**CSE5306, Distributed Systems**
**Fall 2021, Project 1**
Due date: 11:59pm Sept. 30, submission through Canvas

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 nor 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.

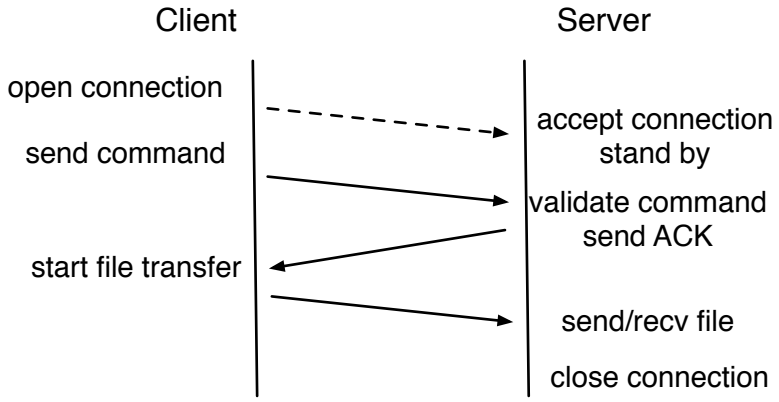
The file server supports four basic operations: UPLOAD, DOWNLOAD, DELETE, and RENAME. We assume that the file service is implemented using a connection-oriented protocol, in which the client and server first establish a network connection, negotiate the operation to be performed, and carry out the file 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.

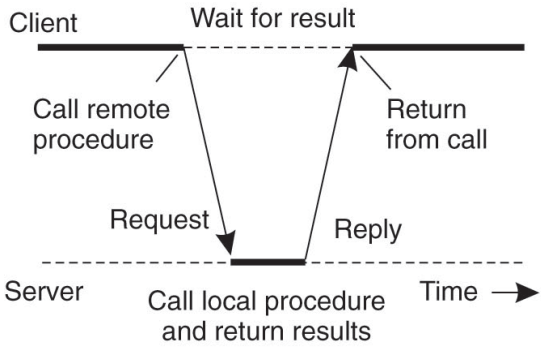
```
fd = accept(sockfd, (struct sockaddr*)NULL, NULL);
```

**Assignment-0 (0pts)**
Create a virtual machine on your laptop or desktop and install a Linux operating system in the VM. You can use any Linux distributions. Ubuntu is recommended. Use the Linux environment to test your programs.

**Assignment-1 (20pts)**
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 (20pts)**
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 do not need to consider locking in your multi-threaded server at this stage. Use multiple clients to test your server.

**Remote procedure call (RPC) based communication**
**Assignment-3 (35pts)**
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)

**Assignment-4 (25pts)**

Re-implement the computation server using asynchronous and deferred synchronous RPCs. For asynchronous RPC, the server immediately acknowledges an RPC call before it actually performs a computation. The result of the computation is saved in a table on the server, which can be looked up by the client for the RPC result. The design of the client will be slightly different from that in the synchronous RPC. Instead of waiting for a synchronous RPC to return, the client using an asynchronous RPC switches to other computations and queries the server for the RPC result at a later time. For deferred synchronous RPC, you need to devise a mechanism to interrupt the client to return the RPC result to the client when the server has completed its local computation.

**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 when 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.