

Московский Авиационный Институт  
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовая работа по курсу  
«Операционные системы»**

**ПРОЕКТИРОВАНИЕ КЛИЕНТ-СЕРВЕРНОЙ ИГРЫ**

Студент: Мирошников Дмитрий Евгеньевич

Группа: М8О–210Б–22

Вариант: 4

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_

Дата: \_\_\_\_\_

Подпись: \_\_\_\_\_

Москва, 2023

## **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.

Вариант: 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.

### **Общие сведения о программе**

Архитектура программы - клиент-серверная. К серверу подключаются два процесса - участники игры. Далее им предоставлена возможность осуществить сражение по правилам морского боя.

### **Общий алгоритм решения**

При запуске исполняемого файла клиента, он связывается с сервером с помощью специального сокета. Далее пользователю предлагается ввести логин, после чего определиться с тем, хочет ли он приглашать другого игрока в “комнату”. Для второго пользователя порядок действий аналогичный. Основная идея решения состоит в общении между клиентами и сервером в формате “запрос-ответ”, при котором сервер выполняет определённые действия в зависимости от типа запроса. Также при реализации была использована технология многопоточности для обработки приглашений на игру.

Для реализации поставленной задачи необходимо:

- 1) Вспомнить механизмы работы очереди сообщений
- 2) Написать программы, реализующие сервер и клиент, а также заголовочный файл, который содержит структуру сообщения
- 3) Протестировать

