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-203Б-23

Студент: Кириллова Е.К.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: _____

Москва, 2024

Постановка задачи

Консоль-серверная игра. Необходимо написать консоль-серверную игру. Необходимо написать 2 программы: сервер и клиент. Сначала запускается сервер, а далее клиенты соединяются с сервером. Сервер координирует клиентов между собой. При запуске клиента игрок может выбрать одно из следующих действий (возможно больше, если предусмотрено вариантом):

- Создать игру, введя ее имя
- Присоединиться к одной из существующих игр по имени игры

10. «Быки и коровы» (угадывать необходимо числа). Общение между сервером и клиентом необходимо организовать при помощи очередей сообщений (например, ZeroMQ). При создании каждой игры необходимо указывать количество игроков, которые будут участвовать. То есть угадывать могут несколько игроков. Должна быть реализована функция поиска игры, то есть игрок пытается войти в игру не по имени, а просто просит сервер найти ему игру.

Описание работы программы

Принцип работы программы заключается в следующем: 1. Клиентская часть (main.cpp): Пользователь может выбрать одно из нескольких действий: создать игру, подключиться к существующей или найти игру. Когда игрок решает создать игру, программа отправляет сообщение на сервер с просьбой создать игру с указанным названием и количеством игроков. После этого клиент ожидает других игроков. Когда игрок подключается к существующей игре, программа отправляет запрос на сервер, и если игра существует, она сообщает о подключении и ждет, пока все игроки присоединятся. Игра начинается, когда все игроки подключены. Затем игроки начинают угадывать число (игра типа "Быки и коровы"), и каждый игрок по очереди вводит свои предположения, пока не угадает правильное число. Игра завершена, когда один из игроков находит правильное число (4 быка). После завершения игры клиент получает информацию о победителе. 2. Серверная часть (server.cpp): Сервер управляет играми и игроками. Он принимает запросы от клиентов на создание игры, подключение к игре и другие действия. Когда клиент отправляет запрос на создание игры, сервер проверяет, существует ли игра с таким именем. Если нет, сервер создает новую игру, добавляет игрока в игру и ждет, пока не соберется нужное количество игроков. Когда все игроки подключены, сервер публикует сообщение о готовности игры, отправляя эту информацию всем участникам через сокет типа PUB. Игры работают с числовым кодом, сгенерированным сервером, который игроки пытаются угадать. Сервер отслеживает состояние игры и сообщает всем игрокам, кто победил. Основные взаимодействия: Сокеты: zmq::socket_t типа req используется для отправки запросов от клиента на сервер и получения ответов. zmq::socket_t типа sub используется для подписки на обновления игры, например, для получения сообщений о готовности игры или о победителе. zmq::socket_t типа pub используется для публикации сообщений о состоянии игры, например, когда игра готова или когда есть победитель. Логика игры: Игра — это классическая игра "Быки и коровы", где игроки пытаются угадать 4-значное число, получая подсказки о количестве быков (правильных цифр в правильных позициях) и коров (правильных цифр, но не в тех позициях). Как только кто-то угадывает число, он выигрывает, и сервер информирует об этом всех участников. Важные аспекты: Для корректной работы используются механизмы ожидания и таймаутов.

Код программы

main.cpp:

