Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовой проект по курсу**

**«Операционные системы»**

Студент: Бурдинский Владислав Дмитриевич

Группа: М8О–203Б–23

Вариант: 3

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2024.

Цель курсового проекта

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.

**Код программы**

**Client.cpp**

// client.cpp

#include <iostream>

#include <string>

#include <zmq.hpp>

static void sendRequest(zmq::socket\_t &socket, const std::string &req) {

zmq::message\_t request(req.size());

memcpy((void\*)request.data(), req.data(), req.size());

socket.send(request, zmq::send\_flags::none);

}

static std::string recvReply(zmq::socket\_t &socket) {

zmq::message\_t reply;

socket.recv(&reply);

std::string rep\_str(static\_cast<char\*>(reply.data()), reply.size());

return rep\_str;

}

int main() {

zmq::context\_t context(1);

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

socket.connect("tcp://localhost:5555");

std::cout << "Введите ваш логин: ";

std::string login;

std::cin >> login;

{

std::string req = "LOGIN:" + login;

sendRequest(socket, req);

std::string rep\_str = recvReply(socket);

if (rep\_str.find("OK:") != 0) {

std::cout << "Ошибка входа: " << rep\_str << std::endl;

return 1;

} else {

std::cout << "Успешный вход: " << rep\_str << std::endl;

}

}

bool inGame = false;

bool gameStarted = false;

std::string currentGame;

while (true) {

if (!inGame) {

std::cout << "\nВы вне игры. Доступные действия:\n";

std::cout << "1) CREATE\_GAME\n";

std::cout << "2) JOIN\_GAME\n";

std::cout << "3) LIST\_GAMES\n";

std::cout << "4) INVITE\n";

std::cout << "5) QUIT (выйти из программы)\n";

int choice;

std::cin >> choice;

if (choice == 5) {

break;

}

std::string req;

if (choice == 1) {

std::cout << "Введите имя новой игры: ";

std::string gname;

std::cin >> gname;

req = "CREATE\_GAME:" + gname + "|" + login;

} else if (choice == 2) {

std::cout << "Введите имя игры для подключения: ";

std::string gname;

std::cin >> gname;

req = "JOIN\_GAME:" + gname + "|" + login;

} else if (choice == 3) {

req = "LIST\_GAMES:" + login;

} else if (choice == 4) {

std::cout << "Введите логин для приглашения: ";

std::string other;

std::cin >> other;

req = "INVITE:" + other + "|" + login;

} else {

std::cout << "Неизвестный выбор.\n";

continue;

}

sendRequest(socket, req);

std::string rep = recvReply(socket);

std::cout << "Ответ сервера: " << rep << std::endl;

if (rep.find("OK:GAME\_CREATED") == 0 || rep.find("OK:JOINED") == 0) {

inGame = true;

if (req.find("CREATE\_GAME:") == 0) {

currentGame = req.substr(std::string("CREATE\_GAME:").size());

currentGame = currentGame.substr(0, currentGame.find('|'));

} else if (req.find("JOIN\_GAME:") == 0) {

currentGame = req.substr(std::string("JOIN\_GAME:").size());

currentGame = currentGame.substr(0, currentGame.find('|'));

}

}

} else {

// Получаем статус игры

{

std::string req = "GAME\_STATUS:" + login;

sendRequest(socket, req);

std::string rep = recvReply(socket);

if (rep.find("OK:") == 0) {

if (rep.find("STARTED") != std::string::npos) {

gameStarted = true;

} else {

gameStarted = false;

}

std::cout << "\n--- Состояние игры ---\n" << rep << "\n----------------------\n";

} else {

std::cout << "Ошибка получения статуса: " << rep << std::endl;

}

}

if (!gameStarted) {

// Расставляем корабли

// Нам нужно 3 раза расставить корабли: 1x4, 2x3

std::cout << "\nИгра не началась. Вам нужно расставить 3 корабля.\n";

std::cout << "Введите 1 чтобы поставить корабль, 2 чтобы выйти из игры: ";

int choice;

std::cin >> choice;

if (choice == 2) {

inGame = false;

currentGame.clear();

continue;

}

if (choice == 1) {

std::cout << "Введите координаты корабля (startX startY endX endY): ";

int sx, sy, ex, ey;

std::cin >> sx >> sy >> ex >> ey;

std::string req = "SETUP\_SHIP:" + std::to\_string(sx) + "," + std::to\_string(sy) + "," +

std::to\_string(ex) + "," + std::to\_string(ey) + "|" + login + "|" + currentGame;

sendRequest(socket, req);

std::string rep = recvReply(socket);

std::cout << "Ответ сервера: " << rep << std::endl;

