Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовая работа по курсу**

**«Операционные системы»**

**III Семестр**

**Проектирование консольной клиент-серверной игры**

Студент: Ветошкина София Владимировна

Группа: М8О-203Б-23

Вариант: 4

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2024

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc186295405)

[1. Цель работы 3](#_Toc186295406)

[2. Задание 3](#_Toc186295407)

[РЕАЛИЗАЦИЯ 4](#_Toc186295408)

[1. Общий метод и алгоритм решения 4](#_Toc186295409)

[2. Основные файлы программы 7](#_Toc186295410)

[РЕЗУЛЬТАТЫ 20](#_Toc186295411)

[Демонстрация работы программы 20](#_Toc186295412)

[ЗАКЛЮЧЕНИЕ 25](#_Toc186295413)

# ВВЕДЕНИЕ

## 1. Цель работы

1. Приобретение практических навыков в использовании знаний, полученных в течении курса
2. Проведение исследования в выбранной предметной области

## 2. 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.

# РЕАЛИЗАЦИЯ

## 1. Общий метод и алгоритм решения

Основные функции:

* Регистрация игроков с уникальными логинами.
* Создание игр и отправка приглашений.
* Организация игрового процесса между двумя игроками с синхронизацией действий.

**Общий метод решения**

1. **Выбор архитектуры**:
   * Используется модель клиент-сервер, где сервер выполняет функции координации.
   * Клиенты взаимодействуют через сервер с использованием очередей сообщений ZeroMQ.
2. **Разработка сервера**:
   * Сервер принимает подключения от клиентов.
   * Хранит информацию о зарегистрированных игроках и их статусах.
   * Обрабатывает команды, такие как регистрация, приглашение, принятие приглашений, и управление игровым процессом.
3. **Разработка клиента**:
   * Клиент предоставляет интерфейс для взаимодействия игрока с системой (ввод логина, отправка команд).
   * Обеспечивает синхронизацию с сервером для выполнения игровых действий (расстановка кораблей, стрельба).
4. **Алгоритмы обработки команд**:
   * Сервер и клиенты используют ZeroMQ для обмена сообщениями в формате ключ-значение.
   * Сервер обрабатывает сообщения от клиентов и передает соответствующие ответы.
5. **Игровая логика**:
   * На стороне сервера реализована логика игры "Морской бой", включая проверку расстановки кораблей, обработки выстрелов и определения победителя.
   * Игровое поле каждого игрока представлено двухмерным массивом.

**Алгоритм работы системы**

1. **Инициализация**:
   * Сервер запускает три сокета ZeroMQ: управляющий сокет и сокеты для двух игроков.
   * Клиенты подключаются к управляющему сокету сервера.
2. **Регистрация игроков**:
   * Клиент отправляет запрос на регистрацию с логином.
   * Сервер проверяет уникальность логина и возвращает подтверждение регистрации.
3. **Отправка приглашений**:
   * Игрок может отправить приглашение другому игроку по логину.
   * Сервер проверяет корректность логина адресата и пересылает приглашение.
   * Адресат принимает или отклоняет приглашение. В случае согласия начинается игра.
4. **Игровой процесс**:
   * Расстановка кораблей:
     + Каждый игрок расставляет корабли на своём игровом поле. Сервер проверяет корректность координат и отправляет подтверждение или сообщение об ошибке.
   * Ходы игроков:
     + Сервер поочередно запрашивает координаты выстрелов у игроков.
     + Проверяет попадания или промахи, обновляет игровое поле и уведомляет обоих игроков о результатах.
   * Завершение игры:
     + Сервер проверяет, остались ли у игрока непотопленные корабли. Если нет, объявляется победитель.
5. **Закрытие соединения**:
   * После завершения игры сервер освобождает ресурсы и разрывает соединения с клиентами.

**Пример взаимодействия клиента и сервера**

1. **Регистрация игрока**:
   * Клиент: login:<PID>:<login>
   * Сервер: Ok:<player\_id> или Error:NameAlreadyExist
2. **Отправка приглашения**:
   * Клиент: invite:<player\_id>:<friend\_login>
   * Сервер: invite:<sender\_login> (адресату), затем accept или reject (ответ).
3. **Ход игрока**:
   * Сервер: your\_turn
   * Клиент: coords:<x>:<y>
   * Сервер: shooted или miss

Представленный подход обеспечивает:

* Стабильное взаимодействие между клиентами через сервер.
* Простую и масштабируемую архитектуру, использующую ZeroMQ для обмена сообщениями.
* Интуитивный пользовательский интерфейс и синхронизацию действий игроков.

