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: Агафонов А.C.

Преподаватель: Миронов Е.С.

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

Дата: 27.12.24

Москва, 2024

**Постановка задачи**

**Вариант 3.**

**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.

Морской бой. Общение между сервером и клиентом необходимо организовать при помощи очередей сообщений (например, ZeroMQ). Каждый игрок должен при запуске ввести свой логин. Для каждого игрока должна вестись статистика игр (сколько побед/поражений). Игрок может посмотреть свою статистику.

**Общий метод и алгоритм решения**

Использованные системные вызовы:

* pid\_t getpid(void);– Возвращает идентификатор (PID) текущего процесса.
* int kill(pid\_t pid, int sig);– Отправляет сигнал процессу с указанным PID.

1. **Клиент-серверная архитектура через ZeroMQ**

* Сервер слушает «основные» команды (login, create, join, stats) на одном сокете (**server** = ZMQ\_REP, **client** = ZMQ\_REQ).
* Для «игры» создаются дополнительные сокеты: **сервер** = ZMQ\_REQ, **клиент** = ZMQ\_REP. Так сервер может «посылать первым», а клиент «отвечать» в логике пошагового боя.

1. **Авторизация и статистика**

* При запуске клиент вводит **логин** и отправляет его серверу (login:<логин>:<pid>).
* Сервер хранит в хэш-таблице статистику игроков (wins, loses).

1. **Создание и присоединение к игре**

* Клиент может отправить create <имя> → сервер формирует «комнату» (GameRoom), где пока 1 игрок.
* Другой клиент отправляет join <имя> → добавляется второй игрок в ту же комнату.
* Если в комнате теперь 2 игрока, сервер вызывает game.play().

1. **Расстановка кораблей**

* Сервер поочерёдно запрашивает у клиента: «Введите ориентацию (V/H)», «Разместите корабль (x y)».
* Проверяется, не пересекаются ли корабли. Если всё хорошо — «рисуем» корабль и показываем обновлённое поле. Если нет — отправляем клиенту Error, просим заново.

1. **Игровой цикл** (ходы)

* Сервер по очереди шлёт игрокам «your\_turn»/«not\_your\_turn».
* Тот, кто ходит, получает от сервера команду shoot, вводит координаты выстрела (coords:x:y).
* Сервер проверяет попадание/промах, обновляет поле, при попадании позволяет ходить ещё раз.

1. **Окончание игры**

* Когда у одного из игроков все корабли потоплены, сервер шлёт ему lose, а противнику win.
* Сервер обновляет статистику побед и поражений.
* Комната освобождается, но **сервер** и **клиенты** продолжают работу. Клиенты могут снова создать/присоединиться к новой игре, пока не введут команду exit.

**Код программы**

**Client.cpp**

#include <iostream>

#include <string>

#include <sstream>

#include <unistd.h>

#include <csignal>

#include <thread>

#include <chrono>

#include "myMQ.h"

int PORT\_ITER = 0;

static zmq::socket\_t\* gPlayerSocket = nullptr;

void gameLoop() {

    while (true) {

        std::string msg = receive\_message(\*gPlayerSocket);

        if (msg.empty()) {

            std::cout << "[Client] Пустое сообщение, завершаем игровой режим.\n";

            return;

        }

        if (msg == "your\_turn") {

            send\_message(\*gPlayerSocket, "ok");

        }

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

            send\_message(\*gPlayerSocket, "ok");

        }

        else if (msg == "shoot") {

            int x, y;

            std::cout << "[Client] Ваш ход! Введите x y: ";

            std::cin >> x >> y;

            std::stringstream ss;

            ss << "coords:" << x << ":" << y;

            send\_message(\*gPlayerSocket, ss.str());

        }

        else if (msg == "shooted") {

            std::cout << "[Client] Попадание!\n";

            send\_message(\*gPlayerSocket, "ok");

        }

        else if (msg == "miss") {

            std::cout << "[Client] Промах!\n";

            send\_message(\*gPlayerSocket, "ok");

        }

        else if (msg.rfind("board", 0) == 0) {

            std::string boardStr = msg.substr(5);

            std::cout << "[Client] Текущее поле:\n" << boardStr << std::endl;

            send\_message(\*gPlayerSocket, "ok");

        }

        else if (msg == "win") {

            std::cout << "[Client] Вы выиграли!\n";

            send\_message(\*gPlayerSocket, "ok");

            return;

        }

        else if (msg == "lose") {

            std::cout << "[Client] Вы проиграли!\n";

            send\_message(\*gPlayerSocket, "ok");

            return;

        }

        else if (msg.rfind("Введите ориентацию", 0) == 0) {

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

            std::string orientation;

            std::cin >> orientation;

            send\_message(\*gPlayerSocket, orientation);

        }

        else if (msg.rfind("Разместите", 0) == 0) {

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

            int x, y;

            std::cin >> x >> y;

            std::stringstream ss;

            ss << "coords:" << x << ":" << y;

            send\_message(\*gPlayerSocket, ss.str());

