Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовой проект по курсу**

**«Операционные системы»**

Студент: Баталин Дмитрий Андреевич

Группа: М8О–212Б–20

Вариант: 4

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023

1. **Цель работы**
2. Приобретение практических навыков в использовании знаний, полученных в течении курса
3. Проведение исследования в выбранной предметной области
4. **Задание**

Необходимо спроектировать и реализовать программный прототип в соответствии с выбранным вариантом. Произвести анализ и сделать вывод на основании данных, полученных при работе программного прототипа.

**Проектирование консольной клиент-серверной игры**

На основе любой из выбранных технологий:

1. Pipes
2. Sockets
3. Сервера очередей
4. И другие

Создать собственную игру более, чем для одного пользователя. Игра может быть устроена по принципу: клиент-клиент, сервер-клиент.

Консоль-серверная игра. Необходимо написать консоль-серверную игру. Необходимо написать 2 программы: сервер и клиент. Сначала запускается сервер, а далее клиенты соединяются с сервером. Сервер координирует клиентов между собой. При запуске клиента игрок может выбрать одно из следующих действий (возможно больше, если предусмотрено вариантом):

* Создать игру, введя ее имя
* Присоединиться к одной из существующих игр по имени игры

**Вариант 4:**

Морской бой. Общение между сервером и клиентом необходимо организовать при помощи очередей сообщений (например, ZeroMQ). Каждый игрок должен при запуске ввести свой логин. Должна быть предоставлена возможность отправить приглашение на игру другому игроку по логину

1. **Общий метод и алгоритм решения**

В проекте есть два основных файла main.cpp, client.cpp.

В файле main.cpp хранится класс игрока, класс игры и сам сервер, к которому подключаются игроки, на котором хранится состояние игры, через который проходит запрос на приглашение в игру. Используя zeromq сервер общается с клиентами, а клиенты с сервером.

В файле client.cpp хранится вся логика общения клиента с сервером. Отправка приглашения другому игроку, регистрация и взаимодействие с игрой происходит через клиент.

1. **Основные файлы программы**

**client.cpp:**

#include <iostream>

#include <unistd.h>

#include <stdlib.h>

#include <string>

#include <sstream>

#include <signal.h>

#include <zmq.hpp>

#include <thread>

#include <chrono>

**using** **namespace** std::chrono\_literals;

**const** **int** TIMER = **50000**;

**const** **int** SMOL\_TIMER = **500**;

**const** **int** DEFAULT\_PORT = **5050**;

**int** n = **2**;

**int** port\_iter;

std::string command;

**pthread\_mutex\_t** mutex;

zmq::**context\_t** context(**2**);

zmq::**socket\_t** player\_socket(context, ZMQ\_REP);

**bool** **send\_message**(zmq::**socket\_t** &socket, **const** std::string &message\_string) {

zmq::**message\_t** message(message\_string.size());

memcpy(message.data(), message\_string.c\_str(), message\_string.size());

**return** socket.send(message);

}

std::string receive\_message(zmq::**socket\_t** &socket) {

zmq::**message\_t** message;

**bool** ok = false;

try {

ok = socket.recv(&message);

}

**catch** (...) {

ok = false;

}

std::string recieved\_message(**static\_cast**<**char**\*>(message.data()), message.size());

**if** (recieved\_message.empty() || !ok) {

**return** "";

}

**return** recieved\_message;

}

**void** create\_node(**int** id, **int** port) {

**char**\* arg0 = strdup("./client");

**char**\* arg1 = strdup((std::to\_string(id)).c\_str());

**char**\* arg2 = strdup((std::to\_string(port)).c\_str());

**char**\* args[] = {arg0, arg1, arg2, NULL};

execv("./client", args);

}

std::string get\_port\_name(**const** **int** port) {

**return** "tcp://127.0.0.1:" + std::to\_string(port);

}

**void**\* check\_invite(**void** \*param) {

std::string invite\_tmp;

// std::string answer;

// std::cout << "Поток клиента для получения инвайтов начал работу и ждет запрос" << std::endl;

pthread\_mutex\_lock(&mutex);

std::string invite\_msg = receive\_message(player\_socket);

std::stringstream invite\_ss(invite\_msg);

std::getline(invite\_ss, invite\_tmp, ':');

**if** (invite\_tmp == "invite") {

std::this\_thread::sleep\_for(**100**ms);

std::getline(invite\_ss, invite\_tmp, ':');

std::cout << "Игрок с ником " + invite\_tmp + " приглашает вас в игру!" << std::endl;

std::cout << "Вы согласны? (y/n)" << std::endl;

std::cin >> command;

std::cout << "Ваш ответ: " + command + "**\n**";

