Московский авиационный институт
(национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Курсовой проект по курсу «Операционные системы»

Пинг-понг

Студент: Инютин М. А.
Группа: М8О-207Б-19
Преподаватель: Миронов Е. С.
Дата:
Оценка:

Москва, 2020

1. Постановка задачи

Целью курсового проекта является приобретение практических навыков, использование знаний, полученных в рамках курса, и проведение исследования в выбранной предметной области.

Создать собственную игру для нескольких пользователей «Пинг-понг» с использованием технологии очередей сообщений.

2. Теоретическая часть

Очередью сообщений называется структура данных, которая позволяет храненить и передавать данные между процессами. Она обладает следующими свойствами:

1. Наличие транзакций
2. Максимальный объём
3. Приоритет сообщений
4. Хранение необработанных сообщений
5. Возможность журналирования очереди сообщений
6. Политики безопасности
7. Система имён
8. Производительность
9. Персистентность очереди сообщений

ZeroMQ — это кроссплатформенная высокопроизводительная библиотека, предоставляющая функции для асинхронного обмена сообщениями.

Она поддерживает основные шаблоны обмена сообщениями и основывается на стандартных интерфейсах для взаимодействия с сетью — сокетах.

3. Описание алгоритма

Я использовал библиотеку *ncurses* для создания простого интерфейса и работы с окнами. Между игроками постоянно происходит обмен сообщениями о внутриигровой информации на основе сокета *ZMQ_PAIR*.

Один из игроков является хостом и постоянно обрабатывает всё, что связано с мячом, и посылает эту информацию другому игроку. По отдельности программы игроков считывают нажатия клавиш, обрабатывают положение игрока, пытаются отправить и получить сообщения от программы другого игрока. В случае успеха обновляют данные и отрисовывают игровую ситуацию на экран. Если один из игроков отключится от игры, то обмен сообщениями станет невозможен.

4. Листинг программы

Программа состоит из трёх файлов: zmq_std.hpp с оболочками над системными вызовами *ZeroMQ*, game.hpp с классом игры и main.cpp с самим приложением, вводом команд и пользовательским интерфейсом.

