OS – Assignment 1

Learning Objective: To observe and understand the difference between single threaded and multi threaded processes in terms of performance in the context of a simple client server program.

Hints: Feel free to refer to this gist for code snippets and code structure.

Libraries and Tools: For folks using Java, you will not needed any external libraries. Use postman for sending in requests (alternatively you can also use your browser). Use any language you like to implement the task. Do bear in mind that some languages are not multi threaded but implementations built on top of them to allow them to handle multi-threaded scenarios (NodeJS in case of JS for example). Thus you may not be able to implement a *single threaded server* even if you wanted to or at least it will be rather complicated to achieve.

Tasks

Task 1: We will start by writing a function that will simulate a compute heavy job – in other words, a function, which when executed, will take a good amount of run to completion. An example of such job would be a database search but since we are just interested in simulating it, we'll do it using ***Thread.sleep*** statement.

Start by creating your ***Main*** class. A good semantic folder structure is always appreciated so feel free to follow my directory structure. So right now you have an empty ***Main*** class with an empty ***Main*** function. To finish this task, we will implement the ***SearchSimulator*** class as shown in the folder structure. We will be revising this implementation again in **Task 2**. But for now, you can implement this as shown here.

You can verify that it indeed blocks the thread it is called on by simply calling it in your ***Main*** function and observing its output.

Task 2: Now that we have a compute heavy job ready, we can move on to implement a server that will run on a *single thread*. It will accept a request and respond with a simple HTML string output. You don't have to worry about the HTML string, I have provided the code for it (make sure to add it in the right place in your code).

To implement the server itself, you can create ***SingleTheadedServer*** class in the ***servers*** package (check dir structure). There are a couple of things that we'll have to address here:

1. The server itself runs on an independent thread created from the main thread.
2. You will have to implement the ***Runnable*** interface to execute the instance of ***SingleTheadedServer*** on its own thread. A question that you might want to think about is why do we want to run the server on its own thread? Can we not run it on the main thread itself? You can use my code as a guide for your implementation.

3. Once you are done, we can go back to ***SearchSimulator*** and modify it to not only block the thread but also write the HTML response we want to send back to the client. For that, we will need to pass the ***Socket*** object into our ***processClientRequest*** function and then use ***ResponseGenerator*** to generate the HTML string.
4. Finally you will need to modify ***Main*** in order to start the server.

If the steps seem too vague, feel free to post your questions on discord for discussion and hints.

