

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-208Б-23

Студент: Денисов К.Д.

Преподаватель: Живалев Е.А.

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.

2. Авторизация и статистика

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

3. 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()`.

4. Расстановка кораблей

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

5. 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.

6. 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] Промаш!\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 {
    // Промax
    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:

```
[Client] Ваш ход! Введите x y: 0 0
[Client] Попадание!
[Client] Ваш ход! Введите x y: 0 1
[Client] Попадание!
[Client] Ваш ход! Введите x y: 0 2
[Client] Попадание!
[Client] Ваш ход! Введите x y: 0 3
[Client] Попадание!
[Client] Ваш ход! Введите x y: 0 4
[Client] Промех!
[Client] Текущее поле:
```

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

Разместите 1-палубный корабль (укажите начальные координаты x y)

```
7 6
[Client] Текущее поле:
```

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

[Client] Вы выиграли!

=====

Доступные команды:

```
create <имя_игры> - создать комнату
join <имя_игры>   - присоединиться к комнате
stats             - показать свою статистику
exit             - выйти
```

=====

Введите команду: stats

[Client] Статистика: Stats: wins=1, loses=0

[Client] Вы проиграли!

=====

Доступные команды:

```
create <имя_игры> - создать комнату
join <имя_игры>   - присоединиться к комнате
stats             - показать свою статистику
exit             - выйти
```

=====

Введите команду: stats

[Client] Статистика: Stats: wins=0, loses=1

```
root@983c3166cd08:/workspaces/os_base/cp/src# ./server
Сервер запущен.
[SERVER] Получено сообщение: login:1:14082
[SERVER] Получено сообщение: create:1:1
[SERVER] Получено сообщение: login:2:14185
[SERVER] Получено сообщение: join:2:1
[SERVER] Начинаем игру в комнате 1
[Server] Игра началась между (PID 14082) и (PID 14185)!
[Server] Игра завершена!
[SERVER] Получено сообщение: stats:2
[SERVER] Получено сообщение: stats:1
```

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)
```

```
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\1\177", 10, 0, NULL, 0) = 10
```

```
20325 recvfrom(11, "\377\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)

[illegible][illegible]

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

```
20325 sendto(11, "4&5READY\vSocket-Type\0\0\03REQ\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, "\\100\\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)
```

[illegible][illegible]

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

```
20326 sendto(13, "\431\5READY\vSocket-Type\0\0\03REP", 27, 0, NULL, 0) = 27
```

```
20326 recvfrom(13, "\4&\5READY\Socket-Type\0\0\03REQ\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.