ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

ассистент		Д.А. Кочин
старший преподаватель	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №5

« Межсетевое взаимодействие между процессами»

по дисциплине: Операционные Системы

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР.	4236		Л. Мвале
		подпись, дата	инициалы, фамилия

Санкт-Петербург
2024

1. Цель работы

Организация межсетевого взаимодействия средствами WinAPI и POSIX.

Задание

Организовать взаимодействие типа клиент-сервер средствами WinAPI и POSIX в соответствии с индивидуальным заданием.

1. Вычислить номер варианта по списку в журнале и сохранить его в файл TASKID.txt в репозитории.

2. Выбрать индивидуальное задание в соответствии с номером варианта. По номеру варианта также определить:

- протокол, с использованием которого должен происходить обмен данными между сервером и клиентом;
- операционную систему, под которую необходимо разработать приложение сервера;
- операционную систему, под которую необходимо разработать приложение клиента.

3. В разделе Issues репозитория создать не менее трех задач. Например: "Разработка приложения сервера", "Разработка приложения клиента", "Отладка клиент-серверного взаимодействия". Последовательно выполнить эти задачи, написав код и разместив его в репозитории. Решению каждой задачи должен соответствовать свой отдельный коммит, который должен быть привязан к конкретной задаче. Итого в репозитории должно быть не менее трех коммитов. При необходимости создать дополнительные задачи.

4. Код приложения клиента необходимо разместить в файле client.cpp в корне репозитория, код приложения сервера - в файле server.cpp там же. При необходимости использовать дополнительные заголовочные файлы. Код, предназначенный для

выполнения в ОС Linux, должен собираться командной g++ client.cpp или g++ server.cpp.

5. В репозитории имеется тест, проверяющий корректность оформления кода в соответствии с Google C++ Style Guide. Данный тест запускает линтер, который проверяет соответствие кода стандарту языка и правилам оформления (отступы, разделители, комментарии и т.п.). В некоторых случаях линтер может дать совет, как улучшить код используя общепринятые практики.

6. Автоматическое тестирование работоспособности кода отсутствует. Необходимо загрузить рабочий код в репозиторий, а затем защитить лабораторную работу у преподавателя, продемонстрировав корректную совместную работу клиентского и серверного приложений.

7. Допускается выполнение задания не в полном объеме. Правила начисления рейтинга для этого случая приведены в разделе Рейтинг.

Схема взаимодействия между клиентом и сервером

Общие требования для всех заданий:

- клиент и сервер являются консольными приложениями;
- номер порта, на котором работает сервер, указывается при его запуске в качестве аргумента командной строки;
- доменное имя (ip-адрес), на котором работает сервер, а также его номер порта указываются в качестве аргументов командной строки при запуске клиента;
- сервер выводит в консоль все сообщения, которые получает от клиента;

- клиент выводит в консоль все сообщения, которые получает от сервера.

Вариант 24

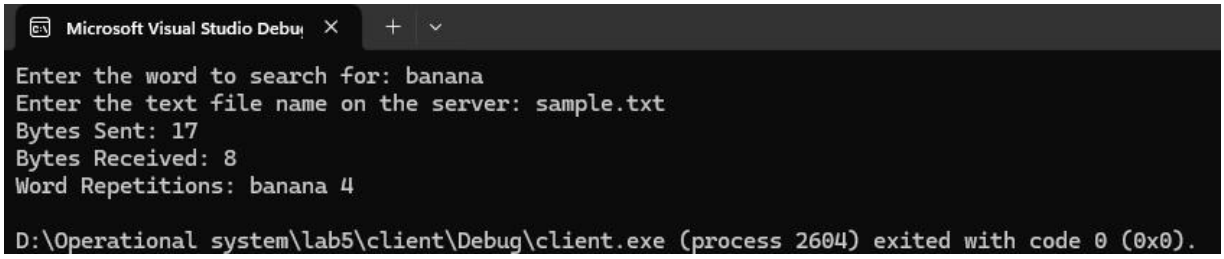
7. Повторения слов. Приложение-клиент запрашивает у пользователя ввод двух строк: слово и имя текстового файла, расположенного на сервере. Эти строки передаются на сервер, который подсчитывает количество повторений указанного слова в выбранном текстовом файле. Полученное число возвращается клиенту.