Дальнейшие улучшения могут включать поддержку больше двух игроков, оптимизацию проверки выстрелов и улучшение визуализации игрового процесса.

## 2. Основные файлы программы

**main\_client.cpp:**

#include <iostream>

#include <unistd.h>

#include <string>

#include <sstream>

#include <signal.h>

#include <zmq.hpp>

#include <thread>

#include <chrono>

#include <pthread.h>

using namespace std::chrono\_literals;

const int DEFAULT\_PORT = 5050;

// commandMutex (для защиты при проверке приглашения)

pthread\_mutex\_t commandMutex;

zmq::context\_t zmqContext(2);

zmq::socket\_t playerSocket(zmqContext, ZMQ\_REP);

int playerId;

std::string userCommand;

// Отправка сообщения по сокету

bool sendZmqMessage(zmq::socket\_t &socket, const std::string &msg) {

zmq::message\_t message(msg.size());

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

return socket.send(message);

}

// Приём сообщения по сокету

std::string receiveZmqMessage(zmq::socket\_t &socket) {

zmq::message\_t message;

bool ok = false;

try {

ok = socket.recv(&message);

}

catch (...) {

ok = false;

}

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

if (received.empty() || !ok) {

return "Ошибка получения сообщения!";

}

return received;

}

// Формируем строку вида "tcp://127.0.0.1:<port>"

std::string getPortName(int port) {

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

}

// Параметры для потока (пока структура пустая)

typedef struct {

} CheckInviteParams;

// Функция, которая в отдельном потоке проверяет, пришло ли приглашение

void\* checkInvite(void \*param) {

std::string inviteTempString;

pthread\_mutex\_lock(&commandMutex);

std::string inviteMsg = receiveZmqMessage(playerSocket);

std::stringstream inviteStream(inviteMsg);

std::getline(inviteStream, inviteTempString, ':');

if (inviteTempString == "invite") {

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

std::getline(inviteStream, inviteTempString, ':');

std::cout << "Игрок с ником " << inviteTempString << " приглашает вас в игру!" << std::endl;

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

std::cin >> userCommand;

std::cout << "Ваш ответ: " << userCommand << "\n";

if (!userCommand.empty() && userCommand[0] == 'y') {

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

sendZmqMessage(playerSocket, "accept");

pthread\_mutex\_unlock(&commandMutex);

pthread\_exit(nullptr);

} else {

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

pthread\_mutex\_unlock(&commandMutex);

sendZmqMessage(playerSocket, "reject");

}

}

pthread\_exit(nullptr);

}

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

// Сокет для связи с сервером

zmq::context\_t contextLocal(2);

zmq::socket\_t serverSocket(contextLocal, ZMQ\_REQ);

// Подключаемся к основному серверу, слушающему порт 5555

serverSocket.connect(getPortName(5555));

// Инициализируем мьютекс

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

// Параметры для потока

CheckInviteParams checkInviteParams;

pthread\_t inviteThread;

int processId = getpid();

std::string serverResponse;

std::string tempString;

int iteration = 1;

while(true) {

// login

if (iteration == 1) {

iteration++;

std::string userLogin;

std::cout << "Введите ваш логин: ";

std::cin >> userLogin;

// Формируем запрос

std::string loginMessage = "login:" + std::to\_string(processId) + ":" + userLogin;

sendZmqMessage(serverSocket, loginMessage);

serverResponse = receiveZmqMessage(serverSocket);

std::stringstream ss(serverResponse);

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

// Обрабатываем ответ

if (tempString == "Ok") {

// Считываем, какой номер порта/идентификатор у нашего игрока

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

playerId = std::stoi(tempString);

// Подключаемся к сокету, который сервер выделил нашему клиенту

playerSocket.connect(getPortName(5555 + playerId));

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

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

std::cin >> tempString;

if (!tempString.empty() && tempString[0] == 'n') {

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

pthread\_create(&inviteThread, NULL, checkInvite, &checkInviteParams);

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

break;

} else {

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

}

} else if (tempString == "Error") {

// Смотрим конкретную ошибку

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

if (tempString == "NameAlreadyExist") {

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

iteration--;

}

}

} else {

// Ждём от пользователя команды

std::cin >> userCommand;

if (userCommand == "invite") {

std::string friendLogin;

std::cin >> friendLogin;

std::cout << "Вы пригласили игрока с ником " << friendLogin << std::endl;

