Operating Systems 2023-2024

Session 5: Sockets (II) 02/11/23

Goal

The objective of this session is to get a first introduction to the sockets, a mechanism of communication between processes that may be running on the same or different systems.

Motivation

More specifically, with this session, the student has to exercise:

- Creation of sockets (socket)
- Establishment of connections (connect)
- Connection closure (close)
- Sending messages between client and server (read, write)
- Passing parameters to an executable (argc, argv)

Previous documentation

To complete this session, it is recommended that you read the following references:

SALVADOR J., CANALETA X. (2014). *Programació en UNIX per a pràctiques de Sistemes Operatius*, Publicacions d'Enginyeria i Arquitectura La Salle (Edició PDF). Capítulo 7. Pág. 97 en adelante.

Tic Tac Toe

In the last session, La Salle Bonanova asked us to make the client for a Tic Tac Toe game and that the **server** will be done by someone else. Apparently, the server that was done by the 3rd part provided was not very good, since the students of SO did an excellent with the client implementation, they have asked as to make the server from scratch.

As always, they have asked us to make the game at last minute and with urgency… So, we only wave the time of this session to develop it. We are lucky that we only need to implement the server, since the client was done last session. The communication process between the client and the server will be made with the following frames:

NewPlayer
To start a game, the player sends to the server: **NP#<player name>#**
The server answers acknowledging he has received the player's name: **NP#OK#**

StartGame
Once two players have joined the game, the server tells both players that the game can be started and which piece the player is going to use: **SG#X#** / **SG#O#**
Note: Here the client must indicate if using X or O and display an empty board.

YourTurn and MyPiece
Every time it is a player turn, the server will notify to the client with the frame: **YT#**
The client must answer to the server indicating which row (1-3) and column (A-C) is going to be placed the piece: **MP#<column>#<row>#**
Note: The client should update the board with the new piece.

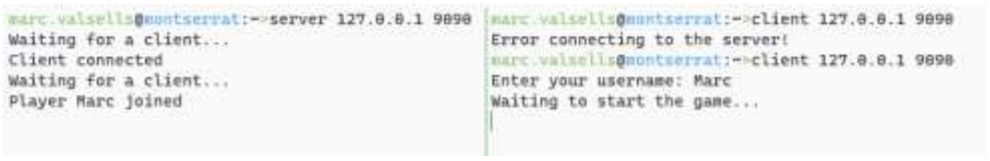
NewPice
When the other player places a piece, the server will send to the client which position has been occupied: **NP#<column>#<row>#**
Note: The client should update the board with the new piece.

EndGame
When a player wins, the server will send to both clients the name of the winner, in case of a tie it will send both players names: **EG#<winner/s name/s>#**

A function to check if a player has won or if there is a tie after every move is provided below in this statement, it is up to you if you want to use it or modify it for your implementation of the server.

Execution example

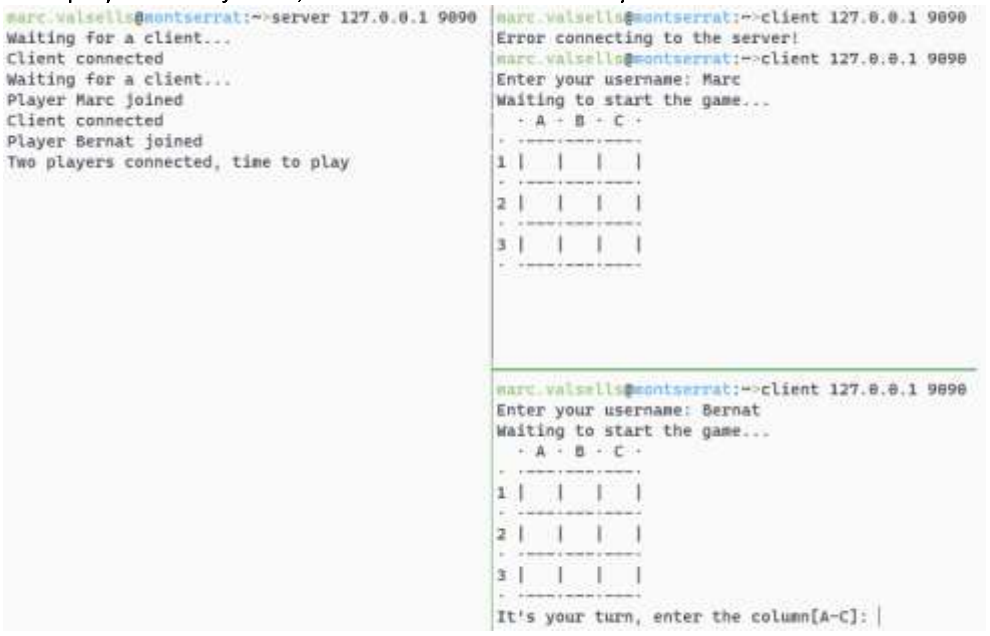
We can see an error when server is not running, and then how once it's started the 1st player is connected.



```
marc.valsells@montserrat:~>server 127.0.0.1 9090
Waiting for a client...
Client connected
Waiting for a client...
Player Marc joined
```

```
marc.valsells@montserrat:~>client 127.0.0.1 9090
Error connecting to the server!
marc.valsells@montserrat:~>client 127.0.0.1 9090
Enter your username: Marc
Waiting to start the game...
```