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

```
#include <sys/types.h>
#include <unistd.h>
#include <string>
#include <algorithm>
#include <thread>
#include <chrono>


int flag = 0;
std::string winner;
std::vector<std::string> waiter(int mode,int& current_players, int& all_players, zmq::socket_t& sock_sub,
zmq::message_t& topic, zmq::message_t& payload,zmq::message_t& game_number ) {
	if (payload.to_string().length() != 0 || mode == 1) {
		for (int i = current_players; i < all_players; i++) {
			sock_sub.recv(topic);
			sock_sub.recv(payload);
			if (payload.to_string().length() == 0) {
				break;
			}
			else {
				current_players += 1;
				std::cout << "Connected: " << payload.to_string() << "! " << current_players << "/"
<< all_players << "\n";
			}
		}
		int timeout = 180000;
		sock_sub.setsockopt(ZMQ_RCVTIMEO, &timeout, sizeof(timeout));
		sock_sub.recv(topic);
		sock_sub.recv(payload);
		sock_sub.recv(game_number);
		std::cout << "game number(test)"<< game_number.to_string() << "\n";
		std::string gamenumber = game_number.to_string();
		std::vector<std::string> v{ payload.to_string(),gamenumber };
		return v;
	}
	else {
			std::string s =  "timeout";
			std::vector<std::string> v{s};
			return  v;
	}
}
std::string getGameName(std::string& str) {
   std::string st = "Connected to game: ";
   int n = st.length();
   std::string num;
   int i = n;
   while (str[i] != ' ') {
```

```
        num += str[i];
        ++i;
    }
    return num;
}
void waiting(zmq::socket_t& sub,std::string &gamename, int& realpid) {
        sub.setsockopt(ZMQ_SUBSCRIBE, gamename.c_str(), gamename.size());
        zmq::message_t topic;
        zmq::message_t payload;
        zmq::message_t winner_name;
        sub.recv(topic);
        sub.recv(payload);
        sub.recv(winner_name);
        winner = winner_name.to_string();
        if (payload.to_string() != std::to_string(realpid)) {

                flag += 1;
    }
        else if (payload.to_string() == std::to_string(realpid)) {
                flag = -1;
        }
}

int  checknumber(std::string& inp,std::string &gamenumber) {
        int cows = 0;
        int bulls = 0;
        for (int i = 0; i < 4; i++) {
                for (int j = 0; j < 4; j++) {
                        if (inp[j] == gamenumber[i] && j == i)
                                bulls += 1;
                        if (inp[j] == gamenumber[i] && j != i) {
                                cows += 1;
                        }
                }
        }
        if (bulls == 4) {
                return 1;
        }
        else {
                std::cout << "bulls: " << bulls << " " << "cows: " << cows << "\n";
                return -1;
        }
}
int game(std::string& gamenumber, zmq::socket_t& sub, zmq::socket_t& req,std::string& gamename) {
        int realpid = getpid();
        std::thread sender_thread(waiting,std::ref(sub),std::ref(gamename),std::ref(realpid));
```

```
	int win_condition = 0;
	while (flag != 1) {
		std::cout << "input number: ";
		std::string inp;
		std::cin >> inp;
		if (checknumber(inp, gamenumber) == 1) {
			win_condition += 1;
			break;
		}
	}
	if (win_condition == 1) {
		std::string message = "4 " + gamename + " "+ std::to_string(realpid);
		zmq::message_t m_out(message);
		req.send(m_out, zmq::send_flags::none);
		zmq::message_t m_in_topic;
		zmq::message_t m_in_payload;
		req.recv(m_in_topic);
		req.recv(m_in_payload);
	}
	std::this_thread::sleep_for(std::chrono::milliseconds(1000));
	if (flag == -1) {
		std::cout << "Your win\n";
	}
	else {
		std::cout << "You lose\n";
		std::cout << "Winner is: " << winner << "\n";
	}
	sender_thread.join();
	flag = 0;
	return 0;
}

#ifndef UNIT_TEST
int main() {
	zmq::context_t ctx;
	zmq::socket_t sock0(ctx, zmq::socket_type::req);
	sock0.connect("tcp://127.0.0.1:5555");
	zmq::socket_t sock_sub(ctx, zmq::socket_type::sub);
	sock_sub.connect("tcp://127.0.0.1:5545");
	std::string player_name;
	std::cout << "Input your player name\n";
	std::cin >> player_name;
	std::cout << "Your name is: " << player_name << "\n";
	while (1) {
		std::cout << "1.Create game 2.Connect to the game 3.Find game \n";
		int c;
```

```
		std::cin >> c;
		if (c == 1) {
			std::cout << "Input game name\n";
			std::string gamename;
			std::cin >> gamename;
			sock_sub.setsockopt(ZMQ_SUBSCRIBE, gamename.c_str(), gamename.size());
			zmq::message_t topic;
			zmq::message_t payload;
			zmq::message_t game_number;
			zmq::message_t m_in_topic;
			zmq::message_t m_in_payload;
			std::string number_of_players;
			std::cout << "cin number of players\n";
			std::cin >> number_of_players;
			int my_pid = getpid();
			std::string message = "1 " + gamename + " " + std::to_string(my_pid) + " " + player_name + "
" + number_of_players;
			zmq::message_t m_out(message);
			sock0.send(m_out, zmq::send_flags::none);
			sock0.recv(m_in_topic);
			sock0.recv(m_in_payload);
			int code = 0;
			int topico = stoi(m_in_topic.to_string());
			if (topico == code) {
				std::cout << "waiting for other people in lobby...\n";
				int  timeout = 180000;
				sock_sub.setsockopt(ZMQ_RCVTIMEO, &timeout, sizeof(timeout));
				int h = stoi(number_of_players);
				int current_players = 1;
				std::vector<std::string> v = waiter(1,current_players, h, sock_sub, topic, payload,
game_number);