}

} else {

// Игра началась

std::cout << "\nИгра идет. Доступные действия:\n";

std::cout << "1) MOVE (сделать выстрел)\n";

std::cout << "2) GAME\_STATUS (посмотреть состояние)\n";

std::cout << "3) QUIT\_GAME (вернуться в главное меню)\n";

int choice;

std::cin >> choice;

if (choice == 3) {

inGame = false;

currentGame.clear();

continue;

} else if (choice == 2) {

std::string req = "GAME\_STATUS:" + login;

sendRequest(socket, req);

std::string rep = recvReply(socket);

std::cout << "Ответ сервера:\n" << rep << std::endl;

} else if (choice == 1) {

std::cout << "Введите координаты выстрела x y: ";

int x,y;

std::cin >> x >> y;

std::string req = "MOVE:" + std::to\_string(x) + "," + std::to\_string(y) + "|" + login + "|" + currentGame;

sendRequest(socket, req);

std::string rep = recvReply(socket);

std::cout << "Ответ сервера: " << rep << std::endl;

// Если это был WIN - игра окончена

if (rep.find("WIN") != std::string::npos) {

std::cout << "Вы победили! Игра окончена.\n";

inGame = false;

currentGame.clear();

gameStarted = false;

}

} else {

std::cout << "Неизвестный выбор.\n";

}

}

}

}

std::cout << "Выход.\n";

return 0;

}

server.cpp

// server.cpp

#include <iostream>

#include <string>

#include <unordered\_map>

#include <vector>

#include <algorithm>

#include <cassert>

#include <zmq.hpp>

#include <thread>

#include <mutex>

struct PlayerInfo {

std::string login;

bool inGame = false;

std::string currentGame;

};

enum class CellState {

EMPTY,

SHIP,

HIT,

MISS

};

// Конфигурация кораблей (для упрощения)

// 1 корабль длиной 4, 2 корабля длиной 3

// Сохраним оставшееся количество кораблей для расстановки

struct ShipConfig {

int ships4 = 1;

int ships3 = 2;

bool done() {

return ships4 == 0 && ships3 == 0;

}

};

struct Board {

static const int SIZE = 10;

CellState cells[SIZE][SIZE]; // Игровое поле

ShipConfig config;

int totalShipCells = 0; // кол-во клеток кораблей на поле

Board() {

for (int i=0; i<SIZE; i++) {

for (int j=0; j<SIZE; j++) {

cells[i][j] = CellState::EMPTY;

}

}

}

bool placeShip(int sx, int sy, int ex, int ey) {

if (!inRange(sx, sy) || !inRange(ex, ey)) return false;

int length = 0;

if (sx == ex) {

length = std::abs(ey - sy) + 1;

} else if (sy == ey) {

length = std::abs(ex - sx) + 1;

} else {

return false; // корабль по диагонали

}

if (length != 3 && length != 4) return false;

// Проверим конфигурацию

if (length == 4 && config.ships4 == 0) return false;

if (length == 3 && config.ships3 == 0) return false;

// Проверим что нет занятых клеток

int stepX = (ex == sx) ? 0 : ((ex > sx) ? 1 : -1);

int stepY = (ey == sy) ? 0 : ((ey > sy) ? 1 : -1);

int x = sx;

int y = sy;

for (int i=0; i<length; i++) {

if (cells[y][x] != CellState::EMPTY) {

return false;

}

x += stepX;

y += stepY;

}

// Ставим корабль

x = sx;

y = sy;

for (int i=0; i<length; i++) {

cells[y][x] = CellState::SHIP;

x += stepX;

y += stepY;

}

totalShipCells += length;

if (length == 4) {

config.ships4--;

} else {

config.ships3--;

}

return true;

}

bool allShipsSet() {

return config.done();

}

bool inRange(int x, int y) {

return (x >=0 && x<SIZE && y>=0 && y<SIZE);

}

bool isAllSunk() {

return totalShipCells == 0;

}

std::string printBoard() {

std::string out;

for (int i=0; i<SIZE; i++) {

for (int j=0; j<SIZE; j++) {

if (cells[i][j] == CellState::SHIP) out += "S ";

else if (cells[i][j] == CellState::HIT) out += "H ";

else if (cells[i][j] == CellState::MISS) out += "M ";

else out += ". ";