Both players have joined, no one has made a move yet.

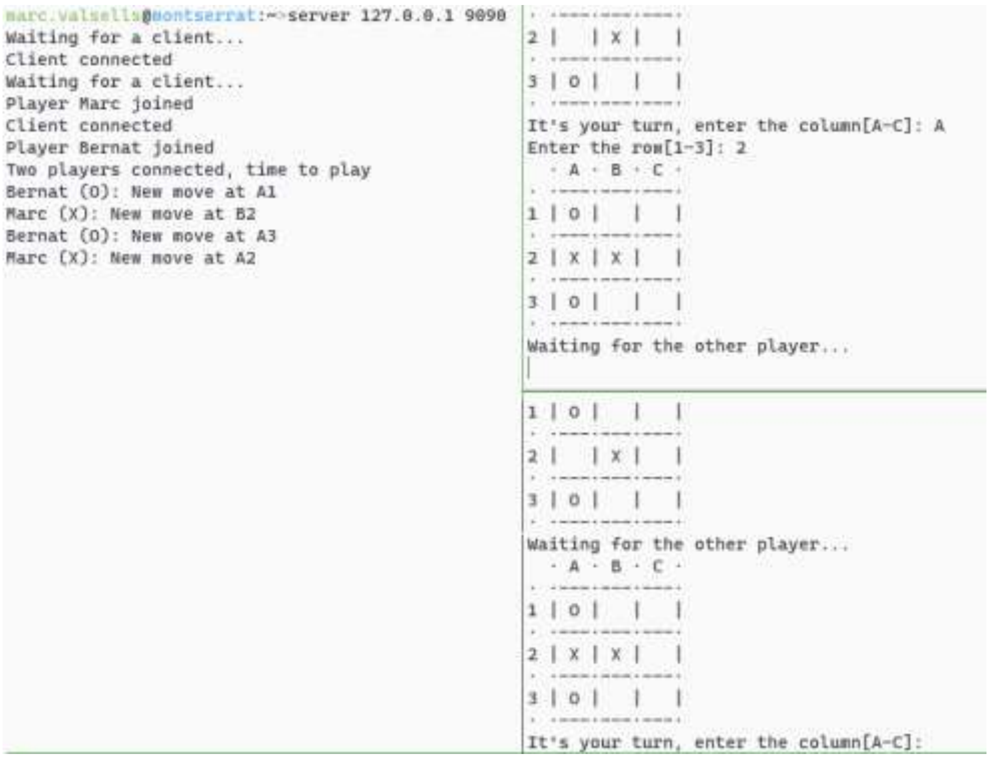


```
marc.valsells@montserrat:~>server 127.0.0.1 9090
Waiting for a client...
Client connected
Waiting for a client...
Player Marc joined
Client connected
Player Bernat joined
Two players connected, time to play
```

```
marc.valsells@montserrat:~>client 127.0.0.1 9090
Error connecting to the server!
marc.valsells@montserrat:~>client 127.0.0.1 9090
Enter your username: Marc
Waiting to start the game...
   · A · B · C ·
·  ·---·---·---·
1  |   |   |   |
·  ·---·---·---·
2  |   |   |   |
·  ·---·---·---·
3  |   |   |   |
·  ·---·---·---·
```

```
marc.valsells@montserrat:~>client 127.0.0.1 9090
Enter your username: Bernat
Waiting to start the game...
   · A · B · C ·
·  ·---·---·---·
1  |   |   |   |
·  ·---·---·---·
2  |   |   |   |
·  ·---·---·---·
3  |   |   |   |
·  ·---·---·---·
It's your turn, enter the column[A-C]: 
```

After several turns of both players.

