# PROJECT – 1

Distributed Systems

CSE 5306

**PARTNER NAME: Md Rajib Hossen**

**ID: 1001626295**

**PARTNER NAME:** Mohamed Ghouse Parvez Mohammed Akmal

**ID:** 1001625800

I have neither given or received unauthorized assistance on this work
Signed: Date: 10/06/2018

Mohamed Ghouse Parvez Mohammed Akmal

Md Rajib Hossen

## ASSIGNMENT 1

Files: server.py, client.py

We created a basic single-threaded file server using python. Server is able to accept connection from client and can serve client request successfully. The details of the operations and coding are given below.

**Server code:**

We used “socket.socket(socket.AF_INET, socket.SOCK_STREAM)” to initialize the socket and “socket.bind” to bind the socket to address (host: 127.0.0.1, port:8080).Then we used “socket.accept” to get the connection from the client. Once we received the request from the client, we used a switch case to determine the request and called the required function. Client message includes the operation type (delete/update…) and the respective filename/data. The server splits the message received from the client using split() function. The part before the first space in the message is the request and the part after it is the data required for that request. We used the following functions to implement the various operations (upload, download, delete and rename) accordingly.

upload_file(filename, connection, address):

This is used to upload the file from the client to the server. ”connection.recv()” was used to get the file data from the client. The main challenge we faced here are receiving large file using byte stream. So, We used while loop and received the file in parts and simultaneously wrote the data on the server side.

Rename_operation(old_name, new_name, connection, address):

This is used to rename an already existing file in the server. The old name and the new name of the file is received from the client. First we check if the file exists in the server directory using “os.path.isfile(old_name)”. If the file exists we rename the file using “os.rename()”.

delete_operation(filename, connection, address):

This is used to delete a file from the server. First, we check if the file is present in the server directory. If it is present we remove the file using os.remove().

download_operation(filename, connection, address):

This is used to download a file from server to the client. We check if the file is present in the server, if it is present we send the file to the client using “connection.send()”, before we send the file we send a message “prep” to let the client know that we have the file and are ready to send it.

**Client code:**

We used “socket.socket(socket.AF_INET, socket.SOCK_STREAM)” to initialize the socket and “socket.connect()” to connect to the server giving the server address.

Once we are connected to the server we get the required action from the user and use a switch case to call the required function. We send the data in the following format (“request “+data), a space is used to separate the data from the request. We are using the following functions in the program.

<u>get_socket():</u>

This is used to initialize the socket and connect to the server.

<u>upload_file(filename,connection):</u>

This is used to upload the file to the server from the client. We used the code “connection.send("UPLOAD " + filename)” to the server. Then we check if the file is in the client directory. if it is present we send the data to the server.

<u>rename_file(old_name, new_name, connection):</u>

This is used to rename a file on the server directory. We send both the old name and the new name of the file to the server and wait for the server to send a message which says whether the operation was successful.

<u>delete_file(filename, connection):</u>

This is used to delete a file on the server side. The name of the file to be deleted is sent to the server and wait for the server to send a message which says whether the operation was successful.

<u>download_file(filename, connection):</u>

This is used to download file from the server. When the server is ready to send the file (when the message “prep” is received), we use a while loop to get the data in parts and simultaneously write it on the client side just as it was done in the server side.

## <u>**ASSIGNMENT 2**</u>

To satisfy this requirements, we changed in the server code. Server was single-threaded before. So, we changed code to support multiple client thread to connect to server. For this, we didn’t change any code in client side.

We used python default thread class. When a client connect to the server, we start a new thread and pass client info to the thread. We implement a threaded function. Then we pass client info to that function using this piece of code: “start_new_thread(threaded, (connection, address,))”.

We also had to use locking mechanism to protect files downloading and uploading/renaming simultaneously. To do this, we used python default threading locking mechanism (threading.Lock()). Then in start of each operation of upload and rename, we acquired lock using “acquire()” and then after finishing task, we released the lock, “release()”.

# ASSIGNMENT 3

In “rpc_server.py” and “rpc_cleint.py”, we have implemented RPC based communication. The required RPCs have been implemented on both server and client side as follows:

**Addition of two numbers:**

In this RPC, the data sent by the client is encoded as shown: “Add number1 number2”. We split the input message with respect to the space using the split () function and store it in a list on the server side. We access the two integers using the list and find it’s sum and send it back to the client.

**Calculate_pi:**

When the client calls for the RPC, we use the BBP(Bailey–Borwein–Plouffe) formula to calculate the value of pi and return it to the client.

**Sorting:**

For this RPC we initially tried to send each element of the array individually through the message but we later improved it by serializing the array using pickle (pickle.dumps()- to serialize, pickle.loads()-to get back the data). We later found that we can use pickle to serialize class objects, serializing class objects will also help us to implement the “matrix_multiply()” RPC. So we declared class foo and we made the input array a member of the class. The class was then serialized using pickle and sent to the server, the server gets the object from serialized data, and the array is sorted using “quicksort.py” where quicksort is performed. Once the array is sorted and updated in the class foo, it is serialized and sent back to the server where it is displayed.

**Matrix_multiply:**

This is done just like sorting, the input matrices and their sizes are obtained from the user and they are added to the class foo. Then the class is serialized and it is sent to the server. The server gets the data and performs the matrix multiplication. The result is stored in the class foo, serialized and sent back to the client where it is displayed.

## BONUS ASSIGNMENT

we wrote a new file for this task. Filename is “background_thread.py”. This file runs for every 10s and synchronize with server. For the first time, it uploaded all of the files to the server. Then for every other iteration, it stores some information like last update time, last updated files. Then it looks for any changes in the files and new or deleted files. Then it perfectly synchronize with server.

The helper thread has the following functions defined in it:

**get_socket():**

This is used to initialize the socket and connect to the server.

**get_creation_time(filepath):** we iterate over client directory and got all the files names. Then we get modified time and creation time for each files.

**upload_file(filename) and delete_file(filename):**

These are same as the “upload_file()” and “delete_file()” functions defined in the client side of assignment 1 and 2.

**synchronize ():**

This is the main part of the helper thread. When the program starts all the files currently present in the client folder are appended to a list “next_iteration_files”, we use this list to keep track of the changes made to the files. We also note down the update_time using “time.time()”. For the first time, In order to synchronize both server and client directory we upload all the files from the “next_iteration_files” to the server, once this is done both server and client directories are in sync. Then we use “threading.Timer(10, synchronize).start()” to start the helper thread again after 10 seconds, now it checks if there is any change in the client directory by comparing the current files with the files in “next_iteration_files”. If there are extra files then they are uploaded to the server else if there are less files the missing files are also deleted from the server. Apart from this we also check for the last modified time of each file using “get_creation_time()” and compare it with the previously stored update time, If there is any change the file is uploaded to the server in order to update the file on the server side. Finally we change the update time to current time and we synchronize again after 10 seconds. This keeps repeating till the program is closed.

## RESULT

After our project we learned how to implement a server client model and handle multiple clients using multithreaded server. We also learned how to send different parameters over the socket while implementing RPC calls. Moreover, we learned about how to synchronize with server like dropbox and got some ideas.