**if** (command[**0**] == 'y') {

std::cout << "Вы приняли запрос!" << std::endl;

send\_message(player\_socket, "accept");

pthread\_mutex\_unlock(&mutex);

pthread\_exit(**0**);

}

**else** {

std::cout << "Вы отклонили запрос!" << std::endl;

pthread\_mutex\_unlock(&mutex);

send\_message(player\_socket, "reject");

}

}

pthread\_exit(**0**);

}

**typedef** **struct** {

} check\_params;

**int** **main**(**int** argc, **char**\*\* argv) {

zmq::**context\_t** context(**2**);

zmq::**socket\_t** main\_socket(context, ZMQ\_REQ);

**int** pid = getpid();

main\_socket.connect(get\_port\_name(**5555**));

pthread\_mutex\_init(&mutex, NULL);

check\_params check\_param;

**pthread\_t** tid;

// pthread\_join(tid, NULL);

// std::cout << "Поток клиента для отправки команд начал работу" << std::endl;

std::string received\_message;

std::string tmp;

**int** iteration = **1**;

**while**(true) {

// login

**if** (iteration == **1**) {

iteration += **1**;

std::string login;

std::cout << "Введите ваш логин: ";

std::cin >> login;

// формируем запрос

std::string login\_msg = "login:" + std::to\_string(pid) + ":" + login;

send\_message(main\_socket, login\_msg);

received\_message = receive\_message(main\_socket);

std::stringstream ss(received\_message);

// обрабатываем ответ

std::getline(ss, tmp, ':');

**if** (tmp == "Ok") {

std::getline(ss, tmp, ':');

port\_iter = std::stoi(tmp);

player\_socket.connect(get\_port\_name(**5555** + port\_iter));

std::cout << "Вы успешно авторизовались!" << std::endl;

std::cout << "Вы хотите пригласить друга? (y/n)" << std::endl;

std::cin >> tmp;

**if** (tmp[**0**] == 'n') {

std::cout << "Ждем приглашения от друга..." << std::endl;

pthread\_create(&tid, NULL, check\_invite, &check\_param);

std::this\_thread::sleep\_for(**1000**ms);

**break**;

}

**else** {

std::cout << "Чтобы пригласить друга напишите invite (friend\_login)" << std::endl;

}

}

**else** **if** (tmp == "Error") {

std::getline(ss, tmp, ':');

**if** (tmp == "NameAlreadyExist") {

std::cout << "ERROR: Это имя уже занято! Попробуйте другое" << std::endl;

iteration -= **1**;

}

}

}

**else** {

std::cin >> command;

**if** (command == "invite") {

std::string invite\_login;

std::cin >> invite\_login;

std::cout << "Вы пригласили игрока с ником " + invite\_login << std::endl;

std::cout << "Ждем ответ..." << std::endl;

std::string invite\_cmd = "invite:" + std::to\_string(port\_iter) + ":" + invite\_login;

send\_message(main\_socket, invite\_cmd);

received\_message = receive\_message(main\_socket);

std::stringstream ss(received\_message);

std::getline(ss, tmp, ':');

**if** (tmp == "accept") {

std::cout << "Запрос принят!" << std::endl;

**break**;

}

**else** **if** (tmp == "reject") {

std::cout << "Запрос отклонен! С тобой не хотят играть(" << std::endl;

// надо не делать ничего и вернуться к выбору команды хз

}

**else** **if** (tmp == "Error") {

std::getline(ss, tmp, ':');

**if** (tmp == "SelfInvite") {

std::cout << "ERROR: Вы отправили запрос на игру самому себе. Попробуйте снова" << std::endl;

}

**else** **if** (tmp == "LoginNotExist") {

std::cout << "ERROR: Игрока с таким ником не существует. Попробуйте снова" << std::endl;

}

**else** **if** (tmp == "AlreadyInviting") {

std::cout << "ERROR: Другой игрок уже хочет вас пригласить. Дадим ему это сделать" << std::endl;

pthread\_create(&tid, NULL, check\_invite, &check\_param);

**break**;

}

}

}

**else** {

std::cout << "Вы ввели несуществующую команду. Попробуйте снова" << std::endl;

}

}

}

// std::cout << "ИГРА НАЧАЛАСЬ!!" << std::endl;

pthread\_mutex\_lock(&mutex);

received\_message = receive\_message(player\_socket);

std::string answer;

**if** (received\_message == "ping") {

std::cout << "Вы готовы к игре? (y/n)" << std::endl;

std::cin >> answer;

