Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

**Курсовая работа по курсу**

**«Операционные системы»**

Группа: М8О-215Б-23

Студент: Павлов О.В.

Преподаватель: Миронов Е.С.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 14.03.25

Москва, 2025

**Постановка задачи**

**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

**Общий метод и алгоритм решения**

Разработанная система представляет собой клиент-серверное приложение для игры "Морской бой", реализованное на C++ с использованием библиотеки ZeroMQ для сетевого взаимодействия. Алгоритм работы можно разделить на несколько ключевых этапов:

**1. Архитектурная модель**

Система строится по схеме централизованного сервера с двумя клиентами:

- **Сервер**:

- Управляет игровой логикой

- Координирует взаимодействие игроков

- Проверяет валидность действий

- Хранит состояние игровых полей

- **Клиенты**:

- Обеспечивают интерфейс для игроков

- Отправляют запросы на сервер

- Отображают игровое состояние

**Транспортный протокол**: ZeroMQ (REQ-REP паттерн) с использованием портов 5555-5557.

**2. Алгоритм работы сервера**

**Этап 1. Инициализация**

1. Создание контекста ZeroMQ

2. Привязка трех сокетов:

- `5555` – управляющий сокет (авторизация)

- `5556` – сокет игрока 1

- `5557` – сокет игрока 2

**Этап 2. Обработка подключений**

**Этап 3. Подготовка к игре**

1. Ожидание подтверждения готовности (`ping-pong`)

2. Последовательная расстановка кораблей:

- Для каждого корабля (1-4 палубные):

- Получение координат от игрока

**Этап 4. Игровой процесс**

**3. Алгоритм работы клиента**

**Этап 1. Подключение**

1. Установка соединения с сервером (`tcp://localhost:5555`)

2. Авторизация:

**Этап 2. Лобби**

**- Вариант 1: Инициатор игры**

**- Вариант 2: Ожидание приглашения**

**Этап 3. Расстановка кораблей**

**Этап 4. Боевые действия**

**4. Ключевые алгоритмические решения**

**1. Валидация расстановки кораблей:**

- Проверка границ поля

- Контроль интервалов между кораблями

- Учет ориентации (горизонтальная/вертикальная)

**2. Механизм очереди ходов:**

- Циклический переключатель игроков

- Блокировка интерфейса при ходе соперника

**3. Сетевой протокол:**

- Формат сообщений: `команда:параметр1:параметр2`

- Примеры:

- login:12345:Player1

- invite:2:Player2

- coords:3:5:H

**4. Обработка ошибок:**

- Повторный запрос при невалидных данных

- Таймауты соединения

- Защита от самоприглашения

**5. Схема данных**

Серверные структуры:

cpp

struct Player {

vector<vector<char>> board; // 'O'-корабль, 'X'-попадание, '\*'-промах

string login;

int id;

bool ready;

}

struct GameState {

Player player1;

Player player2;

int current\_turn;

bool game\_active;

}

**6. Оптимизации и особенности**

**1. Асинхронная обработка приглашений:**

- Использование POSIX Threads для параллельного ожидания

- Мьютексы для синхронизации доступа к ресурсам

**2. Экономия трафика:**

- Дифференцированная отправка данных:

- Полное поле – владельцу

- Маскированное поле – сопернику

**3. Механизм keep-alive:**

- Регулярные ping-запросы

- Автоматическое восстановление соединения

**7. Последовательность выполнения**

1. Сервер запускается и переходит в режим ожидания

2. Клиент 1 подключается, авторизуется как "Player1"

3. Клиент 2 подключается, авторизуется как "Player2"

4. Инициация игровой сессии через систему приглашений

5. Поочередная расстановка кораблей с валидацией

6. Циклическое выполнение ходов до победы

7. Завершение игры с выводом результатов

Данная архитектура обеспечивает четкое разделение ответственности, минимальную задержку взаимодействия и устойчивость к сетевым ошибкам.

