Exercise 1: Introduction to Socket Programming

1 Goal

Gain experience with socket programming by implementing a simple game (Tic-tac-toe) using both UDP and TCP sockets. In this game, two players take turns marking on spaces in a 3 x 3 grid. The player who first successfully in marking all spaces in a row, in a column or in a diagonal line wins the game.

2 Overview

In this exercise, you will implement a networking version of Tic-tac-toe, which will comprise two C++ programs, a client program and a server program. Two players run two client programs that interact with the server program to simulate the gameplay of Tic-tac-toe. This exercise will help you to gain experience with socket programming using Berkeley socket interface. In order to focus more on the details of socket programming part, the game and most of the command parsing are provided as a skeleton code.

3 Specification

In this exercise, you will implement a client and a server program for the game Tic-tac-toe. The server program and client program will interact by exchanging messages to simulate the gameplay. A simple message/packet format and a simple protocol are provided in the file: packet.h. The server and client programs will handshake through TCP and the gaming interaction will base on UDP.

3.1 Tic-tac-toe

The server program hosts a Tic-tac-toe game for one pair of client programs (players). For each pair of client programs, the server program hosts a board consists of a 3 x 3 grid. Initially all the grids are empty. The server program sends the board to the client programs and also informs one of the client program that it is its turn. The client program who is taking the turn marks a grid by telling the server program which position it wants to mark. Upon receiving the position to be marked from the client program, the server marks the
grid, sends the updated board to both players and lets the other client program take the next turn. This board marking keeps going until one player who marked all grids in a row, in a column or in a diagonal line wins the game or until the game ties (that all grids are marked without a winner).

### 3.2 packet.h: packet format and protocol

The packet in the lab are defined as having only two fields: 1. message type: unsigned int integer and 2. message buffer: char buffer[256]. The protocol consists of several message types defined in packet.h. The protocol is visualized in Figure 1.
3.3 The server program

Example invocation: ./lab1_server

The server program that takes no argument is responsible for listening for incoming client requests. For each pair of clients, the server program handles the Tic-tac-toe gameplay for them.

The server program creates a TCP socket and waits for incoming TCP connections. This TCP socket’s port will be assigned by the operating system and printed to the console. We assume that the clients know this TCP port number, because the clients are started after the server. When a client connects to the server, it sends a JOIN message to the server and the server assigns the mark (either nought O or cross X) to the client in a JOIN_GRANT message.

After two clients have join the game, the server program creates a UDP socket, whose port is also assigned by the operating system. The server program waits a GET_UDP_PORT message from the pair of client programs. The server program discards ANY message that is not GET_UDP_PORT, and in this case, terminates the program. If the incoming packet’s type is GET_UDP_PORT, the server program responds a message of type UDP_PORT. Via the TCP socket, the server program returns a packet of type UDP_PORT and UDP port number in the buffer.

The game (marking the grid, sending the board and exiting) is played using the UDP socket. When both client programs get the UDP port, they start playing the game by sending GET_BOARD message. The GET_BOARD message’s buffer comes with the mark assigned to the client. The server distinguishes the clients by their marks (either nought O or cross X). The server program determines which client program takes the turn and respond a YOUR_TURN message to only to that client.

Suppose the two clients are denoted as client A and client B and client A is the client that takes the first turn. The server does not send any message to client B before client A has done her turn. When client A receives a YOUR_TURN message, the player will be prompt that it is her turn and she is allowed to mark a grid. The player can specify which position she wants to mark. For example, if the player wants to mark on position E by entering the command MARK E. Client A sends a PLAYER_MARK message including the position to be marked to the server program. The server program checks if client A has won the game and responds according to the game result. If client A has won the game, the server sends a YOU_WIN message to client A and sends a YOULOSE message to client B. If the game ties after client A’s move, the server sends TIE message to both client A and B. If client A has not won the game, the server sends a UPDATE_BOARD message including the updated board to both client A and client B. When client A and client B receives the UPDATE_BOARD message, both of them sends a GET_BOARD message for next turn. The server sends YOUR_TURN to client B this time. These steps repeat until one of the client wins or the game ties when all grids are marked. Moreover, when a client sends an EXIT message, the server grants this exit, returns an EXIT_GRANT to both clients and terminates the child server process hosting the game. More games can be created by the parent server process.
A skeleton server code is provided: `lab1_server.cc`. A `parse_args()` function is also provided in `lab1_server.h`.