**if** (answer[**0**] == 'y') {

send\_message(player\_socket, "pong");

std::cout << "Вы согласились. Дождитесь других игроков и мы начнем!" << std::endl;

}

**else** {

send\_message(player\_socket, "no\_pong");

std::cout << "Вы отказались. До свидания!" << std::endl;

**return** **0**;

}

}

**else** {

std::cout << "Мне прилетел не пинг" << std::endl;

}

**if** (port\_iter == **1**) {

std::cout << "Начинаем игру" << std::endl;

}

**else** {

std::cout << "Начинаем игру. Подождите, пока другой пользователь расставит корабли" << std::endl;

}

std::cout << "Чтобы расставить ваши корабли (формат: x, y и ориентация (H или V) через пробелы):" << std::endl;

**while**(**1**) {

std::string inside\_msg;

std::string send\_msg;

inside\_msg = receive\_message(player\_socket);

// std::cout << "Команда: " + inside\_msg << std::endl;

std::stringstream strs(inside\_msg);

strs >> tmp;

**if** (tmp == "Разместите") {

std::cout << inside\_msg << std::endl;

**char** orientation;

**int** x, y;

std::cin >> y >> x >> orientation;

send\_msg = "coords:" + std::to\_string(x) + ":" + std::to\_string(y) + ":" + orientation;

send\_message(player\_socket, send\_msg);

}

**else** **if** (tmp == "board") {

std::cout << inside\_msg.substr(**5**, inside\_msg.size()) << std::endl;

send\_message(player\_socket, "ok");

}

**else** **if** (tmp == "Error") {

std::cout << inside\_msg << std::endl;

send\_message(player\_socket, "ok");

}

**else** **if** (tmp == "your\_turn") {

send\_message(player\_socket, "ok");

std::cout << "Ваш ход:" << std::endl;

inside\_msg = receive\_message(player\_socket);

**if** (inside\_msg == "shoot") {

**int** x, y;

std::cout << "Введите координаты выстрела (формат: x y):" << std::endl;

std::cin >> y >> x;

send\_msg = "coords:" + std::to\_string(x) + ":" + std::to\_string(y);

send\_message(player\_socket, send\_msg);

inside\_msg = receive\_message(player\_socket);

**if** (inside\_msg == "shooted") {

std::cout << "Попадание!" << std::endl;

send\_message(player\_socket, "ok");

}

**else** **if** (inside\_msg == "miss") {

std::cout << "Промах!" << std::endl;

send\_message(player\_socket, "ok");

}

}

}

**else** **if** (tmp == "not\_your\_turn") {

std::cout << "Ход соперника: " << std::endl;

send\_message(player\_socket, "ok");

inside\_msg = receive\_message(player\_socket);

**if** (inside\_msg == "shooted") {

std::cout << "Вас подстрелили!" << std::endl;

send\_message(player\_socket, "ok");

}

**else** **if** (inside\_msg == "miss") {

std::cout << "Противник промахнулся" << std::endl;

send\_message(player\_socket, "ok");

}

}

**else** **if** (tmp == "win") {

std::cout << "Вы выиграли!" << std::endl;

send\_message(player\_socket, "ok");

**return** **0**;

}

**else** **if** (tmp == "lose") {

std::cout << "Вы проиграли!" << std::endl;

send\_message(player\_socket, "ok");

**return** **0**;

}

}

}

**main.cpp:**

#include <iostream>

#include <unistd.h>

#include <string>

#include <vector>

#include <sstream>

#include <signal.h>

#include <cassert>

#include <zmq.hpp>

#include <chrono>

#include <thread>

#include <exception>

#include <map>

#include <cstdlib>

#include <ctime>

**using** **namespace** std::chrono\_literals;

**const** **int** TIMER = **2000**;

**const** **int** SMOL\_TIMER = **500**;

**int** n = **2**;

**const** **int** BOARD\_SIZE = **10**;

zmq::**context\_t** context(**3**);

zmq::**socket\_t** main\_socket(context, ZMQ\_REP);

// общая функция для отправки сообщения в дочерний процесс

**bool** **send\_message**(zmq::**socket\_t** &socket, **const** std::string &message\_string) {

zmq::**message\_t** message(message\_string.size());

memcpy(message.data(), message\_string.c\_str(), message\_string.size());

**return** socket.send(message);

}

std::string receive\_message(zmq::**socket\_t** &socket) {

zmq::**message\_t** message;

**bool** ok = false;

try {

ok = socket.recv(&message);

}

**catch** (...) {

ok = false;

}

std::string recieved\_message(**static\_cast**<**char**\*>(message.data()), message.size());

