Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики Кафедра вычислительной математики и программирования

# Курсовой проект по курсу

**«Операционные системы»**

Группа:М8О-215Б-23

Студент: Венгер Ирина Витальевна Преподаватель: Миронов Е.С. Оценка:

Дата:

Москва, 2024.

Содержание

1. Постановка задачи.
2. Общие сведения о программе.
3. Общий метод и алгоритм решения.
4. Код программы.
5. Демонстрация работы программы.
6. Вывод.

# Постановка задачи

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 7: 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

# Общие сведения о программе

Программа написана на языке C++ и представляет собой

клиент-серверное приложение для игры в "Быки и коровы" (Bulls and Cows). Игра заключается в угадывании загаданного сервером четырёхбуквенного слова. Игроки отправляют свои предположения на сервер, а сервер возвращает количество "быков" (правильные буквы на правильных позициях) и "коров" (правильные буквы на неправильных позициях). Игра продолжается до тех пор, пока игрок не угадает слово.

Основные компоненты программы

1. Сервер (server.cpp):
   * Управляет созданием игр, подключением игроков и обработкой их предположений.
   * Поддерживает несколько игр одновременно.
   * Использует многопоточность для обработки подключений клиентов.
   * Генерирует случайные четырёхбуквенные слова для каждой игры.
2. Клиент (client.cpp):
   * Подключается к серверу и позволяет игроку отправлять команды.
   * Отображает ответы сервера (например, количество быков и коров).

**Общий метод и алгоритм решения**

1. **Разделение ответственности**:
   * **Сервер**:
     + Управляет созданием игр.
     + Обрабатывает подключения клиентов.
     + Проверяет предположения игроков и возвращает результаты.
     + Поддерживает состояние игр (название,

загаданное слово, список игроков).

# Клиент:

* + - Подключается к серверу.
    - Отправляет команды (создание игры, подключение к игре, угадывание слова).
    - Получает и отображает ответы сервера.

# Многопоточность:

* + Сервер обрабатывает каждого клиента в отдельном потоке, что позволяет поддерживать несколько игроков одновременно.

# Синхронизация:

* + Для защиты общих данных (списка игр) используется мьютекс (std::mutex).

# Генерация случайного слова:

* + Сервер генерирует случайное четырёхбуквенное слово для каждой игры.

# Логика игры:

* + Игроки отправляют свои предположения.
  + Сервер проверяет предположения и возвращает количество "быков" и "коров".

# Алгоритм решения задачи

1. Серверная часть

# Инициализация сервера:

* + - Создание сокета.
    - Привязка сокета к адресу и порту.
    - Переход в режим прослушивания.

# Обработка подключений:

* + - Сервер принимает подключения от клиентов.
    - Для каждого клиента создаётся отдельный поток

(std::thread).

# Обработка команд клиента:

* + - В каждом потоке сервер читает команды от клиента и обрабатывает их:

# Создание игры:

* + - * + Генерация случайного слова.
        + Добавление игры в список.

# Подключение к игре:

* + - * + Поиск игры по названию.
        + Добавление игрока в игру, если есть свободные места.

# Поиск игры:

* + - * + Поиск первой доступной игры с свободными местами.

# Угадывание слова:

* + - * + Проверка предположения игрока.
        + Возврат количества быков и коров.
        + Если слово угадано, сообщение о победе.

# Синхронизация:

* + - Доступ к общим данным (списку игр) защищается мьютексом.

# Завершение работы:

* + - При отключении клиента поток завершает работу.

1. Клиентская часть

# Подключение к серверу:

* + Создание сокета.
  + Подключение к серверу по указанному IP-адресу и порту.

# Отправка команд:

* + Клиент отправляет команды серверу:
    - create <название\_игры> — создать игру.
    - join <название\_игры> — присоединиться к игре.
    - find — найти доступную игру.
    - play <слово> — сделать предположение.
    - exit — завершить работу.

# Получение ответов:

* + Клиент читает ответы сервера и выводит их на экран.

# Завершение работы:

* + При вводе команды exit клиент завершает работу.

# Алгоритм игры "Быки и коровы":

1. **Генерация слова**:
   * Сервер генерирует случайное четырёхбуквенное слово

(например, "word").

