Homework 2, Non-Blocking Sockets

Network Programming, ID1212

1 Goal

- You can develop a distributed application using non-blocking TCP or/and UDP sockets.
- You can use concurrent threads in nodes of a distributed application using non-blocking sockets, in order to improve scalability and performance (e.g. response time), and to hide communication latency.

2 Grading

The grading is as follows:

0 points Not passed

1 point Passed, but **no points** for higher grades

2 points Passed, and **one point** for higher grades because an accepted solution was submitted before deadline.

3 points Passed, and **two points** for higher grades because an accepted solution including the optional higher grade task was submitted before deadline.

3 Auto-Generated Code and Copying

You must be able to explain and motivate every single part of your code. You are *not* allowed to copy entire files or classes from the example programs on the course web, even if you understand it and/or change it. However, you are allowed to write code which is very similar to the example programs on the course web. You are also allowed to use GUI builders and other tools that generate code.

4 Tasks

You are to solve *one* of the following two tasks. You do not get any extra points from solving both. If you consider solving the optional task, you are advised to read and consider it before solving the mandatory task, since the solution of the optional task might affect how you solve the mandatory task.

Task 1, The Hangman Game With Non-Blocking Sockets

This task is identical to task one of homework one, except that you are now required to use non-blocking sockets. This means a lot of your code might be identical to code you wrote for homework one, which is perfectly fine. Note, however, that threads can not be used the same way as in the homework one application, which used blocking sockets.

Requirements on Your Report

You do not have to repeat anything that was shown in the hw1 report. The report for hw2 must show the following, but nothing else is required.

- That the solution has an acceptable layered architecture and is well designed. This means it must follow the guidelines of the lecture on architecture, and of the programming examples on the course web. Your are, however, not required to use exactly the same layers as in those examples.
- That only non-blocking sockets are used.
- That both client and server are multithreaded. This means the same as in homework one, that the server can handle multiple simultaneous clients and that the client has a responsive user interface. Note that threads must be handled as described in the videos on non-blocking sockets, which is different from how they were handled when using blocking sockets.
- That the program still works as expected. You show this by including screenshots of the user interface.

Task 2, The Rock-Paper-Scissors Game With Non-Blocking Sockets

This task is identical to task two of homework one, except that you are now required to use non-blocking sockets. This means a lot of your code might be identical to code you wrote for homework one, which is perfectly fine. Note, however, that threads can not be used the same way as in the homework one application, which used blocking sockets.

Requirements on Your Report

You do not have to repeat anything that was shown in the hw1 report. The report for hw2 must show the following, but nothing else is required.

- That the solution has an acceptable layered architecture and is well designed. This means it must follow the guidelines of the lecture on architecture, and of the programming examples on the course web. Your are, however, not required to use exactly the same layers as in those examples.
- That only non-blocking sockets are used.
- That nodes are multithreaded and have a responsive user interface. Note that threads must be handled as described in the videos on non-blocking sockets, which is different from how they were handled when using blocking sockets.
- That the program still works as expected. You show this by including screenshots of the user interface.

5 Optional Higher Grade Task For Both Mandatory Tasks

The architecture presented in the video on non-blocking sockets and the non-blocking chat program is a bit simplified. The non-blocking chat server delegates *all* requests to a thread pool. This is not necessary, since tasks that are not time-consuming could be handled directly by the thread performing socket communication, without invoking any other thread. Only tasks performing blocking IO, for example read or write to disk, and tasks that perform lengthy computations, have to be delegated to the thread pool. This is the *single-threaded event loop* architecture used in Node.js and many other non-blocking servers.

Your task is now to implement such a single-threaded event loop architecture in your game server (the client remains unchanged). If you have implemented the hangman game, it means that reading the file with words from disk shall be done in a separate thread, as illustrated in the videos, while all other operations can be done in the same thread as the socket communication. If you have implemented the rock-paper-scissors game, there is no operation that does any blocking IO, which means all operations can be performed in the same thread as the socket communication. However, to implement also the thread pool part, you have to invent an operation performing blocking IO, and execute that in a separate thread. A suggestion for such an operation is to store the current score in a file on disc.