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

// Отправляем приглашение на сервер

std::string inviteCmd = "invite:" + std::to\_string(playerId) + ":" + friendLogin;

sendZmqMessage(serverSocket, inviteCmd);

serverResponse = receiveZmqMessage(serverSocket);

std::stringstream ss(serverResponse);

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

if (tempString == "accept") {

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

break;

}

else if (tempString == "reject") {

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

}

else if (tempString == "Error") {

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

if (tempString == "SelfInvite") {

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

}

else if (tempString == "LoginNotExist") {

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

}

else if (tempString == "AlreadyInviting") {

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

pthread\_create(&inviteThread, NULL, checkInvite, &checkInviteParams);

break;

}

}

} else {

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

}

}

}

// Ожидаем "ping" от сервера, чтобы войти в игру

pthread\_mutex\_lock(&commandMutex);

serverResponse = receiveZmqMessage(playerSocket);

std::string playerAnswer;

if (serverResponse == "ping") {

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

std::cin >> playerAnswer;

if (!playerAnswer.empty() && playerAnswer[0] == 'y') {

sendZmqMessage(playerSocket, "pong");

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

} else {

sendZmqMessage(playerSocket, "no\_pong");

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

return 0;

}

} else {

std::cout << "Пришло неизвестное сообщение вместо 'ping'!" << std::endl;

}

if (playerId == 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). Подождите приглашения к размещению." << std::endl;

while(true) {

std::string incomingMessage = receiveZmqMessage(playerSocket);

std::stringstream strs(incomingMessage);

strs >> tempString;

if (tempString == "Разместите") {

std::cout << incomingMessage << std::endl;

char orientation;

int x, y;

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

std::string sendMsg = "coords:" + std::to\_string(x) + ":" + std::to\_string(y) + ":" + orientation;

sendZmqMessage(playerSocket, sendMsg);

}

else if (tempString == "board") {

// Выводим доску после слова "board"

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

sendZmqMessage(playerSocket, "ok");

}

else if (tempString == "Error") {

std::cout << incomingMessage << std::endl;

sendZmqMessage(playerSocket, "ok");

}

else if (tempString == "your\_turn") {

sendZmqMessage(playerSocket, "ok");

std::cout << "Ваш ход:" << std::endl;

incomingMessage = receiveZmqMessage(playerSocket);

if (incomingMessage == "shoot") {

int x, y;

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

std::cin >> y >> x;

std::string shootMsg = "coords:" + std::to\_string(x) + ":" + std::to\_string(y);

sendZmqMessage(playerSocket, shootMsg);

incomingMessage = receiveZmqMessage(playerSocket);

if (incomingMessage == "shooted") {

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

sendZmqMessage(playerSocket, "ok");

} else if (incomingMessage == "miss") {

std::cout << "Промах!" << std::endl;

sendZmqMessage(playerSocket, "ok");

}

}

}

else if (tempString == "not\_your\_turn") {

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

sendZmqMessage(playerSocket, "ok");

incomingMessage = receiveZmqMessage(playerSocket);

if (incomingMessage == "shooted") {

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

sendZmqMessage(playerSocket, "ok");

} else if (incomingMessage == "miss") {

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

sendZmqMessage(playerSocket, "ok");

}

}

else if (tempString == "win") {

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

sendZmqMessage(playerSocket, "ok");

return 0;

}

else if (tempString == "lose") {

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

sendZmqMessage(playerSocket, "ok");

return 0;

}

}

}

**main\_server.cpp:**

#include <iostream>  
#include <unistd.h>  
#include <string>  
#include <vector>  
#include <sstream>  
#include <signal.h>  
#include <zmq.hpp>  
#include <chrono>  
#include <thread>  
#include <exception>  
#include <map>  
  
using namespace std::chrono\_literals;  
  
// Определим размер поля  
const int BOARD\_SIZE = 10;  
zmq::context\_t serverContext(3);  
zmq::socket\_t serverControlSocket(serverContext, ZMQ\_REP);  
  
bool sendZmqMessage(zmq::socket\_t &socket, const std::string &message) {  
 zmq::message\_t msg(message.size());  
 memcpy(msg.data(), message.c\_str(), message.size());  
 return socket.send(msg);  
}  
  
std::string receiveZmqMessage(zmq::socket\_t &socket) {  
 zmq::message\_t message;  
 bool ok = false;  
 try {  
 ok = socket.recv(&message);  
 } catch (...) {  
 ok = false;  
 }  
 std::string received(std::string(static\_cast<char\*>(message.data()), message.size()));  
 if (received.empty() || !ok) {  
 return "Ошибка получения сообщения!";  
 }  
 return received;  
}  
  
