

Systems programming

2021/2022

Project Assignment – Part B

New-PongPong

In the first part of the project, students implemented two different versions of the pong game, that had a few limitations with respect to user interaction and communication with the server

In the second part of the project students will continue the development of Relay-Pong and Super-Pong taking advantage of threads and Internet stream sockets to solve some of the identified issues.

The overall idea of both versions remains the same:

- **New-Relay-Pong** - Each user takes turns controlling the paddle and hitting a ball, and only one user controls the paddle. The main changes are:
 - The ball moves at a speed of one place per second independently on the paddle movement
 - The user can quit the game pressing **Q** and any time
 - It is the server that changes every 10 seconds the control to another user
- **New-Super-Pong** - In the second version, users control the paddles simultaneously, on the same board, and all are trying to hit the same ball. The new differences are:
 - The communication should be done using Internet Stream sockets
 - The ball on the server moves at a speed of one place per second, even if no user moves the paddle.
 - All clients receive the ball/paddles update instantaneously
 - The users can press the **Q** key to quit and remove the paddle from the game

1 New-Relay-Pong

In the **New-Relay-Pong**, users take turns playing the pong game individually on their client, until the server releases their ball and the control goes to other user.

When a client is not controlling the ball, his paddle will not move, but the ball position (controlled by another client) is updated on the screen.

When a client starts controlling the ball, this client can start to move the paddle and hit the ball, and move the ball. Even if the paddle does not move, the ball should move one place every second.

Any user can press the **Q** key to quit at any time.

This version still uses internet datagram sockets.

1.1 Ball controlling client operation

The client that is holding and controlling the ball should allow two concurrent operation:

- calculation and update of the new ball position
- movement of the user paddle from the keyboard.

The client should calculate the new ball position every second and send it to the server. The client should also read the keyboard and, if the user presses any cursor key, move the user paddle. The user can press the **Q** key to exit the game.

In this new version it is responsibility of the server to decide that a new client should become the ball controlling client. The server now sends the **Release_ball** and **Send_ball** messages when the control of the ball need to go to another client.

In this new version of the game the ball controlling client sends to the server the following messages:

- **Connect**
- **Move_ball**

- **Disconnect**

The ball controlling client can receive from the server a notification to release the ball:

- **Release_ball**

1.2 Idle client operation

The idle client should concurrently read from the keyboard and from the socket:

- If the user presses the **Q** key it should quit and send a **Disconnect** message to the server
- Every time it receives a **Move_ball** message it should update the screen
- When it receives the **Send_ball** it starts controlling the ball

The idle client receives the following messages from the server:

- **Move_ball**
- **Send_ball**

and send the following:

- **Disconnect**

1.3 Server operation

The server is responsible to receive and process the following messages

- **Connect** – inserts a new user in the list
- **Disconnect** – remove a user from the list
- **Move_ball** – send the ball movement to all idle clients

In this new version the server is also responsible for deciding new ball controlling clients. Every 10 second the server chooses a new client to control the ball and sends the following messages:

- **Release_ball** – to the client that was controlling the ball

- **Send_ball** - to the chosen idle client that will start controlling the ball

1.4 Interactions

The essential interaction between the clients and the server is presented in Figure 1. The server should wait for messages from the clients and decide when to change the controlling client concurrently and execute the correct actions.

