Concurrency Assignment

Andrea Minuto
Reza Hassanpour
Afshin Amighi

Description:

The course of concurrency requires to create a concurrent server program that handles several clients over the network (at the same time).

Implementation requirements and behaviour description:

A group is composed 1 or 2 students (no exceptions).
Each group needs to provide the implementation of a server program that can process and collect the information of several clients using concurrent programming solutions.

Each instance of the client program will create a connection to the server.
The server will reply with a welcome message (a string).
The client will send to the server a string of the following json format:

Format of the message

```
{
        "studentnr": "A Studentnumber",
        "classname": "A Class Code",
        "clientid": 1,
        "teamname": "A team name",
        "ip": "An IP",
        "secret": "",
        "status": ""
}
```

example (in json object):

```
{
        "studentnr": "0123456",
        "classname": "INF2A",
        "clientid": 1,
        "teamname": "Team1",
        "ip":'127.0.0.1',
        "secret": None,
        "status": None
 }
```

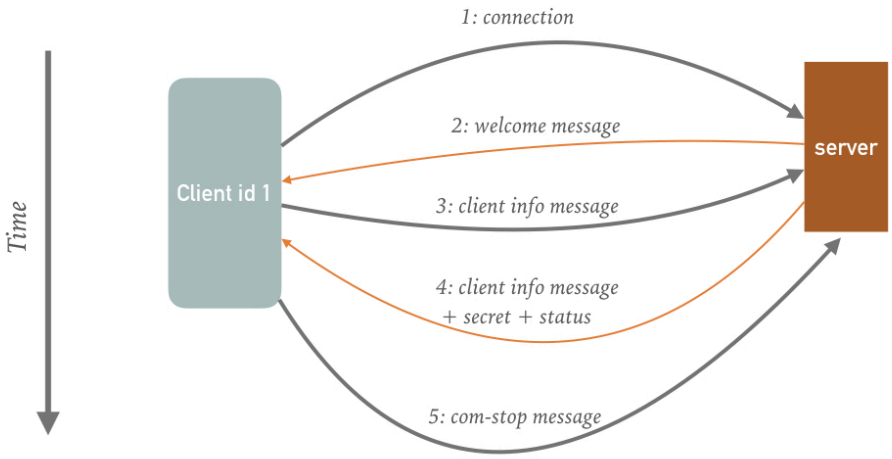
The server will reply with the same format, but added information on *secret* and *END status*. Then, the client will send a message to the server to stop the communication and will close the connection.
When all the clients have communicated, at the end there will be a client with **id=-1** , *that informs the server to stop*. When the server receives the message from the ending client (with id = -1), it must print all collected information and the number of the communicated clients.

Messaging must comply with this set:

```
{
        welcome = "WELCOME"
        stopCommunication = "COMC-STOP"
        statusEnd = "STAT-STOP"
        secret = “SECRET"
}
```

Below we present a high level schema of the communication

Schema of the connections



Client-Server Communication

Extra info:

C# is the official language for the project. The protocol is based on TCP/IP. A sequential version of this assignment is implemented and provided in the exercises GitHub repository, folders: ***SocketClient*, *SocketServer*.** You are expected to fill provided todos. Messages MUST NOT be altered.

Submission:

Instructions for the submissions:

- The concurrent server program MUST be able to handle around 250 concurrent clients.
- For the final submission the student has to submit ONLY the C# program file of the server. No extra files, no zip, no rar, no client program.
- No extra library is needed for this assignment (not already used in the sequential version).
- In order to solve the problem, the student can choose one of the two multi-threading and/or asynchronous methods.
- Before submission, the student MUST test his/her solution with the provided sequential version. The communication must continue for all the clients without any error.

- The student can implement the **concurrent** version of the clients to have a better/complete testing.
- Submitted file MUST contain group's requested information. A list of todos are specified in the server program.
- Do not change the port number provided in the sample version (the server should communicate as in the sample).