Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Курсовая работа по курсу
«Операционные системы»
Проектирование консольной клиент-серверной игры

Студент: Велиев Рауф Рамиз оглы
Группа: М8О-209Б-23
Вариант: 4
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2024

Содержание

Репозиторий

https://github.com/velievrauf/OS/tree/main/cp

Постановка задачи

Цель работы

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

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.

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.

Исходный код

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;
}
```

Демонстрация работы программы

rauf@rauf-VirtualBox:~/os_course_project_ffinal$ g++ -o main_server main_server.cpp -lzmq

rauf@rauf-VirtualBox:~/os_course_project_ffinal$ g++ -o main_client_1 main_client.cpp -lzmq

rauf@rauf-VirtualBox:~/os_course_project_ffinal$ g++ -o main_client_2 main_client.cpp -lzmq

rauf@rauf-VirtualBox:~/os_course_project_ffinal$ ./main_server

```
Сервер запущен и ожидает подключения...
На сервер поступил запрос: 'login:4562:rauf'
Логин игрока номер 1: rauf
На сервер поступил запрос: 'login:4579:archi'
Логин игрока номер 2: archi
На сервер поступил запрос: 'invite:1:archi'
Обрабатываю приглашение
Игрок rauf пригласил archi
Игрок archi принял запрос
Запрашиваем готовность у игроков...
Игрок rauf готов!
Игрок archi 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:1:4:V
Получил запрос: coords:4:3:V
Получил запрос: coords:4:7:V
Получил запрос: coords:9:0:H
Получил запрос: coords:9:6:H
Получил запрос: coords:4:9:V
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:1:4:V
Получил запрос: coords:4:3:V
Получил запрос: coords:4:7:V
Получил запрос: coords:9:0:H
Получил запрос: coords:9:6:H
Получил запрос: coords:4:9:V
Ход игрока 1:
Попадание!
Ход игрока 1:
Промах!
Ход игрока 2:
Попадание!
Ход игрока 2:
Попадание!
Ход игрока 2:
Попадание!
Ход игрока 2:
Попадание!
Ход игрока 2:
Попадание!
Ход игрока 2:
Попадание!
Ход игрока 2:
Попадание!
Ход игрока 2:
```

```
Попадание!
Ход игрока 2:
Попадание!
Ход игрока 2:
Попадание!
Ход игрока 2:
Попадание!
Ход игрока 2:
Попадание!
Ход игрока 2:
Попадание!
Ход игрока 2:
Попадание!
Ход игрока 2:
Попадание!
Ход игрока 2:
Попадание!
Ход игрока 2:
Попадание!
Ход игрока 2:
Попадание!
Ход игрока 2:
Попадание!
Ход игрока 2:
Промах!
Ход игрока 1:
Промах!
Ход игрока 2:
Попадание!
Ход игрока 2:
Попадание!
Победил игрок 2
Игра завершена!
```

rauf@rauf-VirtualBox:~/os_course_project_ffinal$ ./main_client_1

```
Введите ваш логин: rauf
Вы успешно авторизовались!
Вы хотите пригласить друга? (y/n)
y
Чтобы пригласить друга, напишите: invite (friend_login)
invite archi
Вы пригласили игрока с ником archi
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):
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


Разместите корабль 2 (1x2):
4 1 V

  0 1 2 3 4 5 6 7 8 9
0 O
1         O
2     O   O
3
4
5           O
6
7               O
8
9
```

```
Разместите корабль 2 (1x2):
3 4 V

  0 1 2 3 4 5 6 7 8 9
0 O
1         O
2     O   O
3
4       O
5       O   O
6
7               O
8
9


Разместите корабль 2 (1x2):
7 4 V

  0 1 2 3 4 5 6 7 8 9
0 O
1         O
2     O   O
3
4       O       O
5       O   O   O
6
7               O
8
9


Разместите корабль 3 (1x3):
0 9 H

  0 1 2 3 4 5 6 7 8 9
0 O
1         O
2     O   O
3
4       O       O
5       O   O   O
6
7               O
8
9 O O O


Разместите корабль 3 (1x3):
6 9 H

  0 1 2 3 4 5 6 7 8 9
0 O
1         O
2     O   O
3
4       O       O
5       O   O   O
6
7               O
8
9 O O O       O O O
```

```
Разместите корабль 4 (1x4):
9 4 V

  0 1 2 3 4 5 6 7 8 9
0 O
1         O
2     O   O
3
4       O       O   O
5       O   O   O   O
6                   O
7               O   O
8
9 O O O       O O O


Ваш ход:
Введите координаты выстрела (формат: x y):
0 0
Попадание!
Ваш ход:
Введите координаты выстрела (формат: x y):
0 1
Промах!

  0 1 2 3 4 5 6 7 8 9
0 X
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!
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:
Противник промахнулся

  0 1 2 3 4 5 6 7 8 9
0 O
1         X
2     X   X
3
4       X       X   X
5       X   X   X   X
6                   X
7               X   O
8
9 X X X       X X X *


Ваш ход:
Введите координаты выстрела (формат: x y):
3 3
Промах!