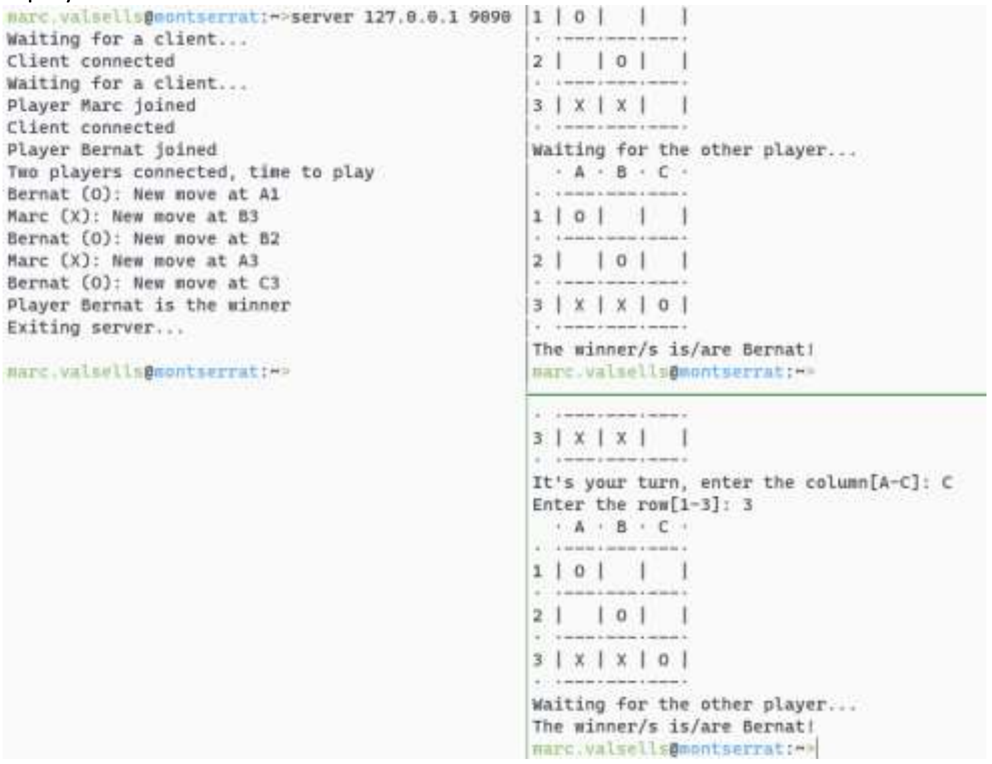


```
marc.valsells@montserrat:~>server 127.0.0.1 9090
Waiting for a client...
Client connected
Waiting for a client...
Player Marc joined
Client connected
Player Bernat joined
Two players connected, time to play
Bernat (O): New move at A1
Marc (X): New move at B2
Bernat (O): New move at A3
Marc (X): New move at A2
```

```
. .---.---.---.
2 |   | X |   |
. .---.---.---.
3 | O |   |   |
. .---.---.---.
It's your turn, enter the column[A-C]: A
Enter the row[1-3]: 2
  · A · B · C ·
. .---.---.---.
1 | O |   |   |
. .---.---.---.
2 | X | X |   |
. .---.---.---.
3 | O |   |   |
. .---.---.---.
Waiting for the other player...
```

```
1 | O |   |   |
. .---.---.---.
2 |   | X |   |
. .---.---.---.
3 | O |   |   |
. .---.---.---.
Waiting for the other player...
  · A · B · C ·
. .---.---.---.
1 | O |   |   |
. .---.---.---.
2 | X | X |   |
. .---.---.---.
3 | O |   |   |
. .---.---.---.
It's your turn, enter the column[A-C]:
```

A player has made 3 in a row and wins.