        }

        else if (msg.rfind("Error", 0) == 0) {

            std::cout << "[Client] Ошибка: " << msg << std::endl;

            send\_message(\*gPlayerSocket, "ok");

        }

        else {

            std::cout << "[Client] Неизвестное сообщение: " << msg << std::endl;

        }

    }

}

int main() {

    zmq::context\_t context(2);

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

    main\_socket.connect(GetConPort(5555));

    std::cout << "Добро пожаловать в Морской бой!\n";

    // Авторизация

    std::string login;

    while (true) {

        std::cout << "Введите ваш логин: ";

        std::cin >> login;

        pid\_t pid = getpid();

        std::stringstream ss;

        ss << "login:" << login << ":" << pid;

        send\_message(main\_socket, ss.str());

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

        if (resp.rfind("Ok:", 0) == 0) {

            auto colPos = resp.find(':');

            int socketIndex = std::stoi(resp.substr(colPos+1));

            gPlayerSocket = new zmq::socket\_t(context, ZMQ\_REP);

            gPlayerSocket->connect(GetConPort(5556 + socketIndex));

            std::cout << "[Client] Авторизация прошла успешно. Игровой порт: "

                      << (5556 + socketIndex) << std::endl;

            break;

        }

        else if (resp.find("Error:NameAlreadyExist") == 0) {

            std::cout << "Ошибка: логин уже существует, попробуйте другой.\n";

        }

        else if (resp.find("Error:NoFreeSockets") == 0) {

            std::cout << "Ошибка: на сервере нет свободных слотов.\n";

            return 0;

        }

        else {

            std::cout << "Ошибка: " << resp << std::endl;

        }

    }

    // Цикл команд

    while (true) {

        std::cout << "\n=========================================\n"

                  << "Доступные команды:\n"

                  << "  create <имя\_игры>  - создать комнату\n"

                  << "  join <имя\_игры>    - присоединиться к комнате\n"

                  << "  stats              - показать свою статистику\n"

                  << "  exit               - выйти\n"

                  << "=========================================\n"

                  << "Введите команду: ";

        std::string command;

        std::cin >> command;

        if (command == "create") {

            std::string roomName;

            std::cin >> roomName;

            std::stringstream ss;

            ss << "create:" << login << ":" << roomName;

            send\_message(main\_socket, ss.str());

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

            std::cout << "[Client] Сервер ответил: " << resp << std::endl;

            if (resp.rfind("Ok:RoomCreated", 0) == 0) {

                std::cout << "[Client] Ожидаем второго игрока...\n";

                gameLoop();

            }

        }

        else if (command == "join") {

            std::string roomName;

            std::cin >> roomName;

            std::stringstream ss;

            ss << "join:" << login << ":" << roomName;

            send\_message(main\_socket, ss.str());

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

            std::cout << "[Client] Сервер ответил: " << resp << std::endl;

            if (resp.rfind("Ok:RoomJoined", 0) == 0) {

                gameLoop();

            }

        }

        else if (command == "stats") {

            std::stringstream ss;

            ss << "stats:" << login;

            send\_message(main\_socket, ss.str());

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

            std::cout << "[Client] Статистика: " << resp << std::endl;

        }

        else if (command == "exit") {

            std::cout << "[Client] Завершение работы.\n";

            return 0;

        }

        else {

            std::cout << "[Client] Неизвестная команда.\n";

        }

    }

    return 0;

}

**Game.h**

#pragma once

#include <iostream>

#include <string>

#include <sstream>

#include <csignal>

#include "player.h"

/\*

 \* Класс, управляющий логикой Морского боя между двумя игроками

 \*/

class Game {

public:

    Player player1;

    Player player2;

    void play(zmq::socket\_t& player1\_socket,

              zmq::socket\_t& player2\_socket,

              pid\_t first\_player\_pid,

              pid\_t second\_player\_pid)

    {

        std::cout << "[Server] Игра началась между (PID "

                  << first\_player\_pid << ") и (PID "

                  << second\_player\_pid << ")!\n";

        player1.num = 1;

        player2.num = 2;

        player1.placeShips(player1\_socket, first\_player\_pid, second\_player\_pid);

        player2.placeShips(player2\_socket, first\_player\_pid, second\_player\_pid);

        int turn = 0;

        while (!gameOver()) {

            bool alive = try\_recv(first\_player\_pid, second\_player\_pid);

            if (!alive) {

                std::cout << "[Server] Игра прервана (один из процессов умер)\n";

                kill(first\_player\_pid, SIGTERM);

                kill(second\_player\_pid, SIGTERM);

                exit(0);

            }

            if (turn % 2 == 0) {

                send\_message(player1\_socket, "your\_turn");

                receive\_message(player1\_socket);

                send\_message(player2\_socket, "not\_your\_turn");

                receive\_message(player2\_socket);

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

                    if (gameOver()) {

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

                        receive\_message(player1\_socket);

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

                        receive\_message(player2\_socket);

                        break;