### 3.4 The client program

Example invocation: `./lab1_client -p 48192 -s homer.cse.msu.edu`

The client program is required to accept the following arguments.

- `-s` is the server address (domain name or IP address).
- `-p` is the TCP port number that the server listen for incoming connection.

The client resolves the server address using `gethostbyname()` and connects to this server over TCP. The client then sends a new game request `JOIN` to the server, in order to inform the server that she wants to play the game. The server responds with a `JOIN_GRANT` message to the client along with the mark (nought O or cross X) assigned to the client. As mentioned in previous section, when there are two clients connected to the server, the server program creates a UDP socket. Both client programs send a `GET_UDP_PORT` message to get the UDP port number.

After the clients obtain the UDP port number, a game is created at the server side. The client programs send `GET_BOARD` messages to start a turn. The server determines who gets this turn and respond to that client program with a `YOUR_TURN` message. On receiving `YOUR_TURN` message, the client program prompts the player and asks which position she wants to mark. When the player enters the position she wants to mark, a `PLAYER_MARK` message will be sent to the server. The server responds different kinds of messages according to the game result, as mentioned in previous section.

Note that the message `JOIN` is sent by the client program automatically right after the client’s TCP connection to the server is established. The player does NOT need to issue a `JOIN` command and neither does the player need to issue `GET_UDP_PORT`. The player can only issue two types of commands: `MARK` and `EXIT`. The parsing function `get_command()` that only accepts those two commands is provided in `lab1_client.h`.

A skeleton client code is provided: `lab1_client.cc`. Several helping functions, including command parsing and argument parsing, are provided in `lab1_client.h`.

### 4 Deliverables

This exercise is optional. It has no due date. This exercise should be used to improve your understanding of sockets.
5 Example

Follows is an example of output from the client and server. Your output may differ.

1. Invoke the server (executed on ned.cse.msu.edu)
   ./lab1_server
   [SYS] Parent process for TCP communication.
   [TCP] Tic-Tac-Toe server started...
   [TCP] Port: 57775

2. Invoke the first client. The client connects to the server. The server responds with a JOIN_GRANT message and assigns the mark 0 to first client. The first client sends a GET_UDP_PORT right after it gets JOIN_GRANT.
   First client:
   ./lab1_client -s ned.cse.msu.edu -p 57775
   [TCP] Tic Tac Toe client started...
   [TCP] Connecting to server: ned.cse.msu.edu:57775
   [TCP] Sent: JOIN
   [TCP] Rcvd: JOIN_GRANT 0
   [TCP] Sent: GET_UDP_PORT

   Server:
   >./lab1_server
   [SYS] Parent process for TCP communication.
   [TCP] Tic-Tac-Toe server started...
   [TCP] Port: 57775
   [TCP] Recv: JOIN
   [TCP] Sent: JOIN_GRANT 0

3. Invoke the second client. The client connects to the server. The server responds with a JOIN_GRANT message and assigns the mark X to second client.
   Second client:
   >./lab1_client -s ned.cse.msu.edu -p 57775
   [TCP] Tic Tac Toe client started...
   [TCP] Connecting to server: ned.cse.msu.edu:57775
   [TCP] Sent: JOIN
   [TCP] Rcvd: JOIN_GRANT X
   [TCP] Sent: GET_UDP_PORT

   First client:
./lab1_client -s ned.cse.msu.edu -p 57775
[TCP] Tic Tac Toe client started...
[TCP] Connecting to server: ned.cse.msu.edu:57775
[TCP] Sent: JOIN
[TCP] Rcvd: JOIN_GRANT O
[TCP] Sent: GET_UDP_PORT

Server:
>./lab1_server
[SYS] Parent process for TCP communication.
[TCP] Tic-Tac-Toe server started...
[TCP] Port: 57775
[TCP] Rcv: JOIN
[TCP] Sent: JOIN_GRANT O
[TCP] Rcv: JOIN
[TCP] Sent: JOIN_GRANT X

4. Right after the second client connects, the server program creates a UDP socket. It receives the two GET_UDP_PORT messages and respond with the UDP port to the clients.