game.hpp

```
#include <math.h>
#include <ncurses.h>
#include <random>
#include <stdexcept>
#include <time.h>
#include <unistd.h>

#include "zmq_std.hpp"

class game {
private:
     /* Determines bind or connect */
     bool mode;

     /*
      * Playing field is rectangle with width FIELD_WIDTH
      * and height FIELD_HEIGHT. Programm will adopt to
      * user's temrinal LINES and COLS
      */
     const unsigned int FIELD_FACTOR = 10;
     const unsigned int FIELD_HEIGHT = 9;
     const unsigned int FIELD_WIDTH = 16;

     const double PONG_SPEED = 4.0;
     const double BALL_SPEED = 1.5;
     const double FIELD_LEFT = 0;
     const double FIELD_UP = 0;
     const double FIELD_RIGHT = FIELD_LEFT + FIELD_WIDTH *
FIELD_FACTOR;
     const double FIELD_DOWN = FIELD_UP + FIELD_HEIGHT *
FIELD_FACTOR;
     const double PONG_SIZE = (FIELD_DOWN - FIELD_UP) / 5.0;

     const double SCALING_FACTOR_X = (LINES - 2) / (FIELD_DOWN -
FIELD_UP);
     unsigned int transform_x(const double & cord) {
          return cord * SCALING_FACTOR_X + 1;
     }

     const double SCALING_FACTOR_Y = (COLS - 1) / (FIELD_RIGHT -
FIELD_LEFT);
     unsigned int transform_y(const double & cord) {
          return cord * SCALING_FACTOR_Y;
     }
```

```
    struct game_info_t {
        double player1_pos = 0;
        double player2_pos = 0;
        double ball_pos_x = 0;
        double ball_pos_y = 0;
        double ball_vec_x = 0;
        double ball_vec_y = 0;
        unsigned long long score1 = 0;
        unsigned long long score2 = 0;
    };

    void gen_ball(game_info_t & info) {
        info.ball_pos_x = (FIELD_DOWN - FIELD_UP) / 2.0;
        info.ball_pos_y = (FIELD_RIGHT - FIELD_LEFT) / 2.0;
        double angle = (5236 + std::abs(rand()) % 5236) /
10000.0;
        info.ball_vec_x = BALL_SPEED * std::cos(angle);
        info.ball_vec_y = BALL_SPEED * std::sin(angle);
    }

    /* System call return code */
    int rc;

    WINDOW* window;
    WINDOW* player1;
    WINDOW* player2;

    void* context = NULL;
    void* socket = NULL;

public:
    game(const bool & input_mode, const std::string & IP) :
mode(input_mode) {
        window = newwin(LINES, COLS, 0, 0);
        wborder(window, ' ', ' ', 0, 0, ACS_HLINE, ACS_HLINE,
ACS_HLINE, ACS_HLINE);
        keypad(window, TRUE);
        nodelay(window, TRUE);
        wrefresh(window);

        context = zmq_ctx_new();
        if (context == NULL) {
            throw std::runtime_error("Error creating context!");
        }
        socket = zmq_socket(context, ZMQ_PAIR);
        if (socket == NULL) {
            throw std::runtime_error("Error creating socket!");
        }
        int linger_period = 1000;
```

```
          rc = zmq_setsockopt(socket, ZMQ_LINGER, &linger_period,
sizeof(int));
          if (rc != 0) {
               throw std::runtime_error("Error setting linger!");
          }
          if (mode) {
               rc = zmq_bind(socket, IP.c_str());
               if (rc != 0) {
                    throw std::runtime_error("Cannot bind to " +
IP);
               }
          } else {
               rc = zmq_connect(socket, IP.c_str());
               if (rc != 0) {
                    throw std::runtime_error("Cannot connect to " +
IP);
               }
          }
          std::srand(time(NULL));
     }

     void play() {
          game_info_t info1;
          game_info_t info2;
          if (mode) {
               gen_ball(info1);
          }
          info1.player1_pos = (FIELD_DOWN - FIELD_UP) / 2.0;
          info1.player2_pos = (FIELD_DOWN - FIELD_UP) / 2.0;

          int c;
          while (1) {
               usleep(30000);

               /* Game window and ball */
               wborder(window, ' ', ' ', 0, 0, ACS_HLINE,
ACS_HLINE, ACS_HLINE, ACS_HLINE);
               mvwprintw(window, 0, COLS / 2 - 15, "> P1: %d <",
info1.score1);
               mvwprintw(window, 0, COLS / 2 + 5, "> P2: %d <",
info1.score2);
               mvwprintw(window, transform_x(info1.ball_pos_x),
transform_y(info1.ball_pos_y), "O");
               wrefresh(window);

               /* Player 1 */
               player1 = newwin(transform_x(PONG_SIZE), 1,
transform_x(info1.player1_pos), transform_y(FIELD_LEFT));
               wborder(player1, ' ', ACS_VLINE, ' ', ' ', ' ',
ACS_URCORNER, ' ', ACS_LRCORNER);
               wrefresh(player1);
```

```
                /* Player 2 */
                player2 = newwin(transform_x(PONG_SIZE), 1,
transform_x(info1.player2_pos), transform_y(FIELD_RIGHT) - 1);
                wborder(player2, ' ', ACS_VLINE, ' ', ' ', ' ',
ACS_ULCORNER, ' ', ACS_LLCORNER);
                wrefresh(player2);

                c = wgetch(window);
                if (c == 10) {
                     break;
                } else if (c == KEY_UP and info1.player1_pos >=
FIELD_UP) {
                     info1.player1_pos -= PONG_SPEED;
                } else if (c == KEY_DOWN and info1.player1_pos +
PONG_SIZE <= FIELD_DOWN) {
                     info1.player1_pos += PONG_SPEED;
                }

                if (mode) {
                     if (info1.ball_pos_x + info1.ball_vec_x <=
FIELD_UP) {
                          info1.ball_vec_x = -info1.ball_vec_x;
                     }

                     if (info1.ball_pos_x + info1.ball_vec_x >=
FIELD_DOWN) {
                          info1.ball_vec_x = -info1.ball_vec_x;
                     }

                     if (info1.ball_pos_y + info1.ball_vec_y <=
FIELD_LEFT) {
                          if (info1.player1_pos <= info1.ball_pos_x
and info1.ball_pos_x <= info1.player1_pos + PONG_SIZE) {
                               info1.ball_vec_y = -info1.ball_vec_y;
                          } else {
                               ++info1.score2;
                               gen_ball(info1);
                          }
                     }

                     if (info1.ball_pos_y + info1.ball_vec_y >=
FIELD_RIGHT) {
                          if (info1.player2_pos <= info1.ball_pos_x
and info1.ball_pos_x <= info1.player2_pos + PONG_SIZE) {
                               info1.ball_vec_y = -info1.ball_vec_y;
                          } else {
                               ++info1.score1;
                               gen_ball(info1);
                               info1.ball_vec_x = -info1.ball_vec_x;
                               info1.ball_vec_y = -info1.ball_vec_y;
```

```
                    }
                }

                info1.ball_pos_x += info1.ball_vec_x;
                info1.ball_pos_y += info1.ball_vec_y;
            }

            zmq_std::send_msg_dontwait(&info1, socket);

            /*
             * Programm will recieve more messages to
             * decrease ping emptying the queue
             */
            for (int i = 0; i < 10; ++i) {
                if (zmq_std::recieve_msg_dontwait(info2,
socket)) {
                    info1.player2_pos = info2.player1_pos;
                    if (!mode) {
                        info1.score1 = info2.score2;
                        info1.score2 = info2.score1;

                        info1.ball_pos_x = info2.ball_pos_x;
                        info1.ball_pos_y = FIELD_RIGHT -
info2.ball_pos_y;

                        info1.ball_vec_x = info2.ball_vec_x;
                        info1.ball_vec_y = -info2.ball_vec_y;
                    }
                }
            }
            delwin(player1);
            delwin(player2);
            wclear(window);
        }
    }

    ~game() {
        rc = zmq_close(socket);
        assert(rc == 0);
        rc = zmq_ctx_term(context);
        assert(rc == 0);
        wrefresh(window);
        delwin(window);
    }
};
```