                    }

                    continue;

                } else {

                    turn++;

                }

            } else {

                send\_message(player2\_socket, "your\_turn");

                receive\_message(player2\_socket);

                send\_message(player1\_socket, "not\_your\_turn");

                receive\_message(player1\_socket);

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

                    if (gameOver()) {

                        // Победил 2

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

                        receive\_message(player2\_socket);

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

                        receive\_message(player1\_socket);

                        break;

                    }

                    continue;

                } else {

                    turn++;

                }

            }

        }

        std::cout << "[Server] Игра завершена!\n";

    }

private:

    bool gameOver() const {

        return allShipsDead(player1) || allShipsDead(player2);

    }

    bool allShipsDead(const Player& pl) const {

        for (auto& row : pl.board) {

            for (auto c : row) {

                if (c == 'O') {

                    return false;

                }

            }

        }

        return true;

    }

    bool playerTurn(Player& attacker, Player& defender,

                    zmq::socket\_t& attacker\_socket,

                    zmq::socket\_t& defender\_socket)

    {

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

        std::string recv = receive\_message(attacker\_socket);

        std::stringstream ss(recv);

        std::string cmd, sx, sy;

        std::getline(ss, cmd, ':'); // coords

        std::getline(ss, sx, ':');

        std::getline(ss, sy, ':');

        if (sx.empty() || sy.empty()) {

            // Неверный формат

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

            receive\_message(attacker\_socket);

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

            receive\_message(defender\_socket);

            return false;

        }

        int x = std::stoi(sx);

        int y = std::stoi(sy);

        // Проверка

        if (x < 0 || x >= BOARD\_SIZE || y < 0 || y >= BOARD\_SIZE) {

            // Мимо

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

            receive\_message(attacker\_socket);

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

            receive\_message(defender\_socket);

            return false;

        }

        // Попал?

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

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

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

            receive\_message(attacker\_socket);

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

            receive\_message(defender\_socket);

            return true;

        } else {

            // Промах

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

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

            }

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

            receive\_message(attacker\_socket);

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

            receive\_message(defender\_socket);

            // Покажем поле

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

            receive\_message(attacker\_socket);

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

            receive\_message(defender\_socket);

            return false;

        }

    }

};

**myMQ.h**

#pragma once

#include <zmq.hpp>

#include <iostream>

#include <signal.h>

#include <string>

extern int PORT\_ITER;

inline std::string GetConPort(int port) {

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

}

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

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

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

    return bool(socket.send(message, zmq::send\_flags::none));

}

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

    zmq::message\_t message;

    bool ok = false;

    try {

        ok = bool(socket.recv(message, zmq::recv\_flags::none));

    } catch(...) {

        ok = false;

    }

    if (!ok) {

        return "";

    }

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

}

inline bool try\_recv(pid\_t first\_player\_pid, pid\_t second\_player\_pid) {

    if (kill(first\_player\_pid, 0) != 0 || kill(second\_player\_pid, 0) != 0) {

        return false;

    }

    return true;

}

**player.h**

#pragma once

#include <iostream>

#include <vector>

#include <string>

#include <sstream>

#include <csignal>

#include <cmath>

#include "myMQ.h"

// Размер игрового поля

const int BOARD\_SIZE = 10;

/\*

 \* Класс, описывающий игрока и его поле

 \*/

class Player {

public:

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

    int num;

    Player() {

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

        num = 0;

    }

    void placeShips(zmq::socket\_t& player\_socket, pid\_t first\_player\_pid, pid\_t second\_player\_pid) {

        bool alive;

        std::vector<int> shipSizes = {4,3,3,2,2,2,1,1,1,1};

        for (int size : shipSizes) {

            while (true) {

                std::string orientationMsg = "Введите ориентацию (V/H) для корабля на " + std::to\_string(size) + " палуб(ы)";

                send\_message(player\_socket, orientationMsg);

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

                {

                    // Удалим лишние пробелы

                    std::stringstream tmpSS(orientation);

                    tmpSS >> orientation;

                }

                std::string placeMsg = "Разместите " + std::to\_string(size) + "-палубный корабль (укажите начальные координаты x y)";

                send\_message(player\_socket, placeMsg);

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

                alive = try\_recv(first\_player\_pid, second\_player\_pid);

                if (!alive) {

                    std::cout << "[Server] Игра прервана из-за смерти процесса\n";

                    kill(first\_player\_pid, SIGTERM);

                    kill(second\_player\_pid, SIGTERM);

                    exit(0);

                }

                int startX = -1, startY = -1;

                {

                    std::stringstream ss(resp);

                    std::string tmp;

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

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

                    startX = std::stoi(tmp);

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

                    startY = std::stoi(tmp);

                }

                // Проверим корректность ориентации

                if (orientation != "V" && orientation != "H") {

                    send\_message(player\_socket, "Error: Неверная ориентация (V/H)");

                    receive\_message(player\_socket);

                    continue;

