Introduction to Network Programming

2016 Fall Mid-Term Exam

Closed Network, Closed USB disk, and Open Textbook

1) 40% (a) 20% Write an iterative server that functions like a cloud drive for each client. Each client should have its own and independent cloud drive on the server. Each time when a client connects to the cloud drive server, it should "see" an empty cloud drive.
   - ✓ When the server receives a command line sent from the client, it should perform the specified operation. The format of the command line is "Command Filename1 Filename2 \n", where Command can be "GET", "PUT", "LIST" or "EXIT" and Filename1 and Filename2 are the names of the specified files for the "GET" and "PUT" commands. For the "LIST" and "EXIT" commands, there is no need to provide Filename1 and Filename2. For example, "PUT test1.txt test2.txt", "GET test2.txt test3.txt", "LIST", and "EXIT" are all valid examples.
   - ✓ The "PUT test1.txt test2.txt" command means that the client wants to upload a file named "test1.txt" to its cloud drive and rename it as "test2.txt" in its cloud drive. The "GET test2.txt test3.txt" command means that the client wants to download a file named "test2.txt" from its cloud drive and rename the received file as "test3.txt" in its local disk. The "LIST" command means that the client wants to see a filename list of all the files that currently reside in its cloud drive. Notice that this list must be generated by and returned from the server and must not be generated by the client itself. The "EXIT" command means that the client wants to close its connection to the cloud drive server.

   (b) 20% Write a client that does the following functions:
   - ✓ Take input from the keyboard, one line at a time.
   - ✓ For each input line, the user will input a command line. The formats of all allowed commands are already explained above. After receiving a command line from the user, the client should send it to the cloud drive server.
   - ✓ After sending a command line to the server, the server and the client should collaboratively perform the specified operations on their respective sides based on the command.
   - ✓ For example, after the client sends the "PUT test1.txt test2.txt" command to the server, it should start sending the content of the test1.txt file to the

server. The server should start receiving the content of the test1.txt file and store the received content in a file named “test2.txt” in its local disk. When the file transfer is done, the client should print “PUT test1.txt test2.txt succeeded” on its screen. If the command is “GET test2.txt test3.txt”, when the file transfer is done, the client should print “GET test2.txt test3.txt succeeded” on its screen. If the command is “LIST”, then after printing a list of the name of the files in the client’s cloud drive (the name of each file should be shown on a different line), the client should print “LIST succeeded” on its screen. If the command is “EXIT”, the client should close the connection without printing anything on the screen.

(c) The following shows a possible input/output on the screen of the client:

PUT hello1.txt hello2.txt (typed in from keyboard)
PUT hello1.txt hello2.txt succeeded
PUT world.doc world.doc (typed in from keyboard)
PUT world.doc world.doc succeeded
LIST (typed in from keyboard)
hello2.txt
world.doc
LIST succeeded
GET hello2.txt hello3.txt (typed in from keyboard)
GET hello2.txt hello3.txt succeeded
EXIT (typed in from keyboard)

2) 20% The requirements for this problem are the same as those for Problem 1. You can reuse the client program developed for Problem 1. However, in this problem, you should use the fork() system call and the signal(SIGCHLD,) system call to write a concurrent server to replace the iterative server that you developed for Problem 1. When a forked child server detects that the client that it is servicing has closed the connection, it should print the message “Client has closed the connection.” to the screen and then call exit() to terminate itself.

An important requirement is that when many clients are connected to the cloud drive server, each client should have its own independent cloud drive. That is, it can only “see” the files in its cloud drive when using the “LIST” command and should not “see” files belonging to other clients. As stated above, the filename list must be generated by and returned from the server and must not be generated by the client itself.

3) 20% The requirements for this problem are the same as those for problem 1. You can reuse the client program developed for Problem 1. However, in this problem, you should use the select() system call to write a server to service multiple clients at the same time. While servicing its clients, the server should be able to accept new client making connections to it.

As in Problem 2, an important requirement here is that when many clients are connected to the cloud drive server, each client should have its own independent cloud drive. That is, it can only "see" the files in its cloud drive when using the "LIST" command and should not "see" files belonging to other clients. As stated above, the filename list must be generated by and returned from the server and must not be generated by the client itself.

4) 20% In the above 3 problems, the cloud drive for each client is empty when a client connects to the cloud drive server. That is, each time when a client connects to its cloud drive, it sees an empty cloud drive. Besides, after it closes its connection to the cloud drive server, all of the files that it had uploaded to the cloud drive while it was connected to the server will be lost.

In contrast, in this problem, you should redesign the client and server so that each time when a client connects to the cloud drive server, it can see the files that currently reside in its cloud drive. That is, the files that were uploaded by the client while it was connected to the server should reside in the client's cloud drive after the client closes its connection to the cloud drive server.

To support this function, after the client connects to the server, the first command that it should send to the server must be "CLIENTID: XXXXX", where XXXXX is a five-digit number. For example, "CLIENTID: 10001" is a valid example. With this information sent from a client to the server, you should modify the client and server programs that you have developed for problem 3 to support this function.

We define a session as follows: A client user makes a connection to the cloud drive server, sends its CLIENTID to the server, performs file operations, and then closes the connection. For a client user, it may do sessions at different time and in each of its session the client user will send its same CLIENTID to the server so that the server knows that these different connections are all initiated from the

same client user.

Notice that you can assume that the cloud drive server will run continuously and no one will shut it down unless it crashes by itself.

The following shows a possible input/output on the screen of the client with a CLIENTID of 20000:

（In the first session）

CLIENTID: 20000 (typed in from keyboard)

LIST (typed in from keyboard)

LIST succeeded (Note that the cloud drive is empty at the beginning of the first session. Therefore, there is no filenames shown above.)

PUT hello1.txt hello2.txt (typed in from keyboard)

PUT hello1.txt hello2.txt succeeded

LIST (typed in from keyboard)

hello2.txt

LIST succeeded

EXIT (typed in from keyboard)

（In the second session）

CLIENTID: 20000 (typed in from keyboard)

PUT world.doc world.doc (typed in from keyboard)

PUT world.doc world.doc succeeded

LIST (typed in from keyboard)

hello2.txt

world.doc

LIST succeeded

EXIT (typed in from keyboard)