**Код программы**

**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); // Сокет для управления (REP-сокет)

/\*

\* Функция отправки сообщения через ZeroMQ сокет

\* socket - целевой сокет

\* message - строка для отправки

\* Возвращает: true если отправка успешна

\*/

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);

}

/\*

\* Функция приема сообщения через ZeroMQ сокет

\* socket - исходный сокет

\* Возвращает: принятую строку или сообщение об ошибке

\*/

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:

\* Содержит логику игрового поля и взаимодействия с игроком

\*/

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;

}

/\*

\* Метод расстановки кораблей:

\* playerSocket - сокет для общения с клиентом

\* Логика:

\* 1. Для кораблей размером от 1 до 4

\* 2. Запрашивает координаты у игрока

\* 3. Проверяет валидность размещения

\* 4. Отправляет обновленное поле

\*/

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++)

{

// запрос

std::string msg = "Разместите корабль " + std::to\_string(shipSize)

+ " (1x" + std::to\_string(shipSize) + "): ";

sendZmqMessage(playerSocket, msg);

// ответ

std::string receivedMessage = receiveZmqMessage(playerSocket);

std::cout << "Получил запрос: " << receivedMessage << std::endl;

// Парсинг координат

std::string token;

std::stringstream strs(receivedMessage);

std::getline(strs, token, ':'); // "coords"

if (token == "coords")

{

// Извлечение координат из сообщения

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;

/\*

\* Основной игровой цикл:

\* 1. Поочередно дает ходы игрокам

\* 2. Проверяет попадания

\* 3. Определяет победителя

\*/

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);

// Проверка завершения игры

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"); // Основной порт управления

firstPlayerSocket.bind("tcp://\*:5556"); // Порт игрока 1

secondPlayerSocket.bind("tcp://\*:5557"); // Порт игрока 2

std::cout << "Сервер запущен и ожидает подключения..." << std::endl;

std::map<int, std::string> playerLoginMap;

int currentPlayerId = 1;

while (true)

{

// Прием сообщений

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"/...)

// Логика авторизации

// Проверка уникальности логина

// Назначение ID игрока

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++;

}

}

}

// Логика приглашения

// Проверка существования игрока

// Отправка приглашения

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;

}

**main\_client.cpp**

// Реализует взаимодействие с сервером и игровую логику для пользователя

#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; // Мьютекс для синхронизации доступа к командам

zmq::context\_t zmqContext(2); // Контекст ZeroMQ с 2 потоками

zmq::socket\_t playerSocket(zmqContext, ZMQ\_REP); // Сокет для обмена сообщениями с сервером (REP-сокет)

int playerId;

std::string userCommand;

/\*

\* Функция отправки сообщения через ZeroMQ сокет

\* socket - целевой сокет

\* msg - сообщение для отправки

\* Возвращает: true если сообщение успешно отправлено

\*/

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);

}

/\*

\* Функция приема сообщения через ZeroMQ сокет

\* socket - сокет для приема сообщений

\* Возвращает: принятое сообщение или сообщение об ошибке

\*/

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;

}

/\*

\* Формирование строки подключения к сокету

\* port - номер порта

\* Возвращает: строку вида "tcp://127.0.0.1:port"

\*/

std::string getPortName(int port)

{

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

}

typedef struct {

} CheckInviteParams;

/\*

\* Функция для проверки входящих приглашений в отдельном потоке

\* param - указатель на параметры (не используется)

\*/

void\* checkInvite(void \*param)

{

std::string inviteTempString;

pthread\_mutex\_lock(&commandMutex); // Блокируем мьютекс

// Получаем и парсим сообщение

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); // Разблокируем мьютекс

sendZmqMessage(playerSocket, "reject");

}

}

pthread\_exit(nullptr);

}

/\*

\* Главная функция клиента

\* argc - количество аргументов командной строки

\* argv - массив аргументов командной строки

\*/

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

{

// Сокет для связи с сервером

zmq::context\_t contextLocal(2); // Локальный контекст ZeroMQ

zmq::socket\_t serverSocket(contextLocal, ZMQ\_REQ); // Сокет для связи с сервером

// Подключаемся к основному серверу, слушающему порт 5555

serverSocket.connect(getPortName(5555));

// Инициализируем мьютекс

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

// Параметры для потока

CheckInviteParams checkInviteParams; // Параметры для потока (пустые)

pthread\_t inviteThread; // Идентификатор потока

int processId = getpid(); // PID процесса

std::string serverResponse; // Ответ сервера

std::string tempString; // Временная строка

int iteration = 1; // Счетчик итераций

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);

// Получаем ответ сервера

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); // Получаем назначенный ID

// 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;

}

}

}

**Протокол работы программы**

**Тестирование:**

$ mkdir build

$ cd build

$ cmake ..