№ варианта	Индивидуальное задание	Протокол	Сервер	Клиент
24	7	UDP	Linux	Windows

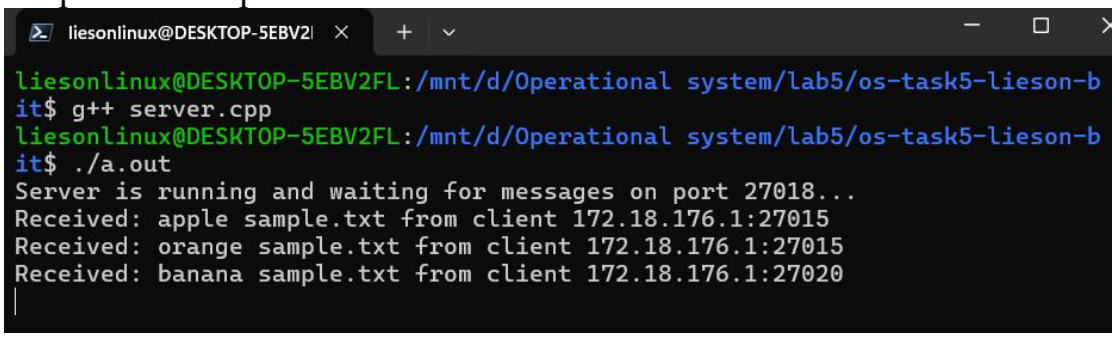
2. Результат выполнения работы

Два клиентских процесса с разными номерами портов, запущенных в Windows, отправляют данные для обработки на сервере

```
PS D:\Operational system\lab5\os-task5-lieson-bit\Debug> g++ -o client clien
t.cpp -lws2_32 && .\client.exe 27018
Enter the word to search for: apple
Enter the text file name on the server: sample.txt
Bytes Sent: 16
Bytes Received: 8
Word Repetitions: apple 10
PS D:\Operational system\lab5\os-task5-lieson-bit\Debug> g++ -o client clien
t.cpp -lws2_32 && .\client.exe 27018
Enter the word to search for: orange
Enter the text file name on the server: sample.txt
Bytes Sent: 17
Bytes Received: 8
Word Repetitions: orange 5
PS D:\Operational system\lab5\os-task5-lieson-bit\Debug>
```



Два разных клиента подключились к серверам. Один клиент отправил 2 запроса



3. Исходный код программы с комментариями

Для client

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#define _WINSOCK_DEPRECATED_NO_WARNINGS
#include <winsock2.h>
#include <string.h> // For strlen()
#pragma comment(lib, "ws2_32.lib") // Link with Winsock library

#define SERVER_IP "172.18.182.131"
#define CLIENT_PORT 27015 // Custom client port
#define BUFFER_SIZE 512

int main(int argc, char *argv[]) {
    // Check if server port is passed as a command-line argument
    if (argc != 2) {
        printf("Usage: client.exe <SERVER_PORT>\n");
        return 1;
    }

    // Parse server port from the command line
    int serverPort = atoi(argv[1]);
    if (serverPort <= 0 || serverPort > 65535) {
        printf("Invalid port number. Please provide a valid port number
(1-65535).\n");
```

```
        return 1;
    }

    // Initialize Winsock
    WSADATA wsaData;
    int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if (iResult != NO_ERROR) {
        printf("Error at WSAStartup()\n");
        return 1;
    }

