Taylor Johnston

CSS 432

Boxiang Pan

12/17/2021

# Final Report

# Code

## Server

The server is separated into two parts. The first part is tftp\_server which acts as the main driver, and includes a receive loop and a thread updating the state of all transactions and also maintains a list of all . The second part is tftp, which includes all of the classes and methods used to define and update transactions. The server is expected to be running at "10.158.82.39" (csslab9.uwb.edu).

Diagram

Description automatically generated

## Client

The client is entirely in tftp\_client. This is separated into two parts, read requests and write requests. Read requests send an ACK after receiving DATA, or resend previous ACK after waiting TIMEOUT seconds without receiving any data. The transaction ends when the client read request receives a data packet with less than 512 bytes of data, a write request sends less than 512 bytes of data, an error packet, or retrying consecutive 10 times.

The client will expect the server to be at ip address "10.158.82.39" (csslab9.uwb.edu).

# Completion

I believe I’ve completed all steps for this assignment, implementing both the TFTP protocol and the steps defined in the assignment. The tftp application can send and write large files, specifying the action and port # through the command line, handles timeouts and retries, errors, and the server can handle multiple connections simultaneously using a combination of a queue and p\_thread.

Server will not accept read requests to files that don’t exists or write requests to files that do.

I’ve tested RRQ and WRQ while the server was down and closing the server or client in the middle of a RRQ and WRQ.

I considered cases where during a read or write request, on either the client or server a data or ack packet is sent out of order. If a data packet is received out of order, the last ack should be resent, while this is not the case for receiving an out of order data packet.

I confirmed that the server and client work when sending and receiving both very small and very large files and both terminate properly after completing the transfer.

# Output

Server write retrying and exiting after losing connection to client:

Text

Description automatically generated

Client read retrying and exiting after losing connection to server:

Text

Description automatically generated

Client write retrying and exiting after losing connection to server:

Graphical user interface, text

Description automatically generated

Successful read:

Graphical user interface, text

Description automatically generated

Successful write:

Graphical user interface, text

Description automatically generated

Two simultaneous writes:

A screenshot of a computer

Description automatically generated with medium confidence

Successful simultaneous read & write:

Graphical user interface, text, chat or text message

Description automatically generated