*Operating Systems and Systems Programming*
Semester 1, 2024
Eike Ritter, David Oswald

The University of Birmingham
30 September, 2024

# Exercise 1

**Deadline: 14 November, 4pm**

## The Task

Write a set of programs which maintains the firewall configuation.

1. One program is a server program. Once started, the server should run forever. It waits for requests, process them and produce a specific response.

   The server maintains a collection of firewall rules. For each rule in the collection of rules, the server also maintains a list of pairs of IP addresses and ports which have been queried and found to match this rule. The server also maintains a list of all requests submitted to it. The server program should accept and process requests and return a response as follows:

   - Listing all requests in the order they were given. The command for this request is `R`
   - Adding a rule to the stored rules. This request expects a string which contains a firewall rule as specified below. If the rule is valid, the rule is added the set of firewall rules. The response is `Rule added` if the rule has been added, and `Invalid rule` if the string is not a valid rule. The command for this request is

     ```
     A <rule>
     ```

   - Checking whether a given IP address and port are allowed according to the rules. If this is the case, the IP address and port are added to list of matched queries for this rule. If the given address and port are allowed according to several rules, you should add the IP address and port to only one rule. If the given IP address and port is checked again, they may be added to a different rule. If the input does not provide a valid IP address and a valid port, the response should be `Illegal IP address or port specified`. If the IP address and port are valid and should be accepted according to the rules, the response should be `Connection accepted`. If the IP address and port are valid and should be rejected according to the rules, the response should be `Connection rejected`. The command for this request is

     ```
     C <IPAddress> <port>
     ```

- Delete a valid rule from the stored rules. This should also delete the list of IP addresses and ports stored for this rule. The rule should only be deleted if the server stores exactly the same rule. In all other cases, no action should be taken except returning a suitable message. If the specification of the rule is invalid, the response should be `Rule invalid`. If the rule is successfully deleted, the response should be `Rule deleted`. If the rule is valid but not found on the server, the response should be `Rule not found`. The command for this request is

  ```
  D <rule>
  ```
- Return all rules and for each rule list all IP addresses and ports which are stored with it. More precisely, for each rule, the response should be

  ```
  Rule: <rule>
  ```

  where `<rule>` is the specification of the rule as given below, followed by

  ```
  Query: <IPaddress> <port>
  ```

  for each query stored with this firewall rule. The command for this request is `L`
- For any other request, the response should be `Illegal request`.

There are two ways of interacting with the server program.

- The first one is to start the server with

  ```
      <ServerProgram> -i
  ```

  where `<ServerProgram>` is the way to call the server program, typically `./server`. In this mode the server reads an command from standard input, prints the response and waits for the next input.
- The second way of running the server is

  ```
  <ServerProgram> <port>
  ```

  where `<ServerProgram>` is the way to call the server program, typically `./server`, and `<port>` is the port on which the server listens for incoming connections. A client program, which you will have to write as well, makes it possible to send commands to the server program and display the results on standard output. The client should be called with

  ```
  <ClientProgram> <serverHost> <serverPort> <command>
  ```

  without any quotes, e.g.

  ```
      ./client localhost 2200 A 147.188.193.15 22
  ```

  All other ways of calling the client program should return an error message.

The specification of a firewall rule is below.

A rule is of the form

```
<IPAddresses> <ports>
```

where `<IPAddresses>` is either a single IP address, which has the form `xxx.xxx.xxx.xxx`, where `xxx` is a number between 0 and 255, or `<IPAddress1>-<IPAddress2>`, where `<IPAddress1>` and `<IPaddress2>` are IP addresses, and `<IPAddress1>` is smaller than `<IPaddress2>`. Assume that IP addresses and ports are separated by exactly one space character.

Similarly, `<ports>` is either a single port number, which is a number between 0 and 65535, or `<port1>-<port2>`, where `<port1>` and `<port2>` are ports and `<port1>` is smaller than `<port2>`.

Examples of such rules would be

```
147.188.193.0-147.188.194.255 21-22
147.188.192.41 443
```

Your server program should maximise the degree of concurrency and not contain any memory leaks.

## Submission and Marking

- You should submit the file `server.c` and `client.c` on the appropriate canvas-quiz. The marking scripts will execute the command `make` with the Makefile provided in the archive on canvas to produce all required binaries.
- The zip-archive on canvas contains a Makefile and a basic test script is provided in the file `test.sh`. The Makefile and the test script will only work if `server.c` and `client.c` are in the same directory as the Makefile and the test script respectively. The test script tests the cases of adding a rule both interactively and via sockets. It is run by changing into the directory containing the binaries and then running

  ```
  ./test.sh
  ```

  We will use more advanced scripts for marking which test all the specified functionality, including concurrency and memory leaks. If the basic test script does not work, all other scripts are likely to fail as well.
- Code which does not compile on the virtual machine provided will be awarded 0 marks.

## Hints

- You should start by implementing the interactive version of the server (the version running with `server -i`) and the `R`-command. Sockets and concurrency will be explained in later lectures.

- You may use the code provided in the lectures in your assignments as you see fit.