Programming Guide : Machine Problem 2

This is a quick guide on how to start your MP2 with the given skeleton code of a client program. The objective of the skeleton is to let students focus on gRPC programming, and also to evaluate your homework fairly by having the same skeleton. The skeleton is almost similar to that of MP1

- **The skeleton**
  1. client.h : contains an abstract class that has three pure virtual functions to be implemented by you and a couple of data structures.
  2. tsc.cc : contains basic constructs and information on what to do and where
  3. tsd.cc : contains basic constructs and information on what to do and where
  4. sns.proto : contains description of the service and it's different functionalities
  5. Makefile

- **Derive IClient class and make your own Client**
  Every business logic has been implemented except for major functions that use gRpc and communicate with the server. What you are supposed to do is just make a concrete class of IClient and implement three virtual functions. Following is one possible example of the concrete class.

```
class Client : public IClient {
  public:
    Client(const std::string& hname,  const std::string& uname,  const std::string& p)
        :hostname(hname), username(uname), port(p)
        //, stub_(/* some parameters */)
        {}
  protected:
    virtual int connectTo();
    virtual IReply processCommand(std::string cmd);
    virtual void processTimeline();
  private:
    std::string hostname;
    std::string username;
    std::string port;
    // You can have an instance of the client stub
    // as a member variable.
    std::unique_ptr<ClassNameOfYourStub> stub_;
};
```

```
int Client::connectTo()
{
  // your implementation
  return 1; // return negative if failed
}

IReply Client::processCommand(std::string input)
{
  // your implementation
  IReply ire;
  return ire;
}

void Client::processTimeline()
{
  // your implementation
}
```

function “connectTo”
In this function, you are supposed to create a stub so that you call service methods in the processCommand/processTimeline function. That is, the stub should be accessible when you want to call any service methods. I recommend you have the stub as a member variable in your own Client class. Please refer to the gRPC tutorial on how to create a stub.

function “processCommand”
In this function, you are supposed to parse the input command and call an appropriate service method. Then, you need to return IReply that contains results taken from the response of the service method. The skeleton will display the result based on the IReply. IStatus and IReply are almost the same as Status and Reply of MP1.

```
enum IStatus
{
  SUCCESS,
  FAILURE_ALREADY_EXISTS,
  FAILURE_NOT_EXISTS,
  FAILURE_INVALID_USERNAME,
  FAILURE_INVALID,
  FAILURE_UNKNOWN
};
```

```
struct IReply
{
   grpc::Status grpc_status;
   enum IStatus comm_status;
   std::vector<std::string> users;
   std::vector<std::string> following_users;
};
```

function “processTimeline”

In this function, you are supposed to get into timeline mode. You may need to call a service method to communicate with the server. Use getPostMessage/displayPostMessage functions for both getting and displaying messages in timeline mode. You should use them as you did in MP1.

```
// in client.h

std::string getPostMessage()
{
   // this function has been implemented for you.
}

void displayPostMessage(std::string sender, std::string message, std::time_t time)
{
   // this function has been implemented for you
}
```

You must invoke the “start_client” function

In order to start the business logic provided by the skeleton, you should invoke “start_client” function that is implemented in IClient when you are ready to go. Following code snippet can be an example

```
int main(int argc, char** argv) {
  // process command line argument
  // do something else for your program

   Client client(hostname, username, port);
   client.run_client();

   return 0;
}
```

- **Completing the server code**
  The server code (tsd.cc) essentially implements the functionalities described by the sns.proto file. Comments have been written within each function describing what is expected of that function. Each of these functions ideally would return a status message stating whether if the service call was successful or not and also populate the Reply object, which would be consumed at the client end

- **Naming & invocation conventions**
  Ensure that the project can be invoked as follows :
  - **Server:** ./tsd -p <port>
  - **Client:** ./tsc -h <hostname> -p <port> -u <username>

- **Do not print anything on the screen**
  The skeleton does print everything on screen for you to interact with a user. You can print some debugging information during the development phase, but you must erase them when you submit your homework.

- **Ensure that commands are accepted in uppercase**
  To be consistent with the examples provided and the testing mechanism, it is recommended that your program be able to accept all commands in uppercase

- **Test your program with given test cases**
  Your submission will be graded based on six test cases (not same as the sample cases, but similar) that cover most of the scenarios in MP2. You can see some sample test cases and its results in the provided excel file. Before testing each test case, remove your data file and restart your server/client program to have the same output as the given output in the excel file.

- **References**
  gRPC basics
  gRPC API