

Министерство образования Республики Беларусь  
Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

**ОТЧЕТ**

к лабораторной работе № 8

на тему «Интерфейс сокетов и основы сетевого программирования  
(Windows). Программное взаимодействия через сеть с использованием  
интерфейса сокетов. Реализация сетевых протоколов: собственных или  
стандартных»

Выполнил:  
студент гр. 153504  
Подвальников А.С.

Проверил:  
Гриценко Н.Ю.

Минск 2023

## СОДЕРЖАНИЕ

1 Цели работы .....	3
2 Краткие теоретические сведения.....	4
3 Полученные результаты .....	5
Выводы .....	6
Список использованных источников .....	7
Приложение А .....	8

## **1 ЦЕЛИ РАБОТЫ**

Изучить интерфейс сокетов и основы сетевого программирования Windows. Изучить программное взаимодействия через сеть с использованием интерфейса сокетов. Изучить сетевые протоколы. Реализовать клиент-серверное приложение для обмена текстовыми сообщениями с использованием TCP сокетов.

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Интерфейс сокетов и сетевое программирование в Windows предоставляют возможности для программного взаимодействия через сеть с использованием сокетов. Сокеты являются стандартными механизмами обмена данными по сети, а Win32 API предоставляет функции и объекты для их создания, настройки и использования.

Для реализации программы, использующей интерфейс сокетов и основы сетевого программирования в Windows, можно использовать следующие подходы и функции:

Создание сокета – функция *socket* для создания сокета, указав тип (например, `'SOCK_STREAM'` для потокового соединения) и протокол (например, `'IPPROTO_TCP'` для TCP-соединения).

Привязка сокета – функцию *bind* для привязки сокета к определенному адресу и порту. Это позволит программе принимать входящие соединения или отправлять данные по указанному адресу.

Прослушивание входящих соединений – функция *listen* для установки сокета в режим прослушивания входящих соединений. Это позволяет программе ожидать входящие запросы на соединение.

Принятие входящих соединений – функция *accept* для принятия входящих соединений от клиентов. Функция *accept* создает новый сокет, который можно использовать для обмена данными с клиентом.

Отправка и прием данных – функция *send* и *recv* для отправки и приема данных по сокету. Эти функции позволяют передавать информацию между клиентом и сервером.

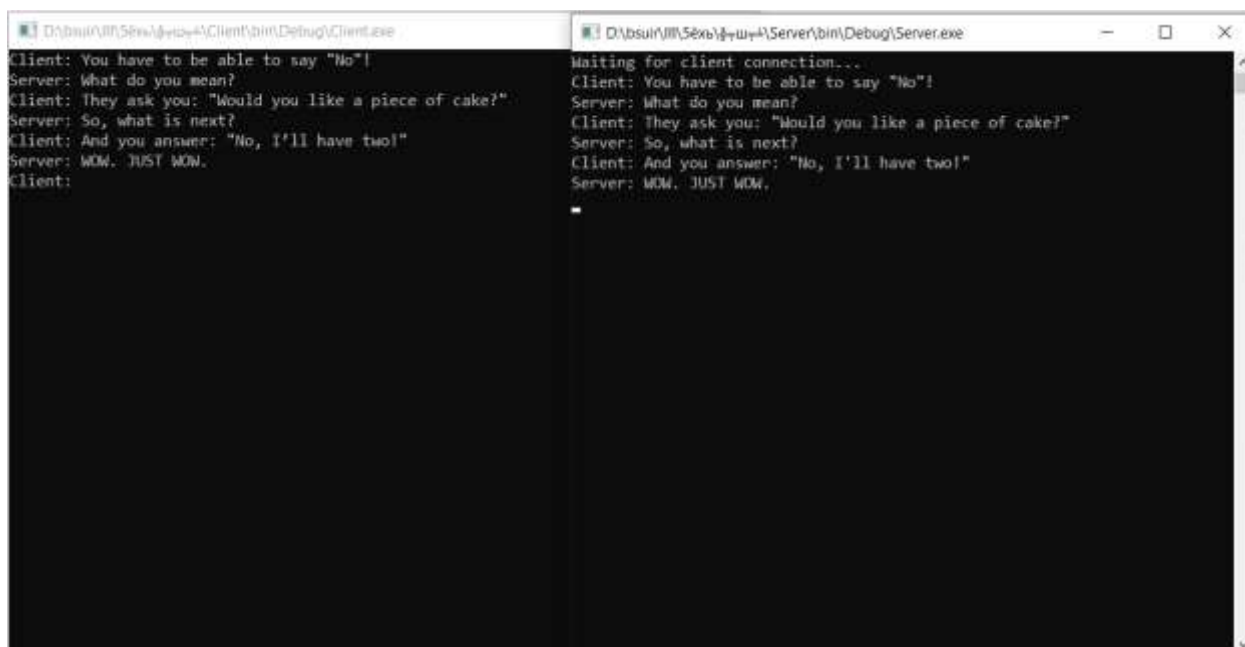
Реализация сетевых протоколов может быть осуществлена на базе созданных сокетов. Можно разрабатывать собственные протоколы для организации передачи данных между клиентами и сервером, либо использовать стандартные протоколы, такие как TCP или UDP.