## Основные файлы программы

### client.cpp

```
#include <iostream>
#include <unistd.h>
#include <stdlib.h>
#include <string>
#include <sstream>
#include <signal.h>
#include <thread>
#include <chrono>
#include <ctime>

#include "myMQ.hpp"

std::string command;
pthread_mutex_t mutex;

zmq::context_t context(2);
zmq::socket_t player_socket(context, ZMQ_REP);

void* check_invite(void *param) {
    pid_t* mypid = static_cast<pid_t*>(param);
    std::string invite_tmp;
    pthread_mutex_lock(&mutex);
    std::string invite_msg = receive_message(player_socket);
    std::stringstream invite_ss(invite_msg);
    std::getline(invite_ss, invite_tmp, ':');
    std::this_thread::sleep_for(std::chrono::milliseconds(100));
    std::getline(invite_ss, invite_tmp, ':');
    std::cout << "Игрок с ником " + invite_tmp + " приглашает вас в игру!" << std::endl;
    std::cout << "Вы согласны? (y/n)" << std::endl;
    std::cin >> command;
    std::cout << "Ваш ответ: " + command + "\n";
    if (command[0] == 'y') {
        std::cout << "Вы приняли запрос!" << std::endl;
        send_message(player_socket, "accept");
        pthread_mutex_unlock(&mutex);
        pthread_exit(0);
    } else {
        std::cout << "Вы отклонили запрос!. Игра окончена, даже не начавшись :(" << std::endl;
        pthread_mutex_unlock(&mutex);
        send_message(player_socket, "reject");
    }
    pthread_exit(0);
}

int main(int argc, char** argv) {
    zmq::context_t context(2);
    zmq::socket_t main_socket(context, ZMQ_REQ);
    pid_t pid = getpid();
    main_socket.connect(GetConPort(5555));
    pthread_mutex_init(&mutex, NULL);

    pthread_t tid;
    std::string received_message;
    std::string tmp;
    bool login_stage = false;
    bool pid_flag = false;
    bool move_on_flag = false;
```

```
while(true) {  
    if (!login_stage) {  
        login_stage = true;
```

```
std::string login;  
std::cout << "Введите ваш логин: ";  
std::cin >> login;
```

```
std::string login_msg = "login:" + login + ":" + std::to_string(pid);  
send_message(main_socket, login_msg);  
received_message = receive_message(main_socket);  
std::stringstream ss(received_message);
```

```
std::getline(ss, tmp, ':');  
if (tmp == "Ok") {  
    std::getline(ss, tmp, ':');  
    PORT_ITER = std::stoi(tmp);  
    player_socket.connect(GetConPort(5555 + PORT_ITER));  
    std::cout << "Авторизация прошла успешно" << std::endl;  
    std::cout << "Вы хотите пригласить друга? (y/n)" << std::endl;  
    std::cin >> tmp;  
    std::string yes_no_message;  
    if (tmp[0] == 'n') {  
        yes_no_message = "no:" + std::to_string(pid);  
        send_message(main_socket, yes_no_message);  
        received_message = receive_message(main_socket);  
        if (received_message == "good") {  
            std::cout << "Ждем приглашения от друга..." << std::endl;  
            pthread_create(&tid, NULL, check_invite, &pid);  
            std::this_thread::sleep_for(std::chrono::microseconds(100));  
            break;  
        } else {  
            std::cout << "Тебе нужно пригласить второго игрока, чтобы начать игру"  
<< std::endl;  
            std::cout << "Чтобы пригласить игрока напишите invite и его логин  
через пробел" << std::endl;  
        }  
    } else if (tmp[0] == 'y') {  
        yes_no_message = "yes:" + std::to_string(pid);  
        send_message(main_socket, yes_no_message);  
        received_message = receive_message(main_socket);  
        if (received_message == "good") {  
            std::cout << "Чтобы пригласить игрока напишите invite и его логин  
через пробел" << std::endl;  
        } else {  
            std::cout << "Тебя уже хочет пригласить другой игрок, дождись его" <<  
std::endl;  
            pthread_create(&tid, NULL, check_invite, &pid);  
            std::this_thread::sleep_for(std::chrono::microseconds(100));  
            break;  
        }  
    }  
} else if (tmp == "Error") {  
    std::getline(ss, tmp, ':');  
    if (tmp == "NameAlreadyExist") {  
        std::cout << "ERROR: Это имя уже занято! Попробуйте другое" << std::endl;  
        login_stage = false;  
    }  
}
```

```

    } else {
        std::cin >> command;
        if (!move_on_flag) {
            received_message = receive_message(player_socket);
            send_message(player_socket, "do:nothing");
            move_on_flag = true;
        }
        if (command == "invite") {
            std::string invite_login;
            std::cin >> invite_login;
            std::cout << "Вы пригласили игрока с ником " + invite_login << std::endl;
            std::cout << "Ждем ответ..." << std::endl;
            std::string invite_cmd = "invite:" + std::to_string(PORT_ITER) + ":" +
invite_login;
            send_message(main_socket, invite_cmd);
            received_message = receive_message(main_socket);
            std::stringstream ss(received_message);
            std::getline(ss, tmp, ':');
            if (tmp == "accept") {
                std::cout << "Запрос принят!" << std::endl;
                break;
            } else if (tmp == "reject") {
                std::cout << "Запрос отклонен! " << std::endl;
            } else if (tmp == "Error") {
                std::getline(ss, tmp, ':');
                if (tmp == "SelfInvite") {
                    std::cout << "ERROR: Вы отправили запрос на игру самому себе.
Попробуйте снова" << std::endl;
                } else if (tmp == "LoginNotExist") {
                    std::cout << "ERROR: Игрока с таким ником не существует. Попробуйте
снова" << std::endl;
                }
            }
        } else {
            std::cout << "Вы ввели несуществующую команду. Попробуйте снова" << std::endl;
        }
    }
}

```

```

pthread_mutex_lock(&mutex);

std::this_thread::sleep_for(std::chrono::milliseconds(10));
if (PORT_ITER == 1) {
    std::cout << "Начинаем игру" << std::endl;
} else {
    std::cout << "Начинаем игру. Подождите, пока другой пользователь расставит корабли" <<
std::endl;
}

```

```

while(1) {
    std::string inside_msg;
    std::string send_msg;
    inside_msg = receive_message(player_socket);
    std::stringstream strs(inside_msg);
    strs >> tmp;
}

```

```

if (tmp == "Разместите") {
    if (inside_msg[inside_msg.size() - 1] == '0') {
        std::cout << inside_msg.substr(0, inside_msg.size() - 1) << std::endl;
    }
}