    // Create a UDP socket
    SOCKET udpSocket = socket(AF_INET, SOCK_DGRAM,
IPPROTO_UDP);
    if (udpSocket == INVALID_SOCKET) {
        printf("Error creating socket: %d\n", WSAGetLastError());
        WSACleanup();
        return 1;
    }

    // Bind the socket to a specific client port
    struct sockaddr_in clientAddr;
    clientAddr.sin_family = AF_INET;
    clientAddr.sin_addr.s_addr = INADDR_ANY; // Bind to any local
interface
    clientAddr.sin_port = htons(CLIENT_PORT); // Use the custom
client port

    if (bind(udpSocket, (struct sockaddr*)&clientAddr,
sizeof(clientAddr)) == SOCKET_ERROR) {
        printf("Bind failed with error: %d\n", WSAGetLastError());
        closesocket(udpSocket);
        WSACleanup();
        return 1;
    }

    // Configure server address using the port passed as argument
    struct sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = inet_addr("172.18.182.131"); // Use
correct server IP
    serverAddr.sin_port = htons(serverPort); // Port passed from
command line
```

```
// Interactive input from the user
char word[256], filename[256];
printf("Enter the word to search for: ");
scanf("%255s", word); // Limit input to prevent buffer overflow
printf("Enter the text file name on the server: ");
scanf("%255s", filename);

// Prepare the message to send to the server
char sendbuf[BUFFER_SIZE];
snprintf(sendbuf, sizeof(sendbuf), "%s %s", word, filename);

// Send data to the server
int bytesSent = sendto(
    udpSocket,
    sendbuf,
    (int)strlen(sendbuf),
    0,
    (struct sockaddr*)&serverAddr,
    sizeof(serverAddr)
);

if (bytesSent == SOCKET_ERROR) {
    printf("Send failed with error: %d\n", WSAGetLastError());
    closesocket(udpSocket);
    WSACleanup();
    return 1;
}
printf("Bytes Sent: %d\n", bytesSent);

// Receive data from the server (the word count)
char recvbuf[BUFFER_SIZE] = { 0 };
int serverAddrLen = sizeof(serverAddr);
int bytesRecv = recvfrom(
    udpSocket,
    recvbuf,
    BUFFER_SIZE - 1,
    0,
    (struct sockaddr*)&serverAddr,
    &serverAddrLen
);

if (bytesRecv == SOCKET_ERROR) {
```

```
        printf("Recv failed with error: %d\n", WSAGetLastError());
    }
    else {
        recvbuf[bytesRecv] = '\0'; // Null-terminate the received data
        printf("Bytes Received: %d\n", bytesRecv);
        printf("Word Repetitions: %s\n", recvbuf);
    }

