

# תרגיל בשפת C

## הגשה בזוגות

### PART A: Learn to use datagram sockets by example.

Read and understand programs `Udp-Client.c` and `Udp-Server.c`. Compile the programs (C9) and execute them both in your local host.

**First**, run **Udp-Server** (note that this program contains an infinite loop, so you will want to put in the background with `&` so you can have your shell prompt back, or just open another terminal window).

**Then**, run **Udp-Client** several times, passing `localhost` as command line parameter, and observe what happens.

1. Go back to the source code and do your best to comment each line where a system call is issued explaining what the line accomplishes. Your comments should be short and straight to the point.
2. Write the code for a new function called `printsin()` according to the specifications below, uncomment the calls to `printsin` in `Udp-Server.c` and build the executable.

The function prototype must be:

```
void printsin(struct sockaddr_in *sin, char *pname, char* msg)
```

The function will print the string indicated by `pname` ended with a line break, then it will print the string indicated by `msg`, then the string `ip=` follow by the IP address of the host associated with this socket address structure in dotted-decimal notation, a comma, and finally the string `port=` followed by the port number associated with this socket address structure. When this function is called, it should produce output as in the examples below:

function call: `printsin(&s_in, "UDP-SERVER:", "Local socket is:");`

output:

UDP-SERVER:

Local socket is: ip= 0.0.0.0, port= 9000

function call: `printsin((struct sockaddr_in*)&from, "UDP-SERVER:", "Packet from:");`

UDP-SERVER:

Packet from: ip= 127.0.0.1, port= 33080

3. Change the server and client code as following: The client will send your name to the server and then will receive the server response. On the server side, the server will receive the client name (instead of the current msg structure) and will send back its name.

4. Copy and change the server and client code as following: node A wants to send messages to node C via node B (router). When node B receiving message from A, it first chose a random number and if it will greater than 0.3 it will forward it otherwise it will delete. On the vice versa, it always forwards it to A.

## Part B: IP addresses, hostnames and ... HTTP

- Use the files `tcp-server.c` and `tcp-client.c` and compile them. Review the programs to be sure you understand them.
- Compile both programs. In one terminal window, run `tcp-server`. In another, run `tcp-client`. Briefly explain what you see (by writing relevant comments in the code).
- Type the command `nslookup <hostname>`, where `hostname` is the name of the computer you are working on, to learn the IP address of that computer (you can use another way to check it as well).
- In `tcp-client.c`, change the definition of `IP_ADDRESS` so that it is the address of the computer you are working on. Recompile `tcp-client.c`.
- Now, run `tcp-server` and `tcp-client` again. Explain what you see.
- Suppose you run `tcp-client` when `tcp-server` is not running? Try this and explain what you see (use `TCPDUMP` to check your assumption).
- Wouldn't it be great if we could run `tcp-server` and `tcp-client` on any computer, without having to change the IP address and recompile the client? The way we will do this is by using the `getaddrinfo` system call. This function lets us supply a hostname as a string, and it will resolve that hostname to a result of type `struct sockaddr`. The program `nslookup.c` illustrates the use of `getaddrinfo`. Use this program, compile it, run it a few times supplying different hostnames as arguments, and review the code to understand how it works (add relevant comments in the code).
- Now, modify `tcp-client.c` so that it takes the hostname as a command-line argument. Use code from `nslookup.c` to resolve this hostname to a result of type `struct sockaddr`, and then use the result you obtain to connect to this address rather than the hard-coded `IP_ADDRESS`.

The `wget` program allows you to fetch the contents of a URL and save it to a new file. If you've never used `wget`, try using it to fetch the file named by

`http://www.yahoo.com`. In this part of the lab, we will build a simple analog to `wget`.

Our program will take a URL as a command-line argument and write the response from the web server to `STDOUT`.

- One problem we will face in building our simple web client is that of parsing URLs. Luckily, we can build a very simple web client while only parsing a limited class of URLs: those of the form `<protocol>://<hostname>/<path>` or `<protocol>://<hostname>:<port>/<path>`. The method illustrates a simple parser for URLs of this form.
- To put together your simple web client, make an appropriately named copy of your program and then make the following changes.
  - The command-line argument should be a URL rather than a hostname.
  - Parse the URL and connect to the hostname and port specified.

- The HTTP protocol, in its simplest form, is very, very simple. After connecting the socket, write a request of the following form (where `\n` indicates a newline character):

```
GET url HTTP/1.0\n
```

```
HOST: hostname\n
```

```
\n
```

**\*\*** when *url* is indicated, it's correlated to *path* from the URL re-structuring

- At this point, data from the web server should start arriving. Repeatedly read chunks of data from the socket into a character array and then write the data to STDOUT. You will know all of the data has been read when the return value from the `read(...)` syscall is zero (indicating 0 bytes read). When all of the data has been read, close the socket. (Note that the data read from the socket will *not* end with a null (`\0`) character. You'll need to account for this somehow when you are writing the contents of the buffer to STDOUT.)
- Examine the result of running your program from previous step for the URL `http://www.yahoo.com`. The *HTTP header* is separated from the contents of the file by a blank line. What information do you see in the header?
- Examine the result of running your program from step previous for the URL `http://www.yahoo.com/does-not-exist`. How is the HTTP header different from what you saw?

## **Hand in:**

- **All your code in a zip file**
  - **Don't forget to add comments in the code**
  - **Add pcap file with detail explanation about the messages in the file regarding your code**
- **Add doc file explaining what your program doing including notes stating whether your programs work to specifications or not. If you know that they don't behave correctly, make sure to explain what is right and what is wrong.**