CS3205: Introduction to Computer Networks

Assignment 2 [Deadline: 8th April, 2022]
Total Marks: 100 [Weightage: 20%]

1. TicTacToe on a LAN [60 Marks]

You have to design and implement a two-player tic-tac-toe game using the traditional client-server architecture. In tic-tac-toe, there are totally two players and a 3×3 empty board so that if player 1 chooses a symbol 'X' and then player 2 chooses 'O'. if any column or any row or any cross gets filled with the same symbol then the player corresponding to that symbol will win. The players will act as clients and the server should handle those two clients as mentioned below.

(You can refer to this link for more info about tic tac toe game)

1. The game server should be first started and wait for players (choose a port of your choice) to join in.

```
$SERVER$ ./gameserver
$SERVER$ Game server started. Waiting for players.
```

2. The client joins, he is assigned an ID and he needs to wait for a game partner...

```
$CLIENT1$ ./gameclient
$CLIENT1$ Connected to the game server. Your player ID is
1. Waiting for a partner to join . . .
```

3. The second client joins.

```
$CLIENT2$ ./gameclient
$CLIENT2$ Connected to the game server. Your player ID is
2. Your partner's ID is 1. Your symbol is 'X'
$CLIENT2$ Starting the game …
$CLIENT2$ __ | __ | __
$CLIENT2$ __ | __ | __
$CLIENT2$ __ | __ | __
```

4. Meanwhile at client 1,

```
$CLIENT1$ Connected to the game server. Your player ID is
1. Waiting for a partner to join . . .
$CLIENT1$ Your partner's ID is 2. Your symbol is 'O'.
$CLIENT1$ Starting the game …
$CLIENT1$ __ | __ | __
$CLIENT1$ __ | __ | __
$CLIENT1$ __ | __ | __
```

```
$CLIENT1$ Enter (ROW, COL) for placing your mark: 1 3
$CLIENT1$ __ | __ | O
$CLIENT1$ __ | __ | __
$CLIENT1$ __ | __ | __
```

Handle illegal entries, for example a player cannot make two successive entries or cannot put a position that is already occupied etc. When a player makes an entry the updated state of the game should be visible to all players simultaneously and the relevant player should be prompted to enter his intended move.

If any of the partners quits/disconnects, the gameserver should display a message to the other player (e.g., “Sorry, your partner disconnected”) and disconnect him.
When a particular player wins or the game is a draw, this message will be displayed in both the player terminals and they need to be asked whether to replay. The game will *only* be replayed if both players enter YES, otherwise both the players will be disconnected from the server.Once you have this basic functionality implemented you can consider the following add-ons. **[30 marks]**

Extra features:

a. Extend the basic code you have developed to support multiple games simultaneously. When there are an odd number of players, the one who joined last will wait for some player to join in and a new game starts with a new game ID. **[10 marks]**

b. Time outs: If a certain player takes more than 15 seconds to make a move, the game will be automatically quitted, the player and his partner will be asked whether they want to replay. If yes, a new game starts with a new game ID, else they will be disconnected from the server. **[10 marks]**

c. Server maintains and updates all game related statistics in a log file. For example, time of the game, how long the game lasted, which player won and the sequence of moves given by the players **.[10 marks]**

2. YAPP - Yet Another Ping Program [40 marks]

You need to write a very simple version of the ping program (*yapp*) in C/C++ to find out whether a remote networked system is alive and responds to ping requests. *yapp* must be compatible with the conventional ping program (even if with bare minimum functionality) that is ubiquitous in all operating systems.

Functional requirements:

```
$ ./yapp X.X.X.X
```

Takes and checks against a valid IPv4 address (X.X.X.X) provided as the sole command line argument. [helpful libraries/functions for handling IPv4 address strings, https://linux.die.net/man/3/inet_aton] and sends one ping request.

- If the provided IPv4 address is invalid, it should print a console message “Bad hostname”
- If the intended host is unreachable or the request timed out (you can use a timeout threshold), it should print a console message “Request timed out or host unreacheable”
- Otherwise, it should print a console message “Reply from X.X.X.X. RTT = N ms” X.X.X.X is the valid IPv4 address and N is the round trip time in milliseconds.

Start with the ping’s man-page here, https://man7.org/linux/man-pages/man8/ping.8.html. In essence, ping sends an ICMP ECHO_REQUEST datagram that triggers the remote system to send an ECHO_RESPONSE which is intercepted by the local host. You would be interested to know what exactly happens (in terms of system calls invoked) when a ping is executed. One quick (hacky) way is to make use of the Linux utility `strace` for understanding the sequence of syscall events that fire under the hood.

```
# strace ping 8.8.8.8
```

Pay particular attention to the calls relating to sockets, socket read/write and their respective arguments.

`socket(AF_INET, SOCK_DGRAM, IPPROTO_ICMP)` - example syscall instance

This simple hacking session will unearth the basic skeleton that is needed for *yapp* to work. You simply need to stitch together these invocations from your code and get it working. Ignore packet corruptions or checking against corrupted packets using ICMP checksum.This RFC (http://www.networksorcery.com/enp/rfc/rfc792.txt) has all details relating to ICMP, this will be particularly useful when populating the various fields in the ICMP packet header.When use a RAW socket, i.e. IPPROTO_RAW as the protocol type, we have to set the ICMP identifier when sending and check if it’s the same on receipt of an ICMP reply that whether it is meant for us or not. We don’t need to do that for IPPROTOCOL_ICMP since the kernel automatically does all those bookkeeping tasks for us.

A quick scan through a real version of the ping program may be helpful, https://github.com/iputils/iputils/blob/master/ping/ping.c This can be overwhelming at first, but after making a few passes you will observe that it handles a lot of special conditions, input

arguments/flags and corner cases that your *yapp* will not need. You need to extract the barebone functionality needed to support the minimal functionality as stated above. In all, don't expect your code to exceed way more than 150 lines in C, including comments.

All code needs to be implemented in C/C++. Submit your code in a zip file with proper documentation and references. This code will be subjected to plagiarism checks across students and several public domains. You will also be asked to run the code and show a demo to the TA. Note that we will test your code across multiple physical systems within the same LAN, i.e., each client and the server will run in separate systems.