				if (v[0] == "ready"){
					std::cout << "the game is ready\n";
					std::cout << "Game starting\n";
					game(v[1], sock_sub, sock0, gamename);
				}
				else {
					std::cout << "error timeout\n";
					std::cout << v[0];
				}
			}
			else {
				std::cout << m_in_payload.to_string() << "\n";
			}
		}
		if (c == 2) {
			std::cout << "cin name of the game\n";
```

```
                std::string gamename;
                std::cin >> gamename;
            sock_sub.setsockopt(ZMQ_SUBSCRIBE, gamename.c_str(), gamename.size());
                zmq::message_t topic;
                zmq::message_t payload;
                zmq::message_t game_number;
                zmq::message_t m_in_topic;
                zmq::message_t m_in_payload;
                int my_pid = getpid();
                std::string message = "2 " + gamename + " " + player_name+ " " + std::to_string(my_pid);
                zmq::message_t m_out(message);
                sock0.send(m_out, zmq::send_flags::none);
                sock0.recv(m_in_topic);
                sock0.recv(m_in_payload);
                int code = 0;
                int topico = stoi(m_in_topic.to_string());
                if (topico == code) {
                        std::cout << "waiting for other people\n";
                        std::string new_message = m_in_payload.to_string();
                        char str1 = new_message[new_message.length() - 3];
                        char str2 = new_message[new_message.length() - 1];
                        int current_players = str1 - '0';
                        int all_players = str2 - '0';
                        int timeout = 180000;
                        sock_sub.setsockopt(ZMQ_RCVTIMEO, &timeout, sizeof(timeout));
                        sock_sub.recv(topic);
                        sock_sub.recv(payload);
                        std::vector<std::string> v = waiter(0,current_players, all_players, sock_sub, topic,
payload, game_number);
                        if (v[0] == "ready") {
                                std::cout << "the game is ready\n";
                                std::string gamenumber = game_number.to_string();
                                game(v[1], sock_sub, sock0, gamename);
                        }
                        else {
                                std::cout << "error timeout\n";
                        }
                }
                else {
                        std::cout << m_in_payload << "\n";
                }
        }
        if (c == 3) {
                std::cout << "Waitging for the game\n";
                int my_pid = getpid();
                std::string message = "3 " + player_name + " " + std::to_string(my_pid);
```

```
                        zmq::message_t m_out(message);
                        sock0.send(m_out, zmq::send_flags::none);
                        zmq::message_t m_in_topic;
                        zmq::message_t m_in_payload;
                        sock0.recv(m_in_topic);
                        sock0.recv(m_in_payload);
                        int code = 0;
                        int topico = stoi(m_in_topic.to_string());
                        if (topico == code) {
                                std::cout << m_in_payload.to_string() << "\n";
                                std::cout << "waiting for other people\n";
                                std::string message_in = m_in_payload.to_string();
                                std::string gamename = getGameName(message_in);
                                std::cout << gamename;
                                sock_sub.setsockopt(ZMQ_SUBSCRIBE, gamename.c_str(), gamename.size());
                                zmq::message_t topic;
                                zmq::message_t payload;
                                zmq::message_t game_number;
                                std::string new_message = m_in_payload.to_string();
                                char str1 = new_message[new_message.length() - 3];
                                char str2 = new_message[new_message.length() - 1];
                                int current_players = str1 - '0';
                                int all_players = str2 - '0';
                                int timeout = 180000;
                                sock_sub.setsockopt(ZMQ_RCVTIMEO, &timeout, sizeof(timeout));
                                sock_sub.recv(topic);
                                sock_sub.recv(payload);
                                std::vector<std::string> v = waiter(0, current_players, all_players, sock_sub, topic,
payload, game_number);
                                if (v[0] == "ready") {
                                        std::cout << "the game is ready\n";
                                        game(v[1], sock_sub, sock0, gamename);
                                }
                                else {
                                        std::cout << "error\n";
                                }
                        }
                        else {
                                std::cout << m_in_payload.to_string() << "\n";
                        }
                }
                if (c == 4) {
                        exit(1);
                }
        }
```

```
	return 0;

}
#endif // UNIT_TEST
```

