Lab 2.5. Sockets

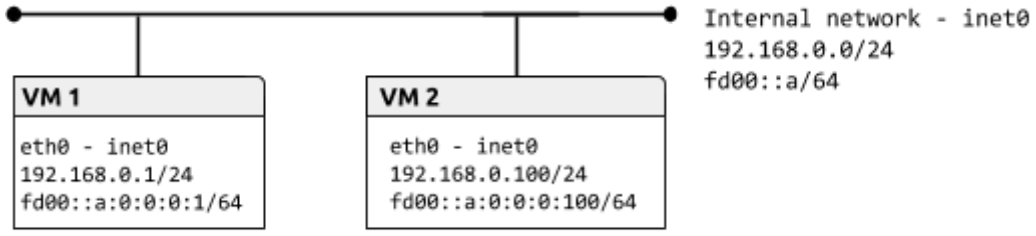
Objectives

In this lab, we will get used to the sockets programming interface, as a the base to program network-based applications. We will see the programming differences between UDP and TCP protocols. Finally, we will learn how to program applications independently of the network protocol family used (IPv4 or IPv6).

Contents

Environment Preparation

We will configure the network topology shown in the following figure. As in previous labs, we will create the topology with `vtopol` and an appropriate topology file. Configure the network interfaces as indicated in the figure and check the connectivity between the VMs.



Note: Observe that the VMs have a network interface with dual stack IPv6 - IPv4.

Address Management

Exercise 1. Write a program to get all the possible addresses to create a socket associated to a host given as the first argument. For each address, show the numeric address, the family and socket type. Check the result for:

- A valid IPv4 address (e.g. `147.96.1.9`).
- A valid IPv6 address (e.g. `fd00::a:0:0:0:1`).
- A valid FQDN (e.g. `www.google.com`).
- A valid name in `/etc/hosts` (e.g. `localhost`).
- Invalid examples of the previous cases.

The program will use the `getaddrinfo(3)` function to get a list of possible socket addresses (`struct sockaddr`). The numerical value of each address will be shown, using `getnameinfo(3)` with flag `NI_NUMERICHOST`, as well as the address family and socket type.

Note: To check the behaviour with the DNS service, **perform this exercise in the physical host**.

Example

```
# Families 2 and 10 are AF_INET and AF_INET6, respectively (see socket.h)
# Types 1, 2 and 3 are SOCK_STREAM, SOCK_DGRAM and SOCK_RAW, respectively
$ ./gai www.google.com
66.102.1.147 2     1
66.102.1.147 2     2
66.102.1.147 2     3
2a00:1450:400c:c06::67    10     1
2a00:1450:400c:c06::67    10     2
2a00:1450:400c:c06::67    10     3
$ ./gai localhost
::1    10     1
::1    10     2
::1    10     3
127.0.0.1    2     1
127.0.0.1    2     2
127.0.0.1    2     3
$ ./gai ::1
::1    10     1
::1    10     2
::1    10     3
$ ./gai 1::3::4
Error getaddrinfo(): Name or service not known
$ ./gai noexite.ucm.es
Error getaddrinfo(): Name or service not known
```

UDP Time Server

Exercise 1. Using as a base the UDP server studied in class, write a time server that:

- Takes the address and port as the first and second arguments of the program. Addresses can be specified in any format (host name, decimal notation…). Also, the server should work with IPv4 and IPv6 addresses.
- Receives a command codified in a character, so that ‘t’ returns the time, ‘d’ returns the date and ‘q’ finalizes the server.
- Prints the client name and port on each message, using function `getnameinfo(3)`.

Check the behaviour of the server with netcat (`nc` or `ncat`) as client.

Note: Since the server must work with IPv4 and IPv6 addresses, an appropriate structure is needed to accommodate any of them, for example, in `recvfrom(2)`. For that purpose, the API defines **`struct`** `sockaddr_storage`.

Example for the UDP time server

Server:

```
$ ./time_server :: 3000
2 bytes from ::FFFF:192.168.0.100:58772
2 bytes from ::FFFF:192.168.0.100:58772
2 bytes from ::FFFF:192.168.0.100:58772
Command not supported X
2 bytes from ::FFFF:192.168.0.100:58772
Exiting...
```

Client:

```
$ nc -u 192.168.0.1 3000
t
10:30:08 PMd
2014-01-14X
q
^C
```

Note: The server doesn’t send ‘\n’, the next command (marked with bold letters in the example) is shown in the same line as the answer.

Exercise 2. Write a client for the time server. The client will have as arguments the server address, the server port and the command, e.g. `./time_client 192.168.0.1 3000 t`, to request the time.

Exercise 3. Modify the server so that, besides receiving commands through the network, it is also able to receive them directly from the terminal, by reading two characters (command and '\n') from the standard input. Multiplex the use of both data channels using the `select(2)` function.

Exercise 4 (Optional). Transform the UDP server into a multi-process server following a *pre-fork* model. Once the socket is associated to the address with the `bind(2)` call, make a call to `recvfrom(2)` in different processes so that each one will attend a connection from a different client. Print the PID of the server process to check this.

TCP Echo Server

Exercise 1. Using as a base the TCP server studied in class, write an echo server that listens incoming connections in a given port. When an connection is received, it must show the client address and port. From that moment, it will send to the client all data received from it (echo). Check its behaviour with netcat (`nc or ncat`) as client. Check what happens if several clients try to connect at the same time.

Example for the TCP echo server	

Server:
`$ ./echo_server :: 2222`
`Connection from fd00::a:0:0:0:1 53456`
`Connection terminated`
`$`
	Client:
	`$ nc -6 fd00::a:0:0:0:1 2222`
	`Hello`
	`Hello`
	`How are you?`
	`How are you?`
	`^C`
	`$`

Exercise 2. Write a client to connect to the echo server. The client takes the server address and port as arguments. Once the connection with the server is established, the client will send what the user types and will show the answer received from the server. When the user writes the character 'Q' as the single character of a line, the client will close the connection with the server and will exit.

Example for exercise 2	

Server:
`$ ./echo_server :: 2222`
`Connection from fd00::a:0:0:0:1 53445`
`Connection terminated`
`$`
	Client:
	`$ ./echo_client fd00::a:0:0:0:1 2222`
	`Hello`
	`Hello`
	`Q`
	`$`

Exercise 3. Modify the server to handle each connection in a different process with `fork(2)`. The server will accept several simultaneous connections and each one will be managed in a child process (*accept-and-fork* model). The parent process must close the socket returned by `accept(2)`.

Exercise 4. Add the needed logic for the server to synchronize child processes (e.g. capturing the `SIGCHLD` signal), so that no process remains in *zombie* status.