OS 2014 – EX5

Reliable File Transfer

Due Date: 17/6/2014

Assignment Description

In this exercise you will implement reliable file transfer between a client and a server. The protocol should use TCP to transmit the file from a client to the server. You will create two programs, one that represents the server (srftp.c or srftp.cpp), and the other representing a client (clftp.c or clftp.cpp).

The Server

The command line for running the server is:

srftp server-port max-file-size

Notes:

- The server receives a port number, and listens to this port. It waits for clients to connect to this port (using TCP connection) and then it starts to serve these clients.
- The server also receives the maximum supported file size. The server won't accept a transmission of a file which is bigger than *max-file-size* (*max-file-size* is supported, *max-file-size*+1 is not). The client should not send the file contents if the file won't be accepted by the server.
- Once it is up and running, the server accepts requests to store files from clients. The client must supply the file size, the name of the file, and the file's contents. The server saves the file locally (in the working directory) with the given name. You should create a simple transmission protocol. Using this protocol, the client will know if its file's size is supported, and only then it will send the file. Moreover, the server will be able to separate between the name of the file and the file's content.
- If the file already exists, the server will overwrite it. You may assume that the server won't be requested to save two files with the same name *simultaneously*.

- The server is multi-threaded. It handles multiple clients by creating a new thread for each client. The server should create exactly one thread for each client, and therefore it will have clients_number+1 threads exactly.
- The server doesn't finish naturally (it can be killed, e.g. by "Crl + c" in the shell). Open it is killed, there might be open resources (open sockets, files, allocated memory, and more). That's fine. However, during its normal activity, it should not have memory leaks. This means that when a connection with a client is terminated, the server should release all the relevant resources.
- The maximal number of pending connection is 5 (the *backlog* parameter in the *listen* function).

The Client

The command line for running the client is:

clftp server-port server-hostname file-to-transfer filename-in-server

Parameters' description:

server-port	The port used by the server.
server-hostname	The host name of the server.
file-to-transfer	The path of the file to transfer from the client to the server. This can be a relative or absolute path of a file located on the client's computer. The client will transmit the content of this file to the server. The server will store this file's contents in its working directory.
filename-in-server	The file name that the server will use to store the file. The client should send the file's name to the server in addition to the file's contents.

Notes:

- The flow of the client is simple. First, it opens the *file-to-transmit.* Then, it transmits its content and the *filename-in-server* to the server.
- As mentioned above, you should create a simple transmission protocol that transmits the file's content only if its size is supported by the server, and enables the server to separate the file's name and the file's content.

If the requested file is too big and not supported by the server, the client won't send the file's content. Instead, the client will print to the standard output the following message:
"Transmission failed: too big file".
An attempt to send too big file shouldn't cause to any errors in the client nor in the server. The client must release all the allocated resources and exit successfully with no memory leaks (e.g., return 0 from the main).

- When finished the client program terminates. In case of success the client program should return 0.

Error Handling

- There might be a few usage problems that you are requested to check, including:
  - Wrong number of parameters.
  - Invalid port number. The port number is a number between 1 to 65535.
  - Invalid *max-file-size*. The max-file-size should be a non-negative integer.
  - The *file-to-transfer* is a directory or it doesn't exist

  You are expected to check exactly these problems. For example, if the *file-in-server* is an invalid name, we expect the "open" method in the server to fail, and not to print a usage message.
  In case of a usage error you should print (to the standard output) the following message:
  - Usage: clftp server-port server-hostname file-to-transfer filename-in-server
  - Usage: srftp server-port max-file-size

- In case of a system call or library error (e.g., if the program can't open a socket), the program will do the following:
  1. Print the following error message via STDERR:
     "Error: function:SYSTEM_CALL errno:ERRNO.\n", where SYSTEM_CALL is the name of the system call / library function that failed, and ERRNO is the errno number (which indicates the failure reason).
  2. Use pthread_exit to terminate the thread in the server, or exit in the client. Pay attention, this means that a client will be killed if an error occurred. However, the server will be terminated only if it has a failure in the main function. If an error occurred in a single client's thread (e.g. if *filename-in-server* is invalid), only this thread exits and the rest of the servers' activity keeps without change.

  Pay attention – you are not required to deallocate all the resources in case of an error!

Testing Your Program:

- ✓ To verify that the file was transferred properly and has the same data exactly, use "diff" command.
- ✓ Basic text: Run a server and a single client on one machine.
- ✓ Try to transfer both ASCII files (text files) and binary files (program files / images etc.).
- ✓ Also, try different files size (in particular, files that require multiple send / recv and those that require a single one, and files that will be accepted by the server and file that will not).
- ✓ Try to transfer files to a single server from multiple clients simultaneously (this should be done by a script).
- ✓ Check that if the server receives a file with invalid name, other clients still can transmit files to the server.
- ✓ Think how to create test cases where multiple threads must work in parallel in the server.

Useful system calls

socket, bind, connect, listen, accept, send, recv, close, inet_addr, htons, ntohs, gethostbyname

Theoretical part (20 points)

This section is worth 20 points, and should be answered in detail in the README file.

- **Reliable file transfer protocol over UDP connection (10 points).**
  In this EX, you are required to implement a file transfer protocol over TCP connection. The question is how it could be done over **UDP connection**. You shall develop an **efficient** and **reliable** file transfer protocol over UDP connection. You should take into account the problems that arise by using UDP, and solve them in your suggested protocol.
  You are required to describe such protocol, explain how it solves each problem and why it is efficient. Finally, specify what are the main differences in the implementation of such a protocol comparing to your implementation (FTP over TCP), focusing on the server side.

- **Performance evaluation of your program (10 points).**
  In order to evaluate the performance of your file transfer protocol, you shall measure the time spent from the establishment of the connection and until its termination. A possible way to measure this elapsed time is by using gettimeofday(), that you are already familiar with. You are requested to create a graph that summarizes the performance of your program.
    - The X-axis of the graph is the size of the file, and the Y-axis is the time spent during the transmission of the file. You are requested to evaluate the performance of at least 5 different sizes. To receive a reliable evaluation, like always, we suggest you to do it multiple times, and average the results.
    - The server and the client should be located on different computers.
    - The graph's name is "performance.jpg".
    - You can create the graph with any program you wish (Matlab, Excel, Word, Python...). However, pay attention to have a very clear and informative graph, including a title, x-label and y-label.
    - Any detail that is not supplied here (e.g. the sizes, the files' types, etc.) is your choice.

  Finally, you are requested to summarize the experiment and analyze the results in your README file.

Submission

1. The files srftp.c \ srftp.cpp and clftp.c \ clftp.cpp. of course you are allowed to create (and submit) more files if you want.
2. A Makefile which compiles the executables. That means that a simple "make" command creates two executables: clftp and srftp. More requirements of the Makefile appear in the course guidelines.
3. performance.jpg.
4. A README

Good luck!!