zmq_std.hpp

```
#ifndef ZMQ_STD_HPP
#define ZMQ_STD_HPP

#include <assert.h>
#include <errno.h>
#include <string.h>
#include <string>
#include <zmq.h>

namespace zmq_std {
     template<class T>
     bool recieve_msg_dontwait(T & reply_data, void* socket) {
          int rc = 0;
          zmq_msg_t reply;
          rc = zmq_msg_init(&reply);
          assert(rc == 0);
          rc = zmq_msg_recv(&reply, socket, ZMQ_DONTWAIT);
          if (rc == -1) {
               rc = zmq_msg_close(&reply);
               assert(rc == 0);
               return false;
          }
          assert(rc == sizeof(T));
          reply_data = *(T*)zmq_msg_data(&reply);
          rc = zmq_msg_close(&reply);
          assert(rc == 0);
          return true;
     }

     template<class T>
     bool send_msg_dontwait(T* token, void* socket) {
          int rc;
          zmq_msg_t message;
          zmq_msg_init(&message);
          rc = zmq_msg_init_size(&message, sizeof(T));
          assert(rc == 0);
          rc = zmq_msg_init_data(&message, token, sizeof(T), NULL,
NULL);
          assert(rc == 0);
          rc = zmq_msg_send(&message, socket, ZMQ_DONTWAIT);
          if (rc == -1) {
               zmq_msg_close(&message);
               return false;
          }
          assert(rc == sizeof(T));
          return true;
     }
}

#endif /* ZMQ_STD_HPP */
```