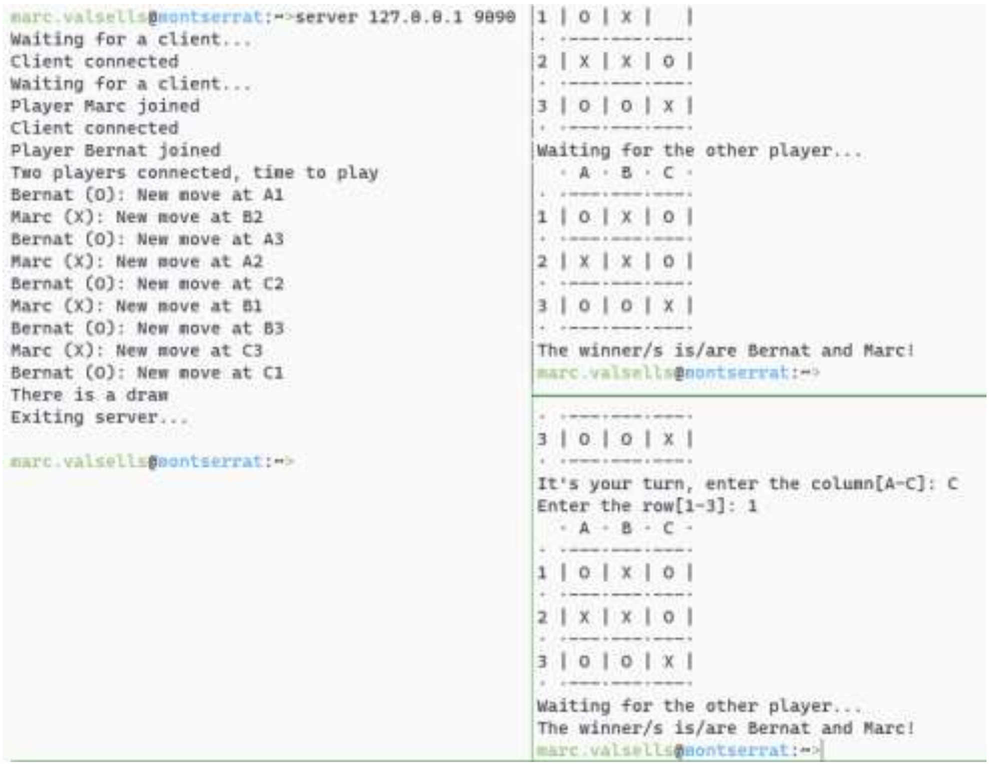
```
marc.valsells@montserrat:~>server 127.0.0.1 9090
Waiting for a client...
Client connected
Waiting for a client...
Player Marc joined
Client connected
Player Bernat joined
Two players connected, time to play
Bernat (O): New move at A1
Marc (X): New move at B3
Bernat (O): New move at B2
Marc (X): New move at A3
Bernat (O): New move at C3
Player Bernat is the winner
Exiting server...

marc.valsells@montserrat:~>
```

```
1 | O |   |   |
. .---.---.---.
2 |   | O |   |
. .---.---.---.
3 | X | X |   |
. .---.---.---.
Waiting for the other player...
  · A · B · C ·
. .---.---.---.
1 | O |   |   |
. .---.---.---.
2 |   | O |   |
. .---.---.---.
3 | X | X | O |
. .---.---.---.
The winner/s is/are Bernat!
marc.valsells@montserrat:~>
```

```
. .---.---.---.
3 | X | X |   |
. .---.---.---.
It's your turn, enter the column[A-C]: C
Enter the row[1-3]: 3
  · A · B · C ·
. .---.---.---.
1 | O |   |   |
. .---.---.---.
2 |   | O |   |
. .---.---.---.
3 | X | X | O |
. .---.---.---.
Waiting for the other player...
The winner/s is/are Bernat!
marc.valsells@montserrat:~>
```

All the squares are filled with a piece and none of the players has made 3 in a row.



```
marc.valsells@montserrat:~>server 127.0.0.1 9090
Waiting for a client...
Client connected
Waiting for a client...
Player Marc joined
Client connected
Player Bernat joined
Two players connected, time to play
Bernat (O): New move at A1
Marc (X): New move at B2
Bernat (O): New move at A3
Marc (X): New move at A2
Bernat (O): New move at C2
Marc (X): New move at B1
Bernat (O): New move at B3
Marc (X): New move at C3
Bernat (O): New move at C1
There is a draw
Exiting server...

marc.valsells@montserrat:~>
```

```
1 | O | X |   |
· ·---·---·---·
2 | X | X | O |
· ·---·---·---·
3 | O | O | X |
· ·---·---·---·
Waiting for the other player...
  · A · B · C ·
· ·---·---·---·
1 | O | X | O |
· ·---·---·---·
2 | X | X | O |
· ·---·---·---·
3 | O | O | X |
· ·---·---·---·
The winner/s is/are Bernat and Marc!
marc.valsells@montserrat:~>
```

```
· ·---·---·---·
3 | O | O | X |
· ·---·---·---·
It's your turn, enter the column[A-C]: C
Enter the row[1-3]: 1
  · A · B · C ·
· ·---·---·---·
1 | O | X | O |
· ·---·---·---·
2 | X | X | O |
· ·---·---·---·
3 | O | O | X |
· ·---·---·---·
Waiting for the other player...
The winner/s is/are Bernat and Marc!
marc.valsells@montserrat:~>
```