// Класс игрока  
class SeaBattlePlayer {  
 public:  
 std::vector<std::vector<char>> board;  
 int playerNumber; // Чтобы понимать, какой это игрок  
  
 SeaBattlePlayer() {  
 board.resize(BOARD\_SIZE, std::vector<char>(BOARD\_SIZE, ' '));  
 playerNumber = 0;  
 }  
  
 // Расставление кораблей  
 void placeShips(zmq::socket\_t &playerSocket) {  
 // Логика расстановки  
 int shipsCount = 5;  
 for (int shipSize = 1; shipSize <= 4; ++shipSize) {  
 shipsCount--;  
 for (int j = 0; j < shipsCount; j++) {  
 std::string msg = "Разместите корабль " + std::to\_string(shipSize)  
 + " (1x" + std::to\_string(shipSize) + "): ";  
 sendZmqMessage(playerSocket, msg);  
  
 std::string receivedMessage = receiveZmqMessage(playerSocket);  
 std::cout << "Получил запрос: " << receivedMessage << std::endl;  
  
 std::string token;  
 std::stringstream strs(receivedMessage);  
 std::getline(strs, token, ':'); // "coords"  
 if (token == "coords") {  
 int x, y;  
 char orientation;  
 std::getline(strs, token, ':'); // X  
 x = std::stoi(token);  
  
 std::getline(strs, token, ':'); // Y  
 y = std::stoi(token);  
  
 std::getline(strs, token, ':'); // orientation  
 orientation = token.empty() ? 'H' : token[0];  
  
 // Проверяем валидность  
 if (orientation != 'H' && orientation != 'V') {  
 sendZmqMessage(playerSocket, "Error : Неверно указана ориентация (H/V)");  
 (void)receiveZmqMessage(playerSocket);  
 j--;  
 continue;  
 }  
 if (isValidPlacement(x, y, shipSize, orientation)) {  
 placeShip(x, y, shipSize, orientation);  
 } else {  
 sendZmqMessage(playerSocket, "Error : Неверное местоположение! Попробуйте еще раз.");  
 (void)receiveZmqMessage(playerSocket);  
 j--;  
 continue;  
 }  
  
 // Отправляем текущее состояние доски  
 std::string boardState = "board" + getBoard();  
 sendZmqMessage(playerSocket, boardState);  
 (void)receiveZmqMessage(playerSocket);  
 }  
 }  
 }  
 }  
  
 // Проверяем, можем ли мы разместить корабль  
 bool isValidPlacement(int x, int y, int size, char orientation) const {  
 if (x < 0 || x >= BOARD\_SIZE || y < 0 || y >= BOARD\_SIZE) {  
 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;  
 }  
 }  
 }  
 // Проверяем окружение, чтобы корабль не касался других  
 for (int i = 0; i < size; i++) {  
 if (orientation == 'H') {  
 if (!isEmptyAround(x, y + i)) {  
 return false;  
 }  
 } else {  
 if (!isEmptyAround(x + i, y)) {  
 return false;  
 }  
 }  
 }  
 return true;  
 }  
  
 // Проверяем клетки вокруг  
 bool isEmptyAround(int row, int col) const {  
 for (int i = row - 1; i <= row + 1; ++i) {  
 for (int j = col - 1; j <= col + 1; ++j) {  
 if (i >= 0 && i < BOARD\_SIZE && j >= 0 && j < BOARD\_SIZE && board[i][j] != ' ') {  
 return false;  
 }  
 }  
 }  
 return true;  
 }  
  
 // Ставим корабль на поле  
 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 {  
 for (int j = y; j < y + size; ++j) {  
 board[x][j] = 'O';  
 }  
 }  
 }  
  
 // Получаем доску для отображения  
 std::string getBoard() const {  
 std::string result;  
 result = "\n 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) {  
 result += board[i][j];  
 result += ' ';  
 }  
 result += '\n';  
 }  
 result += '\n';  
 return result;  
 }  
  
 // Версия доски без отображения кораблей (например, чтобы отправлять противнику)  
 std::string getClearBoard() const {  
 std::string result;  
 result = "\n 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) {  
 if (board[i][j] == 'O') {  
 result += " ";  
 } else {  
 result += board[i][j];  
 result += ' ';  
 }  
 }  
 result += '\n';  
 }  
 result += '\n';  
 return result;  
 }  
};  
  