При разработке собственных протоколов необходимо определить формат сообщений, способы кодирования и декодирования данных, а также логику взаимодействия между клиентом и сервером.

Использование интерфейса сокетов и сетевого программирования в Windows позволяет реализовать программное взаимодействие через сеть и разрабатывать собственные или использовать стандартные сетевые протоколы для обмена данными между клиентами и сервером.

### 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы было реализовано клиент-серверное приложение для обмена текстовыми сообщениями с использованием TCP сокетов. Результат работы программы показан на рисунке 3.1.



```
D:\bsul\III\5exb\4-ш\4\Server\bin\Debug\Client.exe
Client: You have to be able to say "No"!
Server: What do you mean?
Client: They ask you: "Would you like a piece of cake?"
Server: So, what is next?
Client: And you answer: "No, I'll have two!"
Server: WOW. JUST WOW.
Client:

D:\bsul\III\5exb\4-ш\4\Server\bin\Debug\Server.exe
Waiting for client connection...
Client: You have to be able to say "No"!
Server: What do you mean?
Client: They ask you: "Would you like a piece of cake?"
Server: So, what is next?
Client: And you answer: "No, I'll have two!"
Server: WOW. JUST WOW.
```

Рисунок 3.1 – Результат работы программы

## **ВЫВОДЫ**

В ходе выполнения данной лабораторной работы был изучен интерфейс сокетов и основы сетевого программирования Windows. Изучено программное взаимодействие через сеть с использованием интерфейса сокетов. Изучены сетевые протоколы. Реализовано клиент-серверное приложение для обмена текстовыми сообщениями с использованием TCP сокетов.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Щупак Ю. Win32 API. Разработка приложений для Windows. – СПб: Питер, 2008. – 592 с.: ил.
- [2] [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/api/winreg/> – Дата доступа 20.10.2023
- [3] [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/sysinfo/enumerating-registry-subkeys> – Дата доступа 20.10.2023
- [4] [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/api/ws2tcpip/> – Дата доступа 05.11.2023
- [5] [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-socket> – Дата доступа 07.11.2023
- [6] [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/ipc/interprocess-communications> – Дата доступа 07.11.2023
- [7] [Электронный ресурс]. – Режим доступа: <https://firststeps.ru/mfc/net/socket/r.php?1> – Дата доступа 08.11.2023
- [8] [Электронный ресурс]. – Режим доступа: <https://networkprogrammingnotes.blogspot.com/p/windows-socket-api.html> – Дата доступа 08.11.2023

**ПРИЛОЖЕНИЕ А**  
**(обязательное)**  
**Листинг кода**

**Server.cpp**

```
#include <winsock2.h>
#include <ws2tcpip.h>
#include <stdio.h>

#pragma comment(lib, "ws2_32.lib")

#define DEFAULT_PORT "27015"
#define BUFFER_SIZE 512

int main() {
    WSADATA wsaData;
    int result = WSStartup(MAKEWORD(2, 2), &wsaData);
    if (result != 0) {
        printf("WSAStartup failed with error: %d\n", result);
        return 1;
    }

    struct addrinfo* addrResult = NULL;
    struct addrinfo hints;

    ZeroMemory(&hints, sizeof(hints));
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_protocol = IPPROTO_TCP;
    hints.ai_flags = AI_PASSIVE;

    result = getaddrinfo(NULL, DEFAULT_PORT, &hints, &addrResult);
    if (result != 0) {
        printf("getaddrinfo failed with error: %d\n", result);
        WSACleanup();
        return 1;
    }

    SOCKET listenSocket = INVALID_SOCKET;

    listenSocket = socket(addrResult->ai_family, addrResult->ai_socktype,
        addrResult->ai_protocol);
    if (listenSocket == INVALID_SOCKET) {
        printf("socket failed with error: %d\n", WSAGetLastError());
        freeaddrinfo(addrResult);
        WSACleanup();
    }
}
```



```

        return 1;
    }

    result = bind(listenSocket, addrResult->ai_addr, (int)addrResult->ai_addrlen);
    if (result == SOCKET_ERROR) {
        printf("bind failed with error: %d\n", WSAGetLastError());
        freeaddrinfo(addrResult);
        closesocket(listenSocket);
        WSACleanup();
        return 1;
    }

    freeaddrinfo(addrResult);

    result = listen(listenSocket, SOMAXCONN);
    if (result == SOCKET_ERROR) {
        printf("listen failed with error: %d\n", WSAGetLastError());
        closesocket(listenSocket);
        WSACleanup();
        return 1;
    }

    printf("Waiting for client connection...\n");

    SOCKET clientSocket;
    clientSocket = accept(listenSocket, NULL, NULL);
    if (clientSocket == INVALID_SOCKET) {
        printf("accept failed with error: %d\n", WSAGetLastError());
        closesocket(listenSocket);
        WSACleanup();
        return 1;
    }

    closesocket(listenSocket);

    char buffer[BUFFER_SIZE];

    char response[BUFFER_SIZE];

    while (1) {
        result = recv(clientSocket, buffer, BUFFER_SIZE, 0);
        if (result > 0) {
            printf("Client: %.*s\n", result, buffer);

            printf("Server: ");
            fgets(response, BUFFER_SIZE, stdin);
            int responseLen = strlen(response);

            if (response[responseLen - 1] == '\n') {
                response[responseLen - 1] = '\0';
            }
        }
    }