}

out += "\n";

}

return out;

}

};

struct GameRoom {

std::string name;

std::vector<std::string> players; // ровно 2 игрока

bool started = false;

std::unordered\_map<std::string, Board> boards;

int currentPlayerIndex = 0; // чей ход

bool allBoardsSet() {

if (players.size() < 2) return false;

for (auto &p : players) {

if (!boards[p].allShipsSet()) return false;

}

return true;

}

bool isPlayerTurn(const std::string &player) {

if (players.empty()) return false;

return players[currentPlayerIndex] == player;

}

void nextTurn() {

currentPlayerIndex = (currentPlayerIndex + 1) % players.size();

}

std::string getOpponent(const std::string &player) {

for (auto &p : players) {

if (p != player) return p;

}

return "";

}

};

class Server {

public:

void run() {

zmq::context\_t context(1);

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

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

while (true) {

zmq::message\_t request;

socket.recv(&request);

std::string req\_str(static\_cast<char\*>(request.data()), request.size());

std::string reply = handleRequest(req\_str);

zmq::message\_t reply\_msg(reply.size());

memcpy((void\*)reply\_msg.data(), reply.data(), reply.size());

socket.send(reply\_msg, zmq::send\_flags::none);

}

}

private:

std::mutex mtx;

std::unordered\_map<std::string, PlayerInfo> players; // login -> PlayerInfo

std::unordered\_map<std::string, GameRoom> games; // gamename -> GameRoom

std::string handleRequest(const std::string &request) {

// Команды:

// LOGIN:Player

// CREATE\_GAME:GameName|Player

// JOIN\_GAME:GameName|Player

// SETUP\_SHIP:startX,startY,endX,endY|Player|GameName

// MOVE:x,y|Player|GameName

// GAME\_STATUS:Player

// (LIST\_GAMES, INVITE - опционально, можно оставить как есть)

auto colonPos = request.find(':');

if (colonPos == std::string::npos) {

return "ERR:UNKNOWN\_FORMAT";

}

std::string command = request.substr(0, colonPos);

std::string data = request.substr(colonPos + 1);

if (command == "LOGIN") {

return handleLogin(data);

} else if (command == "CREATE\_GAME") {

return handleCreateGame(data);

} else if (command == "JOIN\_GAME") {

return handleJoinGame(data);

} else if (command == "SETUP\_SHIP") {

return handleSetupShip(data);

} else if (command == "MOVE") {

return handleMove(data);

} else if (command == "GAME\_STATUS") {

return handleGameStatus(data);

} else if (command == "LIST\_GAMES") {

return handleListGames(data);

} else if (command == "INVITE") {

return handleInvite(data);

} else {

return "ERR:UNKNOWN\_COMMAND";

}

}

std::string handleLogin(const std::string &login) {

std::lock\_guard<std::mutex> lock(mtx);

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

return "ERR:LOGIN\_ALREADY\_EXISTS";

}

PlayerInfo pi;

pi.login = login;

players[login] = pi;

return "OK:LOGGED\_IN";

}

std::string handleCreateGame(const std::string &data) {

// data: GameName|PlayerLogin

auto sep = data.find('|');

if (sep == std::string::npos) return "ERR:FORMAT";

std::string gameName = data.substr(0, sep);

std::string playerLogin = data.substr(sep+1);

std::lock\_guard<std::mutex> lock(mtx);

if (games.find(gameName) != games.end()) {

return "ERR:GAME\_EXISTS";

}

if (players.find(playerLogin) == players.end()) {

return "ERR:NO\_SUCH\_PLAYER";

}

GameRoom gr;

gr.name = gameName;

gr.players.push\_back(playerLogin);

gr.boards[playerLogin] = Board();

games[gameName] = gr;

players[playerLogin].inGame = true;

players[playerLogin].currentGame = gameName;

return "OK:GAME\_CREATED";

}

std::string handleJoinGame(const std::string &data) {

// data: GameName|PlayerLogin

auto sep = data.find('|');

if (sep == std::string::npos) return "ERR:FORMAT";

std::string gameName = data.substr(0, sep);

std::string playerLogin = data.substr(sep+1);

std::lock\_guard<std::mutex> lock(mtx);

auto gIt = games.find(gameName);

if (gIt == games.end()) {

return "ERR:NO\_SUCH\_GAME";

}

if (players.find(playerLogin) == players.end()) {

return "ERR:NO\_SUCH\_PLAYER";