// Класс Game, реализующий логику морского боя  
class SeaBattleGame {  
 public:  
 SeaBattlePlayer player1;  
 SeaBattlePlayer player2;  
  
 void play(zmq::socket\_t &player1Socket, zmq::socket\_t &player2Socket) {  
 std::cout << "Игра \"Морской бой\" началась!" << std::endl;  
  
 // Запоминаем, что player1 — первый, player2 — второй  
 player1.playerNumber = 1;  
 player2.playerNumber = 2;  
  
 // Предлагаем игрокам расставить корабли  
 player1.placeShips(player1Socket);  
 player2.placeShips(player2Socket);  
  
 int turn = 0;  
 while (!gameOver()) {  
 if (turn % 2 == 0) {  
 // Ход первого игрока  
 sendZmqMessage(player1Socket, "your\_turn");  
 (void)receiveZmqMessage(player1Socket);  
  
 sendZmqMessage(player2Socket, "not\_your\_turn");  
 (void)receiveZmqMessage(player2Socket);  
  
 std::cout << "Ход игрока 1:" << std::endl;  
 bool shotResult = handlePlayerTurn(player1, player2, player1Socket, player2Socket);  
 if (shotResult && gameOver()) {  
 // Игрок 1 добил противника  
 std::cout << "Победил игрок 1" << std::endl;  
 sendZmqMessage(player1Socket, "win");  
 (void)receiveZmqMessage(player1Socket);  
 sendZmqMessage(player2Socket, "lose");  
 (void)receiveZmqMessage(player2Socket);  
 break;  
 }  
 if (!shotResult) {  
 turn++;  
 }  
 } else {  
 // Ход второго игрока  
 sendZmqMessage(player2Socket, "your\_turn");  
 (void)receiveZmqMessage(player2Socket);  
  
 sendZmqMessage(player1Socket, "not\_your\_turn");  
 (void)receiveZmqMessage(player1Socket);  
  
 std::cout << "Ход игрока 2:" << std::endl;  
 bool shotResult = handlePlayerTurn(player2, player1, player2Socket, player1Socket);  
 if (shotResult && gameOver()) {  
 // Игрок 2 добил противника  
 std::cout << "Победил игрок 2" << std::endl;  
 sendZmqMessage(player2Socket, "win");  
 (void)receiveZmqMessage(player2Socket);  
 sendZmqMessage(player1Socket, "lose");  
 (void)receiveZmqMessage(player1Socket);  
 break;  
 }  
 if (!shotResult) {  
 turn++;  
 }  
 }  
 }  
  
 std::cout << "Игра завершена!" << std::endl;  
 }  
  
 // Проверяем, закончилась ли игра (у кого-то больше нет кораблей)  
 bool gameOver() const {  
 return allShipsSunk(player1) || allShipsSunk(player2);  
 }  
  
 bool allShipsSunk(const SeaBattlePlayer &player) const {  
 for (auto &row : player.board) {  
 for (char cell : row) {  
 if (cell == 'O') {  
 return false;  
 }  
 }  
 }  
 return true;  
 }  
  
 // Ход игрока: возвращаем true, если было попадание  
 bool handlePlayerTurn(SeaBattlePlayer &attacker, SeaBattlePlayer &defender,  
 zmq::socket\_t &attackerSocket, zmq::socket\_t &defenderSocket)  
 {  
 // Запрашиваем координаты выстрела  
 sendZmqMessage(attackerSocket, "shoot");  
 std::string incomingShot = receiveZmqMessage(attackerSocket);  
  
 std::stringstream ss(incomingShot);  
 std::string token;  
 std::getline(ss, token, ':'); // "coords"  
  
 // Извлекаем x, y  
 int x, y;  
 std::getline(ss, token, ':');  
 x = std::stoi(token);  
  
 std::getline(ss, token, ':');  
 y = std::stoi(token);  
  
 // Проверяем  
 if (isValidShot(x, y, defender)) {  
 if (defender.board[x][y] == 'O') {  
 // Попадание  
 sendZmqMessage(attackerSocket, "shooted");  
 (void)receiveZmqMessage(attackerSocket);  
  
 sendZmqMessage(defenderSocket, "shooted");  
 (void)receiveZmqMessage(defenderSocket);  
  
 std::cout << "Попадание!" << std::endl;  
 defender.board[x][y] = 'X'; // отмечаем подбитую палубу  
 return true;  
 } else {  
 // Промах  
 sendZmqMessage(attackerSocket, "miss");  
 (void)receiveZmqMessage(attackerSocket);  
  
 sendZmqMessage(defenderSocket, "miss");  
 (void)receiveZmqMessage(defenderSocket);  
  
 std::cout << "Промах!" << std::endl;  
 defender.board[x][y] = '\*';  
  
