CSE5306, Distributed Systems
Fall 2018, Project 1
Due date: 11:59pm Sep. 29, submission through Blackboard

Please read this:
Two students form a team and turn in one submission.
Total points possible: 100 pts.
Please add the following statement in the beginning of your submission.
I have neither given or received unauthorized assistance on this work
Signed: Date:

Introduction

In this programming project, you will implement a simple file upload and download service based on message-oriented client-server communication and a computation service using remote procedure call (RPC) based communication.

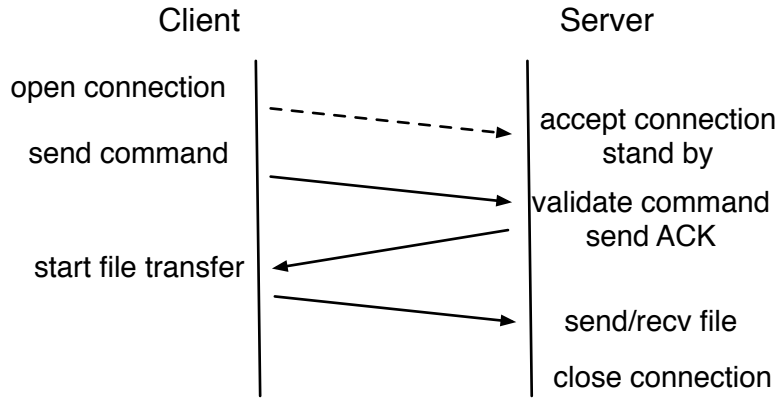
The file server supports four basic operations: UPLOAD, DOWNLOAD, DELETE, and RENAME. We assume that the file service is implemented using a connection-oriented protocol, in which the client and server first establish a network connection, negotiate the operation to be performed, and carry out the file transfer through the same connection. To simplify the design, you can assume that file operations are atomic and the server does not need to support interruptions to a file transfer.

The computation server provides a set of predefined RPCs that can be called from a client. The RPCs support a variety of computations on different types of data structures, including scalar variables, arrays, and matrices.

You can use **any** programming language to implement the servers, though some approach is easier to implement using a specific language.

Message-oriented client-server communication

Use the following protocol to define message-oriented communications.



To establish client-server connections using sockets, the server side creates a socket and binds it to port number 8080 (c program):

```
sockfd = socket(AF_INET, SOCK_STREAM, 0);
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
serv_addr.sin_port = htons(8080);
bind(sockfd, (struct sockaddr*)&serv_addr, sizeof(serv_addr));
```

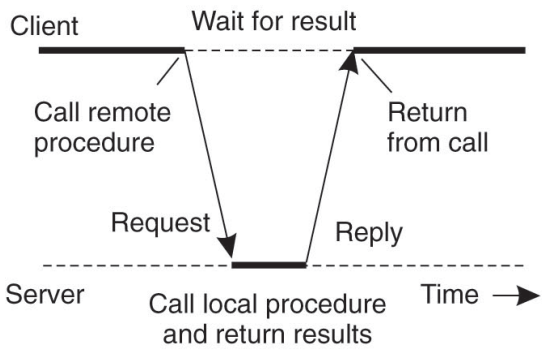
The server then listens to the socket and calls function accept(). The server in put to sleep until an incoming client request establishes a connection with the socket. The accept function wakes the server up and returns a socket representing the established client-server connection.

```
fd = accept(sockfd, (struct sockaddr*)NULL, NULL);
```

Assignment-1 (30pts)
Implement a basic single-threaded file server that supports the four operations listed above. You will use the message-oriented communication protocol. You can assume that the client and the server reside on the same machine but communicate with each other using different ports. Use different folders to hold files downloaded to the client or uploaded to the server.

Assignment-2 (30pts)
Based on the single-threaded server, implement a multi-threaded file server. The client side software does not need any changes but the server should be able to support multiple concurrent operations. Note that you need to consider locking in your multi-threaded server. If a file is being uploaded or renamed, it should not be downloaded or renamed until the previous operation is completed. In contrast, multiple clients can simultaneously download the same file. Use multiple clients to test your server.

Remote procedure call (RPC) based communication
Assignment-3 (40pts)
Use the following design of the synchronous RPCs to implement a computation server. The server supports four RPCs: `calculate_pi()`, `add(i, j)`, `sort(arrayA)`, `matrix_multiply(matrixA, matrixB, matrixC)`. The RPCs represent different ways to pass the parameters to the server. You need to implement a client stub to pack parameters into a message sent to the server and a server stub to unpack the parameters. You are NOT allowed to use Java remote method invocation (RMI) or RPC in other programming languages to implement the RPC-like communication.



(a)

Bonus assignment (20pts)

The file transfer between the client and the server can be made transparent to users and automatically handled by a helper thread. This creates a Dropbox-like synchronized storage service. Whenever changes are made to the synchronized folder at the client side, e.g., creating a new file, updating a file, or deleting a file, the helper thread will establish a connection with the server and automatically send the corresponding operation to the server to update the folder at the server side. You can configure the helper thread to periodically check if there are changes made to the synchronized folder. If the last update time to a file is later than the last check, the file should be synchronized. To simplify the design, we do not need to consider incremental updates. Thus, if the content of a file is updated, the entire file should be sent to the server to overwrite the original copy at the server.

Deliverables

The deliverables include the source code of the client-side and server-side programs, a README file containing instructions on how to compile and run your program, and a report that briefly describes how you implemented the programs, what you have learned, and what issues you encountered. You may want to discuss the challenges in converting the single-threaded server to a multi-threaded version and discuss why distributed locking is necessary for the multi-threaded server. Put all the requirement documents into a zipped folder. Make sure you clearly list your names and student IDs in the report.