Programação de Sistemas – 21/22

Group 23
Marco Mata nº 84297
João Gonçalves nº 85211

1 Implemented functionalities

1.1 Part 1

Table 1: Implemented functionalities (PART 1)

	Not implemented	With faults	Totally Correct
Relay-Pong			
Connect of new client to server			X
List of clients on server		X - Array	
Paddle/ball movement on client		X	
Forward ball movement to all clients			X
Release ball by the client			X
Send ball to another client			X
Update screen (idle clients)			X
Clients disconnect			X
Super-pong			
Connect of new client to server			X
List/Array of clients on server			X
Paddle movement on the client			X
Send paddle movement to server			X
Ball movement		X	
Server update board state			X
User points			X
Send board_update to all clients			X
Client screen update			X
Client Disconnect			X

1.2 Part 2

Table 2: Implemented functionalities (PART 2)

	Not implemented	With faults	Totally Correct
Windows writing synchronization			X
New-relay-Pong			
Ball moves independent of paddle			X
Ball movement (1 Hz)			X
Quit with **Q**			X
Player changes every 10 s			X
Synchronization ball/paddle			X
New-super-pong			
Clients connect		X – Related to Disconnect	
List/Array of clients on server			X
Send Paddle movement to server			X
Ball movement (1 Hz)			X
Update board			X
Send board_update to all clients			X
Synchronization array/list of clients			X
Synchronization paddles/ball			X
Clients disconnect		X	

1.3 Description of faulty functionalities

Relay-Pong

Client list stored in an array instead of a linked list

Horizontal collision between paddle and ball not implemented

Super-Pong

Horizontal collision between paddle and ball not implemented

New-super-pong

When a client (that isn't the last one that connected) disconnects and a new one connects, some of the already connected clients might lose connection due to confusion in the server threads that handle each client.

2 Code structure

2.1 Functions List

In this section students should present a list with every implemented function, divided by the various components implemented (Client, Server, (new-)relay-pong,

(new)-super-pong).
For each function students should describe the objective of each function.

Very small differences in the functions implemented between relay-pong and new-relay-pong, and between super-pong and new-super-pong.

Relay-pong

Client

Move_ball: Computes the next position of the ball, considering potential collisions with the wall or the paddle. Since the paddle updates before the ball, if the paddle is moved on top of the ball, the ball reacts accordingly.

Draw_ball: Draws or deletes the ball on the screen.

Draw_paddle: Draws or deletes the paddle on the screen

Move_paddle: Computes the next position of the paddle based on input direction from the user. Prevents paddle from going out of bounds.

Server

Place_random_ball: When the first client connects to the server, place a ball in a pseudo-random position.

Random_paddle: When a client connects to the server, place their paddle in a pseudo-random position.

Super-pong

Client

Draw_ball: Draws or deletes the ball on the screen.

Draw_paddle: Draws or deletes the paddle on the screen.

Print_scores: Draws all users' scores on the screen, highlighting the current user's score.

Print_empty_scores: Deletes scores of clients that are not participating, useful when a client disconnects.

Server

Move_ball: Computes the next position of the ball, considering potential collisions with the wall or all paddles. When the ball collides with a paddle, increases that user's score.

Move_paddle: Computes the next position of the paddle based on input direction from the user. Prevents paddle from going out of bounds or into another user's paddle.

Place_random_ball: When the first client connects to the server, place a ball in a pseudo-random position.

Random_paddle: When a client connects to the server, place their paddle in a pseudo-random position, making sure to avoid other paddles.

New-relay-pong

Client

Move_ball: Computes the next position of the ball, considering potential collisions with the wall or the paddle

Draw_ball: Draws or deletes the ball on the screen.

Draw_paddle: Draws or deletes the paddle on the screen

Move_paddle: Computes the next position of the paddle based on input direction from the user. Prevents paddle from going out of bounds. Also, if the paddle moves into the ball, pushes the ball with the paddle.

Server

Place_random_ball: When the first client connects to the server, place a ball in a pseudo-random position.

Random_paddle: When a client connects to the server, place their paddle in a pseudo-random position

New-super-pong

Client

Draw_ball: Draws or deletes the ball on the screen.

Draw_paddle: Draws or deletes the paddle on the screen.

Print_scores: Draws all users' scores on the screen, highlighting the current user's score.

