Client-Server exercise

Duration

Up to 2.5 hours

Introduction

In the following exercise you are required to develop a client-server application that communicates with each other.

You will be measured on the following aspects:

- Correctness - The application should behave as described in the sub tasks
- Readability - Your are expected to write clean code that is easy to understand
- Performance - Your application, mainly the server, should respond at reasonable time

Finishing only part of the sections of the assignment perfectly is better than finishing all the assignment imperfectly

You are not allowed to use any online/offline help other than the following official documentation:

- Python - https://docs.python.org/
- Java - https://docs.oracle.com/javase/7/docs/api/
- JS - https://nodejs.org/api/
- C/C++
  - Windows - https://docs.microsoft.com/en-us/windows/win32/api/
  - Linux - https://man7.org/linux/man-pages/man2/

You should not use HTTP/S or any other high level protocol.

Adding tests of all sorts is recommended and appreciated.

The submitted solution should contain 2 executables:

- Server
- Client

Usage

```
$ ./server
Listening on 127.0.0.1:8080
```

```
$ ./client 127.0.0.1 8080
```

Part 1 - Infrastructure

Implement the basic server and client.

Server

The server receives its port and ip address as runtime arguments. It should wait for incoming messages, print them and echo them back to the client.
The server should gracefully shutdown upon “Enter”. Any resources allocated by the server should be released before shutdown.

Client

The client receives the server’s address, port and its identifier as runtime arguments. When executed, the client should send a message to the server, wait for a reply and print it.
The message will contain:
ID - Integer [1-10] (A unique ID - identifying the client)
Message - Random Integer (the message itself to send to the server)

Example

```
$ server                          $ client 127.0.0.1 8080
$ Listening on 127.0.0.1:8080     $ Received "4" from server
$ Received "4" from client [1]
```

Part 2 - Sum

Server

Server should aggregate the sum of the messages and reply it to the client

Client

No Change

Example

```
$ server                          $ client 127.0.0.1 8080
$ Listening on 127.0.0.1:8080     $ Received "sum is 4" from server
$ Received "4" from client [1]    $ Received "sum is 9" from server
$ Received "5" from client [1]
```

Part 3 - Multiple clients support

Server

Support multiple clients concurrently. The server should be able to handle multiple messages received from multiple clients at the same time.

Client

Generate a random id

Attention

Please pay careful attention to concurrency. If multiple messages from the same client arrive at the same time to the server, e.g. “4”, “7”, “6”, all of the values will be summed up.

Example

```
$ server
$ Listening on 127.0.0.1:8080
$ Received "4" from client [1]
$ Received "3" from client [2]
$ Received "5" from client [1]
```

```
$ client 127.0.0.1 8080
$ Received "sum is 4" from server
$ Received "sum is 9" from server
```

```
$ client 127.0.0.1 8080
$ Received "sum is 3" from server
```

Good Luck!