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-B: Pre-forked *TickeZ* Server

The up-and-coming online portal *TickeZ* wants to sell concert tickets for the cantina band [1]. Due to the music group's popularity, there will be a rush at the start of the pre-sale. To cope with this, the portal needs a server that can process several requests in parallel as quickly as possible. You are assigned to implement this server as part of this task as a **pre-forked server**.

1 Requirements

A client initiates a TCP connection with the server and requests a ticket from the server. The server then transfers the ticket file to the client. After the server completely transfers a file to a client, it listens for further file requests. The server must be able to process **four requests simultaneously** using a **pre-forked server structure** and deliver each of the available files. Use the template `tickez_server.py` containing some global variables you might use and code to parse your arguments. It is run by

```
student@host:~/task-b$ python3 tickez_server.py [--port <port>] [--dirpath <dirpath>]
```

with the optional `port` and `dirpath` parameters. The server binds to `127.0.0.1:`*port*, i.e., to the specified port on the local host. If the `port` parameter is omitted, the server binds to port `4711`. If `dirpath` is omitted, the server expects the folder `tickets` in the same directory as the Python script.

For verification, run the `ultra_client` that is explained below **with four connections**. The server-side output should look similar to Example 1. In particular, **the `tickez_server` children must print their process IDs** at initialization!

Example 1

In a correct implementation for the `tickez_server`, this could be serve-side output of a session:

```
student@host:~/task-b$ python3 tickez_server.py
Server listening on 127.0.0.1:4711, dirpath: ./tickets
Child process with PID 39255
Child process with PID 39256
Child process with PID 39257
Child process with PID 39258
...
```

Note that each spawned child process prints its process ID. Your solution *must* do this as well!

The client-side output should look similar to Example 2. The strings `S-127.0.0.1:4711` and `C-127.0.0.1:40460` (or similar) refer to the server and client IP addresses and ports. The description refers to the requested ticket, the received file size, and whether it was successfully received.

Example 2

In a correct implementation for the `ultra_client`, this could be client-side output of a session:

```
student@host:~/task-b$ ./ultra_client 127.0.0.1 4711 4 42
[INFO] Ultra-Client connecting to 127.0.0.1:4711, using 4 threads, seed: 42
S-127.0.0.1:4711 C-127.0.0.1:41362 Ticket ticket_interior.txt requested
S-127.0.0.1:4711 C-127.0.0.1:41366 Ticket ticket_northstand.txt requested
S-127.0.0.1:4711 C-127.0.0.1:41368 Ticket ticket_vip.txt requested
S-127.0.0.1:4711 C-127.0.0.1:41372 Ticket ticket_interior.txt requested
S-127.0.0.1:4711 C-127.0.0.1:41366 Filesize 2428
S-127.0.0.1:4711 C-127.0.0.1:41368 Filesize 4769
S-127.0.0.1:4711 C-127.0.0.1:41362 Filesize 1235
S-127.0.0.1:4711 C-127.0.0.1:41366 Ticket ticket_northstand.txt received
S-127.0.0.1:4711 C-127.0.0.1:41372 Filesize 1235
S-127.0.0.1:4711 C-127.0.0.1:41372 Ticket ticket_interior.txt received
S-127.0.0.1:4711 C-127.0.0.1:41368 Ticket ticket_vip.txt received
S-127.0.0.1:4711 C-127.0.0.1:41362 Ticket ticket_interior.txt received
...
```

2 Client Description

To help you verify your implementation, the `ultra_client` executable is provided to you in the task folder. It can start multiple connections to the server simultaneously. The clients are started by running

```
student@host:~/task-b$ ./ultra_client <server_ip> <port> <connections> <seed>
```

with parameters for `server_ip`, `port`, `connections` and `seed`. The executable starts as many connections as specified by `connections` and each connects to *`server_ip`*`:`*`port`*. The parameter `seed` is used to initialize a random number generator to ensure that tickets are always requested in the same order over multiple sessions.

3 Protocol Format

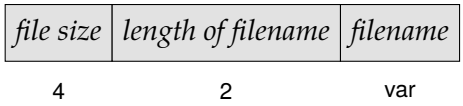
You may assume that a client only requests one ticket per session and an unlimited number of tickets are available. You can find the ticket names in directory `tickets`. The actual exchange of messages proceeds as follows:

1. First, the client sends a message to the `tickez_server` which contains
   - the length of the file name (without path) as *unsigned short*, and
   - the filename (without path) as *UTF-8-encoded string*.

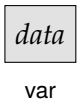
   The message format is as follows:

2. The `tickez_server` responds with a message which contains
   - the file size in bytes as *unsigned integer*,
   - the length of filename (without path) as *unsigned short*, and
   - the filename (without path) as *UTF-8-encoded string*.

   The message format is as follows:



3. Then the file content is transferred. The size has already been announced to the client in Step 2. The message format is as follows:



References

[1] https://youtu.be/0vcoMpDYOjk

Good luck!