 // Отправим обоим актуальные доски  
 std::string defenderBoard = defender.getBoard();  
 std::string defenderClearBoard = defender.getClearBoard();  
  
 sendZmqMessage(attackerSocket, "board" + defenderClearBoard);  
 (void)receiveZmqMessage(attackerSocket);  
  
 sendZmqMessage(defenderSocket, "board" + defenderBoard);  
 (void)receiveZmqMessage(defenderSocket);  
  
 return false;  
 }  
 } else {  
 std::cout << "Неверные координаты! Повторяем ход." << std::endl;  
 // Рекурсия для повторного ввода  
 return handlePlayerTurn(attacker, defender, attackerSocket, defenderSocket);  
 }  
 }  
  
 bool isValidShot(int x, int y, const SeaBattlePlayer &defender) const {  
 return x >= 0 && x < BOARD\_SIZE &&  
 y >= 0 && y < BOARD\_SIZE &&  
 (defender.board[x][y] == ' ' || defender.board[x][y] == 'O');  
 }  
};  
  
int main() {  
 // Создаём сокеты для двух игроков  
 zmq::socket\_t firstPlayerSocket(serverContext, ZMQ\_REQ);  
 zmq::socket\_t secondPlayerSocket(serverContext, ZMQ\_REQ);  
  
 // Привязываемся  
 serverControlSocket.bind("tcp://\*:5555");  
 firstPlayerSocket.bind("tcp://\*:5556");  
 secondPlayerSocket.bind("tcp://\*:5557");  
  
 std::cout << "Сервер запущен и ожидает подключения..." << std::endl;  
  
 std::map<int, std::string> playerLoginMap;  
 int currentPlayerId = 1;  
  
 while (true) {  
 std::string incomingMessage = receiveZmqMessage(serverControlSocket);  
 std::cout << "На сервер поступил запрос: '" << incomingMessage << "'" << std::endl;  
  
 std::stringstream ss(incomingMessage);  
 std::string command;  
 std::getline(ss, command, ':'); // Берём первое слово ("login"/"invite"/...)  
  
 if (command == "login") {  
 if (currentPlayerId > 2) {  
 sendZmqMessage(serverControlSocket, "Error:TwoPlayersAlreadyExist");  
 } else {  
 // Пропускаем PID  
 std::string pidString;  
 std::getline(ss, pidString, ':');  
  
 // Считываем логин  
 std::string playerLogin;  
 std::getline(ss, playerLogin, ':');  
  
 // Проверим, занят ли логин  
 // Допустим, хотим, чтобы login\_map[1] и login\_map[2] были разными логинами  
 if (playerLoginMap[1] == playerLogin || playerLoginMap[2] == playerLogin) {  
 std::cout << "Игрок ввёл занятый логин: " << playerLogin << std::endl;  
 sendZmqMessage(serverControlSocket, "Error:NameAlreadyExist");  
 } else {  
 playerLoginMap[currentPlayerId] = playerLogin;  
 std::cout << "Логин игрока номер " << currentPlayerId  
 << ": " << playerLogin << std::endl;  
 sendZmqMessage(serverControlSocket, "Ok:" + std::to\_string(currentPlayerId));  
 currentPlayerId++;  
 }  
 }  
 }  
 else if (command == "invite") {  
 std::cout << "Обрабатываю приглашение" << std::endl;  
 std::this\_thread::sleep\_for(100ms);  
  
 std::string senderIdString;  
 std::getline(ss, senderIdString, ':');  
 int senderId = std::stoi(senderIdString);  
  
 std::string inviteLogin;  
 std::getline(ss, inviteLogin, ':');  
  