                }

                // Проверим, можно ли разместить корабль

                if (checkShipPlacementOk(startX, startY, orientation, size)) {

                    markShip(startX, startY, orientation, size);

                    send\_message(player\_socket, "board" + getBoard());

                    receive\_message(player\_socket);

                    break;

                } else {

                    send\_message(player\_socket, "Error: Неверное расположение корабля. Повторите ввод");

                    receive\_message(player\_socket);

                }

            }

        }

    }

    /\*

     \* Возвращает поле игрока в виде строки (с кораблями)

     \*/

    std::string getBoard() const {

        std::stringstream ss;

        ss << "\n  0 1 2 3 4 5 6 7 8 9\n";

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

            ss << i << " ";

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

                ss << board[i][j] << " ";

            }

            ss << "\n";

        }

        ss << "\n";

        return ss.str();

    }

    std::string getClearBoard() const {

        std::stringstream ss;

        ss << "\n  0 1 2 3 4 5 6 7 8 9\n";

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

            ss << i << " ";

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

                if (board[i][j] == 'O') {

                    ss << "  ";

                } else {

                    ss << board[i][j] << " ";

                }

            }

            ss << "\n";

        }

        ss << "\n";

        return ss.str();

    }

private:

    bool checkShipPlacementOk(int startX, int startY, const std::string& orientation, int size) const {

        std::vector<std::pair<int,int>> cells;

        cells.reserve(size);

        for (int k = 0; k < size; k++) {

            int xx = startX;

            int yy = startY;

            if (orientation == "V") {

                xx += k;

            } else {

                yy += k;

            }

            if (xx < 0 || xx >= BOARD\_SIZE || yy < 0 || yy >= BOARD\_SIZE) {

                return false;

            }

            cells.emplace\_back(xx, yy);

        }

        for (auto &c : cells) {

            int x = c.first;

            int y = c.second;

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

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

                    if (i >= 0 && i < BOARD\_SIZE && j >= 0 && j < BOARD\_SIZE) {

                        if (board[i][j] == 'O') {

                            return false;

                        }

                    }

                }

            }

        }

        return true;

    }

    void markShip(int startX, int startY, const std::string& orientation, int size) {

        for (int k = 0; k < size; k++) {

            if (orientation == "V") {

                board[startX + k][startY] = 'O';

            } else {

                board[startX][startY + k] = 'O';

            }

        }

    }

};

**server.cpp**

#include <iostream>

#include <string>

#include <map>

#include <unordered\_map>

#include <csignal>

#include <unistd.h>

#include <sstream>

#include <zmq.hpp>

#include "myMQ.h"

#include "game.h"

#include "player.h"

int PORT\_ITER = 0;

struct GameRoom {

    std::string name;

    pid\_t playersPid[2];

    zmq::socket\_t\* playersSock[2];

    int filled = 0;

};

int main() {

    zmq::context\_t context(3);

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

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

    const int MAX\_SOCKETS = 5;

    zmq::socket\_t sockets[MAX\_SOCKETS] = {

        zmq::socket\_t(context, ZMQ\_REQ),

        zmq::socket\_t(context, ZMQ\_REQ),

        zmq::socket\_t(context, ZMQ\_REQ),

        zmq::socket\_t(context, ZMQ\_REQ),

        zmq::socket\_t(context, ZMQ\_REQ),

    };

    for (int i = 0; i < MAX\_SOCKETS; i++) {

        std::string bindStr = "tcp://\*:" + std::to\_string(5556 + i);

        sockets[i].bind(bindStr);

    }

    std::map<std::string, pid\_t> loginMap;                      // login -> pid

    std::map<pid\_t, int> pidToSocketIndex;                      // pid -> socket index

    std::unordered\_map<std::string, std::pair<int,int>> stats;  // login -> (wins, loses)

    std::map<std::string, GameRoom> rooms;                      // roomName -> GameRoom

    std::cout << "Сервер запущен.\n";

    while (true) {

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

        if (request.empty()) {

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

            continue;

        }

        std::cout << "[SERVER] Получено сообщение: " << request << std::endl;

        std::stringstream ss(request);

        std::string cmd;

        std::getline(ss, cmd, ':');

        if (cmd == "login") {

            // "login:<login>:<pid>"

            std::string login;

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

            std::string pidStr;

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

            pid\_t p = (pid\_t)std::stoi(pidStr);

            if (loginMap.find(login) != loginMap.end()) {

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

                continue;

            }

            if (PORT\_ITER >= MAX\_SOCKETS) {

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

                continue;

            }

            loginMap[login] = p;

            pidToSocketIndex[p] = PORT\_ITER;

            if (stats.find(login) == stats.end()) {

                stats[login] = {0, 0}; // wins=0, loses=0

            }

            std::string resp = "Ok:" + std::to\_string(PORT\_ITER);

            send\_message(main\_socket, resp);

            PORT\_ITER++;

        }

        else if (cmd == "create") {

            // "create:<login>:<roomName>"

            std::string login, roomName;

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

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

            if (loginMap.find(login) == loginMap.end()) {

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

                continue;