**if** (recieved\_message.empty() || !ok) {

**return** "Root is dead";

}

**return** recieved\_message;

}

// меняем созданый fork процесс на дочерний, передавая туда нужные нам аргументы (клиент)

**void** create\_node(**int** id, **int** port) {

**char**\* arg0 = strdup("./client");

**char**\* arg1 = strdup((std::to\_string(id)).c\_str());

**char**\* arg2 = strdup((std::to\_string(port)).c\_str());

**char**\* args[] = {arg0, arg1, arg2, NULL};

execv("./client", args);

}

// функция, собирающая полный адрес до дочернего процесса

std::string get\_port\_name(**const** **int** port) {

**return** "tcp://127.0.0.1:" + std::to\_string(port);

}

**bool** is\_number(std::string val) {

try {

**int** tmp = std::stoi(val);

**return** true;

}

**catch**(std::exception& e) {

std::cout << "Error: " << e.what() << "**\n**";

**return** false;

}

}

**class** **Player** {

**public:**

std::vector< std::vector<**char**> > board;

**int** num;

Player() {

board = std::vector< std::vector<**char**> >(BOARD\_SIZE, std::vector<**char**>(BOARD\_SIZE, ' '));

}

**void** placeShips(zmq::**socket\_t** &player\_socket) {

// Расставляем корабли

std::string msg;

std::string tmp;

// msg = "Расставьте ваши корабли (формат: x, y и ориентация (H или V) через пробелы):\n";

// std::cout << "Расставьте ваши корабли (формат: x, y и ориентация (H или V) через пробелы):" << std::endl;

**int** ships\_count = **5**;

**for** (**int** i = **1**; i <= **1**; ++i) {

ships\_count -= **1**;

**for** (**int** j = **0**; j < ships\_count; j++) {

**char** orientation;

**int** x, y;

msg = "Разместите корабль " + std::to\_string(i) + " (1x" + std::to\_string(i) + "): ";

// std::cout << msg << std::endl;

// std::cout << "Разместите корабль " << i << " (1x" << i << "): ";

send\_message(player\_socket, msg);

std::string received\_message = receive\_message(player\_socket);

std::cout << "Получил запрос: " + received\_message << std::endl;

std::stringstream strs(received\_message);

std::getline(strs, tmp, ':');

**if** (tmp == "coords") {

std::getline(strs, tmp, ':');

x = std::stoi(tmp);

std::getline(strs, tmp, ':');

y = std::stoi(tmp);

std::getline(strs, tmp, ':');

orientation = tmp[**0**];

// std::cin >> y >> x >> orientation;

**if** (!((orientation == 'V') or (orientation == 'H'))) {

// std::cout << "Ты по-моему перепутал" << std::endl;

send\_message(player\_socket, "Error : Ты по-моему перепутал");

received\_message = receive\_message(player\_socket);

--j;

**continue**;

}

**if** (isValidPlacement(x, y, i, orientation)) {

placeShip(x, y, i, orientation);

} **else** {

send\_message(player\_socket, "Error : Неверное местоположение! Попробуйте еще раз.");

received\_message = receive\_message(player\_socket);

// std::cout << "Неверное местоположение! Попробуйте еще раз." << std::endl;

--j;

**continue**;

}

std::string boardState = "board";

boardState += getBoard();

// std::cout << boardState.substr(5, boardState.size()) << std::endl;

send\_message(player\_socket, boardState);

received\_message = receive\_message(player\_socket);

}

}

}

}

**bool** isValidPlacement(**int** x, **int** y, **int** size, **char** orientation) **const** {

**if** ((x > **9**) or (x < **0**) or (y > **9**) or (y < **0**)) {

**return** false;

}

**if** (orientation == 'V') {

**if** (x + size - **1** >= BOARD\_SIZE) {

**return** false;

}

**for** (**int** i = x; i < x + size; ++i) {

**if** (board[i][y] != ' ') {

**return** false;

}

}

}

**else** **if** (orientation == 'H') {

**if** (y + size - **1** >= BOARD\_SIZE) {

**return** false;

}

**for** (**int** j = y; j < y + size; ++j) {

**if** (board[x][j] != ' ') {

**return** false;

}

}

}

**for** (**int** i = **0**; i < size; i++) {

**if** (orientation == 'H') {

**if** (!isEmptyAround(x + i, y)) {

**return** false;

}

}

**else** {

**if** (!isEmptyAround(x, y + i)) {

**return** false;

}

}

}

**return** true;

}

**bool** isEmptyAround(**int** x, **int** y) **const** {

**for** (**int** i = x - **1**; i <= x + **1**; ++i) {

**for** (**int** j = y - **1**; j <= y + **1**; ++j) {

**if** (i >= **0** && i < BOARD\_SIZE && j >= **0** && j < BOARD\_SIZE && board[i][j] != ' ') {

**return** false;

}

}

}

**return** true;

}

**void** placeShip(**int** x, **int** y, **int** size, **char** orientation) {

**if** (orientation == 'V') {

**for** (**int** i = x; i < x + size; ++i) {

board[i][y] = 'O';