 // Проверяем на самоприглашение  
 if (inviteLogin == playerLoginMap[senderId]) {  
 std::cout << "Игрок пригласил сам себя" << std::endl;  
 sendZmqMessage(serverControlSocket, "Error:SelfInvite");  
 }  
 // Если отправил игрок 1 -> смотрим, есть ли такой логин  
 else if (inviteLogin == playerLoginMap[2]) {  
 // Приглашаем через secondPlayerSocket  
 std::cout << "Игрок " << playerLoginMap[1]  
 << " пригласил " << playerLoginMap[2] << std::endl;  
 sendZmqMessage(secondPlayerSocket, "invite:" + playerLoginMap[1]);  
  
 std::string inviteAnswer = receiveZmqMessage(secondPlayerSocket);  
 secondPlayerSocket.set(zmq::sockopt::rcvtimeo, -1);  
  
 if (inviteAnswer == "accept") {  
 std::cout << "Игрок " << playerLoginMap[2] << " принял запрос" << std::endl;  
 sendZmqMessage(serverControlSocket, inviteAnswer);  
 break;  
 }  
 else if (inviteAnswer == "reject") {  
 std::cout << "Игрок " << playerLoginMap[2] << " отклонил запрос" << std::endl;  
 sendZmqMessage(serverControlSocket, inviteAnswer);  
 }  
 else {  
 std::cout << "Ошибка при обработке приглашения" << std::endl;  
 }  
 }  
 else if (inviteLogin == playerLoginMap[1]) {  
 // Приглашаем через firstPlayerSocket  
 std::cout << "Игрок " << playerLoginMap[2]  
 << " пригласил " << playerLoginMap[1] << std::endl;  
 sendZmqMessage(firstPlayerSocket, "invite:" + playerLoginMap[2]);  
  
 std::string inviteAnswer = receiveZmqMessage(firstPlayerSocket);  
 if (inviteAnswer == "accept") {  
 std::cout << "Игрок " << playerLoginMap[1] << " принял запрос" << std::endl;  
 sendZmqMessage(serverControlSocket, inviteAnswer);  
 break;  
 }  
 else if (inviteAnswer == "reject") {  
 std::cout << "Игрок " << playerLoginMap[1] << " отклонил запрос" << std::endl;  
 sendZmqMessage(serverControlSocket, inviteAnswer);  
 }  
 else {  
 std::cout << "Ошибка при обработке приглашения." << std::endl;  
 }  
 } else {  
 // Такого логина не существует  
 std::cout << "Логин " << inviteLogin << " не найден в базе" << std::endl;  
 sendZmqMessage(serverControlSocket, "Error:LoginNotExist");  
 }  
 }  
 }  
  
 std::cout << "Запрашиваем готовность у игроков..." << std::endl;  
  
 // Отправляем обоим игрокам "ping"  
 sendZmqMessage(firstPlayerSocket, "ping");  
 sendZmqMessage(secondPlayerSocket, "ping");  
  
 std::string player1Answer = receiveZmqMessage(firstPlayerSocket);  
 std::string player2Answer = receiveZmqMessage(secondPlayerSocket);  
  
 if (player1Answer == "pong") {  
 std::cout << "Игрок " << playerLoginMap[1] << " готов!" << std::endl;  
 } else {  
 std::cout << "Игрок " << playerLoginMap[1] << " отказался от игры." << std::endl;  
 return 0;  
 }  
  
 if (player2Answer == "pong") {  
 std::cout << "Игрок " << playerLoginMap[2] << " готов!" << std::endl;  
 } else {  
 std::cout << "Игрок " << playerLoginMap[2] << " отказался от игры." << std::endl;  
 return 0;  
 }  
  
 std::cout << "Начинаем игру!" << std::endl;  
 SeaBattleGame game;  
 game.play(firstPlayerSocket, secondPlayerSocket);  
  
 return 0;  
}

# РЕЗУЛЬТАТЫ

## Демонстрация работы программы

Для уменьшения логов было ограничено количество кораблей. Используются только однопалубные.

**./server\_cp**

Сервер запущен и ожидает подключения...

На сервер поступил запрос: 'login:2090:sonya'

Логин игрока номер 1: sonya

На сервер поступил запрос: 'login:2104:nikita'

Логин игрока номер 2: nikita

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

Обрабатываю инвайт

Игрок nikita пригласил sonya

Игрок sonya принял запрос

Запрашиваем готовность у игроков...

Игрок sonya готов!

Игрок nikita готов!

Начинаем игру!

Игра "Морской бой" началась!

Получил запрос: coords:0:0:H

Получил запрос: coords:2:2:H

Получил запрос: coords:5:5:H

Получил запрос: coords:7:7:H

Получил запрос: coords:0:0:H

Получил запрос: coords:2:2:H

Получил запрос: coords:5:5:H

Получил запрос: coords:7:7:H

Ход игрока 1:

Попадание!

Ход игрока 1:

Попадание!