server.cpp:
```
#include <iostream>
#include "zmq.hpp"
#include <sys/types.h>
#include <unistd.h>
#include <vector>
#include <string>
#include <algorithm>
#include <ctime>


struct Player {
	int player_pid;
	std::string player_name;
};
class Game {
public:
	std::string game_id;
	std::vector<Player>players;
	int players_quantity;
	std::string number;
	int current_players;
	int cond;
	Game()
	{
		game_id;
		players;
		players_quantity = 0;
		current_players = 0;
		cond = 0;
		std::string number = "";
	}
	void generate_number() {
		srand(static_cast<unsigned int>(time(0)));
		int a = (rand() % 10);
		int b = (rand() % 10);
		while (a == b) {
			b = (rand() % 10);
		}
		int c = (rand() % 10);
```

```
		while (a == c || b == c) {
			c = (rand() % 10);
		}
		int d = (rand() % 10);
		while (a == d || b == d || c == d) {
			d = (rand() % 10);
		}
		number = std::to_string(a) + std::to_string(b) + std::to_string(c) + std::to_string(d);
	}
};
std::string getName(std::string& str) {
	int n = 2;
	std::string name;
	while (str[n] != ' ') {
		name += str[n];
		n += 1;
	}
	return name;
}
int getPid(std::string& str,int n) {
	std::string pid;
	for (int i = n; i < str.length(); i++) {
		if (str[i] == ' ')
			break;
		pid += str[i];
	}
	int user_pid = stoi(pid);
	return user_pid;
}
int getNum(std::string& str) {
	std::string num;
	int i = str.length();
	while (str[i] != ' ') {
		num += str[i];
		--i;
	}
	(std::reverse(num.begin(), num.end()));
	int res = stoi(num);
	return res;

}
std::string getPlayer(std::string& str,int n) {
	std::string player_name;
	while (str[n] != ' ') {
		player_name += str[n];
		n += 1;
```

```
        }
        return player_name;
}
bool findgame(std::vector<Game>& g, Player player, std::string name) {
        for (int i = 0; i < g.size(); i++) {
                if (g[i].game_id == name) {
                        if (g[i].current_players + 1 <= g[i].players_quantity) {
                                g[i].players.push_back(player);
                                g[i].current_players += 1;
                                return true;
                        }
                }
        }
        return false;
}
bool checkgame(std::vector<Game>& g, std::string name) {
        for (int i = 0; i < g.size(); i++) {
                if (g[i].game_id == name) {
                        if (g[i].current_players  == g[i].players_quantity) {
                                return true;
                        }
                }
        }
        return false;
}
bool checkname(std::string& gamename, std::vector<Game>& g) {
        for (int i = 0; i < g.size(); i++) {
                if (g[i].game_id == gamename) {
                        return false;
                }
        }
        return true;
}
void deletegame(std::string& gamename, std::vector<Game>& g) {
    auto it = std::find_if(g.begin(), g.end(), [&gamename](const Game& game) {
        return game.game_id == gamename;
    });
    if (it != g.end()) {
        g.erase(it);
    }
}
void sending(zmq::socket_t &sock,std::string rep,int code) {
        if (code == 0) {
                zmq::message_t reply_topic("0");
                zmq::message_t reply(rep);
                sock.send(reply_topic, zmq::send_flags::sndmore);
```

```
                sock.send(reply, zmq::send_flags::none);
        }
        if (code != 0) {
                zmq::message_t reply_topic(std::to_string(code));
                zmq::message_t reply(rep);
                sock.send(reply_topic, zmq::send_flags::sndmore);
                sock.send(reply, zmq::send_flags::none);
        }

}
std::string getWinner(std::vector<Game>& g, int& win_pid) {
        std::string winner;
        for (int i = 0; i < g.size(); i++) {
                for (int j = 0; j < g[i].players.size(); j++) {
                        if (g[i].players[j].player_pid == win_pid) {
                                winner = g[i].players[j].player_name;
                        }
                }
        }
        return winner;
}
std::string getCurrentstring(std::string& name, std::vector<Game>& g) {
        std::string current_players;
        for (int i = 0; i < g.size(); i++) {
                if (g[i].game_id == name) {
                        for (int j = 0; j < g[i].players.size() ; j++) {
                                current_players += g[i].players[j].player_name + " ";
                        }
                }
        }
        return current_players;
}
std::string getCurrentnumber(std::string& name, std::vector<Game>& g) {
        std::string current_players;
        for (int i = 0; i < g.size(); i++) {
                if (g[i].game_id == name) {