Server:
[TCP] Rcv: GET_UDP_PORT
[TCP] Sent: UDP_PORT 47898

5. Both clients receive the UDP_PORT message and both clients send GET_BOARD message. The first client gets the first turn. The second client does not get any response and it is blocked.

First client:
[UDP] Sent: GET_BOARD O
[SYS] Waiting for response ...
[UDP] Rcvd: YOUR_TURN _________
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>D</td>
<td>E</td>
<td>F</td>
</tr>
</tbody>
</table>

6
Second client:

......
[TCP] Rcvd: JOIN_GRANT X
[TCP] Sent: GET_UDP_PORT
[TCP] Rcvd: UDP_PORT 47898
[UDP] Sent: GET_BOARD X
[SYS] Waiting for response ...

Server:

......
[TCP] Recv: GET_UDP_PORT
[TCP] Sent: UDP_PORT 47898
[TCP] Recv: GET_UDP_PORT
[TCP] Sent: UDP_PORT 47898
[UDP:47898] Rcvd: GET_BOARD O
[UDP:47898] Sent: YOUR_TURN _________
[UDP:47898] Rcvd: GET_BOARD X

6. The first client marks position E. Note that the upper left grid was 0 (zero) before player mark 0 and it becomes O (nought) after player mark 0.

First client:

......
[UDP] Rcvd: YOUR_TURN _________
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>D</td>
<td>E</td>
<td>F</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>G</td>
<td>H</td>
<td>I</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
[SYS] Your turn.
[CMD] MARK E
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
```plaintext
<p>| D | O | F |
|---+---+---|</p>
<table>
<thead>
<tr>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
</table>

[UDP] Sent: PLAYER_MARK E
[UDP] Rcvd: UPDATE_BOARD ____O____

Second client:

......

[UDP] Sent: GET_BOARD X
[SYS] Waiting for response ...
[UDP] Rcvd: UPDATE_BOARD ____O____
[UDP] Sent: GET_BOARD X
[SYS] Waiting for response ...
[UDP] Rcvd: YOUR_TURN ____O____

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>D</td>
<td>O</td>
<td>F</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>G</td>
<td>H</td>
<td>I</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>

[SYS] Your turn.

[CMD]

Server:

......

[TCP] Sent: UDP_PORT 47898
[UDP:47898] Rcvd: GET_BOARD O
[UDP:47898] Sent: YOUR_TURN _________
[UDP:47898] Rcvd: GET_BOARD X
[UDP:47898] Rcvd: PLAYER_MARK E

7. The second client marks position 3. Second client:

......

[UDP] Rcvd: YOUR_TURN ____O____

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>D</td>
<td>O</td>
<td>F</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>G</td>
<td>H</td>
<td>I</td>
</tr>
</tbody>
</table>
```

8
---|---|---|
[SYS] Your turn.
[CMD] MARK I
---|---|---
| A | B | C |
---|---|---
| D | O | F |
---|---|---
| G | H | X |
---|---|---
[UDP] Sent: PLAYER_MARK I
[UDP] Rcvd: UPDATE_BOARD ____O____X
[UDP] Sent: GET_BOARD X
[SYS] Waiting for response ...

First client:
......
[UDP] Sent: PLAYER_MARK E
[UDP] Rcvd: UPDATE_BOARD ____O____
[UDP] Sent: GET_BOARD O
[SYS] Waiting for response ...
[UDP] Rcvd: UPDATE_BOARD ____O____X
[UDP] Sent: GET_BOARD O
[SYS] Waiting for response ...
[UDP] Rcvd: YOUR_TURN ____O____X
---|---|---
| A | B | C |
---|---|---
| D | O | F |
---|---|---
| G | H | X |
---|---|---
[SYS] Your turn.
[CMD]

8. The client keeps playing until a winner is generated or the game is a tie.