ARIZONA STATE UNIVERSITY

SLN 70871 – Computer Networks

Project: Socket Programming - Part 2

Part 2 of this project is an exercise to create a simple FTP application.

Section I: Goal

Design a File Transfer Protocol (FTP) application for communication over a computer network by means of TCP protocol, with client-server network architecture using socket programming. You can build over your code of project part 1, by making necessary changes and additions as described in the requirements of this part of the project.

Section II: File Transfer Protocol

File Transfer Protocol (FTP) is a standard internet protocol, used to transfer files between two hosts. Clients initiate communication with servers. Using FTP, a client can read, write, move, copy, rename and delete a file on a server. In this project we will create a simple FTP application where a client can perform a read/write from/to a file at the server.

Types of Server

A stateless server does not maintain any information about client requests (e.g., the Network File System (NFS)). The client requests service by sending a message to the server. The server executes the request and sends back a response. It's possible that the request or the response might get lost. Both of these cases look the same to the client — it doesn't get an answer. If the client does not receive an answer within a timeout period, it re-issues the request. If the requests to the server are idempotent, meaning that the request can be executed any number of times with the same effect, then re-executing the request doesn't cause a problem.

However, not all services can be made idempotent. Some servers must remember the state of the client and are therefore called *stateful*. For example, in a lock server, executing a request twice can have very bad results. Suppose a client requests a lock on file F. The server grants the lock, but the response is lost. The client times out and re-issues the request. The server responds with a message saying that the file is currently locked. Now, the file is permanently blocked because the client will never send the unlock command.

Section III: Socket Programming Project Part 2 Details:

Step 1. Write the following two programs using C/C++ on Linux.

a. Create a concurrent server program using TCP.
b. Create a client program using TCP

Step 2. Client and Server requirements (5%)

a. Server program should be able to accept port number from command line argument. If a port number is not provided then your program should display an error message and exit.
   server p

b. Client program should be able to accept hostname **h** (string), client number **c** (integer) and port number **p** (integer) from command line arguments. If all the above said arguments are not provided then your program should display an error message and exit.

   client h c p

   The client(s) and the server communicate using port **p**, hence the clients and server must be started with the same port number. Each client must have a unique client number.

   Assumptions:
   1. We are assuming, for this project, that if a client has contacted the server once, it will not contact the server again. Therefore, you can keep a track of all client numbers in your list and deny connections to each duplicate client number.
   2. We are assuming that the order of command line arguments is correctly entered by the application user. Therefore, your program does not require to check for it.

<u>Step 3.</u> FTP application requirements (5% + 5% + 25% + 20%)

a. Implement the server, such that it can handle multiple connections using either fork() system call or threads. *A maximum of 5 clients.*

b. Further, each connection between the client(s) and the server can have a series of requests.
   <u>How to implement:</u> For each connection, server will ask the client if the client wants to go ahead with another request, expecting a reply of **`Y`** or **`N`** for yes and no, respectively. Thereafter the server will act in accordance: If client replies **`Y`**, server will expect another request coming its way. If the client replies **`N`**, server will close the socket for that client.

c. FTP Client
   Client will send a request to server for reading/writing to a file. This request will consist of a <filename> and <mode_of_operation>. Mode of operation can be "r" for reading a file from the server, and "w" for writing a file to the server. Each client request will be of the following form:
   `<filename>, <mode>`
   Example: `file1.txt, r`
   1. If client is reading a file from the server, it should recreate the file received from the server, line by line. Finally, closes the file.
   2. If client is writing to a file to a server, it should read from a file existing at its client end write the file line by line at the server, and finally close the file.

   FTP Server
   The server should receive the name of the file from the client. If the file cannot be opened in the current directory, then the server should return the string "File not found." to the client.
   1. If the file can be opened and a read is requested, then the server should send each line of the file to the client until the end of the file is reached. After sending the complete file to the client, server closes the file.
   2. If the client has requested write to a file. A server opens a new file, receives each line of file from client and writes it to the file at its server end, and finally closes the file.

<u>*How to check that your application has transferred the file correctly:*</u> you can use "diff" command on the original file and the one that has been received through the FTP transfer application.

d. When a client is reading/writing a file to/from server, that file cannot be accessed by any other client. Therefore, it is said that the server will put a lock on the file.
   1. When a client is reading a file from the server, server will put a read lock on that file. This will allow any other client to access the same file for reading only (not for writing).
   2. When a client is writing to a file on the server, server will put a write lock on the file. This will not allow any other client to access the same file in any mode.

   <u>*How to implement:*</u> assign a lock in the database, i.e., table/vector/array/data structure to record the <filename> which is currently being written to client and <lock> to restrict the usage of this file, depending on the mode, by any other client. Once, the file transfer is complete, your program will need to release the lock from the given file.

<u>Step 4.</u> Good Programming Practices (20% + 10%)

a. Describe in the comments sections above your server source code in your solution file: What are the design specifications (data structures) used for creating the table/record of file accessed? Any other design specifications?

b. Your program should be clean and organized and should have a good documentation.

<u>Submission</u> (10% for correct submission)

I. When your project is complete (all four steps, as mentioned above), create a bzipped tarball which will contain all your source code files and header files, if any, and a Makefile to compile your program. This Makefile must work on general.asu.edu.

   Name you bzipped tarball as ***cse434-f15-p02-lastname.tar.bz2***,
   If you worked in a group of *two* then name your PDF file **cse434-f15-p02-*lastname1-lastname2.tar.bz2*** where *lastname1* and *lastname2* are the lastnames of both partners.

   If you are working in a group of *two*, only ***one*** submission is required with both of your names on it; you will each earn the same number of points. Your project solution document must be uploaded to Blackboard by the assignment deadline.

II. At the top of each source code file, add the following information:
   // Name of Author(s): *Complete name of author(s)*
   // Course Number and Name: **CSE 434, Computer Networks**
   // Semester: **Fall 2015**
   // Project Part: 2
   // Time Spent: ?? hours ?? minutes

Submission Deadline: Upload the PDF file to Blackboard using the homework assignment submission link by the deadline which is **10:00 am Sunday September 27th 2015**.