            }

            if (rooms.find(roomName) != rooms.end()) {

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

                continue;

            }

            GameRoom gr;

            gr.name = roomName;

            gr.filled = 1;

            gr.playersPid[0] = loginMap[login];

            gr.playersPid[1] = 0;

            gr.playersSock[0] = &sockets[ pidToSocketIndex[ loginMap[login] ] ];

            gr.playersSock[1] = nullptr;

            rooms[roomName] = gr;

            send\_message(main\_socket, "Ok:RoomCreated");

        }

        else if (cmd == "join") {

            // "join:<login>:<roomName>"

            std::string login, roomName;

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

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

            if (loginMap.find(login) == loginMap.end()) {

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

                continue;

            }

            auto it = rooms.find(roomName);

            if (it == rooms.end()) {

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

                continue;

            }

            GameRoom &gr = it->second;

            if (gr.filled >= 2) {

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

                continue;

            }

            gr.playersPid[1] = loginMap[login];

            gr.playersSock[1] = &sockets[ pidToSocketIndex[ loginMap[login] ] ];

            gr.filled++;

            send\_message(main\_socket, "Ok:RoomJoined");

            if (gr.filled == 2) {

                pid\_t pid1 = gr.playersPid[0];

                pid\_t pid2 = gr.playersPid[1];

                zmq::socket\_t &sock1 = \*(gr.playersSock[0]);

                zmq::socket\_t &sock2 = \*(gr.playersSock[1]);

                std::cout << "[SERVER] Начинаем игру в комнате " << roomName << std::endl;

                Game game;

                game.play(sock1, sock2, pid1, pid2);

                // По окончании игры — определим, кто проиграл

                bool p1lost = false;

                bool p2lost = false;

                {

                    bool p1Dead = true;

                    for (auto &row : game.player1.board) {

                        for (auto c : row) {

                            if (c == 'O') {

                                p1Dead = false;

                                break;

                            }

                        }

                        if (!p1Dead) break;

                    }

                    p1lost = p1Dead;

                }

                {

                    bool p2Dead = true;

                    for (auto &row : game.player2.board) {

                        for (auto c : row) {

                            if (c == 'O') {

                                p2Dead = false;

                                break;

                            }

                        }

                        if (!p2Dead) break;

                    }

                    p2lost = p2Dead;

                }

                std::string login1, login2;

                for (auto &kv : loginMap) {

                    if (kv.second == pid1) login1 = kv.first;

                    if (kv.second == pid2) login2 = kv.first;

                }

                if (!login1.empty() && !login2.empty()) {

                    if (p1lost && !p2lost) {

                        stats[login1].second += 1;

                        stats[login2].first += 1;

                    }

                    if (p2lost && !p1lost) {

                        stats[login2].second += 1;

                        stats[login1].first += 1;

                    }

                }

                rooms.erase(roomName);

            }

        }

        else if (cmd == "stats") {

            // "stats:<login>"

            std::string login;

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

            auto it = stats.find(login);

            if (it == stats.end()) {

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

            } else {

                auto [w, l] = it->second;

                std::stringstream out;

                out << "Stats: wins=" << w << ", loses=" << l;

                send\_message(main\_socket, out.str());