```

```

        responseLen--;
    }
    result = send(clientSocket, response, responseLen, 0);
    if (result == SOCKET_ERROR) {
        printf("send failed with error: %d\n", WSAGetLastError());
        closesocket(clientSocket);
        WSACleanup();
        return 1;
    }
}
else if (result == 0) {
    printf("Connection closed by client.\n");
    break;
}
else {
    printf("recv failed with error: %d\n", WSAGetLastError());
    closesocket(clientSocket);
    WSACleanup();
    return 1;
}
}

closesocket(clientSocket);
WSACleanup();

return 0;
}

```

## Client.cpp

```

#include <winsock2.h>
#include <ws2tcpip.h>
#include <stdio.h>

#pragma comment(lib, "ws2_32.lib")

#define DEFAULT_PORT "27015"
#define SERVER_ADDRESS "127.0.0.1"
#define BUFFER_SIZE 512

int main() {
    WSADATA wsaData;
    int result = WSStartup(MAKEWORD(2, 2), &wsaData);
    if (result != 0) {
        printf("WSStartup failed with error: %d\n", result);
        return 1;
    }

    struct addrinfo* addrResult = NULL;
    struct addrinfo hints;

```

```

ZeroMemory(&hints, sizeof(hints));
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_STREAM;
hints.ai_protocol = IPPROTO_TCP;

result = getaddrinfo(SERVER_ADDRESS, DEFAULT_PORT, &hints,
&addrResult);
if (result != 0) {
    printf("getaddrinfo failed with error: %d\n", result);
    WSACleanup();
    return 1;
}

SOCKET connectSocket = INVALID_SOCKET;

connectSocket = socket(addrResult->ai_family, addrResult->ai_socktype,
addrResult->ai_protocol);
if (connectSocket == INVALID_SOCKET) {
    printf("socket failed with error: %d\n", WSAGetLastError());
    freeaddrinfo(addrResult);
    WSACleanup();
    return 1;
}

result = connect(connectSocket, addrResult->ai_addr, (int)addrResult-
>ai_addrlen);
if (result == SOCKET_ERROR) {
    printf("connect failed with error: %d\n", WSAGetLastError());
    closesocket(connectSocket);
    WSACleanup();
    return 1;
}

freeaddrinfo(addrResult);

char buffer[BUFFER_SIZE];
int bufferLen;

while (1) {
    printf("Client: ");
    fgets(buffer, BUFFER_SIZE, stdin);
    bufferLen = strlen(buffer);

    if (buffer[bufferLen - 1] == '\n') {
        buffer[bufferLen - 1] = '\0';
        bufferLen--;
    }

    result = send(connectSocket, buffer, bufferLen, 0);
    if (result > 0) {

```

```

        result = recv(connectSocket, buffer, BUFFER_SIZE, 0);
        if (result > 0) {
            printf("Server: %.*s\n", result, buffer);
        }
        else {
            break;
        }
    }
    else if (result == 0) {
        printf("Connection closed by server.\n");
        break;
    }
    else {
        printf("send() failed with error: %d\n", WSAGetLastError());
        closesocket(connectSocket);
        WSACleanup();
        return 1;
    }
}

closesocket(connectSocket);
WSACleanup();

return 0;
}

```