main.cpp

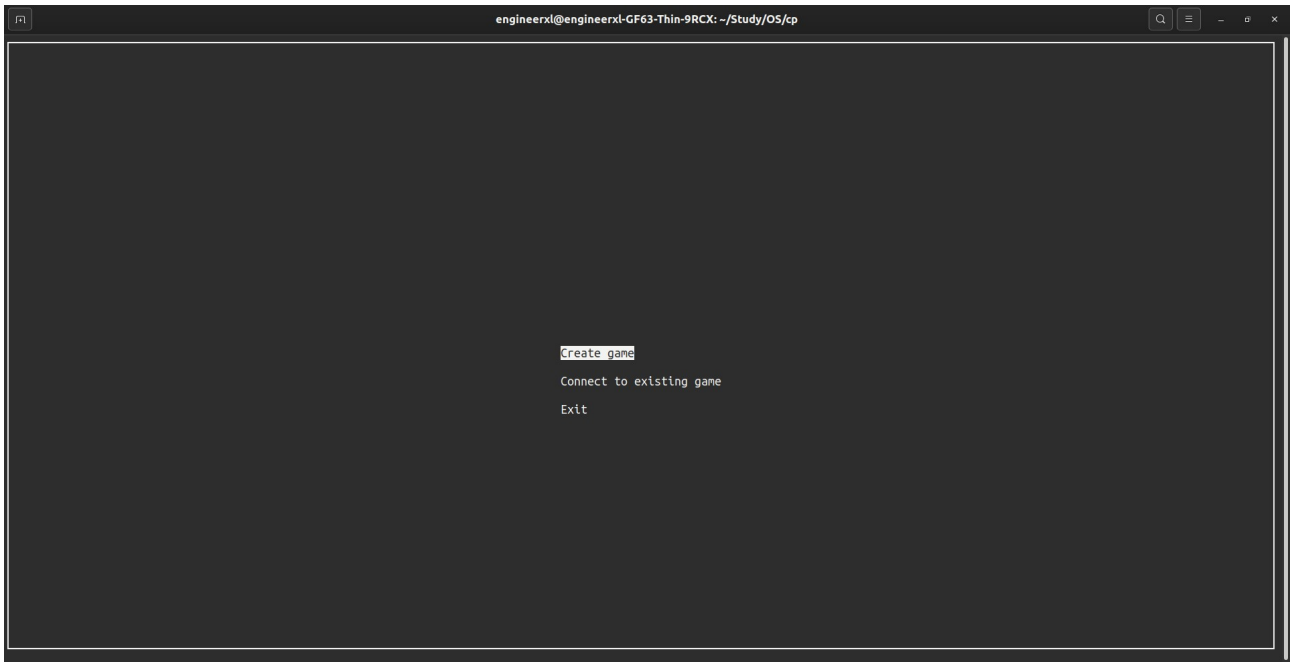
```
#include <iostream>
#include <ncurses.h>
#include <stdio.h>
#include <vector>

#include "game.hpp"
#include "zmq_std.hpp"

std::string read_cords(unsigned int x, unsigned int y) {
	std::string res;
	bool ok = true;
	int c;
	echo();
	WINDOW* window = newwin(4, 10, x, y);
	while (ok) {
		c = wgetch(window);
		if (c == 27) {
			ok = false;
		} else if (c == 10) {
			break;
		} else {
			res = res + (char)c;
		}
	}
	noecho();
	delwin(window);
	if (ok) {
		return res;
	} else {
		return std::string();
	}
}

int main() {
	const char* LOG_NAME = "log.txt";
	FILE* log_file = NULL;
	log_file = fopen(LOG_NAME, "w");
	if (log_file == NULL) {
		std::cout << "Error creating log file!" << std::endl;
		return -1;
	}
	fprintf(log_file, "Starting log...\n");
	initscr();
	if (LINES < 20 or COLS < 80) {
		std::cout << "Invalid terminal size!" << std::endl;
		endwin();
		fclose(log_file);
		return -1;
	}
```

```
	/* Disable line buffering */
	raw();
	const unsigned int start_x = 0;
	const unsigned int start_y = 0;
	const unsigned int end_x = LINES;
	const unsigned int end_y = COLS;
	const unsigned int center_x = (end_x - start_x) / 2;
	const unsigned int center_y = (end_y - start_y) / 2;
	std::vector< std::string > menu_items = {
		"Create game",
		"Connect to existing game",
		"Exit"
	};
	const size_t new_game_ind = 0;
	const size_t connect_ind = 1;
	const size_t exit_ind = 2;
	WINDOW* menu = newwin(end_x - start_x, end_y - start_y,
start_x, start_y);
	/* Do not print wgetch() */
	noecho();
	/* Enables window input */
	keypad(menu, TRUE);
	/* Hide cursor */
	curs_set(0);
	size_t menu_item_ind = 0;
	int c;
	while(1) {
		box(menu, 0, 0);
		for (size_t i = 0; i < menu_items.size(); ++i) {
			if (i == menu_item_ind) {
				/* Highlight current */
				wattron(menu, A_STANDOUT);
			}
			mvwprintw(menu, center_x + i * 2, center_y - 12,
menu_items[i].c_str());
			wattroff(menu, A_STANDOUT);
		}
		wrefresh(menu);
		c = wgetch(menu);
		fprintf(log_file, "Getting key... %d\n", c);
		if (c == KEY_DOWN) {
			if (menu_item_ind < menu_items.size() - 1) {
				++menu_item_ind;
			}
		} else if (c == KEY_UP) {
			if (menu_item_ind > 0) {
				--menu_item_ind;
			}
		} else if (c == 10) {
			if (menu_item_ind == exit_ind) {
				break;
```