```

```

        int x, y;
        std::cin >> x >> y;
        send_msg = "coords:" + std::to_string(x) + ":" + std::to_string(y);
        send_message(player_socket, send_msg);
    } else if (tmp == "Введите") {
        std::cout << inside_msg << std::endl;
        std::string orientation;
        std::cin >> orientation;
        send_msg = "orientation:" + orientation;
        send_message(player_socket, send_msg);
    } else if (tmp == "board") {
        std::cout << inside_msg.substr(5, inside_msg.size()) << std::endl;
        send_message(player_socket, "ok");
    } else if (tmp == "Error") {
        std::cout << inside_msg << std::endl;
        send_message(player_socket, "ok");
    } else if (tmp == "your_turn") {
        send_message(player_socket, "ok");
        std::cout << "Ваш ход:" << std::endl;
        inside_msg = receive_message(player_socket);
        if (inside_msg == "shoot") {
            int x, y;
            std::cout << "Введите координаты выстрела (формат: x y):" << std::endl;
            std::cin >> x >> y;
            send_msg = "coords:" + std::to_string(x) + ":" + std::to_string(y);
            send_message(player_socket, send_msg);
            inside_msg = receive_message(player_socket);
            if (inside_msg == "shot") {
                std::cout << "Попадание!" << std::endl;
                send_message(player_socket, "ok");
            } else if (inside_msg == "miss") {
                std::cout << "Промех!" << std::endl;
                send_message(player_socket, "ok");
            }
        }
    }
} else if (tmp == "not_your_turn") {
    std::cout << "Ход соперника: " << std::endl;
    send_message(player_socket, "ok");
    inside_msg = receive_message(player_socket);
    if (inside_msg == "shot") {
        std::cout << "Вас подстрелили!" << std::endl;
        send_message(player_socket, "ok");
    } else if (inside_msg == "miss") {
        std::cout << "Противник промахнулся" << std::endl;
        send_message(player_socket, "ok");
    }
} else if (tmp == "win") {
    std::cout << "Вы выиграли!" << std::endl;
    send_message(player_socket, "ok");
    pthread_mutex_unlock(&mutex);
    return 0;
} else if (tmp == "lose") {
    std::cout << "Вы проиграли!" << std::endl;
    send_message(player_socket, "ok");
    pthread_mutex_unlock(&mutex);
    return 0;
}
}
}

```

## myMQ.hpp

```
#pragma once
```

```
#include <zmq.hpp>
#include <iostream>
```

```
const int BASE_PORT = 5555;
int PORT_ITER;
```

```
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;
    }
}
```

```
std::string recieved_message(static_cast<char*>(message.data()), message.size());
if (recieved_message.empty() || !ok) {
    return "";
}

return recieved_message;
}
```

```
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;
}
```

```
std::string GetConPort(int id) {
    return "tcp://127.0.0.1:" + std::to_string(id);
}
```

```
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));
}
```

## game.hpp

```
#pragma once
```

```
#include <iostream>
#include <vector>
#include <string>
```

```
#include "myMQ.hpp"
#include "player.hpp"
```

```
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 << "Игра началась!" << std::endl;

```

```

    std::string msg1;
    std::string msg2;
    pid_t pid = getpid();
    bool alive;

```

```

    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()) {
        alive = try_recv(first_player_pid, second_player_pid);
        if (!alive) {
            kill(first_player_pid, SIGTERM);
            kill(second_player_pid, SIGTERM);
            std::cout << "Игра завершена из-за убийства одного или обоих клиентов" <<
std::endl;
            kill(pid, SIGTERM);
        }
        if (turn % 2 == 0) {
            msg1 = "your_turn";
            msg2 = "not_your_turn";
            send_message(player1_socket, msg1);
            receive_message(player1_socket);
            send_message(player2_socket, msg2);
            receive_message(player2_socket);
            std::cout << "Ход игрока 1:" << std::endl;
            if (playerTurn(player1, player2, player1_socket, player2_socket)) {
                if (gameOver()) {
                    std::cout << "Победил игрок 1" << std::endl;
                    send_message(player1_socket, "win");
                    receive_message(player1_socket);
                    send_message(player2_socket, "lose");
                    receive_message(player2_socket);
                    break;
                }
                continue;
            }
        } else {
            turn += 1;
        }
    } else {
        std::cout << "Ход игрока 2:" << std::endl;
        msg1 = "your_turn";
        msg2 = "not_your_turn";
        send_message(player2_socket, msg1);
        receive_message(player2_socket);
        send_message(player1_socket, msg2);
        receive_message(player1_socket);
        if (playerTurn(player2, player1, player2_socket, player1_socket)) {
            if (gameOver()) {
                std::cout << "Победил игрок 2" << std::endl;
                send_message(player2_socket, "win");