                        current_players = std::to_string(g[i].current_players) + "/" +
std::to_string(g[i].players_quantity);

                }
        }
        return current_players;
}
int check_game_condition(std::vector<Game>& g, std::string& name) {
        for (int i = 0; i < g.size(); i++) {
                if (g[i].game_id == name) {
                        if (g[i].cond == 0) {
```

```
                                g[i].cond = 1;
                                return 0;
                        }
                }
        }
                return 1;
}

#ifndef UNIT_TEST
int main() {
        zmq::context_t ctx;
        zmq::socket_t sock_rep(ctx,zmq::socket_type::rep);
        sock_rep.bind("tcp://127.0.0.1:5555");
        zmq::socket_t sock_pub(ctx, zmq::socket_type::pub);
        sock_pub.bind("tcp://127.0.0.1:5545");
        std::vector<Game>games;
        while (1) {
                zmq::message_t m_in;
                sock_rep.recv(m_in);
                std::string message = m_in.to_string();
                std::cout << message;
                if (message[0] == '1') {
                        int user_pid;
                        std::string name;
                        int n = 2;
                        name = getName(message);
                        n += name.length() + 1;
                        std::string player_name;
                        user_pid = getPid(message, n);
                        n += std::to_string(user_pid).length()+1;
                        player_name = getPlayer(message, n);
                        int num_player;
                        num_player = getNum(message);
                        if (checkname(name, games) == true) {
                                Game game;
                                Player player;
                                game.game_id = name;
                                player.player_name = player_name;
                                player.player_pid = user_pid;
                                game.players.push_back(player);
                                game.players_quantity = num_player;
                                game.current_players += 1;
                                game.cond = 0;
                                game.generate_number();
                                std::string gamenumber = game.number;
                                games.push_back(game);
```

```
                        std::string reply = "Game: " + name + " created!";
                        sending(sock_rep, reply,0);
                        if (game.players_quantity == game.current_players) {
                                std::string str = "ready";
                                zmq::message_t topic(name);
                                zmq::message_t payload(str);
                                zmq::message_t game_number(gamenumber);
                                sock_pub.send(topic, zmq::send_flags::sndmore);
                                sock_pub.send(payload, zmq::send_flags::sndmore);
                                sock_pub.send(game_number, zmq::send_flags::none);
                        }
                }
                else {
                        std::string reply = "Game already exists. Please choose other name.";
                        sending(sock_rep, reply,1);
                }
        }
        if (message[0] == '2') {
                int user_pid;
                std::string name;
                name = getName(message);
                std::string player_name;
                player_name = getPlayer(message, 2 + name.length() + 1);
                user_pid = getPid(message, 2 + name.length() + 1+player_name.length()+1);
                Player player;
                player.player_name = player_name;
                player.player_pid = user_pid;
                if (findgame(games, player, name) == true) {
                        std::string reply;
                        std::string current_players = getCurrentstring(name, games);
                        std::string current_number = getCurrentnumber(name, games);
                        reply = "Connected to game:" + name + " in lobby now: " + current_players +
current_number;
                        sending(sock_rep,reply,0);
                        zmq::message_t  top1(name);
                        zmq::message_t pay1(player_name);
                        sock_pub.send(top1, zmq::send_flags::sndmore);
                        sock_pub.send(pay1, zmq::send_flags::none);
                        if (checkgame(games, name) == true) {
                                std::string str = "ready";
