Project 2 - Multitasking

Given a multi-core CPU, we can distribute the computing tasks across the CPU cores so that the tasks can be executed in parallel. This can be done by mapping each of the computing tasks to a thread or a process. This assignment will give you the chance to use multi-threading to improve the performance of a program.

Background: Proof of Work (PoW)

Proof of Work[1] (PoW) is a consensus mechanism, which is widely used by cryptocurrencies. The simplest setting of PoW consists of one server and one client. The server will be asked by the client to perform some time-consuming computing task (i.e., the work). Once the server completes the task, it has to present a proof to the client indicating that it has spent the effort and finished the task. Following is the detailed procedures:

1. Client generates a random string C and sends it to server.
2. Server generates a random string called **seed** S
3. Server appends the client string to the end of the seed string and uses SHA-256 (or some other hash function) to calculate the hash value, i.e., $\mathrm{SHA}(S||C)$
4. Server checks whether the hash value begins with a predefined number of zeros (the number is commonly referred to as the PoW **difficulty**). If not, repeat step2 to step4.
5. Server return the seed S to client.
6. Client calculates $\mathrm{SHA}(S||C)$ and check whether the hash value begins with the predefined number of zeros as the proof that server completes the work.

Task A. Server Program Code

In this assignment, you need to implement a TCP server to handle PoW requests from several clients. The server should be able to process as many PoW requests as possible. You may want to create a couple of threads or processes to parallel the processing of the PoW requests.

➢ TCP Client

The client program is provided by the TA. The client is implemented in Node.js. When you start the client program, it determines the number of CPU cores and forks a corresponding number of client processes that connect to the server (you need to

[1] https://en.wikipedia.org/wiki/Proof_of_work

implement the server). Each client process repeatedly makes PoW requests to the server until the preset timer `testTime` expires. After all the client processes finish their execution, the client program will calculate the total number of PoW requests completed by the sever.

The client program has the following configuration parameters:

- `connectionNumber`: the connections that the client would establish with the server
- `testTime`: total time (ms) of the test
- `PoWDifficulty`: the difficulty of PoW

➢ TCP Server

You can use any language to implement the server. The server should handle as many requests as possible. When the server gets the random string from the client, it should perform PoW to obtain the corresponding seed and reply it to the client. The format of the response is "<random string from the client>,<seed>,<hash value>".

Note:

1. Each message should end with a newline '\n'.
2. Server should listen on localhost and port 20000 by default.
3. Both the length of the random string and the length of the seed should be 5 characters long.
4. Before testing, confirm that server and the client are using the same difficulty value.
5. Before using the client, you should install **Node.js** and **NPM** (Node Package Manager), then run "npm install js-sha256" in the folder of the client code to install js-sha256 package. To run the client, just use "node client.js".

Task B. Report

You should write a report. Please briefly describe:

- how do you implement the server program
- the performance of the server program, and how do you evaluate it

Scoring

Performance (the number of requests per test period) of the server program (60%)
Report (40%)

Submission

Please zip your server program and report, and name "**project2_studentID.zip**".

Named:

- The server program : **project2_studentID**
- Report : **project2_report_studentID**

Reference

- Socket Programming HOWTO - Python 3.8.0 documentation
- multiprocessing - Process-based parallelism - Python 3.8.0 documentation
- threading - Thread-based parallelism - Python 3.8.0 documentation
- Socket Programming in C/C++
- Multithreading in C++