@student.lnu.se,
ro222ij@student.lnu.se

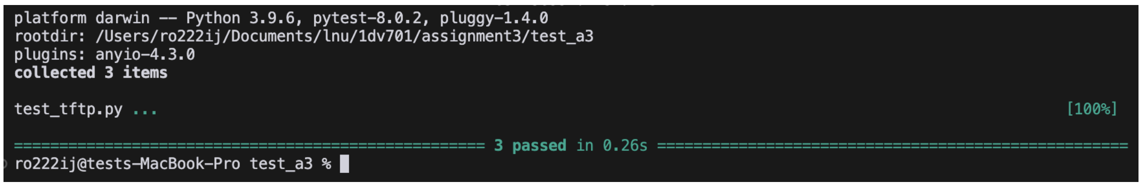
Contents

1 Problem 1



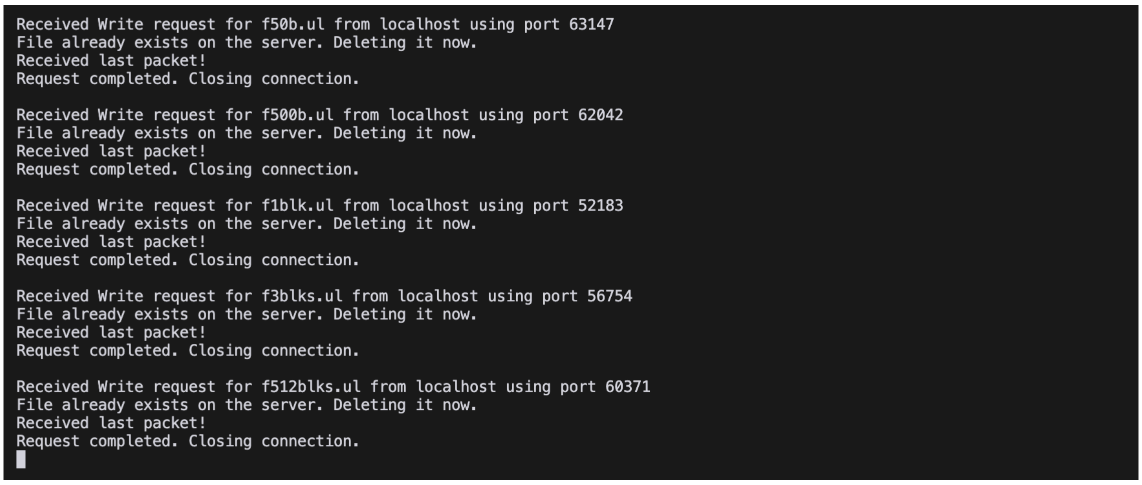
```
Received Read request for f500b.bin from localhost using port 62084
Received last acknowledgment!
Request completed. Closing connection.

Received Read request for f3blks.bin from localhost using port 51969
Received last acknowledgment!
Request completed. Closing connection.

Received Read request for f512blks.bin from localhost using port 50972
Received last acknowledgment!
Request completed. Closing connection.
```

Server log for read requests



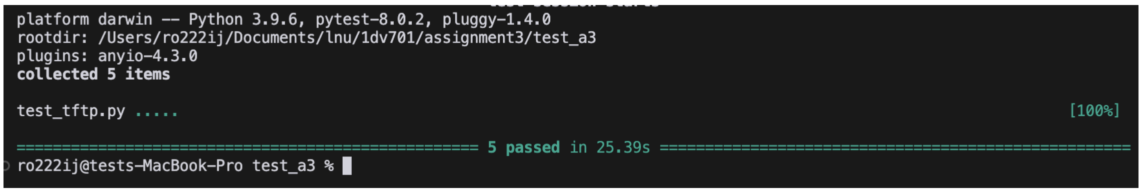
```
platform darwin -- Python 3.9.6, pytest-8.0.2, pluggy-1.4.0
rootdir: /Users/ro222ij/Documents/lnu/1dv701/assignment3/test_a3
plugins: anyio-4.3.0
collected 3 items

test_tftp.py ...                                                                    [100%]

==================================================== 3 passed in 0.26s ====================================================
ro222ij@tests-MacBook-Pro test_a3 %
```