}

} **else** **if** (orientation == 'H') {

**for** (**int** j = y; j < y + size; ++j) {

board[x][j] = 'O';

}

}

}

std::string getBoard() **const** {

std::string result;

std::string probel(**1**, ' ');

result = " 0 1 2 3 4 5 6 7 8 9**\n**";

**for** (**int** i = **0**; i < BOARD\_SIZE; ++i) {

result += std::to\_string(i) + " ";

**for** (**int** j = **0**; j < BOARD\_SIZE; ++j) {

std::string brd(**1**, board[i][j]);

result += brd + probel;

}

result += '\n';

}

result += '\n';

**return** result;

}

std::string getClearBoard() **const** {

std::string result;

std::string space(**1**, ' ');

result = " 0 1 2 3 4 5 6 7 8 9**\n**";

**for** (**int** i = **0**; i < BOARD\_SIZE; ++i) {

result += std::to\_string(i) + " ";

**for** (**int** j = **0**; j < BOARD\_SIZE; ++j) {

std::string brd(**1**, board[i][j]);

**if** (brd == "O") {

result += space + space;

}

**else** {

result += brd + space;

}

}

result += '\n';

}

result += '\n';

**return** result;

}

};

**class** **Game** {

**public:**

Player player1;

Player player2;

**void** **play**(zmq::**socket\_t** &player1\_socket, zmq::**socket\_t** &player2\_socket) {

std::cout << "Игра **\"**Морской бой**\"** началась!" << std::endl;

std::string msg1;

std::string msg2;

// Расставляем корабли для каждого игрока

// pthread\_t tid[2];

// pthread\_create(&tid[0], NULL, player\_placement, &player1);

// pthread\_create(&tid[1], NULL, player\_placement, &player2);

// pthread\_join(tid[0], NULL);

// pthread\_join(tid[1], NULL);

player1.num = **1**;

player2.num = **2**;

player1.placeShips(player1\_socket);

player2.placeShips(player2\_socket);

// Начинаем игру

**int** turn = **0**;

**while** (!gameOver()) {

**if** (turn % **2** == **0**) {

msg1 = "your\_turn";

msg2 = "not\_your\_turn";

send\_message(player1\_socket, msg1);

receive\_message(player1\_socket);

send\_message(player2\_socket, msg2);

receive\_message(player2\_socket);

std::cout << "Ход игрока 1:" << std::endl;

**if** (playerTurn(player1, player2, player1\_socket, player2\_socket)) {

**if** (gameOver()) {

std::cout << "Победил игрок 1" << std::endl;

send\_message(player1\_socket, "win");

receive\_message(player1\_socket);

send\_message(player2\_socket, "lose");

receive\_message(player2\_socket);

**break**;

}

**continue**;

}

**else** {

turn += **1**;

}

}

**else** {

std::cout << "Ход игрока 2:" << std::endl;

msg1 = "your\_turn";

msg2 = "not\_your\_turn";

send\_message(player2\_socket, msg1);

receive\_message(player2\_socket);

send\_message(player1\_socket, msg2);

receive\_message(player1\_socket);

**if** (playerTurn(player2, player1, player2\_socket, player1\_socket)) {

**if** (gameOver()) {

std::cout << "Победил игрок 2" << std::endl;

send\_message(player2\_socket, "win");

receive\_message(player2\_socket);

send\_message(player1\_socket, "lose");

receive\_message(player1\_socket);

**break**;

}

**continue**;

}

**else** {

turn += **1**;

}

}

}

std::cout << "Игра завершена!" << std::endl;

}

**bool** **gameOver**() **const** {

**return** allShipsSunk(player1) || allShipsSunk(player2);

}

**bool** allShipsSunk(**const** Player& player) **const** {

**for** (**const** **auto**& row : player.board) {

**for** (**char** cell : row) {

**if** (cell == 'O') {

**return** false;

}

}

}

**return** true;

}

**bool** playerTurn(Player& attacker, Player& defender, zmq::**socket\_t** &attacker\_socket, zmq::**socket\_t** &defender\_socket) {

**bool** shoot = false;

std::string received\_message;

std::string tmp;