```



```

        receive_message(player2_socket);
        send_message(player1_socket, "lose");
        receive_message(player1_socket);
        break;
    }
    continue;
}
else {
    turn += 1;
}
}
}
std::cout << "Игра завершена!" << std::endl;
}

```

```

bool gameOver() const {
    return allShipsDead(player1) || allShipsDead(player2);
}

```

```

bool allShipsDead(const Player& player) const {
    for (const auto& row : player.board) {
        for (char cell : row) {
            if (cell == '0') {
                return false;
            }
        }
    }
    return true;
}

```

```

bool playerTurn(Player& attacker, Player& defender, zmq::socket_t& attacker_socket,
zmq::socket_t& defender_socket) {
    bool shoot = false;
    std::string received_message;
    std::string tmp;
    int x, y;
    send_message(attacker_socket, "shoot");
    received_message = receive_message(attacker_socket);
    std::stringstream strs(received_message);
    std::getline(strs, tmp, ':');
    std::getline(strs, tmp, ':');
    std::cout << tmp << std::endl;
    x = std::stoi(tmp);
    std::getline(strs, tmp, ':');
    std::cout << tmp << std::endl;
    y = std::stoi(tmp);
}

```

```

if (isValidMove(x, y, defender)) {
    if (defender.board[x][y] == '0') {
        send_message(attacker_socket, "shooting");
        received_message = receive_message(attacker_socket);
        send_message(defender_socket, "shooting");
        received_message = receive_message(defender_socket);
    }
}

```

```

std::cout << "Попадание!" << std::endl;
defender.board[x][y] = 'X';
shoot = true;
return shoot;
} else {
    send_message(attacker_socket, "miss");
}

```

```

received_message = receive_message(attacker_socket);
send_message(defender_socket, "miss");
received_message = receive_message(defender_socket);

```

```

std::cout << "Помех!" << std::endl;
defender.board[x][y] = '*';
std::string board = defender.getBoard();
std::string clearBoard = defender.getClearBoard();

send_message(attacker_socket, "board" + clearBoard);
received_message = receive_message(attacker_socket);
send_message(defender_socket, "board" + board);
received_message = receive_message(defender_socket);
return shoot;
}

```

```

} else {
    std::cout << "Неверные координаты. Игрок потерял ход" << std::endl;
    send_message(attacker_socket, "miss");
    received_message = receive_message(attacker_socket);
    send_message(defender_socket, "miss");
    received_message = receive_message(defender_socket);
    return shoot;
}
}

```

```

bool isValidMove(int x, int y, const Player& defender) const {
    return x >= 0 && x < BOARD_SIZE && y >= 0 && y < BOARD_SIZE && (defender.board[x][y]
== ' ' || defender.board[x][y] == '0');
}
};

```

## player.hpp

```
#pragma once
```

```

#include <iostream>
#include <vector>
#include <string>
#include <signal.h>

```

```
#include "myMQ.hpp"
```

```
const int BOARD_SIZE = 10;
```

```

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, ' '));
    }

```

```

    void placeShips(zmq::socket_t& player_socket, pid_t first_player_pid, pid_t
second_player_pid) {
        pid_t pid = getpid();
        std::string msg;
        std::string tmp;

```

```

std::string orientation;
std::string received_message;
bool alive;
int x, y;
for (int i = 1; i <= 4; ++i) {
    for (int j = 4; j >= i; --j) {
        msg = "Введите ориентацию корабля";
        send_message(player_socket, msg);
        orientation = (receive_message(player_socket));
        orientation = orientation.substr(12, orientation.size());
        msg = "Разместите " + std::to_string(i) + " палубный корабль";
        bool valid_ship = true;
        std::vector<std::pair<int, int>> points;
        std::pair<int, int> prev_point = std::make_pair(-1, -1);
        for (int k = 0; k < i; ++k) {
            alive = try_recv(first_player_pid, second_player_pid);
            if (!alive) {
                kill(first_player_pid, SIGTERM);
                kill(second_player_pid, SIGTERM);
                std::cout << "Игра завершена из-за убийства одного или обоих клиентов"
<< std::endl;

                kill(pid, SIGTERM);
            }
            send_message(player_socket, msg + std::to_string(k));
            received_message = receive_message(player_socket);
            std::cout << "Получил запрос: " + received_message << std::endl;