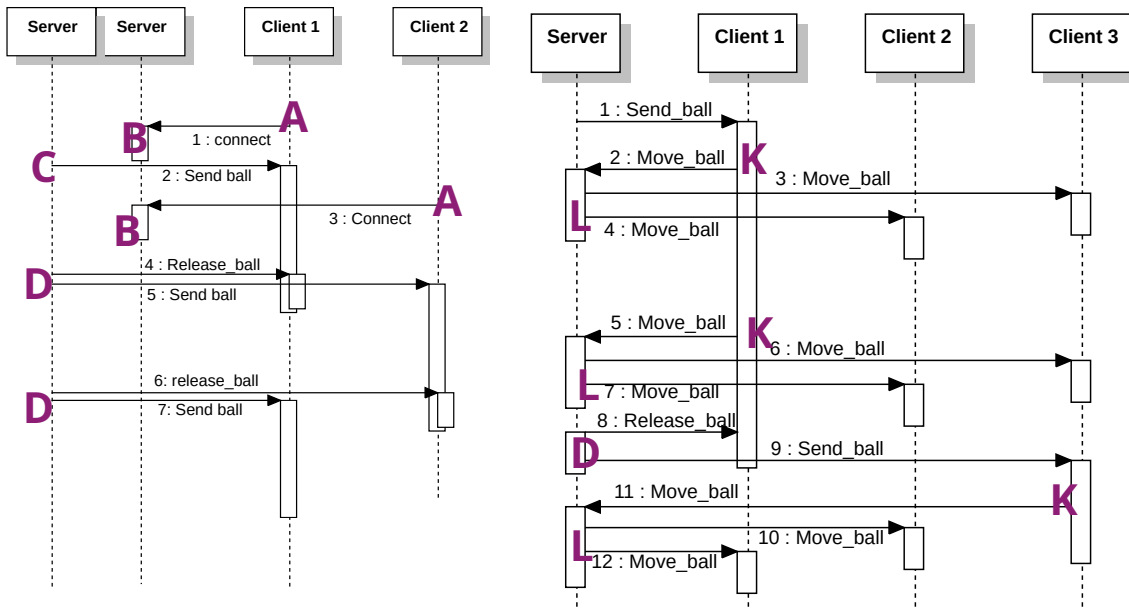


Figure 1: New-Relay-Pong Client - Server Interactions

The messages exchanged between the server and client are:

- **Connect** (from client to the server)
- **Send_ball** (from the server to the client)
- **Release_ball** (from the server to the client)
- **Move_ball** (from client to server and from server to client)
- **Disconnect** (from client to server)

PSis 21/22 - New-PongPong (Part 2)

Every client needs to send a **Connect** message (**step A**) at startup. The server stores the relevant client information in a list (**step B**). If this client is the first one to connect, the server sends as reply a **Send_ball** message (**step C**). The user can now move the paddle and hit the ball (**Controlling the ball**). Any client that connects will be in the **Idle state** and will wait for a **Send_ball** message.

Every 10 seconds the server decides to change the controlling client (**step D**): the server sends a **Release_ball** message to the client controlling the ball and a **Send_ball** message (**step E**) to one of the available clients. If the client that released the ball is the only one connected, it will receive the ball back; otherwise the server should select a different client to receive the ball.

After the client receives a **Send_ball** message, it starts **Controlling the ball**, and the user can start controlling the paddle with the keyboard. Every time the paddle moves, the client performs the following operations (**step K**):

- move the paddle;
- verifies if the paddle hit the ball and calculates the new ball position, and sends the new ball position to the server with the message **Move_ball**.

The client **Controlling the ball** should also move the ball every second, calculate the new position and send it to the server with a **Move_ball** message.

When the server receives the **Move_ball** message, it must forward the same ball movement to all other connected clients (**step L**), so that every client can update the corresponding screen to show the ball in the new position.

When a client terminates orderly, it should send a **Disconnect** message.

1.5 New-Relay-pong Server implementation

The **New-Relay-pong server** is a C program that uses one Internet Domain datagram socket to receive messages from the various clients that want to play **New-Relay-pong**.

PSis 21/22 - New-PongPong (Part 2)

Whenever the server receives one message, it should update its state or send suitable messages following the protocol and description presented in Section 1.4.

The sever should store a list of the clients' information in order to send them messages when necessary. A client is inserted into this list when the server receives a **Connect** message and removed when the server receives a **Disconnect** message from such client.

The server should change the ball controlling client every 10 seconds and send **Release_ball** and **Send_ball** messages to the corresponding clients.

1.6 New-Relay-pong Client

The **New-Relay-pong client** is a C program that interacts with a **New-Relay-Pong server** for the user to play the game. The address of the server should be supplied to the program as a command line argument.

After sending the **Connect** message, the client will wait for a **Send_ball** message. Until this message is received the user can not control the paddle. When receiving this message the client goes into **Ball controlling state** where the paddle can move and hit the ball:

- The ball should move one place in the screen every second independently on the paddle movements.
- Whenever the user presses one of the arrow keys of the keyboard, the paddle position on the screen is updated.
- If the paddle hit the ball (or vice-versa) the ball movement should also be updated.
- Whenever the ball moves, the client should send a **Move_ball** message to the server.

During execution, if the user presses the **Q** key, the client should terminate and send a **Disconnect** message to the server.

The client should print on the screen a message informing the user whether the client is in **Ball controlling state** and that paddle can be controlled.

The screen and data presentation should be that same as in the original Relay-Pong.

2 New-Super-Pong

In the **New-Super-Pong**, all players (connected to the server in multiple clients) can move their paddles and try to hit the ball simultaneous. This is accomplished by reading the paddle movement on each client, and sending such movements to the server, which updates the ball (if hit by a paddle) and sends back to all the clients the new board state.

The clients send every paddle movement to the server, without receiving any explicit reply to this message.