            }

        }

        else {

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

        }

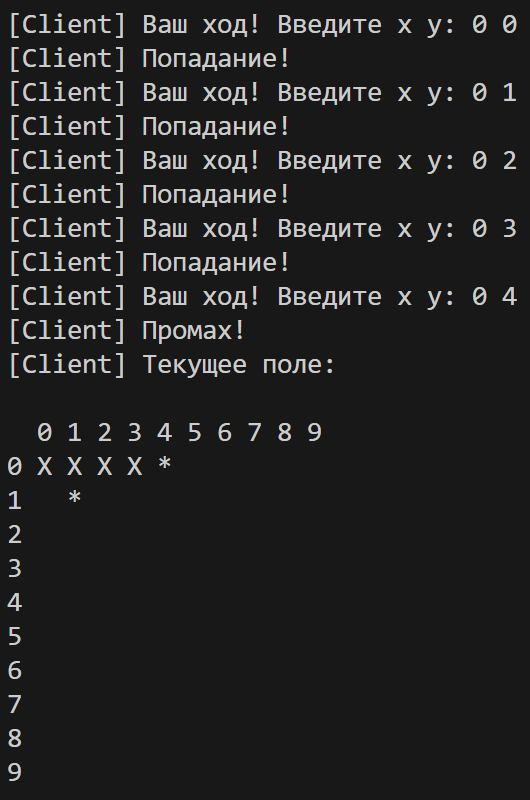
    }

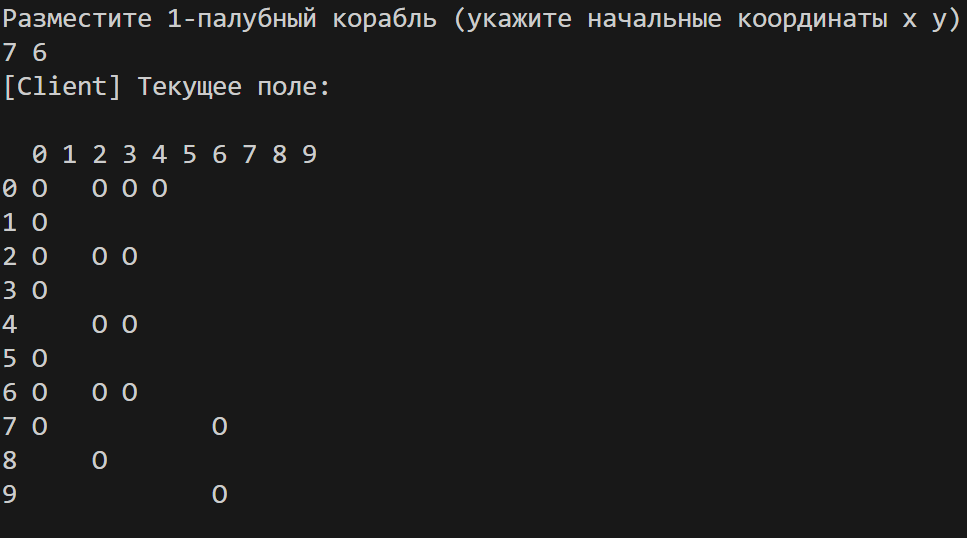
    return 0;

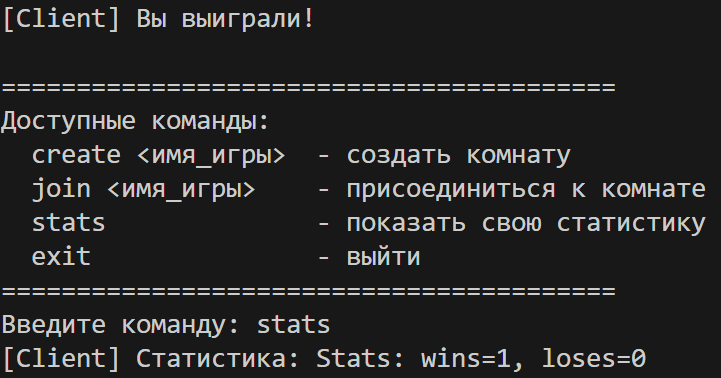
}

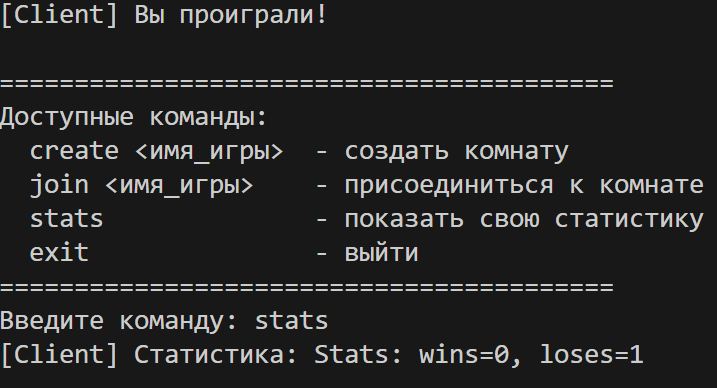
**Протокол работы программы**

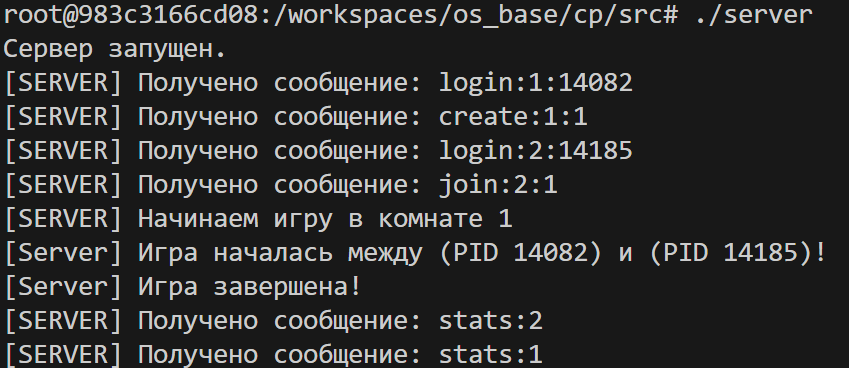
**Тестирование:**











**Strace:**

**Server:**

20241 socket(AF\_INET, SOCK\_STREAM|SOCK\_CLOEXEC, IPPROTO\_TCP) = 13

20241 bind(13, {sa\_family=AF\_INET, sin\_port=htons(5555), sin\_addr=inet\_addr("0.0.0.0")}, 16) = 0

20241 listen(13, 100)                   = 0

20241 socket(AF\_INET, SOCK\_STREAM|SOCK\_CLOEXEC, IPPROTO\_TCP) = 21

20241 bind(21, {sa\_family=AF\_INET, sin\_port=htons(5556), sin\_addr=inet\_addr("0.0.0.0")}, 16) = 0