Print_empty_scores: Deletes scores of clients that are not participating, useful when a client disconnects.

Server

Move_ball: Computes the next position of the ball, considering potential collisions with the wall or all paddles. When the ball collides with a paddle, increases that user's score.

Move_paddle: Computes the next position of the paddle based on input direction from the user. Prevents paddle from going out of bounds or into another user's paddle. Also, if the paddle moves into the ball, pushes the ball with the paddle. If there's a collision with the ball due to the paddle moving, increases the user's score.

Place_random_ball: When the first client connects to the server, place a ball in a pseudo-random position.

Random_paddle: When a client connects to the server, place their paddle in a pseudo-random position, making sure to avoid other paddles.

2.2 Threads

In the section students should present a list of every thread implemented, divided by components (Client, Server, new-relay-pong, or new-super-pong).
Students should describe its overall functioning and objectives.

New-relay-pong

Client

handle_comms: Continuously reads from the server. If necessary, draws or deletes ball and paddle positions.

handle_draw: Updates the ball position every second if the client is active and sends it to the server. If the client is idle, erases the message window below the main game window.

Main: Reads from the keyboard and sends a move_ball message to the server when the user is active. If the user stops being active, deletes the paddle from the screen.

Server

Handle_clients: Every 10 seconds, if more than one client is connected, send a release_ball message to the active client, and a send_ball message to the next client.

Main: Continuously waits for a message from the client and acts accordingly to the type of message received (connect, move_ball, disconnect).

New-super-pong

Client

Handle_comms: Continuously reads from the server to update the ball position and every paddle's position.

Main: Continuously reads from the keyboard to get the direction the user wants their paddle to move in or if the user wants to disconnect and sends the server a message with the user's action.

Server

Handle_ball: Moves the ball every second and sends it to every client.

Handle_client: One of these threads is created for every client. Wait for the designated client to send a message, and act accordingly to the type of message sent (paddle_move, disconnect). If the client disconnected, exit the thread.

Main: Waits for a new client to try to connect to the server and initializes relevant variables. Creates a handle_client thread to communicate with this specific client.

2.3 Shared variables

New-relay-pong

Client

my_win and message_win: WINDOW * variables to draw windows with ncurses. Initialized by the main thread and accessed by all threads whenever it is necessary to draw on the screen.

m_client and m_server: message_t type variables that hold relevant information to be communicated between client and server. Accessed by all threads whenever messages are sent or received. The information they contain is also accessed whenever it is necessary to draw the ball position, since m_client contains the user's ball position, while m_server contains the ball position received when another user is playing.

paddle: paddle_position_t type variable that holds the position of the user's paddle. Accessed by all threads: By main when the user moves the paddle and draws the paddle's position, by handle_draw when the ball's position is updated since the paddle's position is relevant to this calculation, and by handle_comms when the client regains or loses the ability to play to draw/delete the paddle from the screen.

sock_fd: Socket file descriptor, accessed by main to initialize the socket, and accessed by all threads when sending or receiving messages.

server_addr: sockaddr_in type variable used by all threads when messages are received or sent. Stores information about the server's address.

server_addr_size: socklen_t type variable used by all threads when messages are received or sent. Stores the size of the server_addr variable.

m1: pthread_mutex_t type variable used by all threads for synchronization purposes.

Server

sock_fd: Socket file descriptor, accessed by main to initialize the socket, and accessed by both threads when sending or receiving messages.

client_addr: sockaddr_in type variable used by both threads when messages are received or sent. Stores information about the last client the server received a message from's address.

client_addr_size: socklen_t type variable used by both threads when messages are received or sent. Stores the size of the client_addr variable.

m_client and m_server: message_t type variables that hold relevant information to be communicated between client and server. Accessed by both threads whenever messages are sent or received.

clients: Array of 100 client_info_t type variables that holds information about every client currently connected to the server. Accessed by both threads, storing each client's address and play state so the server knows which client is currently playing.

New-super-pong

Client

my_win and message_win: WINDOW * variables to draw windows with ncurses. Initialized by the main thread and accessed by all threads whenever it is necessary to draw on the screen.

sock_fd: Socket file descriptor, accessed by main to initialize the socket, and accessed by both threads when sending or receiving messages.

server_addr: sockaddr_in type variable used by both threads when messages are received or sent. Stores information about the server's address.

server_addr_size: socklen_t type variable used by both threads when messages are received or sent. Stores the size of the server_addr variable.

m_client and m_server: message_t type variables that hold relevant information to be communicated between client and server. Accessed by both threads whenever messages are sent or received.