# Угадывание слова:

* + Игрок отправляет предположение (например, "test").
  + Сервер проверяет предположение:
    - **Быки**: Количество букв, которые совпадают с загаданным словом и находятся на правильных позициях.
      * Например, для слова "word" и предположения "test":
        + Буква "t" не совпадает.
        + Буква "e" не совпадает.
        + Буква "s" не совпадает.
        + Буква "t" не совпадает.
        + Быков: 0.
    - **Коровы**: Количество букв, которые есть в загаданном слове, но находятся на неправильных позициях.
      * Для слова "word" и предположения "test":
        + Буква "t" отсутствует в "word".
        + Буква "e" отсутствует в "word".
        + Буква "s" отсутствует в "word".
        + Буква "t" отсутствует в "word".
        + Коров: 0.

# Победа:

* + Если количество быков равно 4, игрок угадал слово

# Код программы

Код программы смотрите в приложении 1.

**Демонстрация работы программы** (base) boopie@MacBook-Air-Irina KP % ./server Сервер запущен.

Игра "game1" создана. Цель игры: угадать слов. Игроков: 1

Игрок присоединился к игре "game1". Игроков: 2

|  |  |  |  |
| --- | --- | --- | --- |
| Bulls: | 0, | Cows: | 0 |
| Bulls: | 0, | Cows: | 1 |
| Bulls: | 0, | Cows: | 0 |
| Bulls: | 0, | Cows: | 0 |
| Bulls: | 0, | Cows: | 1 |
| Bulls: | 1, | Cows: | 0 |
| Bulls: | 1, | Cows: | 0 |
| Bulls: | 1, | Cows: | 0 |
| Bulls: | 0, | Cows: | 1 |
| Bulls: | 0, | Cows: | 0 |
| Bulls: | 0, | Cows: | 1 |
| Bulls: | 1, | Cows: | 1 |
| Bulls: | 2, | Cows: | 0 |
| Bulls: | 0, | Cows: | 0 |
| Bulls: | 4, | Cows: | 0 |

You won!

(base) boopie@MacBook-Air-Irina KP % ./client

Подключено к серверу.

* create game1

Игра "game1" создана. Цель игры: угадать слово. Игроков: 1

* play qwer

|  |  |  |
| --- | --- | --- |
| Bulls:   * play | 0, Cows:  tqwe | 0 |
| Bulls: | 0, Cows: | 0 |
| * play | uqwe |  |
| Bulls: | 0, Cows: | 1 |
| * play | ausd |  |
| Bulls: | 1, Cows: | 0 |
| * play | ghjk |  |
| Bulls: | 0, Cows: | 1 |
| * play | hqwe |  |
| Bulls: | 0, Cows: | 1 |
| * play | kuhl |  |
| Bulls: | 2, Cows: | 0 |
| * play | buhn |  |
| Bulls: | 4, Cows: | 0 |

You won!

* exit

(base) boopie@MacBook-Air-Irina KP % ./client

Подключено к серверу.

* join game1

Игрок присоединился к игре "game1". Игроков: 2

|  |  |  |
| --- | --- | --- |
| * play | tyui |  |
| Bulls: | 0, Cows: | 1 |
| * play | yqwe |  |
| Bulls: | 0, Cows: | 0 |
| * play | quwe |  |
| Bulls: | 1, Cows: | 0 |
| * play | oupf |  |
| Bulls: | 1, Cows: | 0 |

|  |  |  |
| --- | --- | --- |
| * play | gqwe |  |
| Bulls: | 0, Cows: | 0 |
| * play | quwh |  |
| Bulls: | 1, Cows: | 1 |
| * play | zxcv |  |
| Bulls: | 0, Cows: | 0 |
| * exit |  |  |

# Вывод

В данной работе я познакомилась с **клиент-серверной архитектурой** и принципами её реализации на языке C++. В процессе разработки я изучила следующие ключевые аспекты: научилась создавать сокеты, привязывать их к адресам и портам, а также устанавливать соединение между клиентом и сервером, познакомилась с использованием потоков (std::thread) для обработки нескольких клиентов одновременно, научилась защищать общие данные (например, список игр) от одновременного доступа из нескольких потоков, чтобы избежать состояний гонки (race conditions), реализовала логику игры "Быки и коровы", включая генерацию случайного слова и проверку предположений игроков.