}

if (gIt->second.players.size() >= 2) {

return "ERR:GAME\_FULL";

}

gIt->second.players.push\_back(playerLogin);

gIt->second.boards[playerLogin] = Board();

players[playerLogin].inGame = true;

players[playerLogin].currentGame = gameName;

return "OK:JOINED";

}

std::string handleSetupShip(const std::string &data) {

// data: startX,startY,endX,endY|Player|GameName

auto firstSep = data.find('|');

if (firstSep == std::string::npos) return "ERR:FORMAT";

std::string coords = data.substr(0, firstSep);

std::string rest = data.substr(firstSep+1);

auto secondSep = rest.find('|');

if (secondSep == std::string::npos) return "ERR:FORMAT";

std::string playerLogin = rest.substr(0, secondSep);

std::string gameName = rest.substr(secondSep+1);

int sx, sy, ex, ey;

{

auto csep1 = coords.find(',');

if (csep1 == std::string::npos) return "ERR:FORMAT";

std::string startX = coords.substr(0, csep1);

std::string rest2 = coords.substr(csep1+1);

auto csep2 = rest2.find(',');

if (csep2 == std::string::npos) return "ERR:FORMAT";

std::string startY = rest2.substr(0, csep2);

std::string rest3 = rest2.substr(csep2+1);

auto csep3 = rest3.find(',');

if (csep3 == std::string::npos) return "ERR:FORMAT";

std::string endX = rest3.substr(0, csep3);

std::string endY = rest3.substr(csep3+1);

sx = std::stoi(startX);

sy = std::stoi(startY);

ex = std::stoi(endX);

ey = std::stoi(endY);

}

std::lock\_guard<std::mutex> lock(mtx);

if (players.find(playerLogin) == players.end()) return "ERR:NO\_SUCH\_PLAYER";

auto gIt = games.find(gameName);

if (gIt == games.end()) return "ERR:NO\_SUCH\_GAME";

auto &gr = gIt->second;

if (std::find(gr.players.begin(), gr.players.end(), playerLogin) == gr.players.end()) return "ERR:PLAYER\_NOT\_IN\_GAME";

if (gr.started) return "ERR:GAME\_ALREADY\_STARTED";

auto &board = gr.boards[playerLogin];

if (!board.placeShip(sx, sy, ex, ey)) {

return "ERR:INVALID\_PLACEMENT";

}

// Если оба расставили все корабли

if (gr.players.size() == 2 && gr.allBoardsSet()) {

gr.started = true;

}

return "OK:SHIP\_PLACED";

}

std::string handleMove(const std::string &data) {

// data: x,y|Player|GameName

auto firstSep = data.find('|');

if (firstSep == std::string::npos) return "ERR:FORMAT";

std::string coords = data.substr(0, firstSep);

std::string rest = data.substr(firstSep+1);

auto secondSep = rest.find('|');

if (secondSep == std::string::npos) return "ERR:FORMAT";

std::string playerLogin = rest.substr(0, secondSep);

std::string gameName = rest.substr(secondSep+1);

int x,y;

{

auto csep = coords.find(',');

if (csep == std::string::npos) return "ERR:FORMAT";

std::string sx = coords.substr(0,csep);

std::string sy = coords.substr(csep+1);

x = std::stoi(sx);

y = std::stoi(sy);

}

std::lock\_guard<std::mutex> lock(mtx);

if (players.find(playerLogin) == players.end()) return "ERR:NO\_SUCH\_PLAYER";

auto gIt = games.find(gameName);

if (gIt == games.end()) return "ERR:NO\_SUCH\_GAME";

auto &gr = gIt->second;

if (std::find(gr.players.begin(), gr.players.end(), playerLogin) == gr.players.end()) return "ERR:PLAYER\_NOT\_IN\_GAME";

if (!gr.started) return "ERR:GAME\_NOT\_STARTED";

if (!gr.isPlayerTurn(playerLogin)) return "ERR:NOT\_YOUR\_TURN";

std::string opponent = gr.getOpponent(playerLogin);

if (opponent.empty()) return "ERR:NO\_OPPONENT";

auto &oppBoard = gr.boards[opponent];

if (!oppBoard.inRange(x,y)) {

gr.nextTurn();

return "ERR:OUT\_OF\_RANGE";

}

auto &cell = oppBoard.cells[y][x];

if (cell == CellState::HIT || cell == CellState::MISS) {

// Уже стреляли сюда

gr.nextTurn();

return "OK:MISS"; // повторный выстрел в ту же точку - промах

}

std::string result;

if (cell == CellState::SHIP) {

cell = CellState::HIT;

oppBoard.totalShipCells--;

if (oppBoard.isAllSunk()) {

result = "WIN";

gr.started = false;