Results of python tests for read requests



```
Received Write request for f50b.ul from localhost using port 63147
File already exists on the server. Deleting it now.
Received last packet!
Request completed. Closing connection.

Received Write request for f500b.ul from localhost using port 62042
File already exists on the server. Deleting it now.
Received last packet!
Request completed. Closing connection.

Received Write request for f1blk.ul from localhost using port 52183
File already exists on the server. Deleting it now.
Received last packet!
Request completed. Closing connection.

Received Write request for f3blks.ul from localhost using port 56754
File already exists on the server. Deleting it now.
Received last packet!
Request completed. Closing connection.

Received Write request for f512blks.ul from localhost using port 60371
File already exists on the server. Deleting it now.
Received last packet!
Request completed. Closing connection.
```

Server log for write requests



```
platform darwin -- Python 3.9.6, pytest-8.0.2, pluggy-1.4.0
rootdir: /Users/ro222ij/Documents/lnu/1dv701/assignment3/test_a3
plugins: anyio-4.3.0
collected 5 items

test_tftp.py .....                                                                  [100%]

=================================================== 5 passed in 25.39s ===================================================
ro222ij@tests-MacBook-Pro test_a3 %
```

Results of python tests for write requests

1.1 Discussion

The fist problem we encountered was focused on implementing a basic TFTP server. First things first we focused our efforts on understanding RFC 1350 standards, in particular, request handling process and syntax. After gaining an understanding of what is expected of a TFTP server we moved to implementing the functionality of receiving , parsing and sending data to the client.

We split work between each other in the following manner: Rihards focused on tasks related to read requests, Alex focused on tasks related to write requests. receiveFrom() implementation was given to Rihards and parseRQ() implementation was given to Alex. Since we were working in the same file simultaneously, we knew that we would encounter merge conflicts. Since it is not possible to avoid, we decided to tackle it by separating work into different branches and resolving merge conflicts via rebasing. Every time we hit a minor milestone, we would send a merge request to master, rebase and continue work.

When it comes down to details of implementation, there are several key decisions we made:

1. Leverage DatagramPackets to transfer information. Since DatagramSocket was already used in the template for establishing connection, it is logical to fully use Datagram capabilities for carrying out the task.
2. Limit receive_DATA_send_ACK and send_DATA_receive_ACK methods to handling one DataPackage upon a call. It separates the responsibilities according to the method names.
3. Handle requests that require multiple DataPackages to be exchanged in HandleRQ function. It separates responsibilities according to the method names.
4. Extensively log messages at different stages of request processing. It makes debugging easier.

On top of everything mentioned above, here is the list of bugs we encountered and fixed in the template code:

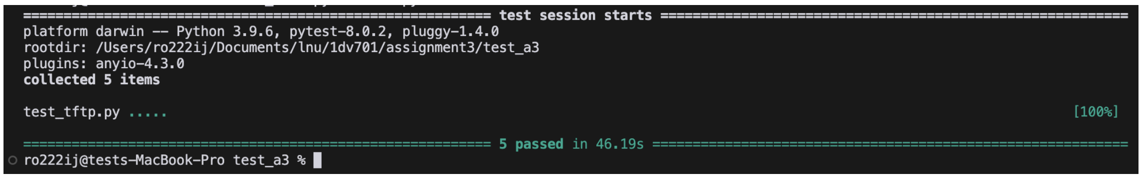
1. Every request with opcode different from read opcode was treated as the write request. We reworked the server to accept read and write requests and throw an error in case of any opcode that doesn't represent any of these two.
2. Undefined variable param being passed as argument and expected as parameter in several places. Apparently, it was placeholder code that was meant to illustrate what structure was expected from us. We deleted it to avoid compilation errors.
3. Missing variable in system print upon receiving a new request. The statement was expecting 4 variables: request type, requested file name, client host name, client port. The requested file name variable was missing in the template code.
4. Socket being closed only if the request was successfully handled. In case an exception was raised, it was handled, yet the socket for this particular client was never close

In the TFTPServer.java code, two sockets are used: socket and sendSocket.

1. **socket:** This is the main server socket that is bound to a specific port (TFTPPORT). It is used to listen for incoming client requests. Once a request is received, the server processes the request to determine the type of operation (read or write) and the file that the client is requesting.
2. **sendSocket:** This socket is created for each new client request. It is used to handle the communication with the specific client that made the request. This includes sending data to the client (in case of a read request) or receiving data from the client (in case of a write request). The sendSocket is connected to the client's address and port, and it is used in a separate thread to handle the client's request independently of other requests.

