Assignment Concurrency 20/21

Reza Hassanpour
Ahmad Omar
Afshin Amighi
Andrea Minuto

Subject:

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

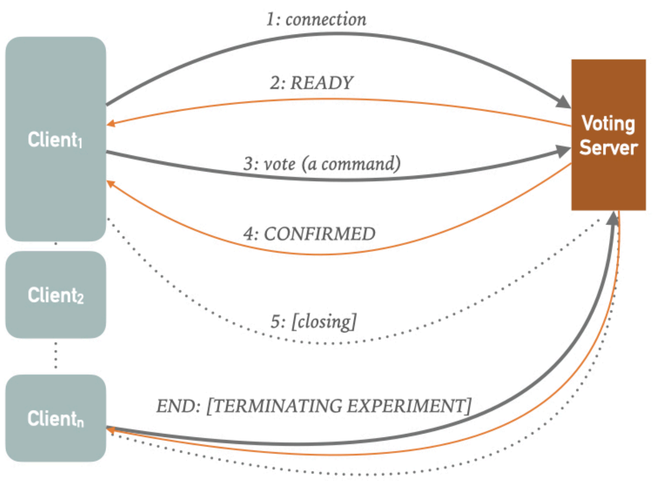
Groups:

An assignment group is composed of 1 or 2 students (no exceptions). All the members of the group must be within the same class (same teacher).
Each group needs to provide the implementation of a server program that can process and collect *votes* coming from simultaneous clients *using concurrent programming solutions*.

Scenario:

On the client side, there must be a program that **simulates** multiple simultaneous clients (for example 150 clients or more). **Each** instance of the client will create a connection to the voting server. The server will reply with a READY message.
The client has its command to vote. It will send the command to the server. The client has to select its command from a list of commands provided in a configuration file. As soon as the command is CONFIRMED by the server, the client will close its connection.
The server is responsible for *accepting* all the requested connections and *collecting* **all** the votes (or commands). At the end of the voting, there will be a client that reports the voting termination. Then, the server reports all the collected votes, the command with the **maximum vote,** and executes the command with

Schema of the connections:



Generic scheme of connection: the timely order of execution depends on the machine state.

Requirements:

the maximum vote. Finally, the server is ready for the next round of voting. A high level schema of the communication is represented below (see the picture).
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 in the provided todos.
Messages and protocol MUST NOT be altered.

Instructions for the submissions:

- The concurrent server program MUST be able to handle at least 150 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 MUST implement the server by using *multi-threading techniques*. Using asynchronous tasks is NOT accepted.
- 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 should implement the **concurrent** version of the client to have a complete testing (edge cases that shows error might be verifiable only with this implementation).*
- Submitted file MUST contain the group's requested information. A list of todos are specified in the server program.
- **DO NOT ALTER the protocol and messages (and global variable if not for testing purposes). The server MUST work correctly with the provided sequential client.**

Assessment:

The assignment will receive a passing grade if the concurrent implementation reflects all the teachings of the course, that means it must implement a correct *concurrent program*.
A correct implementation will have proper regulation of access of shared memory (and only the shared one), will always give the right result for each run, independently from the load of processes, and or memory.