Ход игрока 1:

Промах!

Ход игрока 2:

Попадание!

Ход игрока 2:

Попадание!

Ход игрока 2:

Промах!

Ход игрока 1:

Попадание!

Ход игрока 1:

Попадание!

Победил игрок 1

Игра завершена!

**./client1**

Введите ваш логин: sonya

Вы успешно авторизовались!

Вы хотите пригласить друга? (y/n)

n

Ждем приглашения от друга...

Игрок с ником nikita 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):

0 0 H

0 1 2 3 4 5 6 7 8 9

0 O

1

2

3

4

5

6

7

8

9

Разместите корабль 1 (1x1):

2 2 H

0 1 2 3 4 5 6 7 8 9

0 O

1

2 O

3

4

5

6

7

8

9

Разместите корабль 1 (1x1):

5 5 H

0 1 2 3 4 5 6 7 8 9

0 O

1

2 O

3

4

5 O

6

7

8

9

Разместите корабль 1 (1x1):

7 7 H

0 1 2 3 4 5 6 7 8 9

0 O

1

2 O

3

4

5 O

6

7 O

8

9

Ваш ход:

Введите координаты выстрела (формат: x y):

0 0

Попадание!

Ваш ход:

Введите координаты выстрела (формат: x y):

2 2

Попадание!

Ваш ход:

Введите координаты выстрела (формат: x y):

4 4

Промах!

0 1 2 3 4 5 6 7 8 9

0 X

1

2 X

3

4 \*

5

6

7

8

9

Ход соперника:

Вас подстрелили!

Ход соперника:

Вас подстрелили!

Ход соперника:

Противник промахнулся

0 1 2 3 4 5 6 7 8 9

0 X

1

2 X

3

4 \*

5 O

6

7 O

8

9

Ваш ход:

Введите координаты выстрела (формат: x y):

5 5

Попадание!

Ваш ход:

Введите координаты выстрела (формат: x y):

7 7

Попадание!

Вы выиграли!

**./client2**

Введите ваш логин: nikita

Вы успешно авторизовались!

Вы хотите пригласить друга? (y/n)

y

Чтобы пригласить друга, напишите: invite (friend\_login)

invite sonya

Вы пригласили игрока с ником sonya

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):

0 0 H

0 1 2 3 4 5 6 7 8 9

0 O

1

2

3

4

5

6

7

8

9

Разместите корабль 1 (1x1):

2 2 H

0 1 2 3 4 5 6 7 8 9

0 O

1

2 O

3

4

5

6

7

8

9

Разместите корабль 1 (1x1):

5 5 H

0 1 2 3 4 5 6 7 8 9

0 O

1

2 O

3

4

5 O

6

7

8

9

Разместите корабль 1 (1x1):

7 7 H

0 1 2 3 4 5 6 7 8 9

0 O

1

2 O

3

4

5 O

6

7 O

8

9

Ход соперника:

Вас подстрелили!

Ход соперника:

Вас подстрелили!

Ход соперника:

Противник промахнулся

0 1 2 3 4 5 6 7 8 9

0 X

1

2 X

3

4 \*

5 O

6

7 O

8

9

Ваш ход:

Введите координаты выстрела (формат: x y):

0 0

Попадание!

Ваш ход:

Введите координаты выстрела (формат: x y):

2 2

Попадание!

Ваш ход:

Введите координаты выстрела (формат: x y):

4 4

Промах!

0 1 2 3 4 5 6 7 8 9

0 X

1

2 X

3

4 \*

5

6

7

8

9

Ход соперника:

Вас подстрелили!

Ход соперника:

Вас подстрелили!

Вы проиграли!

# ЗАКЛЮЧЕНИЕ

Работа над курсовым проектом "Морской бой" позволила успешно освоить принципы разработки клиент-серверных приложений с использованием ZeroMQ. Реализация проекта предоставила возможность применить знания о синхронизации данных, многопоточности и обработке сообщений. Полученные результаты демонстрируют работоспособность предложенной архитектуры и алгоритмов, а также их соответствие требованиям задания.

Проект не только подчеркнул важность планирования и структурированного подхода к разработке, но и позволил углубить понимание сетевых технологий и механизмов взаимодействия в многопользовательских системах. В дальнейшем проект может быть расширен за счёт добавления новых функциональных возможностей, улучшения интерфейса и оптимизации производительности. Работа над данным проектом внесла значительный вклад в развитие навыков программирования на с++ и анализа архитектурных решений.