} else {

result = "HIT";

}

} else {

cell = CellState::MISS;

result = "MISS";

}

if (result != "WIN") {

gr.nextTurn();

}

return "OK:" + result;

}

std::string handleGameStatus(const std::string &playerLogin) {

std::lock\_guard<std::mutex> lock(mtx);

auto pIt = players.find(playerLogin);

if (pIt == players.end()) return "ERR:NO\_SUCH\_PLAYER";

if (!pIt->second.inGame) return "ERR:PLAYER\_NOT\_IN\_GAME";

std::string gameName = pIt->second.currentGame;

auto gIt = games.find(gameName);

if (gIt == games.end()) return "ERR:NO\_SUCH\_GAME";

auto &gr = gIt->second;

std::string status = "OK:";

status += (gr.started ? "STARTED\n" : "WAITING\_FOR\_SETUP\n");

status += "Your board:\n";

status += gr.boards[playerLogin].printBoard();

status += "Current turn: " + gr.players[gr.currentPlayerIndex] + "\n";

return status;

}

std::string handleListGames(const std::string &login) {

std::lock\_guard<std::mutex> lock(mtx);

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

return "ERR:NO\_SUCH\_PLAYER";

}

std::string result = "OK:";

bool first = true;

for (auto &kv : games) {

if (!first) result += ",";

result += kv.first;

first = false;

}

return result;

}

std::string handleInvite(const std::string &data) {

// data: InvitedLogin|InviterLogin

auto sep = data.find('|');

if (sep == std::string::npos) return "ERR:FORMAT";

std::string invited = data.substr(0, sep);

std::string inviter = data.substr(sep+1);

std::lock\_guard<std::mutex> lock(mtx);

if (players.find(invited) == players.end()) return "ERR:INVITED\_NOT\_FOUND";

if (players.find(inviter) == players.end()) return "ERR:INVITER\_NOT\_FOUND";

return "OK:INVITE\_SENT";

}

};

int main() {

Server s;

s.run();

return 0;

}

**Пример работы**

**(base) vladislavburdinskij@MacBook-Pro-Vladislav build % ./client**

**Введите ваш логин: 123**

**Успешный вход: OK:LOGGED\_IN**

**Вы вне игры. Доступные действия:**

**1) CREATE\_GAME**

**2) JOIN\_GAME**

**3) LIST\_GAMES**

**4) INVITE**

**5) QUIT (выйти из программы)**

**1**

**Введите имя новой игры: 444**

**Ответ сервера: OK:GAME\_CREATED**

**--- Состояние игры ---**

**OK:WAITING\_FOR\_SETUP**

**Your board:**

**. . . . . . . . . .**

**. . . . . . . . . .**

**. . . . . . . . . .**

**. . . . . . . . . .**

**. . . . . . . . . .**

**. . . . . . . . . .**

**. . . . . . . . . .**

**. . . . . . . . . .**

**. . . . . . . . . .**

**. . . . . . . . . .**

**Current turn: 123**

**----------------------**

**Игра не началась. Вам нужно расставить 3 корабля.**

**Введите 1 чтобы поставить корабль, 2 чтобы выйти из игры: 1**

**Введите координаты корабля (startX startY endX endY): 1 1 4 1**

**Ответ сервера: OK:SHIP\_PLACED**

**--- Состояние игры ---**

**OK:WAITING\_FOR\_SETUP**

**Your board:**

**. . . . . . . . . .**

**. S S S S . . . . .**

**. . . . . . . . . .**

**. . . . . . . . . .**

**. . . . . . . . . .**

**. . . . . . . . . .**

**. . . . . . . . . .**

**. . . . . . . . . .**

**. . . . . . . . . .**

**. . . . . . . . . .**

**Current turn: 123**

**----------------------**

**Игра не началась. Вам нужно расставить 3 корабля.**

**Введите 1 чтобы поставить корабль, 2 чтобы выйти из игры:**

**Вывод**

В ходе работы над курсовым проектом согласно назначенному варианту задания я реализовал игру «Морской бой» на клиент-серверной архитектуре. Общение между клиентами и сервером происходит с помощью очереди сообщений ZMQ, поддерживается возможность создания игры, расстановки кораблей, стрельбы и завершение игры в случае уничтожения всех кораблей.