                                zmq::message_t topic(name);
                                zmq::message_t payload(str);
                                std::string gamenumber;
                                for (int i = 0; i < games.size(); i++) {
                                        if (games[i].game_id == name) {
                                                gamenumber = games[i].number;
```

```
                                        }
                                }
                                zmq::message_t game_number(gamenumber);
                                sock_pub.send(topic, zmq::send_flags::sndmore);
                                sock_pub.send(payload, zmq::send_flags::sndmore);
                                sock_pub.send(game_number, zmq::send_flags::none);
                        }
                }
                else {
                        std::string reply;
                        reply = "Game not found :(";
                        sending(sock_rep, reply,1);
                }

        }
        if (message[0] == '3') {
                std::string player_name = getPlayer(message, 2);
                int user_pid = getPid(message, 2+player_name.length()+1);
                Player player;
                player.player_name = player_name;
                player.player_pid = user_pid;
                std::string gamename;
                std::string gamenumber;
                for (int i = 0; i < games.size(); i++) {
                        if (games[i].current_players < games[i].players_quantity) {
                                gamename = games[i].game_id;
                                gamenumber = games[i].number;
                                break;
                        }

                }
                if (findgame(games, player, gamename) == true) {
                        std::string reply;
                        std::string current_players = getCurrentstring(gamename, games);
                        std::string current_number = getCurrentnumber(gamename, games);
                        reply = "Connected to game: " + gamename+" "+current_players + current_number;;
                        sending(sock_rep, reply,0);
                        sleep(3);
                        zmq::message_t  top1(gamename);
                        zmq::message_t pay1(player_name);
                        sock_pub.send(top1, zmq::send_flags::sndmore);
                        sock_pub.send(pay1, zmq::send_flags::none);
                        if (checkgame(games, gamename) == true) {
                                std::string str = "ready";
                                zmq::message_t topic(gamename);
                                zmq::message_t payload(str);
```

```
                                        sleep(3);
                                        zmq::message_t game_number(gamenumber);
                                        sock_pub.send(topic, zmq::send_flags::sndmore);
                                        sock_pub.send(payload, zmq::send_flags::sndmore);
                                        sock_pub.send(game_number, zmq::send_flags::none);
                                }
                        }
                        else {
                                std::string reply;
                                reply = "Game not found :(";
                                sending(sock_rep, reply,1);
                        }
                }
                if (message[0] == '4') {
                        std::string gamename = getName(message);
                                int win_pid = getPid(message, gamename.length() + 3);
                                std::string reply = "ok";
                                sending(sock_rep, reply, 0);
                                if (check_game_condition(games,gamename)==0) {
                                zmq::message_t topic(gamename);
                                std::string pid = std::to_string(win_pid);
                                zmq::message_t payload(pid);
                                std::string winner = getWinner(games, win_pid);
                                zmq::message_t payload2(winner);
                                sock_pub.send(topic, zmq::send_flags::sndmore);
                                sock_pub.send(payload, zmq::send_flags::sndmore);
                                sock_pub.send(payload2, zmq::send_flags::none);
                                deletegame(gamename, games);
                                }
                }
        }
        return 0;
}
#endif // UNIT_TEST
```