$ make

$ ./server\_cp

$ ./client\_cp

$ ./client\_cp

server\_cp

Сервер запущен и ожидает подключения...

На сервер поступил запрос: 'login:98551:oleg'

Логин игрока номер 1: oleg

На сервер поступил запрос: 'login:98564:egor'

Логин игрока номер 2: egor

На сервер поступил запрос: 'invite:1:egor'

Обрабатываю приглашение

Игрок oleg пригласил egor

Игрок egor принял запрос

Запрашиваем готовность у игроков...

Игрок oleg готов!

Игрок egor готов!

Начинаем игру!

Игра "Морской бой" началась!

Получил запрос: coords:0:0:H

Получил запрос: coords:2:2:H

Получил запрос: coords:4:4:H

Получил запрос: coords:6:6:H

Получил запрос: coords:9:0:H

Получил запрос: coords:1:8:V

Получил запрос: coords:5:0:H

Получил запрос: coords:0:4:V

Получил запрос: coords:0:1:H

Получил запрос: coords:7:0:H

Получил запрос: coords:8:0:H

Получил запрос: coords:8:5:H

Получил запрос: coords:5:8:V

Получил запрос: coords:0:0:H

Получил запрос: coords:1:1:H

Получил запрос: coords:2:2:H

Получил запрос: coords:4:4:H

Получил запрос: coords:6:6:H

Получил запрос: coords:2:0:V

Получил запрос: coords:0:9:V

Получил запрос: coords:9:0:H

Получил запрос: coords:8:0:H

Получил запрос: coords:6:0:H

Получил запрос: coords:2:6:V

Получил запрос: coords:5:0:H

Получил запрос: coords:8:2:H

Получил запрос: coords:8:3:H

Ход игрока 1:

Попадание!

Ход игрока 1:

Попадание!

Ход игрока 1:

Попадание!

Ход игрока 1:

Попадание!

Ход игрока 1:

Попадание!

Ход игрока 1:

Попадание!

Ход игрока 1:

Промах!

Ход игрока 2:

Промах!

Ход игрока 1:

Попадание!

Ход игрока 1:

Попадание!

Ход игрока 1:

Попадание!

Ход игрока 1:

Попадание!

Ход игрока 1:

Промах!

Ход игрока 2:

Промах!

Ход игрока 1:

Попадание!

Ход игрока 1:

Попадание!

Ход игрока 1:

Попадание!

Ход игрока 1:

Попадание!

Ход игрока 1:

Попадание!

Ход игрока 1:

Попадание!

Ход игрока 1:

Попадание!

Ход игрока 1:

Попадание!

Ход игрока 1:

Попадание!

Ход игрока 1:

Попадание!

Победил игрок 1

Игра завершена!

./client\_cp

Введите ваш логин: oleg

Вы успешно авторизовались!

Вы хотите пригласить друга? (y/n)

y

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

invite egor

Вы пригласили игрока с ником egor

Ждем ответ...

Запрос принят!

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). Подождите приглашения к размещению.

