CSE 5306
Distributed Systems
Fall-2019
PROJECT – 1

NAME: Pranjol Sen Gupta
ID: 1001763076

NAME: Zahidur Rahim Talukder
ID: 1001771748

I have neither given or received unauthorized assistance on this work
Signed: **Date: 10/05/2019**
Pranjol Sen Gupta, Zahidur Rahim Talukder

ASSIGNMENT-1

Single_Server.py, Client1.py, Client2.py

In this programming project, we have implemented a simple single threaded file server using Python 2.7. 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, we assumed that file operations are atomic and the server does not need to support interruptions to a file transfer.

The details of the operations and coding are given below.

Server:

The address of host is "127.0.0.1" and the port number is 8080.

s_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) // initialization of socket

s_socket.bind((HOST, PORT))// binding the socket to the port and address

s_socket.listen(3)// listening for connection

Once the server gets the connection from the client, it sends acknowledgement. Once the client sends the request, we use conditional cases to check the request at the server side. The required function to perform the request is also called. The server uses commands.split(' ') to check the request.

For example, connection.send("RENAME " + old_name + " " + new_name), here the first part is request and the last part is data.

upload(filename, link, addrs):

For uploading data from client to server. As the file can be larger, we used while loop to receive the file and wrote it on the server side.

delete(file_name, link, addrs):

To remove an existing file we used os.remove().

rename(prev_name, current_name, link, addrs):

To rename an existing file. For this purpose we used "os.path.isfile(old_name)" to check for existing file. If exists, "os.rename()" is used to rename else 'File not found' is sent.

download(file_name, link):

To download an existing file from server. Basically we just send the data using connection.send.

Client

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) // initialization of socket

s.connect((HOST, PORT))// connecting to host and port

Once the connection has been set, the request is sent to server to perform certain tasks (add, sort…). We send data in the following format (‘request ‘+data). The following functions are used on client side.

upload(file_name, link):

To upload the file. We used “connection.send("UPLOAD " + filename)”. Then on the server side split function is used to check whether it is an upload request. If the file is present in client directory, data is sent to the server.

rename(prev_name, current_name, link):

To rename the file. Both old name and new names are sent to the server as function parameters.

delete(file_name, link):

To delete the file. The filename is sent to server and if there is existing file, it is deleted.

download(filename, connection):

To download the file. If there is existing file in server, the data is sent to the client side and wrote on a file.

Basically this is s single-server, single-client model. We have used two clients namely client1 and client2 to test it. When the server is connected to one client, the other client cannot perform any task. Once one client goes out of connection, immediately the other client can send request to the server. Both clients are connected to the server at the same time but only one can send requests.

ASSIGNMENT-2

Multi_Client_Server.py, Client1.py, Client2.py

For this purpose, we did not change client side code. Only the server side code is changed. To handle multiple threads, we have used python default thread. Each time a client is connected to the server, we start a new thread. We have used start_new_thread(thread,(link, addrs)) for creating new thread. The client can send request and data through this thread.

We did not consider locking in multi-threaded server at this stage. We have used multiple clients to test the server.

But locking is important for multi-threaded server. For example, suppose one client is trying to rename the file in the server. At the same time another client sends the request to rename the same file. What will happen? Two nodes will concurrently work on the same piece of data and the result is corrupted file, data loss, permanent inconsistency or some other serious problems.

ASSIGNMENT 3

RPC_Server.py, RPC_Client.py

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. We have implemented a client stub to pack parameters into a message, sent to the server and a server stub to unpack the parameters.

Calculate_pi():

We have used math function of python to calculate pi. When the client sends request to calculate pi, the server computes it and sends it back to client. The client resumes its operation after getting the result.

add(i, j):

We have used add function to add two numbers. The client sends two numbers back to the server and the server uses split function to get two numbers and add them. Later the server sends the result.

sort(arrayA):

This is one of the trickiest parts where we faced challenge. At first, we sent each element of the array individually through the message, but there was a problem in sending it, as it was an array. Later we serialized the array using pickle.dumps() and sent it to the server. We used pickle.loads() to get back the data at the client side. Pickle can also be used to serialize class objects. So we created class cal to make the input array a member of the class. We then used pickle to serialize the class and sent it to the server. On the server side the object is retrieved from serialized data. The array is sorted using sort function. Then the sorted array is serialized and sent back to the client, where pickle.loads() is used to get back the sorted array from object.

matrix_multiply(matrixA, matrixB, matrixC:

A matrix is a two dimensional array. For array multiplication, the column number of first array must match the row number of second array. The input matrix is sent to the server just like the way we passed array for sorting. We have used numpy.dot function to calculate matrix multiplication. The result is sent back to the client by serializing and pickle.loads() is used to get back the result.

ASSIGNMENT 4

Asynchronous_Server.py, Asynchronous_Client.py//Deferred_Synchronous_Server.py, Deferred_Synchronous_Client.py

For asynchronous part, we have changed the server side and client side. The server does not immediately send the computed result just like the synchronous part. Instead it saves the result in a dictionary based on the key value. For example, dictionary= {'add': 0, 'pie': 0,'sort': [], 'mat': []}. The server computes the result and saves them in each slot. Suppose the client adds two numbers, the server saves it in the dictionary. Now the client sorts an array, the server will also save the result in the dictionary. Later when the client will ask for the result the server will send them based on the task type.

For deferred synchronous system we have changed the server side and the client side. Both need two threads now. Because one thread is needed for client to send the request and data, and another one is needed for receiving data. Similarly, one thread is needed for server to compute the output and another is needed for sending the output. Whenever data is received in one client thread, it interrupts another thread to display the data. In deferred connection, the client uses both threads on different connection, because if the connection is same, the server will refuse one thread.

Result

After the project we have learned how to implement server-client model based on different communications. We have also learned how to build a computational server and how to handle arrays, matrix as input data and output data. We also learned about the importance of using locking in multi-threaded server to reduce the possibility of corrupted data because of race condition between two threads. After implementing the project, we have got a notion how messaging app works.