```

```

std::stringstream strs(received_message);
std::getline(strs, tmp, ':');
if (tmp == "coords") {
    std::getline(strs, tmp, ':');
    x = std::stoi(tmp);
    std::getline(strs, tmp, ':');
    y = std::stoi(tmp);

```

```

        if (!((orientation == "V") or (orientation == "H"))) {
            send_message(player_socket, "Error : Такой ориентации нет");
            received_message = receive_message(player_socket);
            ++j;
            break;
        }
        if (!ValidPointCheck(x, y, prev_point, orientation)) {
            valid_ship = false;
        }
        prev_point.first = x;
        prev_point.second = y;
        points.push_back(std::make_pair(x, y));
    }
}
if (valid_ship) {
    for (auto& elem : points) {
        board[elem.first][elem.second] = 'O';
    }
    std::string boardState = "board";
    boardState += getBoard();
    send_message(player_socket, boardState);
    received_message = receive_message(player_socket);
} else {
    send_message(player_socket, "Error : Неверное местоположение корабля.
Попробуйте ещё раз");

```

```

        received_message = receive_message(player_socket);
        ++j;
    }
}
}
}

```

```

bool ValidPointCheck(int x, int y, std::pair<int, int>& prev_point, std::string&
orientation) const {
    if (x > 9 || x < 0 || y > 9 || y < 0) {
        return false;
    }
}

```

```

    if (prev_point.first == -1 && prev_point.second == -1) {
        if (!isEmptyAround(x, y)) {
            return false;
        }
    } else {
        if (orientation == "V") {
            if (prev_point.second != y || (abs(prev_point.first - x) > 1)) {
                return false;
            }
        }
        if (orientation == "H") {
            if (prev_point.first != x || (abs(prev_point.second - y) > 1)) {
                return false;
            }
        }
    }
    return true;
}

bool isEmptyAround(int x, int y) const {
    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 && board[i][j] != ' ')
        {
            return false;
        }
    }
    return true;
}

std::string getBoard() const {
    std::string result;
    std::string probel(1, ' ');
    result = "  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) {
            std::string brd(1, board[i][j]);
            result += brd + probel;
        }
        result += '\n';
    }
    result += '\n';
    return result;
}

```

```

std::string getClearBoard() const {
    std::string result;
}

```

```

        std::string space(1, ' ');
        result = "  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) {
                std::string brd(1, board[i][j]);
                if (brd == "0") {
                    result += space + space;
                } else {
                    result += brd + space;
                }
            }
            result += '\n';
        }
        result += '\n';
        return result;
    }
};

```

## server.cpp

```

#include <iostream>
#include <unistd.h>
#include <string>
#include <vector>
#include <sstream>
#include <signal.h>
#include <cassert>
#include <chrono>
#include <thread>
#include <exception>
#include <map>
#include <zmq.hpp>
#include <cstdlib>
#include <ctime>

#include "myMQ.hpp"
#include "player.hpp"
#include "game.hpp"

zmq::context_t context(3);
zmq::socket_t main_socket(context, ZMQ_REP);
int main() {
    zmq::socket_t first_player_socket(context, ZMQ_REQ);
    zmq::socket_t second_player_socket(context, ZMQ_REQ);
    main_socket.bind("tcp://*:5555");
    first_player_socket.bind("tcp://*:5556");
    second_player_socket.bind("tcp://*:5557");
    std::cout << "Сервер начал работу" << std::endl;
    std::map<int, std::string> login_map;
    int port_iter = 1;
    int invite_applicants_count = 0;
    bool alive;
    pid_t first_player_pid, second_player_pid, first_applicator_pid, second_applicator_pid;
    std::vector<std::string> yes_no_vector;
    while (true) {
        std::string received_message = receive_message(main_socket);
        std::cout << "На сервер поступил запрос: '" + received_message + "'" << std::endl;
        if (port_iter > 2) {
            alive = try_rcv(first_player_pid, second_player_pid);
            if (!alive) {
                kill(first_player_pid, SIGTERM);
            }
        }
    }
}

