CMSC 257 Assignment 5: Simple FTP with forked server and threaded client

Project due date: 11:59 pm EST, 12/08/17

You MUST use the client/server code from Lab-9 for this assignment. You need to build on top of that code.

Client-Server Set-up

- The goal of this project is to create two programs: a client and a server. The server will operate in a forked mode in order to transfer files to the client that works in a threaded mode over TCP sockets.
- For each connection from the client, the server will read a single file name from the client and then send the file back to the client. The client should first send a request to the server of the form: get <filename>
  The server parses this command, identifies the filename and then starts the file transfer. After completion of file download, the client needs to close the corresponding socket. For multiple file downloads, create a different socket for each file transfer.
- The client needs to take in command line parameters for (i) the server IP address, (ii) the server PORT, (iii) the number of files to be downloaded; each of these downloads are handled in a different thread, and (iv) the sleep time between downloads. Thus a sample execution of the client is as follows:
  ./client 127.0.0.1 3000 10 2
- The server needs to be invoked by typing in: ./server PORT

Code set-up

- Create two directories named “Server_dir” and “Client_dir” under the main assignment directory “Assignment_5” and store the server program named “server.c” and few files that the client might ask for from the server like “file001.txt”, “file002.txt”, etc. The client program “client.c” is saved in the directory “Client_dir”. In order to run the implementation the “server.c” is executed first and then in another terminal window the “client.c” is.

- As both the server program “server.c” and client program “client.c” will be executed on the same machine, assign the IP address as “127.0.0.1” which is a default IP for the current machine.

- Use PORTs 2000-3000 for setting up your sockets.

Requirements:

- Each read/write into sockets needs to use a buffer of 50 characters. Note that the file-name to be transferred should fit the 50 characters requirement as well.
- The server.c should have an infinite loop to cater to multiple client requests.
- The server should use file handling to open the file and start sending chunks of bytes to the client.
- The client.c code needs to use an infinite while loop to keep reading in the contents of the file. As the file is not present, it needs to create a file in “w” mode and write components to it as and when it receives chunks from the server.

- Use a special string: “cmsc257” as a terminal string; the server sends this string at the very end to tell the client that the file has been transferred. The client, on receiving the string, needs to *exit gracefully*.
- Use a signal handler to “gracefully” terminate the server on Ctrl-C; existing client connections should be terminated (by stopping the data transfer or waiting for the transfer to complete) before the server exits.
- Use fork on the server to allow multiple simultaneous file downloads from different clients; you may want to specify the maximum number of client connections that can work in parallel. Note: the forked server.c from Lab-9 is inefficient as we had the wait(NULL) in the parent code itself. This means that the server has to wait for the first child process to complete before it can go back to the while loop and accept the next client. This problem can be overcome by using another signal handler to catch the SIGCHLD signals; this signal is sent by the child process when it completes. In the signal handler function, you can simply put in the waitpid(-1, NULL, WNOHANG) instead of the wait(NULL) in the parent code.
- The client needs to be multi-threaded (use pthreads). Say, for 10 file downloads, the client randomly chooses which 10 files to download out of a total of 25 such files (file001.txt,…,file025.txt). Then it spawns 10 different threads with each thread in charge of an individual file download. The sleep time makes the client call sleep(#secs) as specified by the user between any two file downloads.
- In the server directory, create 25 files of the right nomenclature each of 1MB in size. You can modify the following shell script for this:

```
#! /bin/bash
for n in {1..5}; do
   base64 /dev/urandom | head -c 1000000 > file$( printf %03d "$n" ).txt
   done
```

- Record the total execution time of the client to download the specified number of files. Run tests for different number of files and different sleep times between downloads and create a report.
- Submit a tar file as before with your client/server directories and corresponding files within along with the report.

Note: Late assignments will lose 5 points per day upto a maximum of 3 days. Code must be submitted in the prescribed format.