Приложения

server.cpp

#include <iostream> #include <string>

#include <vector>

#include <unistd.h> #include <ctime>

#include <algorithm>

#include <sys/types.h> #include <sys/socket.h> #include <netinet/in.h> #include <arpa/inet.h> #include <thread>

#include <mutex> #include <random>

const int MAX\_PLAYERS = 4; // Максимальное количество игроков в одной игре

struct Game {

std::string name; int players;

std::string word;

std::vector<int> clients;

};

std::vector<Game> games; std::mutex gamesMutex;

// Функция для проверки наличия буквы в строке

bool contains(const std::string& s, char lit) {

return std::find(s.begin(), s.end(), lit) != s.end();

}

// Функция для генерации случайного слова

std::string generateRandomWord() {

std::string letters = "qwertyuiopasdfghjklzxcvbnm"; std::random\_device rd;

std::default\_random\_engine rng(rd());

std::shuffle(letters.begin(), letters.end(), rng); // Перемешивает случайным образом

std::string word;

for (int i = 0; i < 4; ++i) { word += letters[i];

}

return word;

}

void createGame(const std::string& name, int clientSocket) { std::lock\_guard<std::mutex> lock(gamesMutex);

Game game;

game.name = name; game.players = 1;

game.word = generateRandomWord();

game.clients.push\_back(clientSocket); games.push\_back(game);

// Отправляем подтверждение клиенту

std::string response = "Игра \"" + name + "\" создана. Цель игры: угадать слово. Игроков: 1\n";

write(clientSocket, response.c\_str(), response.length());

std::cout << "Игра \"" << name << "\" создана. Цель игры: угадать слов.

Игроков: 1" << std::endl;

}

void joinGame(std::string& name, int clientSocket) { std::lock\_guard<std::mutex> lock(gamesMutex);

for (Game& game : games) {

if (game.name == name) {

if (game.players < MAX\_PLAYERS) { game.players++;

game.clients.push\_back(clientSocket);

// Отправляем подтверждение клиенту

std::string response = "Игрок присоединился к игре \"" + name + "\". Игроков: " + std::to\_string(game.players) + "\n";

write(clientSocket, response.c\_str(), response.length());

std::cout << "Игрок присоединился к игре \"" << name << "\".

Игроков: " << game.players << std::endl;

} else {

// Отправляем сообщение о том, что игра полная

std::string response = "Игра \"" + name + "\" уже полная.\n"; write(clientSocket, response.c\_str(), response.length());

std::cout << "Игра \"" << name << "\" уже полная." << std::endl;

}

return;

}

}

// Отправляем сообщение о том, что игра не найдена

std::string response = "Игра \"" + name + "\" не найдена.\n"; write(clientSocket, response.c\_str(), response.length());

std::cout << "Игра \"" << name << "\" не найдена." << std::endl;

}

void findGame(int clientSocket) {

std::lock\_guard<std::mutex> lock(gamesMutex); for (Game& game : games) {

if (game.players < MAX\_PLAYERS) { game.players++;

game.clients.push\_back(clientSocket);

// Отправляем подтверждение клиенту

std::string response = "Игрок присоединился к игре \"" + game.name + "\". Игроков: " + std::to\_string(game.players) + "\n";

write(clientSocket, response.c\_str(), response.length());

std::cout << "Игрок присоединился к игре \"" << game.name << "\".

Игроков: " << game.players << std::endl; return;

}

}

// Отправляем сообщение о том, что свободные игры не найдены

std::string response = "Свободные игры не найдены.\n"; write(clientSocket, response.c\_str(), response.length());

std::cout << "Свободные игры не найдены." << std::endl;

}

void playGame(std::string guessWord, int clientSocket) { std::lock\_guard<std::mutex> lock(gamesMutex);

for (Game& game : games) {

if (std::find(game.clients.begin(), game.clients.end(), clientSocket) != game.clients.end()) {

std::string word = game.word; int bulls = 0;

int cows = 0;

for (int i = 0; i < 4; ++i) {

if (word[i] == guessWord[i]) { bulls++;

} else if (contains(word, guessWord[i])) { cows++;

}

}

// Отправляем ответ клиенту

std::string response = "Bulls: " + std::to\_string(bulls) + ", Cows: "