```

```

        kill(second_player_pid, SIGTERM);
        std::cout << "Игра завершена из-за убийства одного или обоих клиентов" <<
std::endl;
        return 0;
    }
}
std::stringstream ss(received_message);
std::string tmp;
std::getline(ss, tmp, ':');
if (tmp == "login") {
    if (port_iter > 2) {
        send_message(main_socket, "Error:TwoPlayersAlreadyExist");
    } else {
        std::getline(ss, tmp, ':');
        if (login_map[1] == tmp) {
            std::cout << "Игрок ввел занятое имя" << std::endl;
            send_message(main_socket, "Error:NameAlreadyExist");
        } else {
            std::cout << "Логин игрока номер " + std::to_string(port_iter) + ": " +
tmp << std::endl;
            std::string login = tmp;
            login_map[port_iter] = login;
            std::getline(ss, tmp, ':');
            if (port_iter == 1) {
                first_player_pid = pid_t(std::stoi(tmp));
            } else {
                second_player_pid = pid_t(std::stoi(tmp));
            }

            send_message(main_socket, "Ok:" + std::to_string(port_iter));
            port_iter += 1;
        }
    }
} else if (tmp == "yes" || tmp == "no") {
    std::string application = tmp;
    std::string reply_message;
    if (application == "yes" && invite_applicants_count == 0) {
        reply_message = "good";
        send_message(main_socket, reply_message);
        ++invite_applicants_count;
    } else {
        if (application == "yes" && invite_applicants_count > 0) {
            reply_message = "bad";
            send_message(main_socket, reply_message);
        }
    }
}
if (application == "no" && invite_applicants_count == 0) {
    if (yes_no_vector.size() == 0) {
        reply_message = "good";
        send_message(main_socket, reply_message);
    } else {
        reply_message = "bad";
        send_message(main_socket, reply_message);
    }
}
if (application == "no" && invite_applicants_count == 1) {
    reply_message = "good";
    send_message(main_socket, reply_message);
}
yes_no_vector.push_back(application);
std::getline(ss, tmp, ':');

```

```

        if (yes_no_vector.size() == 1) {
            first_applicator_pid = pid_t(std::stoi(tmp));
        } else {
            second_applicator_pid = pid_t(std::stoi(tmp));
            if (yes_no_vector[0] == "yes" && yes_no_vector[1] == "no") {
                if (first_applicator_pid == first_player_pid) {
                    send_message(first_player_socket, "move on");
                    receive_message(first_player_socket);
                } else {
                    send_message(second_player_socket, "move on");
                    receive_message(second_player_socket);
                }
            }
            if (yes_no_vector[0] == "yes" && yes_no_vector[1] == "yes") {
                if (first_applicator_pid == first_player_pid) {
                    send_message(first_player_socket, "move on");
                    receive_message(first_player_socket);
                } else {
                    send_message(second_player_socket, "move on");
                    receive_message(second_player_socket);
                }
            }
            if (yes_no_vector[0] == "no" && yes_no_vector[1] == "yes") {
                if (second_applicator_pid == first_player_pid) {
                    send_message(first_player_socket, "move on");
                    receive_message(first_player_socket);
                } else {
                    send_message(second_player_socket, "move on");
                    receive_message(second_player_socket);
                }
            }
            if (yes_no_vector[0] == "no" && yes_no_vector[1] == "no") {
                if (second_applicator_pid == first_player_pid) {
                    send_message(first_player_socket, "move on");
                    receive_message(first_player_socket);
                } else {
                    send_message(second_player_socket, "move on");
                    receive_message(second_player_socket);
                }
            }
        }
    }
} else if (tmp == "invite") {
    std::cout << "Обрабатываю приглашение на игру" << std::endl;
    std::this_thread::sleep_for(std::chrono::milliseconds(100));
    std::string invite_login;
    std::getline(ss, tmp, ':');
    int sender_id = std::stoi(tmp);
    std::getline(ss, invite_login, ':');

    if (invite_login == login_map[sender_id]) {
        std::cout << "Игрок пригласил сам себя" << std::endl;
        send_message(main_socket, "Error:SelfInvite");
    } else if (invite_login == login_map[2]) {
        std::cout << "Игрок " + login_map[1] + " пригласил в игру " + login_map[2] <<
std::endl;

        send_message(second_player_socket, "invite:" + login_map[1]);
        std::string invite_message = receive_message(second_player_socket);
        if (invite_message == "accept") {
            std::cout << "Игрок " + login_map[2] + " принял запрос " << std::endl;
            send_message(main_socket, invite_message);
            break;
        }
    }
}

