Faculty of Engineering *Department of Electrical and Computer Engineering*

ECE 428 – Computer Networks and Security
Project 1: TCP-UDP/IP Socket Programming – Soccer World Cup 2010
Spring 2010
Instructor: *Mahesh V. Tripunitara (tripunit@uwaterloo.ca*)
TA: *Ahmad R. Dhaini (adhaini@uwaterloo.ca)*

1. Objective

The purpose of this project is to make you familiar with the fundamental principles of computer communications. The project will also give you some insights on the client/server socket programming based on the two main Internet communications protocols – TCP and UDP over IP. Since the 2010 soccer World Cup is approaching, you will be required to use these protocols to form the national soccer team of a selected country ☺.

2. Description

Before you start with this project, make sure you understand the main functionalities of TCP and UDP over IP, in particular how to setup a connection using TCP and UDP.

In this project, you are required to implement, using either C/C++ or JAVA programming language, a *client* and a *server* communicating using a UDP connection, and another *client* and *server* communicating using a TCP connection. You are also required to allow each *client* to send a list of soccer players (read from a given input file) and a selected country to the *server*, which will be responsible for forming the national soccer team of the selected country to be sent back to the *client*. The *client* will then print the list of selected players (or the national team) in an output file.
These are the steps that your program should be performing:

I. Given an input file `in.dat` and the name of the country (e.g., Brazil), the client should send each player's name and country to the *server*.

II. Once all the players are received at the *server* side, all the players of the selected country only are sent back to the client.

III. Once all the selected players are received at the *client* side, the *client* will print their names (without the country name) in an output file (`outTCP.dat` or `outUDP.dat`).

a. Compiling your code

You are expected to write a `makefile` that will compile and link your code. A sample `makefile` for Java under Linux can be found at the following link:

http://www.cs.swarthmore.edu/~newhall/unixhelp/javamakefiles.html

Once written, you should be able to run `make`, and get your `.class` file(s).

A sample `makefile` for C/C++ can also be found at the following link:

http://mrbook.org/tutorials/make/

b. Running your program

Running your program should be done using `make run` for each of the UDP and TCP client and server in the following way:

```
> make runTCP
> make runUDP
```

If you are using JAVA, the `make runTCP` command could look like this:

```
> java serverTCP &
> java clientTCP in.dat Brazil
```

c. Expected output

Your programs should generate the following two files:

- `outTCP.dat` : This will contain the list of players of the selected country transferred using the TCP server and client.
- `outUDP.dat` : This will contain the list of players of the selected country transferred using the UDP server and client.

3. Things to Consider

a. IP address

The client and the server do not have to run on different machines. Hint: use loopback address to setup your communication.

b. End of transmission notification

The *server (client)* cannot know when all the players are received unless the *client (server)* notifies him that there are no more messages to be sent. The implementation of such a notification is left to your intuition.

c. Port selection

A static port may be used when *binding* the socket to the *server*. However, the assigned port might not be available at the machine (or *ecelinux* server). One solution might be to *bind* to *null*, which will generate a random available port for you. This port may then be placed in some repository (or file) from where the *client* can it pick up to connect to the *server*. You may also consider other solutions.

d. Compiling java code on ecelinux

To compile your java program on the *ecelinux* server, you must use `gcc` (i.e., `gcj –C`) because `javac` is not installed. Example:

```
> gcj –C program.java
```

e. Terminating the background process

If you choose to run the *server* in background (i.e., `&`), make sure you kill the process after you are done with testing your program. The process id (PID) is normally echoed once a process is run in background. However, you can always use the `ps` command to get it; then use `kill` with the PID as argument to terminate the process.
Failing to do so can exhaust the *ecelinux* server!

4. To Submit

Since each project is done in-group, the group must select a *delegate* at the beginning of the term and only him is allowed to submit the project on behalf of his group. Each group will be assigned a group ID (or number), which will be sent via UW-ACE to all students.

Each project should be submitted as a gzipped tar file through UW-ACE. The naming convention should be as follows:

gN_pM.tar.gz

N stands for the group number and *M* stands for the project number. For example, group 20 shall submit project 1 using:

g10_p1.tar.gz

If you make a mistake during a submission, you may simply resubmit, and the original submission will be ignored. If you feel that the wrong file was marked (UW-ACE requires to manually select the most recent project), please contact the TA.
Make sure to use lower case for "p" and "g"; otherwise your submission will be ignored.

Your gzipped file should contain the following files:

- `serverUDP.java – or – serverUDP.c(cpp)`
- `clientUDP.java – or – clientUDP.c(cpp)`
- `serverTCP.java – or – serverTCP.c(cpp)`
- `clientTCP.java – or – clientTCP.c(cpp)`
- `Makefile`

Do not submit any input file or the generated `.class (.o)` files. That is, make sure that you clean your directory from any file that is not listed above using the following:

```
> make clean
```

Note: Your program is expected to compile and run on the *ecelinux* server.

5. Grading Policy

Grading will be done as follows:

Total is 100 points. You will receive:

- 20 % if your program compiles, but does not work at all.
- 0 points if your program does not compile.
- 0 points for identical (or very similar) programs.
- 60 % if your program passes the basic test case, but is badly written[1].
- 65 % if your program passes the basic test case and is well written[2].
- 95 % if your program passes all the test cases, but is badly written.
- 100 % if your program passes all the test cases and is well written.

Late submissions will not be accepted.

It's better to have something working, even with little functionality, than a program that crashes (or doesn't compile). It is expected that you submit your own program; otherwise you will get a 0 for the project.

6. Deadline

Friday May 21, 2010 before 11:55 PM.

[1] Badly written code means that the code is neither commented nor follows the *programming style*. A good programming style guideline can be found at: http://en.wikipedia.org/wiki/Programming_style
[2] Reviewing your grade because of programming style is not allowed.