Help

Since la Salle Bonanova wants the code done urgently, they have provided us the code to print the board and to make a move, so we can focus on the communication protocol, it is up to every developer if they want to use it or not and if they want to modify some part of it.

```
#define printF(x) write(1, x, strlen(x))
#define BOARD_COLUMNS 3
#define BOARD_ROWS 3
#define BOARD_HORITZONTAL_LINE "· ·---·---·---·\n"
#define BOARD_TOP_LINE "  · A · B · C ·\n"
#define KEEP_PLAYING 0
#define WIN 1
#define DRAW 2

void printBoard(char board[BOARD_COLUMNS][BOARD_ROWS])
{
    int i;
    char *buffer;
    printF(BOARD_TOP_LINE);
    printF(BOARD_HORITZONTAL_LINE);
    for (i = 0; i < BOARD_ROWS; i++)
    {
        asprintf(&buffer, "%c | %c | %c | %c |\n",'1' + i, board[0][i], board[1][i], board[2][i]);
        printF(buffer);
        free(buffer);
        printF(BOARD_HORITZONTAL_LINE);
    }
}


int placePiece(int col, int row, char piece)
{
    int i, j, count;

    // Place piece
    board[col][row] = piece;

    // For debug
    //printBoard();

    // Check row
    count = 0;
    for (i = 0; i < BOARD_COLUMNS; i++)
    {
        if (board[i][row] == piece)
        {
            count++;
        }
    }
    if (count == BOARD_COLUMNS)
    {
        return WIN;
    }

    // Check column
    count = 0;
    for (i = 0; i < BOARD_ROWS; i++)
    {
        if (board[col][i] == piece)
        {
            count++;
        }
    }
    if (count == BOARD_ROWS)
    {
        return WIN;
    }
    // Check diagonal
    if (col == row)
    {
        count = 0;
        for (i = 0; i < BOARD_COLUMNS; i++)
        {
            if (board[i][i] == piece)
            {
                count++;
            }
        }
        if (count == BOARD_COLUMNS)
        {
            return WIN;
        }
    }

    // Check anti-diagonal
    if (col + row == BOARD_COLUMNS - 1)
    {
        count = 0;
        for (i = 0; i < BOARD_COLUMNS; i++)
        {
            if (board[i][BOARD_COLUMNS - 1 - i] == piece)
            {
                count++;
            }
        }
        if (count == BOARD_COLUMNS)
        {
            return WIN;
        }
    }

    // Check if there is a draw
    for (i = 0; i < BOARD_COLUMNS; i++)
    {
        for (j = 0; j < BOARD_ROWS; j++)
        {
            if (board[i][j] == ' ')
            {
                // There are cells to fill, keep playing
                return KEEP_PLAYING;
            }
        }
    }

    // There are no cells to fill, it's a draw
    return  DRAW;
}
```

Print board note: Empty squares of the board are represented by a space (' ') in the board matrix. The board does not have to be printed in the server, but it can be helpful for debugging.

Place piece note: The *col* and *row* parameters are the position of the matrix where the piece is going to be placed, their value must be between 0 and the defined value in *BOARD_COLUMNS* and *BOARD_ROWS*. The *piece* parameter is to indicate if an 'X' or an 'O' must be placed in the provided position.

Considerations

- The IP and port to which the client must connect and the server must listen are passed as a parameter.
- It can be assumed that the format of the input parameters will always be correct. However, must check if the number of parameters is correct.
- The port assigned to the client and server is the one assigned to the group.
- The client is guaranteed to always enter an input followed by a return (\n).
- The server manages who is the winner, clients must only save and display the board distribution.
- The communication between client and server must be carried out following the guidelines of the frames indicated in the statement.
- The output of the program must be similar to that of the statement (see execution examples).
- The use of “system” or “popen” or analogous functions of the same family is not allowed.
- The use of global variables must be reduced to the minimum possible to make the program work correctly.
- All input and output must be done with file descriptors, the use of printf, scanf, FILE*, getchar, or similar is not allowed.
- It must be compiled using the –Wall, –Wextra and -lpthread flags.
- Any practice that contains warnings will be directly discarded.
- All resources must be released.
- A single “S5.c” file must be delivered that will have the names and logins of the group members commented. Otherwise, the practice will not be corrected.
- You may use the client you made last session. However, the correction will be made with the one provided / last session solution.