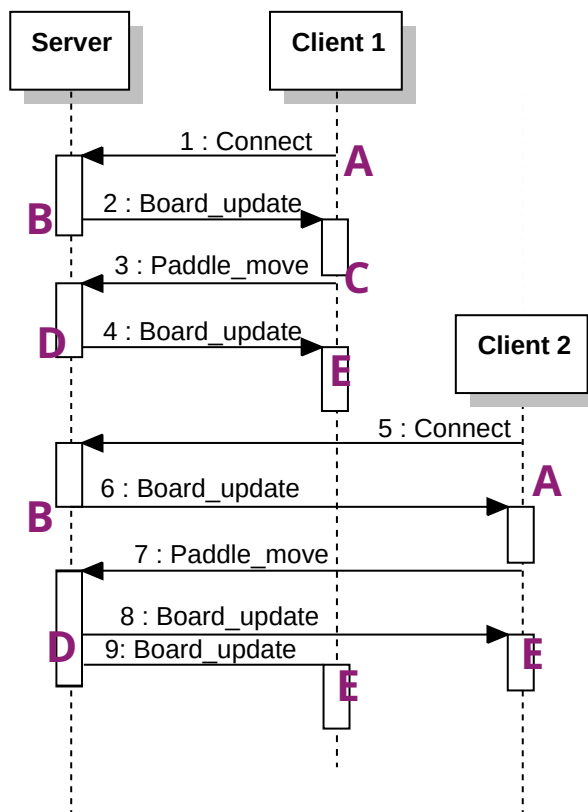


Figure 2: New-Super-Pong Client-server interaction

The **New-Super-Pong** uses Internet stream sockets so the **Connect** and **Disconnect** messages can be omitted and replaced by the socket connect/accept and close.

2.1 Clients / Server interaction

The essential interaction between the clients and the server is presented in Figure 2. The server is waiting for connections and paddle movement messages from the clients, after which it sends the new state of the board to the every client.

The messages exchanged in **New-Super-Pong** are:

- **Connect** – from client to server (can be removed)
- **Paddle_move** – from client to server
- **Board_update** – from server to client
- **Disconnect** – From client to server (should be removed)

Every client starts by connecting to the Server (**step A**). The server inserts the relevant data into the clients' list, defines a random position for the paddle of this client, and replies with a **Board_update** message (**step B**).

The **Board_update** message should include:

- the position of the ball
- the position of this client paddle
- the position of other clients paddles
- players scores

Whenever the user presses one of the arrow keys to move the paddle, the client sends a **Paddle_move** message (**step C**) to the server.

In the **New-Super-Pong**, whenever a server receives a **Paddle_move** and updates the board it should send a **Board_update** message to all connected clients (**step D**). Even if a client does not move his paddle he will receive all updates of the ball and other clients' paddles. Clients should receive any **Board_update** message sent by the server in parallel with the keyboard reading.

When a client terminates orderly, it should send a **Disconnect** message.

2.2 New-Super-pong Server

The **New-Super-pong server** is a C program with an Internet domain stream socket to receive connections from the various clients that want to play **New-Super-pong**.

The maximum number of simultaneous players is ten (10). Students should decide what happens to a client that tries to connect when 10 clients are already connected.

Whenever the server receives one message, it should update the state of the game and send suitable messages following the protocol and description presented in Section 2.1.

The server should store an array of all the clients with all the relevant information (e.g. paddle position). A client is inserted into this list when such client connects and removed when it disconnects.

When the server receives a **Paddle_move** message, it should calculate and update the ball and paddle positions.

2.2.1 Ball movement

The ball movement should be calculated on the server even if no **Paddle_move** messages is received from the clients.

The ball should move every second. After the ball position is updated a **Board_update** message should also be sent to all clients.

The ball should bounce on the wall and when touching the paddles just like in Part A of the project.

2.2.2 Paddles and scores




Paddles should not share the same space. When a user tries to move a paddle to a position already occupied by another paddle, the moving paddle should not move.

If a ball touches a paddle or a paddle touches the ball, the corresponding user will get a point added to the score.

2.3 New-Super-pong Client

The client should connect to a **New-Super-pong server** for the user to play. The address of the server should be supplied to the client as a command line argument.

After connecting, the client will start reading the keyboard. If the user presses one of the arrow keys, a **Paddle_move** should be sent to the server.

Player 1	Player 2	Player 3
		
<p>Each player controls his paddle (marked with =).</p> <p>All screens are really, really synchronized:</p> <ul style="list-style-type: none"> • When a paddle move it is instantaneously/simultaneously updated on all other clients. • The ball position is updated on all clients whenever the server moves it. 		

Whenever the client receives a **Board_update**, the client should draw the ball and all the paddles. The client paddle can be drawn using the = character, to distinguish it from the other paddles (that can be drawn with _).

The various players' points should be written on the screen and updated every time a **Board_update** message is received.

3 Client user interface

The client will use **ncurses** to read the keyboard and draw the paddles and ball on the screen. The interface should be similar to the one of Part A of the project.

4 Project submission

Students should implement the two different Pong games, each composed of a client and server.

The deadline for the submission for the part B of the project will be **4th February at 19h00 on FENIX**.

Before submission, students should create the project groups and register them at FENIX.

Students should submit a zip file containing the code for both games. The files for each game should be placed on two different directories, and if possible students should also provide a Makefile for the compilation of the various programs.

5 Project evaluation

The grade for this project will be given taking into consideration the following:

- Number of functionalities implemented
- Communication
- Code structure and organization
- Error validation and treatment
- Comments