```

```

    } else if (invite_message == "reject") {
        std::cout << "Игрок " + login_map[2] + " отклонил запрос. Игра окончена."
<< std::endl;

        kill(first_player_pid, SIGTERM);
        kill(second_player_pid, SIGTERM);
        return 0;
    }
} else if (invite_login == login_map[1]){
    std::cout << "Игрок " + login_map[2] + " пригласил в игру " + login_map[1] <<
std::endl;

    send_message(first_player_socket, "invite:" + login_map[2]);
    std::string invite_message = receive_message(first_player_socket);
    if (invite_message == "accept") {
        std::cout << "Игрок " + login_map[1] + " принял запрос " << std::endl;
        send_message(main_socket, invite_message);
        break;
    } else if (invite_message == "reject") {
        std::cout << "Игрок " + login_map[1] + " отклонил запрос. Игра окончена."
<< std::endl;

        kill(first_player_pid, SIGTERM);
        kill(second_player_pid, SIGTERM);
        return 0;
    }
} else {
    std::cout << "Ника " + invite_login + " отсутствует в базе" << std::endl;
    std::cout << "Отправляю ошибку игроку" << std::endl;
    std::this_thread::sleep_for(std::chrono::microseconds(100));

    send_message(main_socket, "Error:LoginNotExist");
}

std::getline(ss, tmp, ':');
}

}

std::cout << "Запускаю игру" << std::endl;
Game game;
game.play(first_player_socket, second_player_socket, first_player_pid, second_player_pid);
}

```

## Пример работы

The screenshot shows a Visual Studio Code editor with a C++ project. The main terminal window displays the game's output, including a 10x10 grid for ship placement and a 10x10 grid for the battle. The game is in progress, with the player's turn to move. The right sidebar shows the file explorer with folders for client and server builds.

```

File Edit Selection View Go Run Terminal Help
myMCU.cpp U game.hpp U client.cpp U .gitignore U server.cpp U player.hpp U X
Lab's-git > os_cp > src > player.hpp > Player > AroundCount(int, int) const
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Разместите 4 палубный корабль
2 7
3 7
4 7
5 7
0 1 2 3 4 5 6 7 8 9
0 0 0 0 0 0 0 0 0 0
1
2 0 0
3 0 0
4 0 0
5 0 0
6
7
8 0
9 0 0 0 0 0 0 0 0 0

Ваш ход:
Введите координаты выстрела (формат: x y):
5 0
Попадание!
Ваш ход:
Введите координаты выстрела (формат: x y):
0 0
Пропуск!
0 1 2 3 4 5 6 7 8 9
0
1
2
3
4
5 X
6
7
8
9

Ход соперника:
[]

Error: Неправильное местоположение корабля. Попробуйте ещё раз
Введите ориентацию корабля
Н
Разместите 4 палубный корабль
5 0
5 1
5 2
5 3
0 1 2 3 4 5 6 7 8 9
0 0 0 0 0 0 0 0 0 0
1 0 0
2
3 0 0 0 0 0 0 0 0
4
5 X 0 0 0 0 0 0 0 0
6
7 0 0
8 0 0
9 0 0 0 0 0 0 0 0

Ход соперника:
Вас подстрелили!
Ход соперника:
Противник промахнулся
0 1 2 3 4 5 6 7 8 9
0
1 0 0
2
3 0 0 0 0 0 0 0 0
4
5 X 0 0 0 0 0 0 0 0
6
7 0 0
8 0 0
9 0 0 0 0 0 0 0 0

Ваш ход:
Введите координаты выстрела (формат: x y):
[]

```



### **Вывод**

Курсовая работа помогла мне структурировать знания, полученные при выполнении предыдущих лабораторных работ, в особенности 5-7. Я лучше понял принципы и механизмы работы с очередями сообщений. В целом, мне понравилось работать над данным проектом, поскольку в 3 семестре он является одним из наиболее приближенных к реальным задачам программирования.