**int** x, y;

send\_message(attacker\_socket, "shoot");

received\_message = receive\_message(attacker\_socket);

std::stringstream strs(received\_message);

std::getline(strs, tmp, ':');

std::getline(strs, tmp, ':');

x = std::stoi(tmp);

std::getline(strs, tmp, ':');

y = std::stoi(tmp);

**if** (isValidMove(x, y, defender)) {

**if** (defender.board[x][y] == 'O') {

send\_message(attacker\_socket, "shooted");

received\_message = receive\_message(attacker\_socket);

send\_message(defender\_socket, "shooted");

received\_message = receive\_message(defender\_socket);

std::cout << "Попадание!" << std::endl;

defender.board[x][y] = 'X';

shoot = true;

// std::cout << defender.getBoard();

**return** shoot;

} **else** {

send\_message(attacker\_socket, "miss");

received\_message = receive\_message(attacker\_socket);

send\_message(defender\_socket, "miss");

received\_message = receive\_message(defender\_socket);

std::cout << "Промах!" << std::endl;

defender.board[x][y] = '\*';

std::string board = defender.getBoard();

std::string clearBoard = defender.getClearBoard();

send\_message(attacker\_socket, "board" + clearBoard);

received\_message = receive\_message(attacker\_socket);

send\_message(defender\_socket, "board" + board);

received\_message = receive\_message(defender\_socket);

**return** shoot;

}

} **else** {

std::cout << "Неверные координаты! Попробуйте еще раз." << std::endl;

**return** **playerTurn**(attacker, defender, attacker\_socket, defender\_socket);

}

}

**bool** isValidMove(**int** x, **int** y, **const** Player& defender) **const** {

**return** x >= **0** && x < BOARD\_SIZE && y >= **0** && y < BOARD\_SIZE &&

(defender.board[x][y] == ' ' || defender.board[x][y] == 'O');

}

};

// void\* player\_placement(void \*param) {

// Player \*player = (Player\*) param;

// player->placeShips();

// pthread\_exit(0);

// }

**int** **main**() {

// zmq::context\_t context(3);

// zmq::socket\_t main\_socket(context, ZMQ\_REP);

zmq::**socket\_t** first\_player\_socket(context, ZMQ\_REQ);

zmq::**socket\_t** second\_player\_socket(context, ZMQ\_REQ);

main\_socket.bind("tcp://\*:5555");

first\_player\_socket.bind("tcp://\*:5556");

second\_player\_socket.bind("tcp://\*:5557");

std::cout << "Сервер начал работу" << std::endl;

std::map<**int**, std::string> login\_map;

**int** port\_iter = **1**;

**while** (true) {

std::string received\_message = receive\_message(main\_socket);

std::cout << "На сервер поступил запрос: '" + received\_message + "'" << std::endl;

std::stringstream ss(received\_message);

std::string tmp;

std::getline(ss, tmp, ':');

**if** (tmp == "login") {

**if** (port\_iter > **2**) {

send\_message(main\_socket, "Error:TwoPlayersAlreadyExist");

}

**else** {

std::getline(ss, tmp, ':');

std::getline(ss, tmp, ':');

**if** (login\_map[**1**] == tmp) {

std::cout << "Игрок ввел занятое имя" << std::endl;

send\_message(main\_socket, "Error:NameAlreadyExist");

}

**else** {

std::cout << "Логин игрока номер " + std::to\_string(port\_iter) + ": " + tmp << std::endl;

std::string login = tmp;

login\_map[port\_iter] = login;

send\_message(main\_socket, "Ok:" + std::to\_string(port\_iter));

port\_iter += **1**;

}

}

}

// получили от клиента запрос на игру другому клиенту

**else** **if** (tmp == "invite") {

std::cout << "Обрабатываю инвайт" << std::endl;

std::this\_thread::sleep\_for(**100**ms);

std::string invite\_login;

std::getline(ss, tmp, ':');

**int** sender\_id = std::stoi(tmp);

std::getline(ss, invite\_login, ':');

// если клиент 1 отправил запрос клиенту 2

**if** (invite\_login == login\_map[sender\_id]) {

std::cout << "Игрок пригласил сам себя" << std::endl;

send\_message(main\_socket, "Error:SelfInvite");

}

**else** **if** (invite\_login == login\_map[**2**]) {

std::cout << "Игрок " + login\_map[**1**] + " пригласил в игру " + login\_map[**2**] << std::endl;

send\_message(second\_player\_socket, "invite:" + login\_map[**1**]);

std::string invite\_message = receive\_message(second\_player\_socket);

second\_player\_socket.set(zmq::sockopt::rcvtimeo, -**1**);