    // Clean up
    closesocket(udpSocket);
    WSACleanup();
    return 0;
}
```

Для server

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <cstring>
#include <string>
#include <algorithm>
#include <thread>
#include <map>

#define BUFFER_SIZE 512    // Размер буфера для получения данных
#define SERVER_PORT 27018 // Заданный порт для сервера

// Функция подсчёта количества вхождений слова в файле
int countWordOccurrences(const std::string& filename, const std::string& word) {
    std::ifstream file(filename); // Открываем файл
    if (!file.is_open()) {
        return -1; // Возвращаем -1, если файл не найден или не может быть открыт
    }
```

```
    std::string line;
    int count = 0;

    while (std::getline(file, line)) { // Читаем файл построчно
        std::istringstream iss(line);
        std::string token;
        while (iss >> token) { // Разбиваем строку на слова
            // Приводим слова и искомое слово к нижнему регистру
            std::string tokenLower = token;
            std::string wordLower = word;
            std::transform(tokenLower.begin(), tokenLower.end(),
tokenLower.begin(), ::tolower);
            std::transform(wordLower.begin(), wordLower.end(),
wordLower.begin(), ::tolower);

            if (tokenLower == wordLower) { // Сравниваем слова
                ++count; // Увеличиваем счётчик, если совпадение
найдено
            }
        }
    }

    file.close(); // Закрываем файл
    return count; // Возвращаем количество вхождений
}

// Функция для обработки запроса от клиента
// Function to handle the client request
void handleClientRequest(int serverSocket, sockaddr_in clientAddr,
const std::string& request) {
    std::istringstream iss(request); // Parse the client request
    std::string word, filename;
    if (!(iss >> word >> filename)) { // Check the format of the request
        std::cerr << "Invalid message format!" << std::endl;
        return;
    }

    // Get the count of word occurrences in the file
    int count = countWordOccurrences(filename, word);
    std::map<std::string, int> wordCountMap; // Map to store word
counts

    if (count == -1) {
```

```
        std::cerr << "Error: File not found!" << std::endl;
    } else {
        wordCountMap[word] = count; // Store the word count in the map
    }

    // Create a response string with sorted words, no extra newline at the end
    std::ostringstream response;
    bool first = true;  // Flag to control the newline between entries

    for (const auto& entry : wordCountMap) {
        if (!first) {
            response << std::endl;  // Only add newline if it's not the first entry
        }
        response << entry.first << " " << entry.second;
        first = false;  // After the first entry, set flag to false
    }

    // Send the formatted response to the client
    ssize_t bytesSent = sendto(
        serverSocket, response.str().c_str(), response.str().size(), 0,
        (struct sockaddr*)&clientAddr, sizeof(clientAddr)
    );

    if (bytesSent < 0) {
        std::cerr << "Error sending data to client!" << std::endl;
    }
}

// Функция для обработки входящих сообщений от клиентов
void handleClient(int serverSocket) {
    struct sockaddr_in clientAddr; // Структура для хранения адреса клиента
    socklen_t clientAddrLen = sizeof(clientAddr);
    char buffer[BUFFER_SIZE];

    while (true) {
        // Получаем данные от клиента
        ssize_t bytesReceived = recvfrom(
            serverSocket, buffer, BUFFER_SIZE - 1, 0,
            (struct sockaddr*)&clientAddr, &clientAddrLen
```

```
        );

        if (bytesReceived < 0) {
            std::cerr << "Error receiving data!" << std::endl;
            continue;
        }

        buffer[bytesReceived] = '\0'; // Завершаем строку символом конца строки
        std::cout << "Received: " << buffer << " from client "
                  << inet_ntoa(clientAddr.sin_addr) << ":" << ntohs(clientAddr.sin_port) << std::endl;

        // Создаём новый поток для обработки запроса клиента
        std::thread(handleClientRequest, serverSocket, clientAddr, std::string(buffer)).detach();
    }
}

int main() {
    int serverSocket;
    struct sockaddr_in serverAddr;

    // Создаём UDP-сокет
    serverSocket = socket(AF_INET, SOCK_DGRAM, 0);
    if (serverSocket < 0) {
        std::cerr << "Error creating socket!" << std::endl;
        return 1;
    }

    // Конфигурируем адрес сервера
    memset(&serverAddr, 0, sizeof(serverAddr));
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = INADDR_ANY; // Привязываем ко всем локальным интерфейсам
    serverAddr.sin_port = htons(SERVER_PORT);

    // Привязываем сокет к указанному адресу и порту
    if (bind(serverSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) < 0) {
        std::cerr << "Error binding socket!" << std::endl;
        close(serverSocket);
        return 1;
```

```
    }

    std::cout << "Server is running and waiting for messages on port "
<< SERVER_PORT << "..." << std::endl;

    // Обрабатываем входящие сообщения от клиентов
    handleClient(serverSocket);

    // Завершаем работу
    close(serverSocket);
    return 0;
}
```

4. Вывод

В ходе выполнения задачи был разработан клиент-серверный UDP-приложение. Клиент отправляет серверу слово и имя файла, сервер возвращает количество повторений слова в файле. Реализованы механизмы обработки ошибок, ввода данных и взаимодействия через сокеты.