+ std::to\_string(cows) + "\n";

if (bulls == 4) {

response += "You won!\n";

}

write(clientSocket, response.c\_str(), response.length());

std::cout << "Bulls: " << bulls << ", Cows: " << cows << std::endl;

if (bulls == 4) {

std::cout << "You won!" << std::endl;

}

return;

}

}

// Если игрок не найден в играх

std::string response = "Игрок не найден в играх.\n";

write(clientSocket, response.c\_str(), response.length());

std::cout << "Игрок не найден в играх." << std::endl;

}

void handleClient(int clientSocket) { char buffer[256];

while (true) {

int bytesRead = read(clientSocket, buffer, sizeof(buffer) - 1); if (bytesRead <= 0) {

break;

}

buffer[bytesRead] = '\0';

std::string command(buffer);

if (command.substr(0, 6) == "create") {

std::string name = command.substr(7); createGame(name, clientSocket);

} else if (command.substr(0, 4) == "join") { std::string name = command.substr(5);

joinGame(name, clientSocket);

} else if (command.substr(0, 4) == "find") { findGame(clientSocket);

} else if (command.substr(0, 4) == "play") {

std::string guessWord = command.substr(5); playGame(guessWord, clientSocket);

} else if (command == "exit") { break;

} else {

std::string response = "Неверная команда.\n";

write(clientSocket, response.c\_str(), response.length()); std::cout << "Неверная команда." << std::endl;

}

}

close(clientSocket);

}

int main() {

std::srand(std::time(nullptr)); // Инициализация генератора случайных чисел

int serverSocket = socket(AF\_INET, SOCK\_STREAM, 0); if (serverSocket == -1) {

std::cerr << "Ошибка при создании сокета." << std::endl;

return 1;

}

sockaddr\_in serverAddr;

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(12345); // Порт сервера

serverAddr.sin\_addr.s\_addr = INADDR\_ANY;

if (bind(serverSocket, (sockaddr\*)&serverAddr, sizeof(serverAddr)) == -1) { std::cerr << "Ошибка при привязке сокета." << std::endl;

close(serverSocket); return 1;

}

if (listen(serverSocket, 5) == -1) {

std::cerr << "Ошибка при прослушивании сокета." << std::endl;

close(serverSocket); return 1;

}

std::cout << "Сервер запущен." << std::endl;

while (true) {

sockaddr\_in clientAddr;

socklen\_t clientAddrLen = sizeof(clientAddr);

int clientSocket = accept(serverSocket, (sockaddr\*)&clientAddr, &clientAddrLen);

if (clientSocket == -1) {

std::cerr << "Ошибка при принятии соединения." << std::endl; continue;

}

std::thread clientThread(handleClient, clientSocket); clientThread.detach();

}

close(serverSocket); return 0;

}

client.cpp

#include <iostream> #include <string>

#include <unistd.h>

#include <sys/socket.h> #include <netinet/in.h> #include <arpa/inet.h>

int main() {

// Создаем сокет

int clientSocket = socket(AF\_INET, SOCK\_STREAM, 0); if (clientSocket == -1) {

std::cerr << "Ошибка при создании сокета." << std::endl; return 1;

}

// Указываем адрес и порт сервера

sockaddr\_in serverAddr;

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(12345); // Порт сервера

serverAddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1"); // Локальный адрес

// Подключаемся к серверу

if (connect(clientSocket, (sockaddr\*)&serverAddr, sizeof(serverAddr)) == -1) { std::cerr << "Ошибка при подключении к серверу." << std::endl;

close(clientSocket); return 1;

}

std::cout << "Подключено к серверу." << std::endl;

std::string command; while (true) {

// Выводим приглашение для ввода команды

std::cout << "> ";

std::getline(std::cin, command);

// Если команда "exit", завершаем работу

if (command == "exit") { break;

}

// Отправляем команду серверу

write(clientSocket, command.c\_str(), command.length());

// Читаем ответ от сервера

char buffer[256];

int bytesRead = read(clientSocket, buffer, sizeof(buffer) - 1);

if (bytesRead <= 0) {

std::cout << "Сервер завершил работу." << std::endl; break;

}

buffer[bytesRead] = '\0';

std::cout << buffer; // Выводим ответ сервера

}

// Закрываем сокет

close(clientSocket); return 0;

}