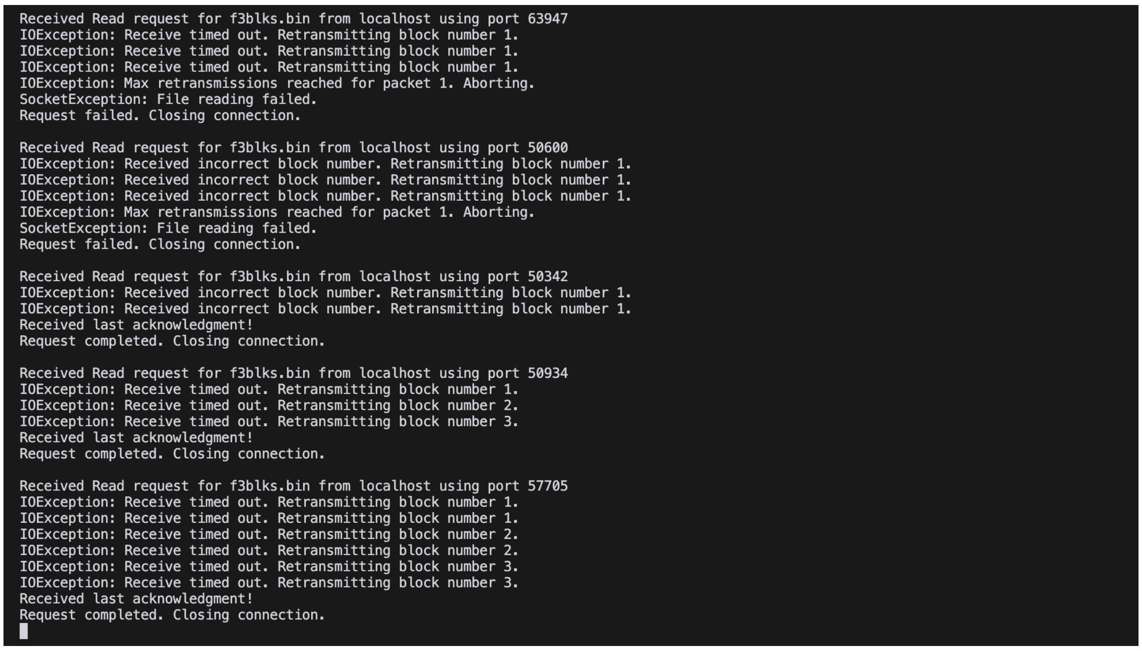
The reason for using two sockets is to allow the server to handle multiple client requests concurrently. The main socket is used to accept new client requests, while the sendSocket is used to handle the communication with each individual client. This design allows the server to continue accepting new requests while it is processing existing ones, thereby improving the server's performance and responsiveness.

2 VG Problem-2



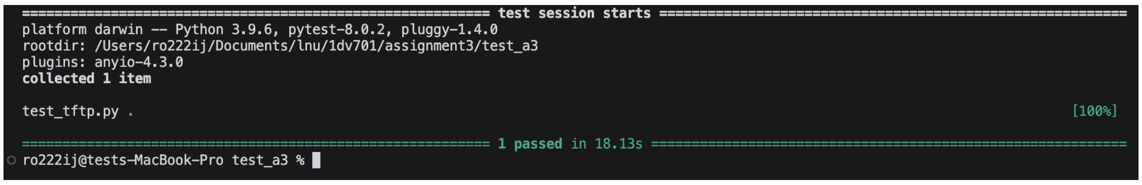
```
======================================================== test session starts =========================================================
platform darwin -- Python 3.9.6, pytest-8.0.2, pluggy-1.4.0
rootdir: /Users/ro222ij/Documents/lnu/1dv701/assignment3/test_a3
plugins: anyio-4.3.0
collected 5 items

test_tftp.py .....                                                                                                              [100%]

========================================================= 5 passed in 46.19s =========================================================
ro222ij@tests-MacBook-Pro test_a3 %
```