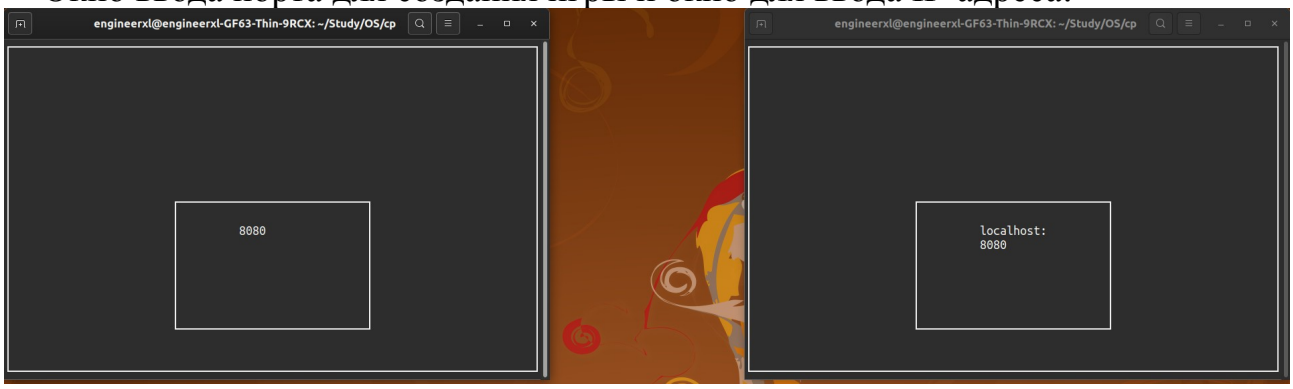
```
                } else if (menu_item_ind == new_game_ind) {
                    /* Creating new game */
                    WINDOW* port_menu = newwin(10, 30, center_x -
1, center_y - 15);
                    box(port_menu, 0, 0);
                    wrefresh(port_menu);
                    std::string port = read_cords(center_x + 1,
center_y - 5);
                    delwin(port_menu);
                    if (!port.empty()) {
                        fprintf(log_file, "Creating game on port
%s\n", port.c_str());
                        try {
                            game new_game(true,
std::string("tcp://*:") + port);
                            new_game.play();
                        } catch (std::exception & ex) {
                            fprintf(log_file, "%s\n", ex.what());
                        }
                    }
                } else if (menu_item_ind == connect_ind) {
                    /* Connecting to existing game */
                    WINDOW* ip_menu = newwin(10, 30, center_x - 1,
center_y - 15);
                    box(ip_menu, 0, 0);
                    wrefresh(ip_menu);
                    std::string ip = read_cords(center_x + 1,
center_y - 5);
                    delwin(ip_menu);
                    if (!ip.empty()) {
                        fprintf(log_file, "Connecting to %s\n",
ip.c_str());
                        try {
                            game new_game(false,
std::string("tcp://") + ip);
                            new_game.play();
                        } catch (std::exception & ex) {
                            fprintf(log_file, "%s\n", ex.what());
                        }
                    }
                }
            }
            wclear(menu);
            wrefresh(menu);
            refresh();
        }
        delwin(menu);
        endwin();
        fprintf(log_file, "End of log...\n");
        fclose(log_file);
}
```