20241 listen(21, 100)                   = 0

20241 socket(AF\_INET, SOCK\_STREAM|SOCK\_CLOEXEC, IPPROTO\_TCP) = 22

20241 bind(22, {sa\_family=AF\_INET, sin\_port=htons(5557), sin\_addr=inet\_addr("0.0.0.0")}, 16) = 0

20241 listen(22, 100)                   = 0

20241 socket(AF\_INET, SOCK\_STREAM|SOCK\_CLOEXEC, IPPROTO\_TCP) = 23

20241 bind(23, {sa\_family=AF\_INET, sin\_port=htons(5558), sin\_addr=inet\_addr("0.0.0.0")}, 16) = 0

20241 listen(23, 100)                   = 0

20241 socket(AF\_INET, SOCK\_STREAM|SOCK\_CLOEXEC, IPPROTO\_TCP) = 24

20241 bind(24, {sa\_family=AF\_INET, sin\_port=htons(5559), sin\_addr=inet\_addr("0.0.0.0")}, 16) = 0

20241 listen(24, 100)                   = 0

20241 socket(AF\_INET, SOCK\_STREAM|SOCK\_CLOEXEC, IPPROTO\_TCP) = 25

20241 bind(25, {sa\_family=AF\_INET, sin\_port=htons(5560), sin\_addr=inet\_addr("0.0.0.0")}, 16) = 0

20241 listen(25, 100)                   = 0

20244 recvfrom(26, 0x7f5b78001318, 12, 0, NULL, NULL) = -1 EAGAIN (Resource temporarily unavailable)

20244 sendto(26, "\377\0\0\0\0\0\0\0\1\177", 10, 0, NULL, 0) = 10

20244 recvfrom(26, "\377\0\0\0\0\0\0\0\1\177", 12, 0, NULL, NULL) = 10

20244 recvfrom(26, 0x7f5b78001322, 2, 0, NULL, NULL) = -1 EAGAIN (Resource temporarily unavailable)

20244 sendto(26, "\3", 1, 0, NULL, 0)   = 1

20244 recvfrom(26, "\3", 2, 0, NULL, NULL) = 1

20244 recvfrom(26, 0x7f5b78001323, 53, 0, NULL, NULL) = -1 EAGAIN (Resource temporarily unavailable)

20244 sendto(26, "\1NULL\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 53, 0, NULL, 0) = 53

20244 recvfrom(26, "\1NULL\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 53, 0, NULL, NULL) = 53

20244 recvfrom(26, 0x7f5b78003b68, 8192, 0, NULL, NULL) = -1 EAGAIN (Resource temporarily unavailable)

20244 sendto(26, "\4\31\5READY\vSocket-Type\0\0\0\3REP", 27, 0, NULL, 0) = 27

20244 recvfrom(26, "\4&\5READY\vSocket-Type\0\0\0\3REQ\10Iden"..., 8192, 0, NULL, NULL) = 40

20244 recvfrom(26, "\1\0\0\rlogin:1:20323", 8192, 0, NULL, NULL) = 17

20244 sendto(26, "\1\0\0\4Ok:0", 8, 0, NULL, 0) = 8

20245 recvfrom(27, 0x7f5b70001318, 12, 0, NULL, NULL) = -1 EAGAIN (Resource temporarily unavailable)

20245 sendto(27, "\377\0\0\0\0\0\0\0\1\177", 10, 0, NULL, 0) = 10

20245 recvfrom(27, "\377\0\0\0\0\0\0\0\1\177", 12, 0, NULL, NULL) = 10

20245 recvfrom(27, 0x7f5b70001322, 2, 0, NULL, NULL) = -1 EAGAIN (Resource temporarily unavailable)

20245 sendto(27, "\3", 1, 0, NULL, 0)   = 1

20245 recvfrom(27, "\3", 2, 0, NULL, NULL) = 1

20245 recvfrom(27, 0x7f5b70001323, 53, 0, NULL, NULL) = -1 EAGAIN (Resource temporarily unavailable)

20245 sendto(27, "\1NULL\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 53, 0, NULL, 0) = 53

20245 recvfrom(27, "\1NULL\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 53, 0, NULL, NULL) = 53

20245 recvfrom(27, 0x7f5b70003b78, 8192, 0, NULL, NULL) = -1 EAGAIN (Resource temporarily unavailable)

20245 sendto(27, "\4&\5READY\vSocket-Type\0\0\0\3REQ\10Iden"..., 40, 0, NULL, 0) = 40

20245 recvfrom(27, "\4\31\5READY\vSocket-Type\0\0\0\3REP", 8192, 0, NULL, NULL) = 27