  0 1 2 3 4 5 6 7 8 9
0 X
1 *
2
3       *
4
5
6
7
8
9


Ход соперника:
Вас подстрелили!
Ход соперника:
Вас подстрелили!
Вы проиграли!
```

rauf@rauf-VirtualBox:~/os_course_project_ffinal$ ./main_client_2

```
Введите ваш логин: archi
Вы успешно авторизовались!
Вы хотите пригласить друга? (y/n)
n
Ждем приглашения от друга...
Игрок с ником rauf 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):
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


Разместите корабль 2 (1x2):
4 1 V

  0 1 2 3 4 5 6 7 8 9
0 O
1         O
2     O   O
3
4
5           O
6
7               O
8
9


Разместите корабль 2 (1x2):
3 4 V

  0 1 2 3 4 5 6 7 8 9
0 O
1         O
2     O   O
3
4       O
5       O   O
6
7               O
8
9


Разместите корабль 2 (1x2):
7 4 V

  0 1 2 3 4 5 6 7 8 9
0 O
1         O
2     O   O
3
4       O       O
5       O   O   O
6
7               O
8
9


Разместите корабль 3 (1x3):
0 9 H

  0 1 2 3 4 5 6 7 8 9
0 O
1         O
2     O   O
3
4       O       O
```

```
5       O   O   O
6
7               O
8
9 O O O


Разместите корабль 3 (1x3):
6 9 H

  0 1 2 3 4 5 6 7 8 9
0 O
1         O
2     O   O
3
4       O       O
5       O   O   O
6
7               O
8
9 O O O       O O O


Разместите корабль 4 (1x4):
9 4 V

  0 1 2 3 4 5 6 7 8 9
0 O
1         O
2     O   O
3
4       O       O   O
5       O   O   O   O
6                   O
7               O   O
8
9 O O O       O O O


Ход соперника:
Вас подстрелили!
Ход соперника:
Противник промахнулся

  0 1 2 3 4 5 6 7 8 9
0 X
1 *       O
2     O   O
3
4       O       O   O
5       O   O   O   O
6                   O
7               O   O
8
9 O O O       O O O


Ваш ход:
Введите координаты выстрела (формат: x y):
2 2
Попадание!
Ваш ход:
Введите координаты выстрела (формат: x y):
4 1
```

```
Попадание!
Ваш ход:
Введите координаты выстрела (формат: x y):
4 2
Попадание!
Ваш ход:
Введите координаты выстрела (формат: x y):
3 4
Попадание!
Ваш ход:
Введите координаты выстрела (формат: x y):
3 5
Попадание!
Ваш ход:
Введите координаты выстрела (формат: x y):
5 5
Попадание!
Ваш ход:
Введите координаты выстрела (формат: x y):
7 4
Попадание!
Ваш ход:
Введите координаты выстрела (формат: x y):
7 5
Попадание!
Ваш ход:
Введите координаты выстрела (формат: x y):
7 7
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 9
Попадание!
Ваш ход:
Введите координаты выстрела (формат: x y):
6 9
Попадание!
Ваш ход:
Введите координаты выстрела (формат: x y):
7 9
Попадание!
Ваш ход:
Введите координаты выстрела (формат: x y):
8 9
Попадание!
Ваш ход:
Введите координаты выстрела (формат: x y):
9 4
Попадание!
Ваш ход:
Введите координаты выстрела (формат: x y):
9 5
Попадание!
Ваш ход:
Введите координаты выстрела (формат: x y):
9 6
```

```
Попадание!
Ваш ход:
Введите координаты выстрела (формат: x y):
9 9
Промах!

  0 1 2 3 4 5 6 7 8 9
0
1         X
2     X   X
3
4       X       X   X
5       X   X   X   X
6                   X
7               X
8
9 X X X       X X X *


Ход соперника:
Противник промахнулся

  0 1 2 3 4 5 6 7 8 9
0 X
1 *       O
2     O   O
3       *
4       O       O   O
5       O   O   O   O
6                   O
7               O   O
8
9 O O O       O O O


Ваш ход:
Введите координаты выстрела (формат: x y):
9 7
Попадание!
Ваш ход:
Введите координаты выстрела (формат: x y):
0 0
Попадание!
Вы выиграли!
```

Выводы

Работа над курсовым проектом позволила мне углубить знания в области создания клиент-серверных приложений с использованием ZeroMQ. В процессе реализации проекта я применил навыки синхронизации данных, многопоточности и обработки сообщений между сервером и клиентом. Полученные результаты продемонстрировали успешную работу предложенной архитектуры и алгоритмов, которые соответствуют поставленным требованиям.

Проект подчеркнул важность структурированного подхода и тщательного планирования на всех этапах разработки. Также он позволил мне лучше понять сетевые технологии и принципы взаимодействия в многопользовательских системах. В будущем проект можно будет развивать, добавив новые функции, улучшив интерфейс и проведя оптимизацию.