5. Пример работы программы

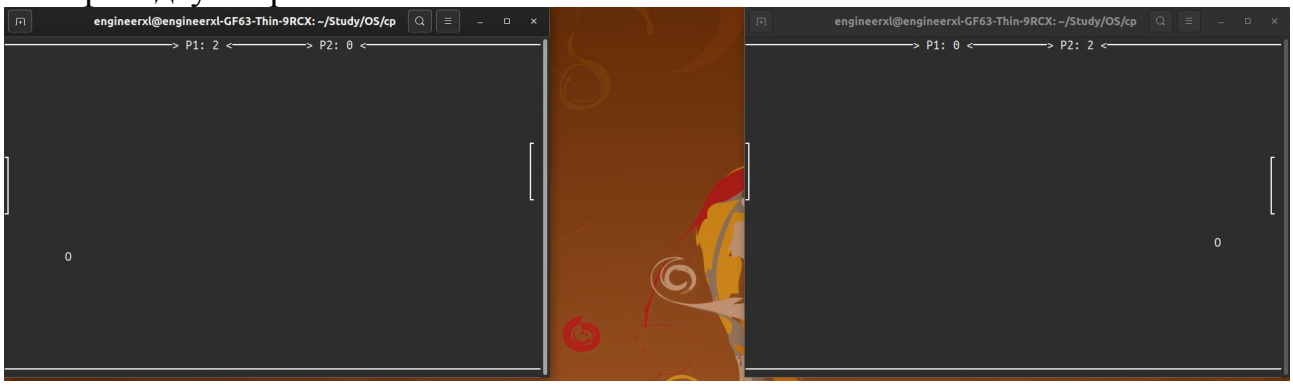
Главное меню

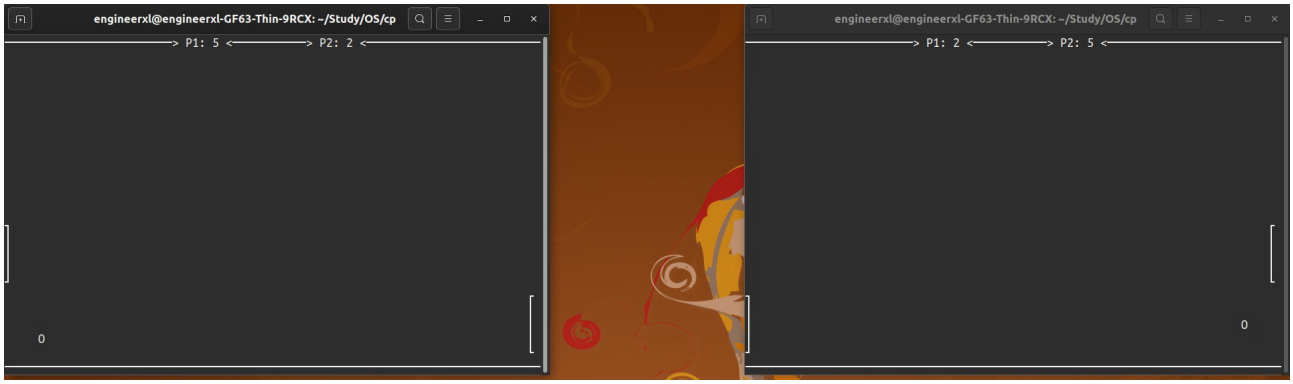


Окно ввода порта для создания игры и окно для ввода IP-адреса:

