МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева (Самарский университет)»

Факультет информатики
Кафедра информационных систем и технологий

Дисциплина
«Операционные системы»

ОТЧЕТ

по дополнительному заданию

Реализация ассоциативного массива в виде WSA2 сервера на языке C++.

Студент: Колбанов Д.О.
Группа: 6204-020302D

Преподаватель: Востокин С.В.

Самара, 2023

Задание.

Задание заключается в реализации сервера на языке C++, обрабатывающего одно подключение за раз по протоколу IP. Сервер и клиенты - на одном узле.

Сервер (ddict) реализует ассоциированный массив (словарь) пар вида (слово, слово).

Клиент сервера может выполнять 2 операции запроса:

1) dict -s <ключ> <значение> -- запись пары в словарь;

2) dict -g <ключ> -- чтение значения по ключу.

Листинг программы.

Server.cpp

```
#pragma comment(lib, "ws2_32.lib")
#include <winsock2.h> // библиотека для работы с сетью
#include <iostream>
#include <map>
#pragma warning(disable: 4996) // чтобы addr.sin_addr.s_addr = inet_addr("127.0.0.1");
работало корректно

using namespace std;

SOCKET connections[100];
int Counter = 0;
map<string, string> my_map;

bool CheckMap(char* word)
{
    bool flag = false;
    for (const auto& element : my_map)
    {
        if (element.first == word)
        {
            return flag = true;
        }
    }
    return flag;
}

void ServerWorker(char* words[], int index)
{
    string message;
    int msg_size;
    if (strcmp(words[0], "write") == 0)
    {
        my_map[words[1]] = words[2];
        message = "Writing is succesful";
        msg_size = message.size();
        send(connections[index], (char*)&msg_size, sizeof(int), NULL);
        send(connections[index], message.c_str(), msg_size, NULL);
        Sleep(10);
    }
    else if (strcmp(words[0], "read") == 0)
    {
        if (CheckMap(words[1]))
        {
            message = my_map[words[1]];
            msg_size = message.size();
```

```
            send(connections[index], (char*)&msg_size, sizeof(int), NULL);
            send(connections[index], message.c_str(), msg_size, NULL);
            Sleep(10);
        }
        else
        {
            message = "There is no value";
            msg_size = message.size();
            send(connections[index], (char*)&msg_size, sizeof(int), NULL);
            send(connections[index], message.c_str(), msg_size, NULL);
            Sleep(10);
        }
    }
    else
    {
        message = "The command is not recognized";
        msg_size = message.size();
        send(connections[index], (char*)&msg_size, sizeof(int), NULL);
        send(connections[index], message.c_str(), msg_size, NULL);
        Sleep(10);
    }
}

void ClientHandler(int index)
{
    int msg_size;
    while (true)
    {
        if (recv(connections[index], (char*)&msg_size, sizeof(int), NULL) > 0)
        {
            char* msg = new char[msg_size + 1];
            msg[msg_size] = '\0';
            recv(connections[index], msg, msg_size, NULL);
            cout << "Client message - " << msg << endl;
            char* words[10];
            int count = 0;
            char* pch = strtok(msg, " ");
            while (pch != NULL)
            {
                words[count] = pch;
                pch = strtok(NULL, " ");
                count++;
            }
            ServerWorker(words, index);
            delete[] msg;
        }
        else
        {
            ::closesocket(connections[index]);
            connections[index] = INVALID_SOCKET;
            cout << "Client disconnected. ID - " << index << endl;
            return;
        }
    }
}

int main(int argc, char* argv[])
{
    cout << "Server is working" << endl;
    WSAData wsaData; // структура WSAData
    WORD DLLVersion = MAKEWORD(2, 1); // запрашиваемая версия библиотеки winsock
    if (WSAStartup(DLLVersion, &wsaData) != 0) // функция для загрузки библиотеки
    {
        cout << "Error: lib connection failed" << endl;
        exit(1);
    }

    SOCKADDR_IN addr; // структура, предназначенная для хранения адреса, для интернет
протоколов используется имя SOCKADDR_IN
    int sizeofaddr = sizeof(addr);
```

```
    addr.sin_addr.s_addr = inet_addr("127.0.0.1"); // хранение IP адреса
    addr.sin_port = htons(1111); // порт для идентификации программы с поступающими данными
    addr.sin_family = AF_INET; // семейство протоколов, для инетернет-протоколов используется константа AF_INET

    // чтобы 2 компьютера смогли установить соединение, один из них должен запустить прослушивание на определенном порту
    SOCKET sListen = socket(
        AF_INET,
        SOCK_STREAM, // указывает на протокол, устанавливающий соединение
        NULL);
    bind(sListen, (SOCKADDR*)&addr, sizeof(addr)); // функция для привязки адреса сокету
    // после того, как локальный адрес и порт привязаны к сокету, нужно приступить к прослушиванию порта в ожидании
    // соединения со стороны клиента, для этого служит фукция listen
    listen(sListen, SOMAXCONN); // второй параметр - максимальное количество запросов, ожидающих обработки
    // фактически мы уже принимаем соединение

    SOCKET newConnection; // новый сокет для удерживания соединения с клиентом
    // newConnection = accept(sListen, (SOCKADDR*)&addr, &sizeofaddr); возвращает указатель на новый сокет, который можно использовать для общения с клиентом
    // после выполнения функции addr будет содержать сведения об IP адресе клиента,
    // который произвел подключение. Эти данные можно использовать для контроля доступа к серверу по IP адресу

    for (int i = 0; i < 100; i++) {
        newConnection = accept(sListen, (SOCKADDR*)&addr, &sizeofaddr);

        if (newConnection == 0)
        {
            std::cout << "Error #2\n";
        }
        else
        {
            std::cout << "Client Connected! ID: " << i << endl;
            connections[i] = newConnection;
            Counter++;
            CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE)ClientHandler, (LPVOID)(i), NULL, NULL);
            string message;
            int msg_size;
            message = "Hello, Client!\nwrite <ключ> <значение>\nread <ключ>\n";
            msg_size = message.size();
            send(connections[i], (char*)&msg_size, sizeof(int), NULL);
            send(connections[i], message.c_str(), msg_size, NULL);
            Sleep(10);
        }
    }

    system("pause");

}
```

