Instituto Superior Técnico

Electrical and Computer Engineering

System Programming

PongPong

Authors:

Pedro [redacted]
Ricardo Barreto [redacted]

Group Number 16

2021/2022 - First semester
January of 2022

1 Implemented functionalities

1.1 Part 1

	Not implemented	With faults	Totally correct
Relay Pong			
Connect of new client to server			X
List of clients on server			X
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 serve			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

	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
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

2 Code Structure

2.1 Functions List

2.1.1 Relay Pong Client

- **void new_paddle (paddle_position_t * paddle, int legth)**
  Initializes the variable "paddle" with length "length" at a specific position
- **void draw_paddle(WINDOW *win, paddle_position_t * paddle, int draw)**
  Draws paddle on window "win"
  mode draw=true: draws paddle
  mode draw=false: cleans paddle
- **void clear_window(WINDOW *win, int window_y, int window_x)**
  Fills window "win" with blank spaces, that is, cleans the window
- **bool check_availability(int y, int x, int length)**
  Checks if paddle move to coordinates "x" and "y" is valid.
  Only takes into account the ball.
- **void moove_paddle (paddle_position_t * paddle, int direction)**
  Calculates the new coordinates of the paddle
- **void place_ball_random(ball_position_t * ball)**
  Initializes the variable "ball" at a random position
- **void paddle_bounce(int current_y, int current_x, ball_position_t *ball)**
  Calculates the ball bouncing in the paddle
- **void moove_ball(ball_position_t * ball)**
  Calculates the new coordinates of the ball
- **void draw_ball(WINDOW *win, ball_position_t * ball, int draw)**
  Draws ball on window "win"
  mode draw=true: draws ball
  mode draw=false: cleans ball
- **void print_message_window(WINDOW *message_win)**
  Prints in message window a message stating that the player is in the play state
- **long long timeInMilliseconds()**
  Returns current time in milliseconds
- **void send_disconnect_message(message m, int sock_fd, struct sockaddr_in server_addr)**
  Sends to server disconnect message and closes all descriptors

2.1.2 Relay Pong Server

- **bool compare_clients(struct sockaddr_in addr1, struct sockaddr_in addr2)**
  Compares if socket addresses "addr1" and "addr2" match
- **node *create_player(struct sockaddr_in addr, int player_id)**
  Creates a node of a new player with address "addr" and id "player_id"
- **void insert_player(node *new_player)**
  Inserts new player "new_player" at the head of the players' list
- **void next_player()**
  Determines the next player who receives the ball
- **void disconnect_player(struct sockaddr_in addr)**
  Removes from players' list, player with address "addr"
- **void print_communication_status(struct sockaddr_in client_addr, int n_bytes, message m, bool status)**
  Print in the terminal information about a message
  mode status=true: message received
  mode status=false: message sent
- **void broadcast(struct sockaddr_in addr, message m)**
  Broadcasts message "m" to all clients with the exception of the one with address "addr"

2.1.3 Super Pong Client

- **void draw_paddle(WINDOW *win, paddle_position_t * paddle, char ch)**
  Draws paddle on window "win"
  mode draw=true: draws paddle
  mode draw=false: cleans paddle
- **void draw_paddle_board(WINDOW *my_win, paddle paddles_list[MAX_PLAYERS], int draw, int my_id, int num_player)**
  Draws all paddles on window "win"
  mode draw=true: draws paddle
  mode draw=false: cleans paddle
- **void clear_window(WINDOW *win, int window_y, int window_x)**
  Fills window "win" with blank spaces, that is, cleans the window
- **void draw_ball(WINDOW *win, ball_position_t * ball, int draw)**
  Draws ball on window "win"
  mode draw=true: draws ball
  mode draw=false: cleans ball
- **void print_message_window(WINDOW *message_win, board_message m, int my_id)**
  Prints in message window the players' scores
- **void copy_paddles(paddle *paddles_list, board_message m)**
  Copy to array all gameboard paddles
- **void send_disconnect_message(status_message m_sent, int sock_fd, struct sockaddr_in server_addr)**
  Sends to server disconnect message and closes all descriptors

