Linnéuniversitetet
Kalmar Växjö

Assignment 3

TFTP Server

Course: 2DT905 (Computer Networks)
Student: William Abrahamsson
Contact: wa222dt@student.lnu.se

Problem 1

Change of skeleton structure

My TFTP (Trivial File Transfer Protocol) server only supports RRQ (Read requests). Therefore i removed the skeleton for “recieve_DATA_send_ACK(params)”. Since this function would be used to send data packets after the client sends a write request. I also divided the function: “send_DATA_recieve_ACK(params)” into two separate functions. One for sending the data and one for receiving the acknowledgement packet (ACK) from the client after a successful packet transfer.

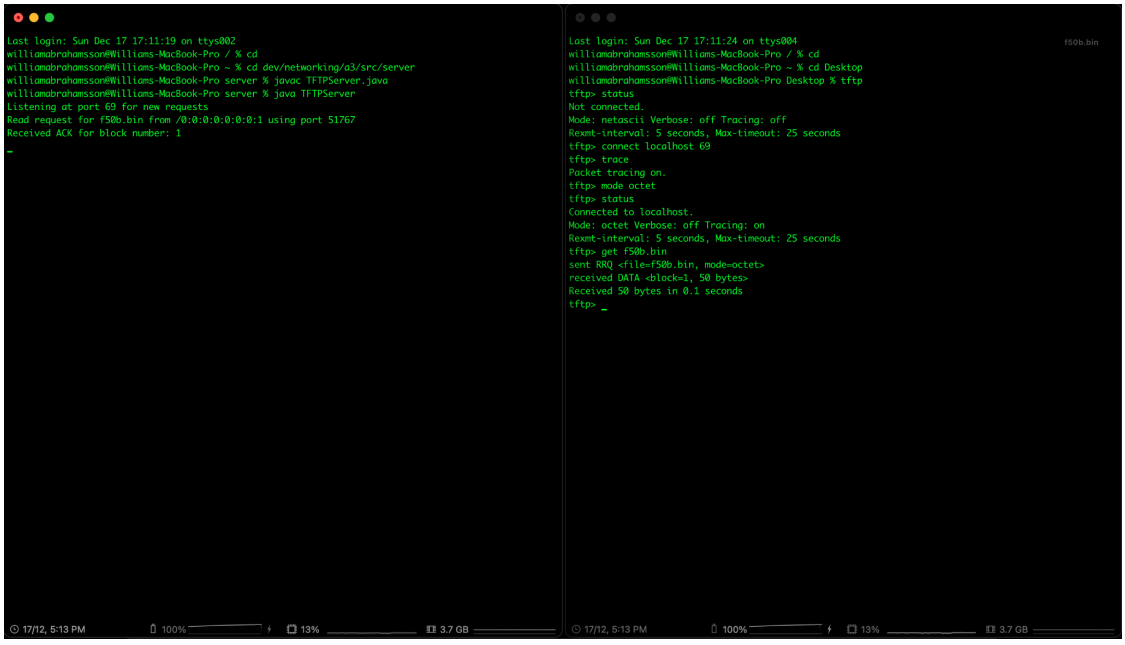
Serving Directory

For testing files i downloaded the python testing client and moved “genfiles.sh” into a directory in src called “files”. I then ran “./genfiles.sh” inside my files repository to generate the testing files. This folder is the one being served by the server. Since the server only supports RRQ requests i removed “WRITEDIR” and modified “READDIR” to “../files/”. This uses the dynamic path at the directory parallel to the server.

Code explanation

The start function contains a loop that listens for client requests. When a client sends a request. The server socket connects to the client and calls the handleRQ method. If the request has the same opcode as OP_RRQ (read request), a loop is started that sends data packets to the client with the size 512 bytes. If a file is larger than this, multiple packets is sent until the whole file is transferred. After each packet is sent, an ACK package is received according to protocol.

Read request of f50b.bin (size: 50 bytes)



Sockets
In the provided code skeleton, two separate datagram sockets, namely socket and sendSocket, serve distinct purposes in managing communication between the server and clients.

The primary socket: “socket” is utilized for receiving incoming packets from clients. It listens to a specific port, in my case: 69, and plays a crucial role in accepting requests from clients. By using this socket, the server is capable of listening for and receiving incoming data packets from clients who are attempting to communicate with the server.

The second socket: “sendSocket” is employed for sending data to clients in response to their requests. This socket is created for each client request within a separate thread. The utilization of separate threads for each client request is particularly significant as it enables the server to concurrently handle multiple requests without having to wait for the completion of one before addressing the next. This concurrent processing enhances the efficiency and responsiveness of the server, allowing it to efficiently manage multiple client interactions simultaneously.