

## Computer Networks

### Skill Check

Prof. Dr. rer. nat. Nils Aschenbruck

Leonhard Brüggemann, Leon Richardt, Alexander Tessmer

#### General information:

- Read the **complete** task sheet **carefully** before you start working on it!
- By starting to work on the task, you declare that you are able to do so both physically and mentally.
- You must only use your assigned computer and work in the directory *Task* on the desktop. This is vital for submission (see below)!
- In case of an attempt to deceive, the assignment of all parties involved will be graded immediately as failed. There is no prior warning. Mobile phones that are switched on and other communication devices (such as smart watches) are also considered an attempt at deception! Changing the font size is also considered an attempt at deception.
- You are not allowed to eat in the digital exam labs, drinking is allowed.
- If you have to go to the bathroom, notify the supervisor. They will make sure that no two people go at the same time.
- Submission:
  - Make sure that your final version is saved in the directory *Task*.
  - Notify the supervisor to present your solution and **do not** turn off your computer.
  - The solution is automatically copied from your computer.
- System Information:
  - The tasks assume you are working on **Ubuntu 22.04**, using **Python 3.10**. We cannot guarantee support for other system environments.
  - Files that are part of this task are provided in the directory *Task* on the Desktop.
- Provided Documentation:
  - Linux man pages
  - Python documentation (might take a few seconds to open)
  - OpenSSL Cookbook
- Recommended IDEs: VS Code, vim, GNU nano, gedit

**Please note:** In order to pass, your solution must fulfill **all requirements** specified in the task. It is not sufficient to only complete a subset.

## Task 2-C: Multiplexed Distributed Calculator Using select

In Task 1-C, you programmed the Distributed Calculator — a server that allows clients to connect via TCP and send commands (ADD, MULTIPLY, RESET, GET), which the server then executes. In this task, you will add the functionality to handle multiple connections simultaneously. **The server must be capable of handling commands from multiple clients separately, which means that it should keep an *individual* running total  $t$  for each client.** It is *mandatory* to implement this using `select` which has been introduced in the lecture.

We slightly modified Section 2. It is now possible to bind the client to a specific port. By using the same port again, a client must be able to reconnect to the prior running total. Otherwise Sections 1–3 are identical to Task 1-C and have only been added to this sheet for completeness.

**You have passed the task when your server can be used as a reliable endpoint for the REPL of *multiple* clients using `select`** (as in Section 2, but with multiple simultaneously used terminals).

### 1 Requirements

The server must keep a running total  $t$  of the commands performed. On startup, set  $t \leftarrow 0$ , and listen for incoming TCP connections on port 4711. For this task, it is sufficient if your implementation can only handle a single client connection at a time. In later pools, you will extend the implementation to support handling multiple clients in parallel.<sup>1</sup>

Let  $x$  be a double-precision (8 bytes) floating-point number. Your server must implement the following commands:

ADD $x$	Performs $t \leftarrow t + x$
MULTIPLY $x$	Performs $t \leftarrow t \cdot x$
RESET	Performs $t \leftarrow 0$
GET	Returns the current value of $t$

The commands are transmitted from the client in a custom message format, which is described in detail in Section 3. Example 1 illustrates the kind of client behavior your server must be able to support.

---

<sup>1</sup>Note:  $t$  should only be reset on reception of the RESET command, or when the server shuts down. Do *not* reset  $t$  when a client disconnects.

**Example 1**

In a correct implementation, one possible client session could look like this:

```
student@host:~/task-c$ ./client
Trying to connect to 127.0.0.1:4711 ... Connected!
> GET
[GET] 0.0
> ADD 42
> MULTIPLY 42
> GET
[GET] 1764.0
> MULTIPLY 0.5
> ADD -881
> GET
[GET] 1.0
> RESET
> GET
[GET] 0.0
> [Ctrl]+[D]
Goodbye!
```

In particular, note that negative and non-integer numbers are supported arguments as well.

**Hint**

When developing your implementation, you might find it useful to call

```
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

before binding the socket to an IP address. This allows the socket to be rebound to the same address even when it was not shut down properly. See [man 7 socket](#) for more information.

## 2 Client Description

To start the client, run

```
student@host:~/task-c$ ./client --server_port <server_port> --client_port <client_port>
```

with optional parameters `server_port` and `client_port` parameter. The client binds to `client_port` and then attempts to open a TCP connection to the specified `server_port` on the local host. When the `server_port` parameter is omitted, the client attempts to

connect to port 4711 per default. When the `client_port` parameter is omitted, the client binds itself to 8888.

After a connection has been established, the client enters a *read-eval-print loop* (REPL). The user may enter a command via the terminal, which is translated into a message according to Figure 1 and sent to the server. In case of the ADD, MULTIPLY, and RESET commands, no response is returned by the server, so the REPL returns to reading input from the user. In case of the GET command, the server returns a response (cf. Figure 2) which the client parses and prints to the terminal before waiting for further input from the user. To exit the application gracefully and close the connection to the server, input `Ctrl + D`.

### 3 Protocol Format

Each of the commands defined in Section 1 is associated with a message. The formats of these messages are shown in Figure 1. Therein, the first field is an identifier for the type of command, represented with a single UTF-8-encoded character (A, M, R, or G). For the commands with a parameter, the second field is the parameter value  $x$ , encoded as an 8-byte floating-point number.

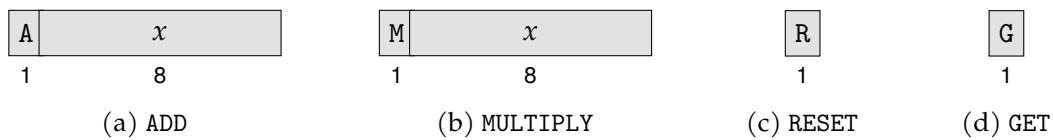


Figure 1: Format of command messages sent by the client. The numbers below the fields indicate the field length in number of bytes.

Since the GET command instructs the server to return the current value of  $t$ , a response message format is defined in Figure 2. Again,  $t$  is encoded as an 8-byte floating-point number.

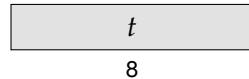


Figure 2: Format of GET response messages sent by the server. The number below the field indicates the field length in number of bytes.

**Good luck!**