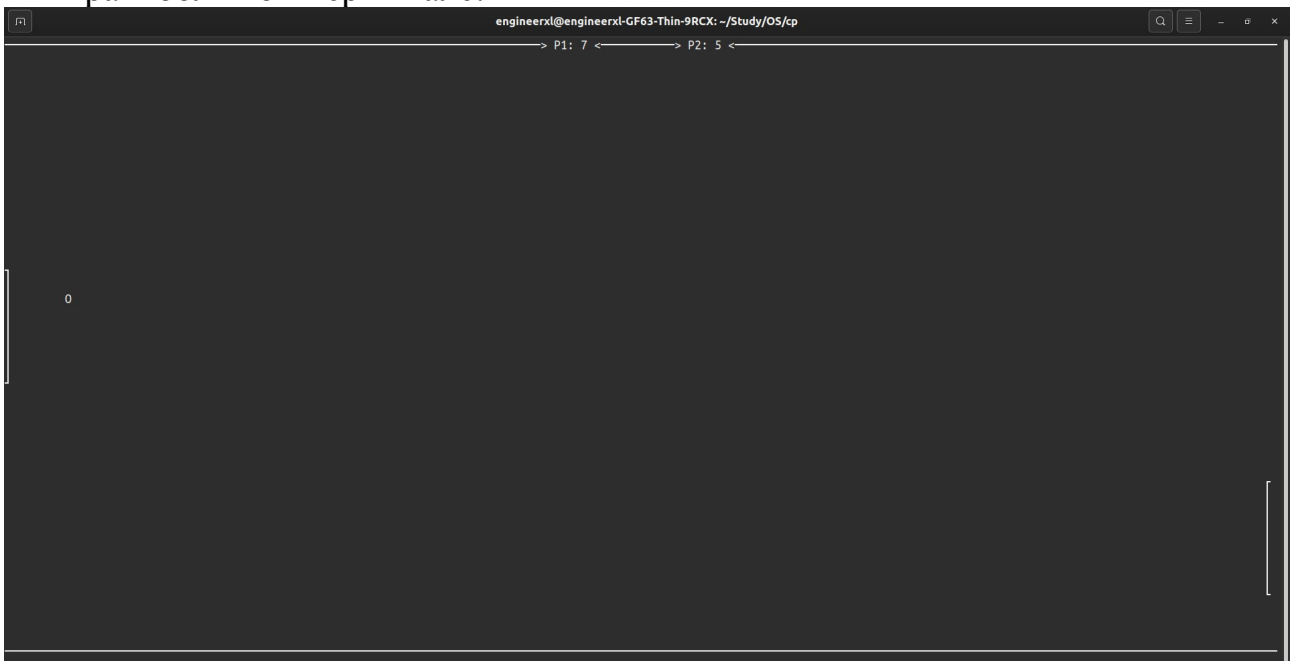


Игра с двух терминалов:





Игра в большом терминале:



6. Выводы

Во время выполнения курсового проекта я познакомился с библиотекой для создания простых консольных приложений *ncurses*, вспомнил принципы работы с очередями сообщений и библиотекой *ZeroMQ*.

Основной сложностью работы было совместить отрисовку игрового поля и работу с сообщениями. Для решения этой проблемы я использовал *ZMQ_DONTWAIT* и пытался получать сообщение несколько раз.

Для игры с другим человеком нужен открытый порт на компьютере, но провайдеры редко дают статический IP-адресс, из-за чего открытие портов не всегда возможно. Программа *Hamachi* помогла мне создать виртуальную локалькую сеть и тестировать программу с другом.

Для многопользовательской игры важно обеспечить быстрый обмен данными, чтобы минимизировать время отклика сервера на команды игрока. Как оказалось, очереди сообщений отлично справляются с этой задачей.

Список литературы

1. NCURSES Programming HOWTO
   URL: https://tldp.org/HOWTO/NCURSES-Programming-HOWTO/
   (дата обращения 25.12.2020)
2. zmq_socket(3) — 0MQ Api — ZeroMQ API
   URL: https://zeromq.org/socket-api/
   (дата обращения 25.12.2020)
3. zmq_bind(3) — 0MQ Api — ZeroMQ API
   URL: http://api.zeromq.org/2-1:zmq-bind
   (дата обращения 25.12.2020)
4. zmq_connect(3) — 0MQ Api — ZeroMQ API
   URL: http://api.zeromq.org/2-1:zmq-connect
   (дата обращения 25.12.2020)
5. Hamachi for Linux — VPN.net — Hamachi by LogMeIn
   URL: https://www.vpn.net/linux
   (дата обращения 26.12.2020)