Results of python tests for timed out and retransmitted on read requests



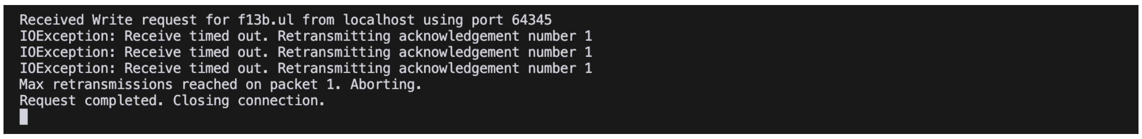
```
Received Read request for f3blks.bin from localhost using port 63947
IOException: Receive timed out. Retransmitting block number 1.
IOException: Receive timed out. Retransmitting block number 1.
IOException: Receive timed out. Retransmitting block number 1.
IOException: Max retransmissions reached for packet 1. Aborting.
SocketException: File reading failed.
Request failed. Closing connection.

Received Read request for f3blks.bin from localhost using port 50600
IOException: Received incorrect block number. Retransmitting block number 1.
IOException: Received incorrect block number. Retransmitting block number 1.
IOException: Received incorrect block number. Retransmitting block number 1.
IOException: Max retransmissions reached for packet 1. Aborting.
SocketException: File reading failed.
Request failed. Closing connection.

Received Read request for f3blks.bin from localhost using port 50342
IOException: Received incorrect block number. Retransmitting block number 1.
IOException: Received incorrect block number. Retransmitting block number 1.
Received last acknowledgment!
Request completed. Closing connection.

Received Read request for f3blks.bin from localhost using port 50934
IOException: Receive timed out. Retransmitting block number 1.
IOException: Receive timed out. Retransmitting block number 2.
IOException: Receive timed out. Retransmitting block number 3.
Received last acknowledgment!
Request completed. Closing connection.

Received Read request for f3blks.bin from localhost using port 57705
IOException: Receive timed out. Retransmitting block number 1.
IOException: Receive timed out. Retransmitting block number 1.
IOException: Receive timed out. Retransmitting block number 2.
IOException: Receive timed out. Retransmitting block number 2.
IOException: Receive timed out. Retransmitting block number 3.
IOException: Receive timed out. Retransmitting block number 3.
Received last acknowledgment!
Request completed. Closing connection.
```

Server log for retransmitted and timed out read requests



```
======================================================== test session starts =========================================================
platform darwin -- Python 3.9.6, pytest-8.0.2, pluggy-1.4.0
rootdir: /Users/ro222ij/Documents/lnu/1dv701/assignment3/test_a3
plugins: anyio-4.3.0
collected 1 item

test_tftp.py .                                                                                                                  [100%]

========================================================= 1 passed in 18.13s =========================================================
ro222ij@tests-MacBook-Pro test_a3 %
```

Results of python tests for timed out and retransmitted write requests



```
Received Write request for f13b.ul from localhost using port 64345
IOException: Receive timed out. Retransmitting acknowledgement number 1
IOException: Receive timed out. Retransmitting acknowledgement number 1
IOException: Receive timed out. Retransmitting acknowledgement number 1
Max retransmissions reached on packet 1. Aborting.
Request completed. Closing connection.
```

Server log for retransmitted and timed out read requests

2.1 Discussion

Our testing strategy for retransmissions and timeouts includes the following tests:

1. **Time out a read request (*test_timeoutMoreThanThreeTimes*):** This test verifies that the server correctly times out when a read request is not acknowledged within the expected time frame.
2. **Send more than 3 incorrect acknowledgements for a read request (*test_getFileWithWrongAckFourTimes*):** This test checks the server's ability to handle multiple incorrect acknowledgements. The server should time out after receiving more than three incorrect acknowledgements.
3. **Send 2 incorrect acknowledgements for a read request, then send all correct acknowledgements (*test_getFileWithWrongAckTwiceThenCorrect*):** This test ensures that the server can recover from initial incorrect acknowledgements and continue processing once correct acknowledgements are received.
4. **Send one incorrect acknowledgement for each incoming data packet for a read request (*test_GMBFail1stAck*):** This test checks the server's ability to handle an incorrect acknowledgement for each data packet. It tests the retransmission functionality at the packet level.
5. **Send two incorrect acknowledgements for each incoming data packet for a read request (*test_GMBFail2ndAck*):** Similar to the previous test, this test sends two incorrect acknowledgements for each data packet to further test the retransmission functionality.
6. **Time out a write request (*test_TimeOutOnFileWrite*):** This test verifies that the server correctly handles a write request that times out during the process.

Please note that the Unix TFTP client and the Windows TFTP client handle timeouts differently. The Unix client doesn't mind if the socket is still open when sending a package and times out as expected. However, the Windows client raises a ConnectionResetError in such a scenario. Our tests are designed to pass when either of these behaviors is observed.

In addition, we simulated timeouts in scenarios where the network is stable, but an issue arises on the client side. This scenario results in all retransmitted packets being received. Consequently, we incorporated a feature into the methods we developed that disregards the same retransmitted data packet once it has been processed.