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):

4 4 H

0 1 2 3 4 5 6 7 8 9

0 O

1

2 O

3

4 O

5

6

7

8

9

Разместите корабль 1 (1x1):

6 6 H

0 1 2 3 4 5 6 7 8 9

0 O

1

2 O

3

4 O

5

6 O

7

8

9

Разместите корабль 2 (1x2):

0 9 H

0 1 2 3 4 5 6 7 8 9

0 O

1

2 O

3

4 O

5

6 O

7

8

9 O O

Разместите корабль 2 (1x2):

8 1 V

0 1 2 3 4 5 6 7 8 9

0 O

1 O

2 O O

3

4 O

5

6 O

7

8

9 O O

Разместите корабль 2 (1x2):

0 5 H

0 1 2 3 4 5 6 7 8 9

0 O

1 O

2 O O

3

4 O

5 O O

6 O

7

8

9 O O

Разместите корабль 3 (1x3):

4 0 V

0 1 2 3 4 5 6 7 8 9

0 O O

1 O O

2 O O O

3

4 O

5 O O

6 O

7

8

9 O O

Разместите корабль 3 (1x3):

1 0 H

Error : Неверное местоположение! Попробуйте еще раз.

Разместите корабль 3 (1x3):

0 7 H

Error : Неверное местоположение! Попробуйте еще раз.

Разместите корабль 3 (1x3):

0 8 H

Error : Неверное местоположение! Попробуйте еще раз.

Разместите корабль 3 (1x3):

5 8 H

0 1 2 3 4 5 6 7 8 9

0 O O

1 O O

2 O O O

3

4 O

5 O O

6 O

7

8 O O O

9 O O

Разместите корабль 4 (1x4):

8 5 V

0 1 2 3 4 5 6 7 8 9

0 O O

1 O O

2 O O O

3

4 O

5 O O O

6 O O

7 O

8 O O O O

9 O O

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

Попадание!

Ваш ход:

Введите координаты выстрела (формат: x y):

6 6

Попадание!

Ваш ход:

Введите координаты выстрела (формат: x y):

0 9

Попадание!

Ваш ход:

Введите координаты выстрела (формат: x y):

1 9

Попадание!

Ваш ход:

Введите координаты выстрела (формат: x y):

2 0

Промах!

0 1 2 3 4 5 6 7 8 9

0 X \*

1

2 X

3

4 X

5

6 X

7

8

9 X X

Ход соперника:

Противник промахнулся

0 1 2 3 4 5 6 7 8 9

0 O O

1 O O

2 O O O

3

4 O

5 O O \* O

6 O O

7 O

8 O O O O

9 O O

Ваш ход:

Введите координаты выстрела (формат: x y):

0 2

Попадание!

Ваш ход:

Введите координаты выстрела (формат: x y):

0 3

Попадание!

Ваш ход:

Введите координаты выстрела (формат: x y):

0 5

Попадание!

Ваш ход:

Введите координаты выстрела (формат: x y):

1 5

Попадание!

Ваш ход:

Введите координаты выстрела (формат: x y):

1 6

Промах!

0 1 2 3 4 5 6 7 8 9

0 X \*

1

2 X X

3 X

4 X

5 X X

6 \* X

7

8

9 X X

Ход соперника:

Противник промахнулся

0 1 2 3 4 5 6 7 8 9

0 O O

1 O O

2 O O O

3

4 O

5 O O \* O

6 O O

7 O

8 O O O O

9 O O \*

Ваш ход:

Введите координаты выстрела (формат: x y):

2 5

Попадание!

Ваш ход:

Введите координаты выстрела (формат: x y):

9 0

Попадание!

Ваш ход:

Введите координаты выстрела (формат: x y):

9 1

Попадание!

Ваш ход:

Введите координаты выстрела (формат: x y):

6 2

Попадание!

Ваш ход:

Введите координаты выстрела (формат: x y):

6 3

Попадание!

Ваш ход:

Введите координаты выстрела (формат: x y):

6 4

Попадание!

Ваш ход:

Введите координаты выстрела (формат: x y):

3 8

Попадание!

Ваш ход:

Введите координаты выстрела (формат: x y):

4 8

Попадание!

Ваш ход:

Введите координаты выстрела (формат: x y):

5 8

Попадание!

Ваш ход:

Введите координаты выстрела (формат: x y):

6 8

Попадание!

Вы выиграли!

./client\_cp

Введите ваш логин: egor

Вы успешно авторизовались!

Вы хотите пригласить друга? (y/n)

n

Ждем приглашения от друга...