ТЕСТЫ

test.cpp:

```
#include <gtest/gtest.h>
#include "zmq.hpp"
#include <thread>
#include <chrono>
#include <vector>
#include <string>
#include <algorithm>
```

```
#include <ctime>

struct Player {
    int player_pid;
    std::string player_name;
};

struct Game {
    std::string game_id;
    std::vector<Player> players;
    int players_quantity;
    std::string number;
    int current_players;
    int cond;
    Game() : players_quantity(0), current_players(0), cond(0) {}
};

extern int checknumber(std::string& inp, std::string& gamenumber);
extern std::string getGameName(std::string& str);
extern std::string getWinner(std::vector<Game>& g, int& win_pid);
extern std::string getCurrentstring(std::string& name, std::vector<Game>& g);
extern std::string getCurrentnumber(std::string& name, std::vector<Game>& g);
extern bool findgame(std::vector<Game>& g, Player player, std::string name);
extern bool checkgame(std::vector<Game>& g, std::string name);
extern void deletegame(std::string& gamename, std::vector<Game>& g);
extern int check_game_condition(std::vector<Game>& g, std::string& name);

TEST(GameLogicTest, CheckNumberTest) {
    std::string gamenumber = "1234";

    std::string correct_guess = "1234";
    EXPECT_EQ(checknumber(correct_guess, gamenumber), 1);

    std::string incorrect_guess = "5678";
    EXPECT_EQ(checknumber(incorrect_guess, gamenumber), -1);

    std::string partial_guess = "1243";
    EXPECT_EQ(checknumber(partial_guess, gamenumber), -1);

    std::string all_cows = "4321";
    EXPECT_EQ(checknumber(all_cows, gamenumber), -1);
}

TEST(GameLogicTest, GetWinnerTest) {
    std::vector<Game> games;
    Game game;
    game.game_id = "game1";
```

```
    Player player1 = {123, "Alice"};
    Player player2 = {456, "Bob"};
    game.players.push_back(player1);
    game.players.push_back(player2);

    games.push_back(game);

    int win_pid = 123;
    EXPECT_EQ(getWinner(games, win_pid), "Alice");

    win_pid = 456;
    EXPECT_EQ(getWinner(games, win_pid), "Bob");

    win_pid = 999;
    EXPECT_EQ(getWinner(games, win_pid), "");
}

TEST(GameLogicTest, GetCurrentstringTest) {
    std::vector<Game> games;
    Game game;
    game.game_id = "game1";

    Player player1 = {123, "Alice"};
    Player player2 = {456, "Bob"};
    game.players.push_back(player1);
    game.players.push_back(player2);

    games.push_back(game);

    std::string expected = "Alice Bob ";
    std::string game1 = "game1";
    EXPECT_EQ(getCurrentstring(game1, games), expected);

    std::string nonexistent = "nonexistent";
    EXPECT_EQ(getCurrentstring(nonexistent, games), "");
}

TEST(GameLogicTest, GetCurrentnumberTest) {
    std::vector<Game> games;
    Game game;
    game.game_id = "game1";
    game.players_quantity = 4;
    game.current_players = 2;
    games.push_back(game);

    std::string expected = "2/4";
```

```
    std::string game1 = "game1";
    EXPECT_EQ(getCurrentnumber(game1, games), expected);

    std::string nonexistent = "nonexistent";
    EXPECT_EQ(getCurrentnumber(nonexistent, games), "");
}

TEST(GameLogicTest, FindGameTest) {
    std::vector<Game> games;
    Game game;
    game.game_id = "game1";
    game.players_quantity = 2;
    game.current_players = 1;

    Player player1 = {123, "Alice"};
    game.players.push_back(player1);

    games.push_back(game);

    Player player2 = {456, "Bob"};
    EXPECT_TRUE(findgame(games, player2, "game1"));
    EXPECT_EQ(games[0].current_players, 2);

    Player player3 = {789, "Charlie"};
    EXPECT_FALSE(findgame(games, player3, "game1"));
}

TEST(GameLogicTest, CheckGameTest) {
    std::vector<Game> games;

    Game game;
    game.game_id = "game1";
    game.players_quantity = 2;
    game.current_players = 2;
    games.push_back(game);

    EXPECT_TRUE(checkgame(games, "game1"));

    Game game2;
    game2.game_id = "game2";
    game2.players_quantity = 2;
    game2.current_players = 1;
    games.push_back(game2);

    EXPECT_FALSE(checkgame(games, "game2"));
}
```

```
TEST(GameLogicTest, DeleteGameTest) {
    std::vector<Game> games;

    Game game;
    game.game_id = "game1";
    games.push_back(game);

    std::string game1 = "game1";
    deletegame(game1, games);
    EXPECT_TRUE(games.empty());

    std::string nonexistent = "nonexistent";
    deletegame(nonexistent, games);
    EXPECT_TRUE(games.empty());
}

TEST(GameLogicTest, CheckGameConditionTest) {
    std::vector<Game> games;
    Game game;
    game.game_id = "game1";
    game.cond = 0;
    games.push_back(game);

    std::string game1 = "game1";
    EXPECT_EQ(check_game_condition(games, game1), 0);
    EXPECT_EQ(games[0].cond, 1);

    EXPECT_EQ(check_game_condition(games, game1), 1);
}

int main(int argc, char **argv) {
    ::testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}
```

Заключение

Разработанная программа реализует игру "Быки и коровы" с использованием технологий сокетов ZeroMQ, что позволяет эффективно управлять взаимодействием между клиентом и сервером. Клиенты могут создавать игры, подключаться к существующим, а затем по очереди пытаться угадать 4-значное число, получая подсказки в виде быков и коров. Сервер отслеживает состояние игры, управляет подключениями и публикует сообщения о готовности игры, а также объявляет победителя. Использование различных типов сокетов (REQ, SUB, PUB) обеспечивает гибкость и производительность системы. Важно отметить, что в процессе работы программы применяются механизмы синхронизации и таймауты, что позволяет избежать блокировки и улучшить взаимодействие между клиентами и сервером.