README

ECEN 602 Network Programming Assignment #3

TRIVIAL FILE TRANSFER PROTOCOL (TFTP) SERVER

Team No: 11

Team Members:

1) Srividhya Balaji : UIN : 827007169
2) Sanjana Srinivasan : UIN : 927008860

Individual Contributions:

1) Srividhya did the UDP connection sequence, file transfer handling for text files, multi-client implementation, testing.
2) Sanjana did the Binary file transmission, Time Out implementation, Error handling.

The package contains the 4 files:

1) tftp_server.c
2) Makefile
3) README.txt
4) Test Cases.pdf

To compile the code, run the Makefile : make
Run the server code using the command line : ./tftp_server server_ip server_port
Run the TFTP client using the command line : tftp
To clean the executables, run : make clean

IMPLEMENTATION OVERVIEW : ARCHITECTURE

A TFTP server capable of handling multiple simultaneous clients is implemented. TFTP is implemented over UDP, which is a connectionless protocol. While the TFTP can transfer files in both directions, the server here only implements reading of files using Read Request function (RRQ).

In order to handle multiple simultaneous clients the server calls the fork() function. And each child process will create a new ephemeral port in order to connect with a client. The TFTP server initially receives an RRQ request from the client containing the filename and the mode of transfer. The contents of the file are read by the server and transferred to the client as DATA packets of size 512bytes. The client then replies by sending an ACK for each of the DATA received.

A timer is set at the sender when a block is first sent, and retransmits the data block on timeout. RRQ implementation makes the server the sender and client the receiver thus timer is implemented at the server. Timeout occurs if the ACK from the client is lost during transmission. In case the server receives a duplicate ACK, it doesn't retransmit the DATA packet.

TFTP supports two modes of transfer: netascii and octet. The netascii mode is used for transferring text files while the octet format is used for transferring binary files. This TFTP server is implemented using block number wrap-around.

TFTP SERVER:

The server was implemented using C. It is started on the user input IP address and Port number permanently to listen for new requests from the clients. Whenever a the server receives a new RRQ request, a new child process is created using fork().

The RRQ request packet sent by the client contains the filename and transfer mode. The file requested is read by the server. The file is then sent from the server to the client in blocks of 512bytes. Each block of DATA is numbered and server waits to get an ACK for each DATA sent before sending the next block. The last data block is indicated by its size being less than 512bytes.

The following socket functions were used,

1) socket() :

Syntax : **int socket(int** *domain*, **int** *type*, **int** *protocol*);
Parameters based on defined architecture , parameter 1: domain. This can be either AF_INET or AF_INET6. Since we are using UDP the type is defined as SOCK_DGRAM.

2) Bind() :

Syntax: **int bind(int** *sockfd*, **const struct sockaddr ***my_addr ", socklen_t " addrlen)

Bind() is used to bind the socket descriptor. Its parameters include, the socket descriptor which we want to bind, the address structure object and the length of the address Our code uses the sockaddr_in structure to define the socket parameters. The server address from this structure is type casted to struct sockaddr type in order to bind the socket address.

3) recvfrom() :

Syntax : **ssize_t recvfrom(int** sockfd, **void** *buff, **size_t** len, **int** flags, **struct sockaddr ***src_addr, **socklen_t ***addrlen);

This function is used to receive data from a socket irrespective of the the link being connection oriented like TCP or connection less like the UDP protocol we implement here in TFTP.

4) fork() :

This creates a new process called the child process with its own unique process ID. This function is called every time a new client sends a RRQ request packet to the server. Since TFTP is implemented using User Datagram Protocol, every time a new child process is created, an ephermeral port is used to establish connection with the client.

5) sendto() :

Syntax : **ssize_t sendto**(**int** sockfd, **const void** *buf, **size_t** len, **int** flags, **const struct sockaddr** *dest_addr, **socklen_t** addrlen);

This function will send a message through a connection-mode or connectionless-mode socket. If the socket is connectionless-mode like the TFTP implementation, then the message will be sent to the address specified by the dest_addr.

6) Readable_timeout() :

This function is used to implement timer at the sender ie TFTP server. The server after sending out a DATA packet sets a timeout timer and waits for the ACK from the client. If the ACK arrives before the timer expires then it is cancelled and the next packet is processed. If the timeout timer expires, which is the case that the ACK is lost then the last DATA packet is retransmitted.

This function makes use of select() in order to implement the timer.

Opcode for the packets using TFTP:

01 = RRQ
02 = WRQ
03 = DATA
04 = ACK
05 = ERROR

IMPLEMENTATION DETAILS :

The server was implemented in C. The main function includes establishing the UDP connection, and listening through a specific port. Once an RRQ request is received from the client, using fork(), we process the child process and wait for the next connection. We transfer the file in blocks of 512 bytes. Opcodes are used to identify the type of packet received. Once the file request is processed, the child is disconnected. Various testcases including multi client scenarios were tested.