Игрок с ником oleg 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. Подождите, пока другой пользователь расставит корабли

Чтобы расставить ваши корабли (формат: x, y и ориентация (H или V) через пробелы). Подождите приглашения к размещению.

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):

1 1 H

Error : Неверное местоположение! Попробуйте еще раз.

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):

4 4 H

0 1 2 3 4 5 6 7 8 9

0 O

1

2 O

3

4 O

5

6

7

8

9

Разместите корабль 1 (1x1):

6 6 H

0 1 2 3 4 5 6 7 8 9

0 O

1

2 O

3

4 O

5

6 O

7

8

9

Разместите корабль 2 (1x2):

0 2 V

0 1 2 3 4 5 6 7 8 9

0 O

1

2 O O

3 O

4 O

5

6 O

7

8

9

Разместите корабль 2 (1x2):

9 0 V

0 1 2 3 4 5 6 7 8 9

0 O O

1 O

2 O O

3 O

4 O

5

6 O

7

8

9

Разместите корабль 2 (1x2):

0 9 H

0 1 2 3 4 5 6 7 8 9

0 O O

1 O

2 O O

3 O

4 O

5

6 O

7

8

9 O O

Разместите корабль 3 (1x3):

0 8 H

Error : Неверное местоположение! Попробуйте еще раз.

Разместите корабль 3 (1x3):

0 6 H

Error : Неверное местоположение! Попробуйте еще раз.

Разместите корабль 3 (1x3):

6 2 V

0 1 2 3 4 5 6 7 8 9

0 O O

1 O

2 O O O

3 O O

4 O O

5

6 O

7

8

9 O O

Разместите корабль 3 (1x3):

0 5 H

0 1 2 3 4 5 6 7 8 9

0 O O

1 O

2 O O O

3 O O

4 O O

5 O O O

6 O

7

8

9 O O

Разместите корабль 4 (1x4):

2 8 H

Error : Неверное местоположение! Попробуйте еще раз.

Разместите корабль 4 (1x4):

3 8 H

0 1 2 3 4 5 6 7 8 9

0 O O

1 O

2 O O O

3 O O

4 O O

5 O O O

6 O

7

8 O O O O

9 O O

Ход соперника:

Вас подстрелили!

Ход соперника:

Вас подстрелили!

Ход соперника:

Вас подстрелили!

Ход соперника:

Вас подстрелили!

Ход соперника:

Вас подстрелили!

Ход соперника:

Вас подстрелили!

Ход соперника:

Противник промахнулся

0 1 2 3 4 5 6 7 8 9

0 X \* O

1 O

2 O X O

3 O O

4 X O

5 O O O

6 X

7

8 O O O O

9 X X

Ваш ход:

Введите координаты выстрела (формат: x y):

5 5

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:

Вас подстрелили!

Ход соперника:

Вас подстрелили!

Ход соперника:

Вас подстрелили!

Ход соперника:

Противник промахнулся

0 1 2 3 4 5 6 7 8 9

0 X \* O

1 O

2 X X O

3 X O

4 X O

5 X X O

6 \* X

7

8 O O O O

9 X X

Ваш ход:

Введите координаты выстрела (формат: x y):

9 9

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:

Вас подстрелили!

Ход соперника:

Вас подстрелили!

Ход соперника:

Вас подстрелили!

Ход соперника:

Вас подстрелили!

Ход соперника:

Вас подстрелили!

Ход соперника:

Вас подстрелили!

Ход соперника:

Вас подстрелили!

Ход соперника:

Вас подстрелили!

Ход соперника:

Вас подстрелили!

Вы проиграли!

**Вывод**

ZeroMQ обеспечил эффективное и надежное сетевое взаимодействие между клиентом и сервером за счет использования паттерна REQ-REP, который идеально подошел для синхронного обмена сообщениями в рамках пошаговой игры. Библиотека упростила реализацию многопоточности и обработку конкурентных запросов, обеспечив стабильность соединения даже при высокой нагрузке. Гибкость ZeroMQ позволила легко масштабировать логику взаимодействия, минимизировать задержки и организовать четкий протокол обмена данными, что критично для реализации игровой механики в реальном времени.