**if** (invite\_message == "accept") {

std::cout << "Игрок " + login\_map[**2**] + " принял запрос " << std::endl;

send\_message(main\_socket, invite\_message);

**break**;

}

**else** **if** (invite\_message == "reject") {

std::cout << "Игрок " + login\_map[**2**] + " отклонил запрос " << std::endl;

send\_message(main\_socket, invite\_message);

}

**else** {

std::cout << "Что-то пошло не так во время обработки запроса на игру" << std::endl;

}

}

// если клиент 2 отправил запрос клиенту 1

**else** **if** (invite\_login == login\_map[**1**]){

std::cout << "Игрок " + login\_map[**2**] + " пригласил в игру " + login\_map[**1**] << std::endl;

send\_message(first\_player\_socket, "invite:" + login\_map[**2**]);

std::string invite\_message = receive\_message(first\_player\_socket);

**if** (invite\_message == "accept") {

std::cout << "Игрок " + login\_map[**1**] + " принял запрос " << std::endl;

send\_message(main\_socket, invite\_message);

**break**;

}

**else** **if** (invite\_message == "reject") {

std::cout << "Игрок " + login\_map[**1**] + " отклонил запрос " << std::endl;

send\_message(main\_socket, invite\_message);

}

**else** {

std::cout << "Что-то пошло не так во время обработки запроса на игру" << std::endl;

}

}

**else** {

std::cout << "Ника " + invite\_login + " нет в базе" << std::endl;

std::cout << "Отправляю ошибку игроку" << std::endl;

std::this\_thread::sleep\_for(**100**ms);

send\_message(main\_socket, "Error:LoginNotExist");

}

std::getline(ss, tmp, ':');

}

}

std::cout << "Опрашиваю игроков" << std::endl;

send\_message(first\_player\_socket, "ping");

send\_message(second\_player\_socket, "ping");

std::string received\_message1 = receive\_message(first\_player\_socket);

std::string received\_message2 = receive\_message(second\_player\_socket);

**if** (received\_message1 == "pong") {

std::cout << "Игрок " + login\_map[**1**] + " готов!" << std::endl;

}

**else** {

std::cout << "Игрок " + login\_map[**1**] + " откзался от игры" << std::endl;

**return** **0**;

}

**if** (received\_message2 == "pong") {

std::cout << "Игрок " + login\_map[**2**] + " готов!" << std::endl;

}

**else** {

std::cout << "Игрок " + login\_map[**2**] + " откзался от игры" << std::endl;

**return** **0**;

}

std::cout << "Начинаю игру!" << std::endl;

Game game;

game.play(first\_player\_socket, second\_player\_socket);

}

1. **Демонстрация работы программы**

**Логи сервера**

**moses@MBP-Dmitrij OS % /Users/moses/Documents/cs/OS/build/kp/kp\_server**

Сервер начал работу

На сервер поступил запрос: 'login:19900:dima'

Логин игрока номер 1: dima

На сервер поступил запрос: 'login:19904:sasha'

Логин игрока номер 2: sasha

На сервер поступил запрос: 'invite:2:dima'

Обрабатываю инвайт

Игрок sasha пригласил в игру dima

Игрок dima принял запрос

Опрашиваю игроков

Игрок dima готов!

Игрок sasha готов!

Начинаю игру!

Игра "Морской бой" началась!

Получил запрос: coords:9:9:H

Получил запрос: coords:0:9:V

Получил запрос: coords:9:0:V

Получил запрос: coords:0:0:V

Получил запрос: coords:1:1:H

Получил запрос: coords:8:8:H

Получил запрос: coords:8:1:H

Получил запрос: coords:1:8:H

Ход игрока 1:

Промах!

Ход игрока 2:

Промах!

Ход игрока 1:

Попадание!

Ход игрока 1:

Промах!

Ход игрока 2:

Попадание!

Ход игрока 2:

Попадание!

Ход игрока 2:

Попадание!

Ход игрока 2:

Попадание!

Победил игрок 2

Игра завершена!

**Вывод первого игрока**

moses@MBP-Dmitrij OS % /Users/moses/Documents/cs/OS/build/kp/kp\_client

Введите ваш логин: dima

Вы успешно авторизовались!

Вы хотите пригласить друга? (y/n)

n

Ждем приглашения от друга...

Игрок с ником sasha приглашает вас в игру!

Вы согласны? (y/n)

y

Ваш ответ: y

Вы приняли запрос!

Вы готовы к игре? (y/n)

y

Вы согласились. Дождитесь других игроков и мы начнем!