2.1.4 Super Pong Server

- **void place_ball_random(ball_position_t * ball)**
  Initializes the variable "ball" at a random position
- **bool check_availability(int y, int x, int length, int my_id)**
  Checks if paddle move to coordinates "x" and "y" is valid
  Only takes into account the other paddles and the ball
- **void new_paddle (paddle_position_t * paddle, int length, int my_id)**
  Initializes the variable "paddle" with length "length" at a specific position
- **void moove_paddle (paddle_position_t * paddle, int direction, int my_id)**
  Calculates the new coordinates of the paddle
- **void paddle_bounce(int current_y, int current_x, ball_position_t *ball)**
  Calculates the ball bouncing in the paddle
- **void moove_ball(ball_position_t * ball)**
  Calculates the new coordinates of the ball
- **bool compare_clients(struct sockaddr_in addr1, struct sockaddr_in addr2)**
  Compares if socket addresses "addr1" and "addr2" match
- **node *create_player(struct sockaddr_in addr, int player_id)**
  Creates a node of a new player with address "addr" and id "player_id"
- **void insert_player(node *new_player)**
  Inserts new player "new_player" at the head of the players' list
- **void disconnect_player(struct sockaddr_in addr)**
  Removes from players' list, player with address "addr"
- **void print_communication_status(struct sockaddr_in client_addr, int n_bytes, status_message m)**
  Print in the terminal information about a received status message
- **void print_board_communication(struct sockaddr_in client_addr, int n_bytes, bool valid)**
  Print in the terminal information about a sent board message
- **void fill_board_message(board_message *m, int number_players, int player_total)**
  Writes to "m" the current state of the board
- **paddle *find_paddle(struct sockaddr_in addr)**
  Returns pointer to paddle with address "addr"

2.1.5 New Relay Pong Client

- **void draw_paddle(WINDOW *win, paddle_position_t * paddle, int draw)**
  Draws paddle on window "win"
  mode draw=true: draws paddle
  mode draw=false: cleans paddle
- **void clear_window(WINDOW *win, int window_y, int window_x)**
  Fills window "win" with blank spaces, that is, cleans the window

- **bool check_availability(int y, int x, int length)**
  Checks if paddle move to coordinates "x" and "y" is valid
  Only takes into account the ball
- **void place_ball_random(ball_position_t * ball)**
  Initializes the variable "ball" at a random position
- **void moove_paddle (paddle_position_t * paddle, int direction)**
  Calculates the new coordinates of the paddle
- **void paddle_bounce(int current_y, int current_x, ball_position_t *ball)**
  Calculates the ball bouncing in the paddle
- **void moove_ball(ball_position_t * ball)**
  Calculates the new coordinates of the ball
- **void draw_ball(WINDOW *win, ball_position_t * ball, int draw)**
  Draws ball on window "win"
  mode draw=true: draws ball
  mode draw=false: cleans ball
- **void print_message_window(WINDOW *message_win)**
  Prints in message window a message stating that the player is in the play state
- **void send_disconnect_message(message m, int sock_fd, struct sockaddr_in server_addr)**
  Sends to server disconnect message and closes all descriptors

2.1.6 New Relay Pong Server

- **bool compare_clients(struct sockaddr_in addr1, struct sockaddr_in addr2)**
  Compares if socket addresses "addr1" and "addr2" match
- **node *create_player(struct sockaddr_in addr, int player_id)**
  Creates a node of a new player with address "addr" and id "player_id"
- **void insert_player(node *new_player)**
  Inserts new player "new_player" at the head of the players' list
- **void place_ball_random(ball_position_t * ball)**
  Initializes the variable "ball" at a random position
- **void new_paddle (paddle_position_t * paddle, int legth)**
  Initializes the variable "paddle" with length "length" at a specific position
- **void next_player()**
  Determines the next player who receives the ball
- **void disconnect_player(struct sockaddr_in addr)**
  Removes from players' list, player with address "addr"
- **void print_communication_status(struct sockaddr_in _client_addr, int n_bytes, message m, bool status)**
  Print in the terminal information about a message
  mode status=true: message received
  mode status=false: message sent
- **void broadcast(struct sockaddr_in addr, message m)**
  Broadcasts message "m" to all clients with the exception of the one with address "addr"