Client.cpp

```
#pragma comment(lib, "ws2_32.lib")
#include <winsock2.h> // библиотека для работы с сетью
#include <iostream>
#include <string>
#pragma warning(disable: 4996)

using namespace std;

SOCKET Connection;

void ClientHandler()
{
```

```
    int size_message;
    while (true)
    {
        if (recv(Connection, (char*)&size_message, sizeof(int), NULL) > 0)
        {
            if (Connection == INVALID_SOCKET)
            {
                continue;
            }
            char* message = new char[size_message + 1];
            message[size_message] = '\0';
            recv(Connection, message, size_message, NULL);
            cout << message << endl;
            delete[] message;
        }
        else
        {
            ::closesocket(Connection);
            Connection = INVALID_SOCKET;
            return;
        }
    }
}


int main()
{
    cout << "Client is working" << endl;
    setlocale(LC_ALL, "Rus");
    WSAData wsaData; // структура WSAData
    WORD DLLVersion = MAKEWORD(2, 1); // запрашиваемая версия библиотеки winsock
    if (WSAStartup(DLLVersion, &wsaData) != 0) // функция для загрузки библиотеки
    {
        cout << "Error: lib connection failed" << endl;
        exit(1);
    }

    SOCKADDR_IN addr; // структура, предназначенная для хранения адреса, для интернет
протоколов используется имя SOCKADDR_IN
    int sizeofaddr = sizeof(addr);
    addr.sin_addr.s_addr = inet_addr("127.0.0.1"); // хранение IP адреса
    addr.sin_port = htons(1111); // порт для идентификации программы с поступающими
данными
    addr.sin_family = AF_INET; // семейство протоколов, для инетернет-протоколов
используется константа AF_INET

    Connection = socket(AF_INET, SOCK_STREAM, NULL); // сокет для соединения с
сервером
    if (connect(Connection, (SOCKADDR*)&addr, sizeof(addr)) != 0 ) // соединение с
сервером
    {
        cout << "Error: connection to server failed" << endl;
        return 1;
    }

    CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE)ClientHandler, NULL, NULL, NULL);

    string message;
    while (Connection != INVALID_SOCKET)
    {
        getline(cin, message);
        int msg_size = message.size();
        if (Connection != INVALID_SOCKET)
        {
            send(Connection, (char*)&msg_size, sizeof(int), NULL);
            send(Connection, message.c_str(), msg_size, NULL);
            Sleep(10);
        }
        else
        {
```

```
            cout << "Server is dead" << endl;
        }
    }
    system("pause");

}
```