– Lab2 –
Introduction to Unix Socket Programming

1 General Instructions

You can solve this lab individually or together with another student. If you work in a group of two, both students must participate actively in the implementation and demonstration of the solutions. Be sure to be well prepared prior to the scheduled lab and use the time to ask questions and to demonstrate your solutions.

Deliverables

- Oral demonstration of your solutions for the lab assistant. Be sure that you understand what you have done – it is not sufficient with a working solution.
- Submit your answers to Canvas. In case two students work together, both must submit their answers to Canvas and you must write both names on the answers.
- All program codes must be well documented (undocumented code will not be approved). As a minimum, you should include program name, author(s), date, how to execute the program, and a general description of the solution. Also, you should write comments on important sections in the code.
  Use your comments to answer the question `why` and now `how` (the code itself often explains how something is done)
- Make sure that you check the return values on all system/library calls (e.g., malloc, socket, etc.) that you use.
- Please check the deadline on Canvas. After this date the demonstration of Lab 3 will be prioritized. Lab 2 can be presented if there are free time-slots in the next lab occasions, or during the next instance of the course.

2 Understanding a simple client/server setup

The purpose of this lab is to give you a basic introduction to Unix socket programming.

Start by reading "Network Programming in Unix" (in `socket.pdf`) to get an introduction to the *Unix socket API*. There is also a lot of information about socket programming in the Unix environment on the web. It is strongly recommended to use Unix socket programming. In principle it is possible to use Windows and *WinSOCK*, Java to implement lab 2 but less help from lab assistants should be expected.

It should be noted that lab 2 requires concurrent sending and receiving of messages between the client and the server; therefore, it is expected that you have familiarity with handling of multitasking problems.

Once you got the feeling that you have some clue about the sockets, you can start trying out an example.

2.1 Example

Start by downloading `lab2.zip` file from Canvas. Next, open a terminal and extract the files using `unzip` as described in the appendix.

The `files` folder contains the following files:

- `server.c` – a simple server that allows multiple clients to connect. The clients send messages, and the server just prints them out.

- `client.c` – a simple client that connect to a server and send messages to it.
- `Makefile` – a makefile needed for compilation.

Compile the client and server by typing `make` in the directory where you extracted the files. You should now have two executable files (named `server` and `client`).

Determine the bostname of your computer by writing `hostname` in the terminal. The hostname should look something like itaXXXXX.u2-XXX.cam-student.mdh.se.

Next, start the server by writing `server` (or `./server`) in a terminal window. Do not start the server as a background process, that is do not append an ampersand (&) to the command. You should now get the following output:

```
[waiting for connections...]
```

The next step is to start the client. We will start by running both the server and the client on the same machine. Open a new terminal and write `client` <server_host_name> (or ./client <server_host_name>). Do not run the client in the background.

The server will now print the following message:

```
Server: Connect from client aaa.bbb.ccc.ddd, port XXXX
```

And the client prints the following message and waits for user input:

```
Type something and press [RETURN] to send it to the server.
Type 'quit' to nuke this program.
>
```

Try to send some messages and see what happens.

Start another client (by opening a new terminal and write `./client` <server_host_name>). Both clients are now connected to the server, and both can communicate with it. Write "quit" in the client window to terminate the program. Press `Ctrl+c` in the server window to terminate the server.

Ok, until now both the clients and the server are running ion the same machine. Log in to three different machines and try it out again. Example:

- first client is running on: `ita05979.u2-039.cam-student.mdh.se`
- second client is running on: `ita05988.u2-039.cam-student.mdh.se`
- and the server is running on: `ita05987.u2-039.cam-student.mdh.se`

NB: You do not need to kick out some of your buddies in order to get three different computers. You know that you can log into remote machines by using `ssh` <name_of_remote_host> command from the computer that you are currently using, right?

Finally, study the source code (in `client.c` and `server.c`) and try to understand how they work. Ask the lab assistant if anything is unclear.

3 Assignment

The example in the previous section only illustrates one-way communication, that is only client-to-server. Your assignment is to make the following modifications to `server.c` and `client.c`.

1. Modify the *server* to send an answer to the connected client(s). Just a simple answer like "I hear you, dude ..."

2. Modify the *client* to print out the answer from the server
3. Modify the *server* to inform all other clients (if any) when a new client connects (BROADCAST to all other clients)
4. Modify the *server* to refuse a connection from a particular client (hard-coded address in the server)

A Useful Linux commands

This appendix introduces some Linux commands that can be useful when working in the command shell (terminal). You can find more information about the commands by writing `man <cmd>`.

A.1 Working with directories

- `cd <dir>` – Change current working directory to `dir`. Use `cd ..` to move to the parent directory.
- `mkdir` <dir> – Make (create) a new directory
- `rmdir` <dir> – Remove directory `dir`.
- `pwd` – Print current working directory
- `ls [dir]` – List directory `dir`. If `dir` is omitted, the contents of the current directory is listed.

A.2 Working with files

- You can edit files using any text editor. Examples of editors includes *Kwrite*, *Kedit*, and *emacs*.
- Execute a file by writing `cmd`. Note: depending on the settings of your $PATH variable, you might need to write `./cmd`.
- `cp <src> <dst>` – Copy file `src` to `dst`
- `mv <src> <dst>` – Move (rename) file `src` to `dst`
- `rm <file>` – Remove (delete) `file`.
- `chmod <mode> <file>` – Change file permission on `file`. For example, `chmod +x file` sets execute permission on `file`.
- `unzip <file.zip>` – Unzip `file.zip`.
- `tar xvzf <file.tar.gz>` - Extracts files from `file.tar.gz`.

A.3 Managing processes

- `ps` – List running processes
- `kill <pid>` – Kill process with process identifier `pid`.

A.4 Working with remote systems

- `ssh <hostname>` – Log into a remote computer.