2.1.7 New Super Pong Client

- **void draw_paddle(WINDOW *win, paddle_position_t * paddle, char ch)**
  Draws paddle on window "win"
  mode draw=true: draws paddle
  mode draw=false: cleans paddle
- **void draw_paddle_board(WINDOW *my_win, paddle paddles_list[MAX_PLAYERS], int draw, int my_id, int num_player)**
  Draws all paddles on window "win"
  mode draw=true: draws paddle
  mode draw=false: cleans paddle
- **void clear_window(WINDOW *win, int window_y, int window_x)**
  Fills window "win" with blank spaces, that is, cleans the window
- **void draw_ball(WINDOW *win, ball_position_t * ball, int draw)**
  Draws ball on window "win"
  mode draw=true: draws ball
  mode draw=false: cleans ball
- **void print_message_window(WINDOW *message_win, board_message m, int my_id)**
  Prints in message window the players' scores
- **void copy_paddles(paddle *paddles_list, board_message m)**
  Copy to array all gameboard paddles
- **void close_client(int sock_fd)**
  Close client descriptors

2.1.8 New Super Pong Server

- **void place_ball_random(ball_position_t * ball)**
  Initializes the variable "ball" at a random position
- **bool check_availability(int y, int x, int length, int my_id)**
  Checks if paddle move to coordinates "x" and "y" is valid
  Only takes into account the other paddles and the ball
- **void new_paddle (paddle_position_t * paddle, int length, int my_id)**
  Initializes the variable "paddle" with length "length" at a specific position
- **void moove_paddle (paddle_position_t * paddle, int direction, int my_id)**
  Calculates the new coordinates of the paddle
- **void paddle_bounce(int current_y, int current_x, ball_position_t *ball)**
  Calculates the ball bouncing in the paddle
- **void moove_ball(ball_position_t * ball)**
  Calculates the new coordinates of the ball
- **node *create_player(int c_sock_fd, int player_id)**
  Creates a node of a new player with "address" addr and id "player_id"

- **void insert_player(node *new_player)**
  Inserts new player "new_player" at the head of the players' list
- **void disconnect_player(int c_sock_fd)**
  Removes from players' list, player with address "addr"
- **void print_communication_status(int n_bytes,int c_sock_fd)**
  Print in the terminal information about a status message
- **void print_board_communication(int n_bytes, int c_sock_fd, bool valid)**
  Print in the terminal information about a board message
- **void fill_board_message(board_message *m, int number_players, int player_total)**
  Writes to "m" the current state of the board
- **paddle *find_paddle(int c_sock_fd)**
  Returns pointer to paddle with address "addr"
- **void broadcast(board_message m_sent)**
  Broadcasts message "m" to all clients

2.2 Threads

Threads are only used in the Second Part of the project.

2.2.1 New Relay Pong

- **client.c**
  - **void *read_socket()**
    Thread that reads and processes the socket messages from server.
  - **void *ball_position()**
    Thread that updates ball position at each second and sends move_ball message to server.
- **server.c**
  - **void *read_socket()**
    Thread that reads and processes the socket messages from all clients.
  - **void *change_client()**
    Thread that sends release_ball to current client and send_ball to next client every 10 seconds.

2.2.2 New Super Pong

- **client.c**
  - **void *read_socket()**
    Thread that receives board message from the server and draws the information on the window.
- **server.c**
  - **void *ball_position()**
    Thread that updates ball position at each second and sends board_message to all registered client.
  - **void *client_thread(void *arg)**
    Thread that reads and processes the socket messages from a specific client with file descriptor *arg.
    There is a thread of this type for each connected client.

2.3 Shared variables

2.3.1 New Relay Pong

- **client.c**
  - **WINDOW * message_win**
  - **WINDOW * my_win**
    These variables are used as pointers to the windows where the ball, paddle and game details will be printed. my_win is also used to get keyboard inputs.
    accessed by: main, read_socket and ball_position
  - **paddle_position_t paddle**
    This variable stores the information of the paddle.
    accessed by: main, read_socket and ball_position
  - **ball_position_t ball**
    This variable stores the information of the ball.
    accessed by: read_socket and ball_position
  - **bool play_state**
    This variable dictates if the current client is in it's play state.
    accessed by: main, read_socket and ball_position
- **server.c**
  - **node *player_info_list**
    List of all current players that are connected to the server.
    accessed by: read_socket and change_client
  - **node *current_player**
    The current player that has the ball.
    accessed by: read_socket and change_client
  - **int number_players**
    Number of players currently connected to the server.
    accessed by: read_socket and change_client
  - **paddle_position_t paddle**
  - **ball_position_t ball**
    Ball and paddle current positions.
    accessed by: read_socket and change_client

2.3.2 New Super Pong

- **client.c**
  There are no shared variables that need synchronization. There are the window pointers from ncurses but we only use them in main before creating the thread that will use them later.
- **server.c**
  - **node *player_info_list**
    List of all current players that are connected to the server.
    accessed by: main, ball_position and client_thread
  - **int player_total**
    Total number of players.
    accessed by: main, ball_position and client_thread

- **int number_players**
  Number of players currently connected to the server.
  accessed by: main, ball_position and client_thread
- **ball_position_t ball**
  Ball position.
  accessed by: main, ball_position and client_thread

2.4 Synchronization

The guarding mechanisms used in this project to avoid problems with concurrency are mutexes.