Server

client_fd_list: Array of int type variables that are the socket file descriptors for each client, accessed by main to initialize the individual client sockets, a specific element of the array is passed as an argument when a handle_client thread is created, and accessed by the handle_ball and handle_client threads when sending or receiving messages.

m1: pthread_mutex_t type variable used by all threads for synchronization purposes.

n_clients: int variably that stores the number of currently connected clients. Accessed by all threads when move_ball or move_paddle are called, and to update the number of clients when that numbers does change.

m_client and m_server: message_c_t (m_client) and message_s_t (m_server) type variables that hold relevant information to be communicated between client and server. m_client is accessed by handle_client and handle_ball threads whenever messages are sent, while m_server is accessed by handle_client and handle_ball threads whenever messages are received, but also by main to update relevant information.

ball: ball_position_t type variable that stores the ball's position and is used by all threads: handle_ball updates this variable when the ball moves every second, handle_client might update this variable when a paddle is moved and causes the ball to move, and main places the ball in a random position and makes use of the position of the ball when placing a random paddle for a newly connected user.

paddle_pos: paddle_position_t type variable that stores the most recent moved or placed paddle's position and used by all threads when the ball is moved, a paddle is moved, or a paddle is randomly placed for a newly connected user.

2.4 Synchronization

We only used mutexes throughout the project.

In New-Relay-Pong, we used mutexes in the client to make sure nothing went wrong when updating the ncurses windows.

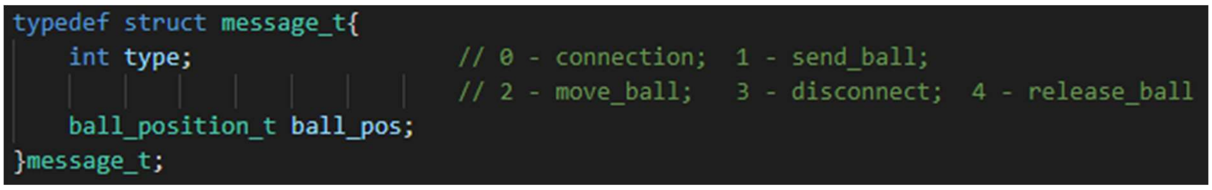
In New-Super-Pong, we used mutexes in the server to make sure none of the variable the various threads share change mid-execution of the thread loop, when the ball is being updated or when a paddle is moved. Since for this game the ncurses windows are updated by one thread only, we did not feel the need to use a mutex, and we confirmed that to the best of our ability in testing.

3 Communication

3.1 Transferred data

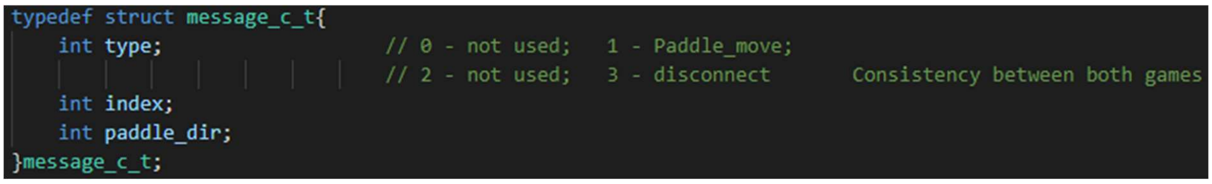
In this section students should present the exchange data structures/messages. Students can present the typedef used and relate them to the messages presented in the assignments.