Начинаем игру

Чтобы расставить ваши корабли (формат: x, y и ориентация (H или V) через пробелы):

Разместите корабль 1 (1x1):

9 9 H

0 1 2 3 4 5 6 7 8 9

0

1

2

3

4

5

6

7

8

9 O

Разместите корабль 1 (1x1):

9 0 V

0 1 2 3 4 5 6 7 8 9

0 O

1

2

3

4

5

6

7

8

9 O

Разместите корабль 1 (1x1):

0 9 V

0 1 2 3 4 5 6 7 8 9

0 O

1

2

3

4

5

6

7

8

9 O O

Разместите корабль 1 (1x1):

0 0 V

0 1 2 3 4 5 6 7 8 9

0 O O

1

2

3

4

5

6

7

8

9 O O

Ваш ход:

Введите координаты выстрела (формат: x y):

3 3

Промах!

0 1 2 3 4 5 6 7 8 9

0

1

2

3 \*

4

5

6

7

8

9

Ход соперника:

Противник промахнулся

0 1 2 3 4 5 6 7 8 9

0 O O

1

2 \*

3

4

5

6

7

8

9 O O

Ваш ход:

Введите координаты выстрела (формат: x y):

1 1

Попадание!

Ваш ход:

Введите координаты выстрела (формат: x y):

5 5

Промах!

0 1 2 3 4 5 6 7 8 9

0

1 X

2

3 \*

4

5 \*

6

7

8

9

Ход соперника:

Вас подстрелили!

Ход соперника:

Вас подстрелили!

Ход соперника:

Вас подстрелили!

Ход соперника:

Вас подстрелили!

Вы проиграли!

moses@MBP-Dmitrij OS %

**Вывод второго игрока**

moses@MBP-Dmitrij OS % /Users/moses/Documents/cs/OS/build/kp/kp\_client

Введите ваш логин: sasha

Вы успешно авторизовались!

Вы хотите пригласить друга? (y/n)

y

Чтобы пригласить друга напишите invite (friend\_login)

invite dima

Вы пригласили игрока с ником dima

Ждем ответ...

Запрос принят!

Вы готовы к игре? (y/n)

y

Вы согласились. Дождитесь других игроков и мы начнем!

Начинаем игру. Подождите, пока другой пользователь расставит корабли

Чтобы расставить ваши корабли (формат: x, y и ориентация (H или V) через пробелы):

Разместите корабль 1 (1x1):

1 1 H

0 1 2 3 4 5 6 7 8 9

0

1 O

2

3

4

5

6

7

8

9

Разместите корабль 1 (1x1):

8 8 H

0 1 2 3 4 5 6 7 8 9

0

1 O

2

3

4

5

6

7

8 O

9

Разместите корабль 1 (1x1):

1 8 H

0 1 2 3 4 5 6 7 8 9

0

1 O

2

3

4

5

6

7

8 O O

9

Разместите корабль 1 (1x1):

8 1 H

0 1 2 3 4 5 6 7 8 9

0

1 O O

2

3

4

5

6

7

8 O O

9

Ход соперника:

Противник промахнулся

0 1 2 3 4 5 6 7 8 9

0

1 O O

2

3 \*

4

5

6

7

8 O O

9

Ваш ход:

Введите координаты выстрела (формат: x y):

2 2

Промах!

0 1 2 3 4 5 6 7 8 9

0

1

2 \*

3

4

5

6

7

8

9

Ход соперника:

Вас подстрелили!

Ход соперника:

Противник промахнулся

0 1 2 3 4 5 6 7 8 9

0

1 X O

2

3 \*

4

5 \*

6

7

8 O O

9

Ваш ход:

Введите координаты выстрела (формат: x y):

0 0

Попадание!

Ваш ход:

Введите координаты выстрела (формат: x y):

0 9

Попадание!

Ваш ход:

Введите координаты выстрела (формат: x y):

9 0

Попадание!

Ваш ход:

Введите координаты выстрела (формат: x y):

9 9

Попадание!

Вы выиграли!

moses@MBP-Dmitrij OS %

1. **Выводы**

После выполнения этого курсового проекта я получил полностью готовую игру Морской бой, которая работает по принципу клиент-сервер. С небольшими изменениями по подключению клиента и сервера я смогу вывести эту игру в онлайн и играть со своими друзьями в эту игру дистанционно.

Благодаря данному курсовому проекту я закрепил все полученные в курсе операционных систем знания. Я получил большой опыт в работе с очередями сообщений, научился логически строить структуру своего проекта.

Самой сложной частью работы была система приглашений, но и с этим у меня получилось справиться.