20245 recvfrom(27,  <unfinished ...>

20244 recvfrom(26,  <unfinished ...>

20245 <... recvfrom resumed>"", 8192, 0, NULL, NULL) = 0

20244 <... recvfrom resumed>"", 8192, 0, NULL, NULL) = 0

20241 --- SIGINT {si\_signo=SIGINT, si\_code=SI\_KERNEL} ---

20245 +++ killed by SIGINT +++

20244 +++ killed by SIGINT +++

20243 +++ killed by SIGINT +++

20246 +++ killed by SIGINT +++

20241 +++ killed by SIGINT +++

**Client:**

20325 socket(AF\_INET, SOCK\_STREAM|SOCK\_CLOEXEC, IPPROTO\_TCP) = 11

20325 connect(11, {sa\_family=AF\_INET, sin\_port=htons(5555), sin\_addr=inet\_addr("127.0.0.1")}, 16) = -1 EINPROGRESS (Operation now in progress)

20325 recvfrom(11, 0x7f0d6c001be8, 12, 0, NULL, NULL) = -1 EAGAIN (Resource temporarily unavailable)

20325 sendto(11, "\377\0\0\0\0\0\0\0\1\177", 10, 0, NULL, 0) = 10

20325 recvfrom(11, "\377\0\0\0\0\0\0\0\1\177", 12, 0, NULL, NULL) = 10

20325 recvfrom(11, 0x7f0d6c001bf2, 2, 0, NULL, NULL) = -1 EAGAIN (Resource temporarily unavailable)

20325 sendto(11, "\3", 1, 0, NULL, 0)   = 1

20325 recvfrom(11, "\3", 2, 0, NULL, NULL) = 1

20325 recvfrom(11, 0x7f0d6c001bf3, 53, 0, NULL, NULL) = -1 EAGAIN (Resource temporarily unavailable)

20325 sendto(11, "\1NULL\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 53, 0, NULL, 0) = 53

20325 recvfrom(11, "\1NULL\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 53, 0, NULL, NULL) = 53

20325 recvfrom(11, 0x7f0d6c003d18, 8192, 0, NULL, NULL) = -1 EAGAIN (Resource temporarily unavailable)

20325 sendto(11, "\4&\5READY\vSocket-Type\0\0\0\3REQ\10Iden"..., 40, 0, NULL, 0) = 40

20325 recvfrom(11, "\4\31\5READY\vSocket-Type\0\0\0\3REP", 8192, 0, NULL, NULL) = 27

20325 sendto(11, "\1\0\0\rlogin:1:20323", 17, 0, NULL, 0) = 17

20325 recvfrom(11, "\1\0\0\4Ok:0", 8192, 0, NULL, NULL) = 8

20326 socket(AF\_INET, SOCK\_STREAM|SOCK\_CLOEXEC, IPPROTO\_TCP) = 13

20326 connect(13, {sa\_family=AF\_INET, sin\_port=htons(5556), sin\_addr=inet\_addr("127.0.0.1")}, 16) = -1 EINPROGRESS (Operation now in progress)

20326 recvfrom(13, 0x7f0d64001be8, 12, 0, NULL, NULL) = -1 EAGAIN (Resource temporarily unavailable)

20326 sendto(13, "\377\0\0\0\0\0\0\0\1\177", 10, 0, NULL, 0) = 10

20326 recvfrom(13, "\377\0\0\0\0\0\0\0\1\177", 12, 0, NULL, NULL) = 10

20326 recvfrom(13, 0x7f0d64001bf2, 2, 0, NULL, NULL) = -1 EAGAIN (Resource temporarily unavailable)

20326 sendto(13, "\3", 1, 0, NULL, 0)   = 1

20326 recvfrom(13, "\3", 2, 0, NULL, NULL) = 1

20326 recvfrom(13, 0x7f0d64001bf3, 53, 0, NULL, NULL) = -1 EAGAIN (Resource temporarily unavailable)

20326 sendto(13, "\1NULL\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 53, 0, NULL, 0) = 53

20326 recvfrom(13, "\1NULL\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 53, 0, NULL, NULL) = 53

20326 recvfrom(13, 0x7f0d64003d18, 8192, 0, NULL, NULL) = -1 EAGAIN (Resource temporarily unavailable)

20326 sendto(13, "\4\31\5READY\vSocket-Type\0\0\0\3REP", 27, 0, NULL, 0) = 27

20326 recvfrom(13, "\4&\5READY\vSocket-Type\0\0\0\3REQ\10Iden"..., 8192, 0, NULL, NULL) = 40

20323 --- SIGINT {si\_signo=SIGINT, si\_code=SI\_KERNEL} ---

20326 +++ killed by SIGINT +++

20325 +++ killed by SIGINT +++

20324 +++ killed by SIGINT +++

20323 +++ killed by SIGINT +++

**Вывод**

В ходе разработки многопользовательской игры «Морской бой» с использованием библиотеки ZeroMQ мне удалось успешно реализовать все основные функциональные требования, включая авторизацию пользователей, создание и присоединение к игровым комнатам, а также ведение статистики побед и поражений. Программа демонстрирует стабильную работу обмена сообщениями между сервером и клиентами, обеспечивая плавный игровой процесс.