New-Relay-Pong

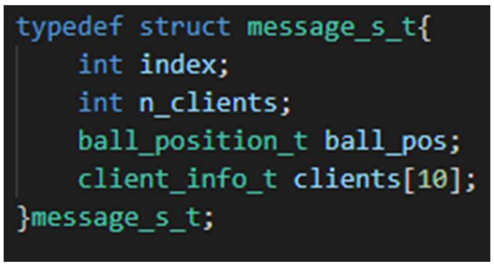


```c
typedef struct message_t{
    int type;                   // 0 - connection;  1 - send_ball;
                                // 2 - move_ball;   3 - disconnect;  4 - release_ball
    ball_position_t ball_pos;
}message_t;
```

New-Super-Pong

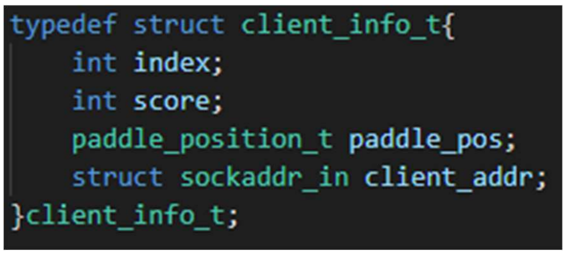


```c
typedef struct message_c_t{
    int type;                   // 0 - not used;   1 - Paddle_move;
                                // 2 - not used;   3 - disconnect      Consistency between both games
    int index;
    int paddle_dir;
}message_c_t;
```

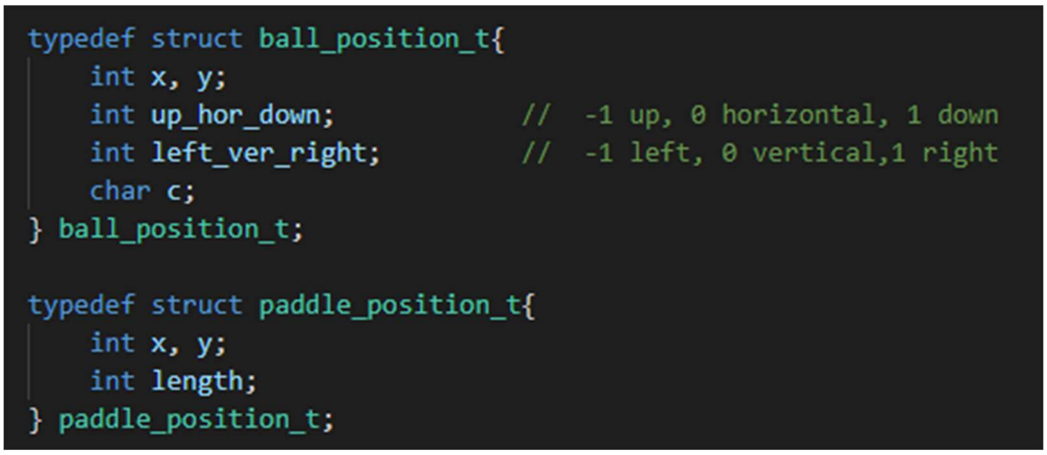


```c
typedef struct message_s_t{
    int index;
    int n_clients;
    ball_position_t ball_pos;
    client_info_t clients[10];
}message_s_t;
```

Types used in the messages



```c
typedef struct client_info_t{
    int index;
    int score;
    paddle_position_t paddle_pos;
    struct sockaddr_in client_addr;
}client_info_t;
```



```c
typedef struct ball_position_t{
    int x, y;
    int up_hor_down;            //  -1 up, 0 horizontal, 1 down
    int left_ver_right;         //  -1 left, 0 vertical,1 right
    char c;
} ball_position_t;

typedef struct paddle_position_t{
    int x, y;
    int length;
} paddle_position_t;
```

3.2 Error treatment

In this section students should present what type of error treatments were implemented related to communication: validation of read data, verification of read/write return errors, ...

(New-)Relay-Pong

Client

Program shuts down if the socket creation fails or if the address passed as an argument when the program is run is invalid.

No error treatment if the client tries to connect to a server that doesn't yet exist and hangs there.

Server

Program shuts down if the socket creation fails or if binding the socket fails.

(New-)Super-Pong

Client

Program shuts down if the socket creation fails or if the client fails to connect to the server

The thread that continually waits for messages from the server only proceeds if the number of bytes read is greater than 0.

Server

Program shuts down if the socket creation fails or binding the socket fails.

If the accept call returns an error for whatever reason, information about a new client is not processed and the program goes back to waiting for a new connection to accept.

The threads that communicate individually with each client only proceed if the number of bytes read is greater than 0.

4 Fixed faults in submission

We have fixed the faults regarding connection and disconnection in the **New-Super-Pong** that we could not submit. The fixed version is ready for demonstration.