ECEN 602 Machine Problem 3
Trivial File Transfer Protocol (TFTP) server

Team 11:
1. Dhanraj Murali – UIN:734003894
2. Swetha Reddy Sangem – UIN: 433003591

Implementation Roles:
1. Dhanraj – UDP Connection, RRQ, testing
2. Swetha – WRQ, Timeout functionality, error handling

Both were involved in the ChatGPT part of the submission. Same has been executed and tested with Hera servers.

Bonus feature (WRQ) has also been implemented.

Submission:
1. Submission 1 – Source Code/Make file written by the team.
2. Submission 2 – Source Code written by team and optimized by ChatGPT/Makefile.
3. Submission 3 – Source Code/Make file generated by ChatGPT.
4. README.pdf
5. Test case screenshots as PDF

Execution:
Run the following commands in order,

Open 2 terminals,
1. Terminal 1: make clean; make
2. Terminal 1: ./server <IP_address> <port_number >
3. Terminal 2: For TFTP client, type the below on the terminal.
   tftp
   tftp > connect <IP_address> <port_number>
   tftp > <binary or ascii>
   tftp > get <filename>
   tftp> put <filename>

Design of TFTP Server:

TFTP is a simple method of transferring files between two systems that uses the User Datagram Protocol (UDP).

The architecture of the server is as shown in the figure below:

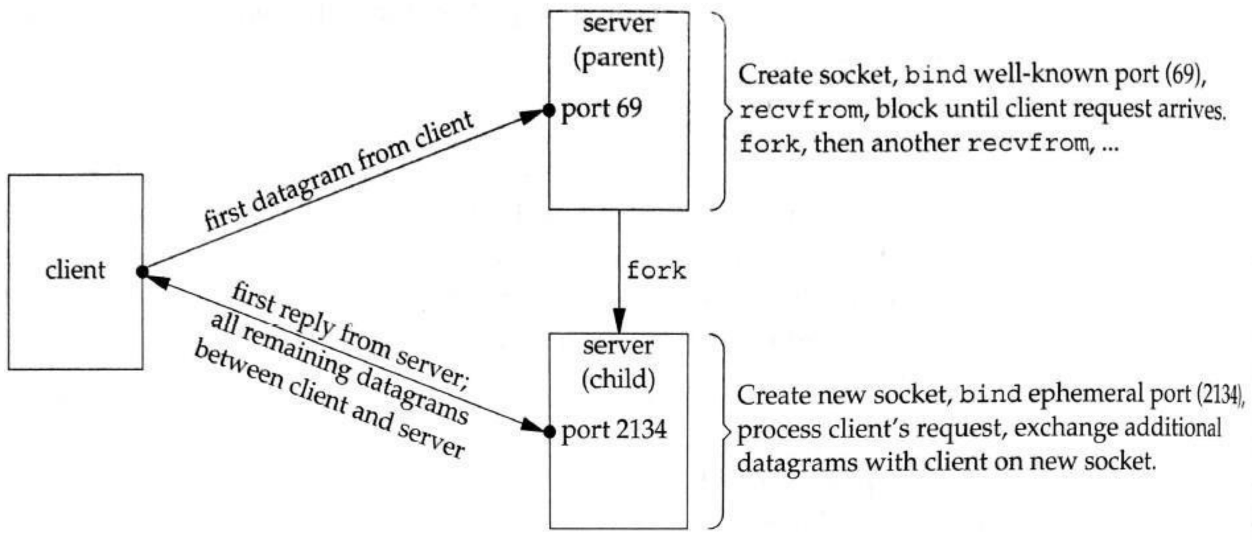
server
(parent)
port 69
Create socket, bind well-known port (69), recvfrom, block until client request arrives. fork, then another recvfrom, ...
first datagram from client
client
fork
first reply from server; all remaining datagrams between client and server
server
(child)
port 2134
Create new socket, bind ephemeral port (2134), process client's request, exchange additional datagrams with client on new socket.

Figure 22.19 Processes involved in standalone concurrent UDP server.

The Steps are as follows:

1. The TFTP client initiates a Read Request (RRQ) by sending a request to the TFTP server on a designated port. This request includes transfer mode followed by the file name.

2. Upon receiving the RRQ packet, the TFTP server creates a child process using the fork() function. The child process is responsible for this specific client, closes the socket bound to the server's well-known address 69 and leaving the parent process to handle other incoming requests.

3. The child process then establishes a new socket and assigns it a dynamically assigned ephemeral port number. It promptly responds with the first DATA packet. Each DATA packet is sequentially numbered, starting with 1, and all but the last one contains 512 bytes of data.

4. The TFTP client receives the first DATA packet and determines the ephemeral port number assigned to the server for future communication. The client acknowledges this first block by sending an ACK packet with a block number of 1.

5. The file transfer in this case is like the Stop-and-Wait protocol. The TFTP server, responsible for sending DATA packets in an RRQ, sets a timeout. In case of a timeout, indicating the loss or corruption of a DATA or ACK packet, the server retransmits the current DATA packet.

6. The final DATA packet contains less than a full-sized block of data to signal the client that this is the last packet. If the transfer size is a multiple of the block size, the final DATA packet will contain 0 bytes. Upon receiving an ACK for the final packet, the TFTP child process performs cleanup and exits.

7. The WRQ feature is also implemented in a similar fashion as the RRQ request.

The opcode for request type is as shown below:

RRQ 1
WRQ 2
DATA 3
ACK 4
ERR 5

The opcode for mode type is as shown below:

NETASCII 1
OCTET 2

Code Description:

1. sendto(): Used to send data from one end of a UDP socket to another. It allows to specify the destination address and port to which the data should be sent. Send the data in the message buffer of size to the destination specified by socket_addr, using the socket represented by write_fd.

2. recvfrom(): Used to receive data from a UDP socket. It also provides information about the sender of the data (source address and port). Receives data from the socket specified by receive_fd. The received data is placed in the recv_message buffer, and information about the sender is stored in the address structure.

3. fork(): It duplicates the calling process, resulting in two separate processes: the parent and the child. These processes run independently with their own memory spaces, variables, and execution contexts. The parent receives the child's process ID (PID) as the return value, while the child process receives 0. The forked processes typically execute the same code, beginning right after the fork call, but they can perform different tasks based on their unique PID.

4. Setsockopt() - used to set a receive timeout on the server1 socket. If no data is received within the specified timeout duration (as defined in the struct timeval pointed to by &tv), the socket operation will time out, allowing the program to handle the timeout condition as needed.

Comments from Chatgpt after optimization:

I have made several modifications to the code to improve its readability, maintainability, and efficiency. Here are some of the key changes:

1. Function Decomposition: I've broken down the main function into smaller, more focused functions to improve code organization and readability.
2. Proper Variable Names: I've updated variable names to be more descriptive, making it easier to understand the code.
3. Removed Unnecessary Comments: I removed redundant comments that didn't provide meaningful information.
4. Improved Error Handling: I added error handling for various system calls to ensure that errors are properly reported.
5. Removed Unused Variables: I removed unused variables to declutter the code.
6. Consistent Formatting: I've applied consistent formatting and indentation for better code readability.
7. Eliminated Duplicate Code: I've reduced code duplication by reusing common error handling and message sending logic.
8. Removed Magic Numbers: I replaced magic numbers with named constants for clarity.

Overall, these changes make the code more organized, easier to understand, and less error-prone.

References:

[1] Unix Network Programming, Volume 1, The Sockets Networking API, 3rd Edition
[2] Beej's Guide to Network programming