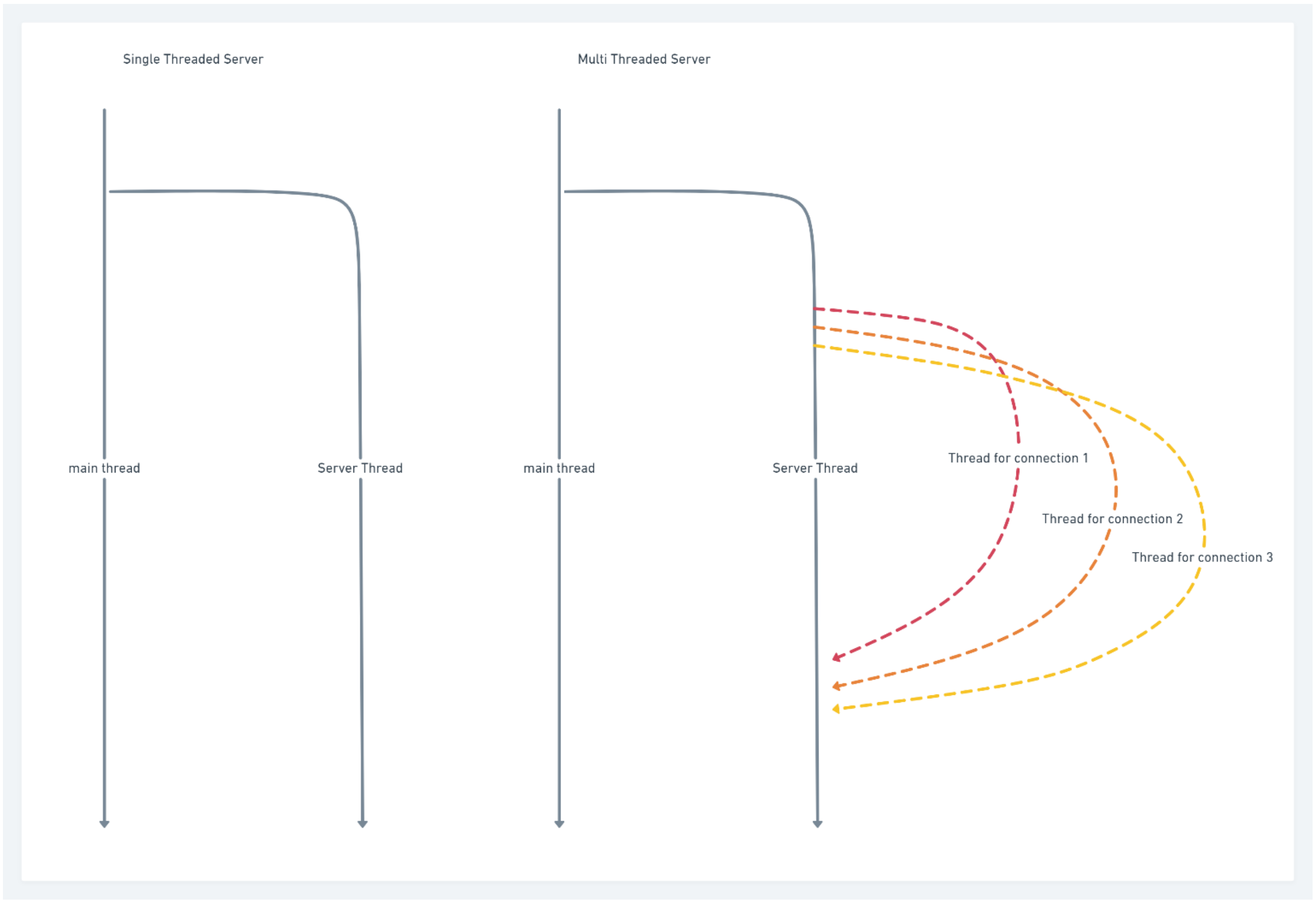
With everything ready, you can use Postman or your browser to send multiple **GET** request to "http://localhost:<portnumber>" and see what happens. Note the timings.

Task 3: Now we are ready to try the same experiment with a multi-threaded setting. But the first question we need to answer is what exactly should be allowed to run on multiple threads? It might appear that multi threading is some sort of a secret sause for performance but usually implementation become very complicated when you start running things in their own thread especially in terms of resource sharing between the parallel threads (think of reading from a shared variable while another thread is modifying it in parallel). Since in our case we are not really writing or reading, it doesn't really matter much but just be aware.

1. It would be wise to create another class and name it ***MultiThreadedServer***.
2. You can go ahead and copy/paste all the code from the ***SingleTheadedServer*** class but this time there will be a slight difference. Instead of running the compute-heavy task in the same thread as the server, we will create a new thread for each client request, effectively running the compute-heavy task on their own threads. This will result in the socket being ready to recive subsequent requests almost instantly.
3. But what need to change now if the implementation of the compute-heavy task itself. Since we want it to be able to run on its own thread, we'll have to implement the ***Runnable*** interface for it. You can use this as your starting point.
4. Finally, you can need to replace the compute-heavy function class with a statement to create a new thread to run the compute-heavy function.

With everything ready, you can use Postman or your browser to send multiple **GET** request to "http://localhost:<portnumber>" and see what happens. Note the timings.

The following diagram might help you think the tasks through.



Report

1. Your repository link
    a. A README.md with instructions to run your code
    b. A section in your README.md or docs/REPORT.md documenting and explaining your observations.
2. A short and concise video of your code in action