2.4.1 New Relay Pong Client

- **mux_curses**
  Mutex used to lock access to variables and functions of the ncurses library.
- **mux_paddle**
  Mutex used to lock access to variable paddle.
- **mux_ball**
  Mutex used to lock access to variable ball.
- **mux_state**
  Mutex used to lock access to variable play_state.

2.4.2 New Relay Pong Server

- **mux_current_player**
  Mutex used to lock access to variable current_player.
- **mux_player_info_list**
  Mutex used to lock operations on the list of all current players that are connected to the server.
- **mux_number_players**
  Mutex used to lock access to variable number_players.
- **message_variables_mux**
  Mutex used to lock access to variables ball and paddle.

2.4.3 New Super Pong Client

Like previously mentioned, there are no shared variables that need synchronization and therefore there is no need to use blocking access mechanics.

2.4.4 New Super Pong Server

- **mux_player_info_list**
  Mutex used to lock operations on the list of all current players that are connected to the server.
- **mux_player_total**
  Mutex used to lock access to variable player_total.
- **mux_number_players**
  Mutex used to lock access to variable number_players.
- **mux_ball**
  Mutex used to lock access to variable ball.

3 Communication

3.1 Transferred data

3.1.1 Relay Pong

In Relay Pong, all messages between server and client (connect, send_ball, release_ball, move_ball and disconnect) are of type *message*.

```
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;

typedef struct message{

    int msg_type;  // 0->Connect, 1->Release_ball, 2->Send_ball 3->Move_ball 4->
    Disconnect
  ball_position_t ball;
  paddle_position_t paddle;
    int num_player;
} message;
```

3.1.2 Super Pong

In Super Pong, all messages from server to client (board_update) are of type *board_message*.

Likewise, messages form client to server (connect, paddle_move and disconnect) are of type *status_message*.

```
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;

typedef struct status_message{

    int msg_type;  // 0->Connect, 1->Paddle, 2->Disconnect
  int direction;
    int player_id;
} status_message;

typedef struct paddle{

    int player_id;
    int score;
    paddle_position_t position;
} paddle;

typedef struct board_message{

```

```
  ball_position_t ball;
  paddle player_paddle[MAX_PLAYERS];
    int num_player;
    int player_id;
    bool status; //true if connect is valid, false otherwise
} board_message;
```

3.1.3 New Relay Pong

In New Relay Pong, all communication messages types are similar to Relay Pong.

```
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;

typedef struct message{

    int msg_type;  // 0->Connect, 1->Release_ball, 2->Send_ball 3->Move_ball 4->
    Disconnect
  ball_position_t ball;
  paddle_position_t paddle;
    int num_player;
} message;
```

3.1.4 New Super Pong

In New Super Pong, board_update is of type *board_message* while paddle_move is of type *paddle_message*.

Connect and disconnect messages are omitted due to Internet socket stream capabilities (accept/connect and close functions).

```
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;

typedef struct paddle_message{

  int direction;
    int player_id;
} paddle_message;

typedef struct paddle{

    int player_id;
    int score;
    paddle_position_t position;
} paddle;

```

```
28 typedef struct board_message{
29
30   ball_position_t ball;
31   paddle player_paddle[MAX_PLAYERS];
32     int num_player;
33     int player_id;
34     bool status; //true if connect is valid, false otherwise
35 } board_message;
```

3.2 Error Treatment

Socket Communication functions working on both protocols, namely TCP (stream sockets) and UDP (datagram sockets), are tested for return value.

For datagram sockets, a return value of -1 on the sendto function indicates that there is a sending message error while a return value of recv with a number of bytes different from the expected size provide for a solid check of incoming messages' correctness.

For stream sockets, the write function error is define by the -1 return value while the read function checks for a return value that is smaller or equal to 0.