Lab Session 1: Introduction to Distributed Systems Programming

April/21/2020

The objective of this lab session is to exercise the use of the Golang "net" package to setup inter-process communication. The "net" package provides a portable interface for network I/O, including TCP/IP, UDP, domain name resolution, and UNIX domain sockets. Although the package provides access to low-level networking primitives, most servers/clients will only need the implementations of the basic interface Dial, Listen, and Accept functions for TCP and the associated Connection (conn in the demo client0) and Listener (ln in the demo server0) objects – see sample code.

Exercise 1:

- Write a simple client-server program that can be run as two different processes (nodes) that by just changing the port number in the listing interfaces and the port and possibly IP addresses in the receiving sockets, they can send string messages to each other using TCP connections. Like in the server0.go and client0.go programs, messages to be sent are read from the terminal and messages received are print in the terminal. There is a message per line, and the communication ends when one of the two processes gets a line with the string "stop" as input. Note that the Dial operation (but not your program) might fail if the server in one of the process is not ready to listen for messages.

Observe that you write a single program and run two copies with different dialling and listening addresses – since a full address comprises an IP address and a port number the difference between addresses can be simply a different port number. You can use the same IP address (e.g., 127.0.0.1) in both processes and only change the listening ports. Hence you can test your two programs in your computer by running the programs in two different terminals.

Exercise 2 (Simple Chat room)

- Extend the program in Exercise 1, so that a process can send and receive string messages to and from multiple nodes which addresses are read from a configuration file. The first line in the file will be the address and port the program will use as the server interface, the other lines are the IP addresses and port of the neighbouring process servers. The name of this file will be passed as a command-line argument to the program when the program is started. Each line in the configuration file will have the following format:

  ```
  <IP address>:<Port number>
  ```

  An example of an input file is:

  ```
  127.0.0.1:6002
  10.80.29.90:6001
  127.0.0.1:6001
  ```

  The IP of the server reading this file is 127.0.0.1 and it will be listening for connections in port 6002. The string read in one terminal will be printed in the terminals of the neighbours.

Submitting your work for evaluation: Create a zip file with all the relevant files needed to run the two exercises together with a Readme.txt file that clearly explains how to test your programs. This is particularly important for Exercise 2. The test should be able to run in a single computer. For that all configuration files in your tests must use the loopback IP address 127.0.0.1 and only vary the port numbers. The Readme.txt file must include 1) the name or names of the authors, 2) a clear explanation of how to compile the program(s) (in case you decide to split your code into several packages), and 3) any information needed in order to test your programs. The evaluation will

consider how easy is to use and test your programs as well as good programming practices such code documentation and organization.

Named your zip file lab1_*<name>*_*<last name>*.zip and e-mail it to (jorge.lobo@upf.edu) no later than **April/28/2020** at 16:30.

Note: you will not be able to complete the second lab assignment before completing the exercises for this assignment.

The following text is from Source file src/net/dial.go and it gives examples of how addresses and ports can be specified:

```
229 // Dial connects to the address on the named network.
230 //
231 // Known networks are "tcp", "tcp4" (IPv4-only), "tcp6" (IPv6-only),
232 // "udp", "udp4" (IPv4-only), "udp6" (IPv6-only), "ip", "ip4"
233 // (IPv4-only), "ip6" (IPv6-only), "unix", "unixgram" and
234 // "unixpacket".
235 //
236 // For TCP and UDP networks, addresses have the form host:port.
237 // If host is a literal IPv6 address it must be enclosed
238 // in square brackets as in "[::1]:80" or "[ipv6-host%zone]:80".
239 // The functions JoinHostPort and SplitHostPort manipulate addresses
240 // in this form.
241 // If the host is empty, as in ":80", the local system is assumed.
242 //
243 // Examples:
244 //       Dial("tcp", "192.0.2.1:80")
245 //       Dial("tcp", "golang.org:http")
246 //       Dial("tcp", "[2001:db8::1]:http")
247 //       Dial("tcp", "[fe80::1%lo0]:80")
248 //       Dial("tcp", ":80")
249 //
```