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 № 7
на тему «Средства обмена данными (Windows). Изучение и использованием средств обмена данными и совместного доступа»

Выполнил:
студент гр. 153504
Подвальников А.С.

Проверил:
Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

1 ЦЕЛИ РАБОТЫ

Изучить средства обмена данными Windows. Изучить средства обмена данными и совместного доступа. Реализовать приложение для обмена текстовыми сообщениями между клиентами по локальной сети с использованием сокетов.

2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Для работы с средствами обмена данными и совместного доступа в Windows разработан ряд функций и объектов.

Очереди сообщений – используются для передачи сообщений между процессами и потоками. Очередь сообщений предоставляет механизмы для отправки сообщений, их приема и обработки. Это позволяет эффективно обмениваться данными и командами между различными частями программы или между разными процессами.

Разделяемая память – представляет собой область памяти, доступную для чтения и записи нескольким процессам и потокам. Разделяемая память позволяет эффективно обмениваться большими объемами данных и обеспечивает быстрый доступ к данным. Это особенно полезно, когда необходимо совместное использование информации между несколькими процессами.

Именованный канал — это именованный односторонний или дуплексный канал для обмена данными между сервером канала и одним или несколькими клиентами канала. Все экземпляры именованного канала имеют одно и то же имя канала, но каждый экземпляр имеет собственные буферы и дескрипторы и предоставляет отдельный канал для обмена данными между клиентом и сервером. Использование экземпляров позволяет нескольким клиентам канала одновременно использовать один и тот же именованный канал.

Заглушки – используются для временного заблокирования выполнения потоков или процессов до наступления определенных условий. Заглушки позволяют синхронизировать потоки и процессы и обеспечивают ожидание выполнения определенных операций или условий. Это может быть полезно, например, для синхронизации потоков при работе с сетевыми запросами или для последовательной обработки данных.

Для создания приложения для обмена текстовыми сообщениями между клиентами по локальной сети с использованием сокетов в Windows можно использовать функции и объекты из Win32 API, такие как сокеты и функции, связанные с сетевым программированием.

Сокеты представляют собой стандартный механизм для обмена данными по сети. Win32 API предоставляет функции для создания, настройки и использования сокетов. Например, функция `socket` позволяет создать сокет, указав тип (например, `SOCK_STREAM` для потокового соединения) и протокол (например, `IPPROTO_TCP` для TCP-соединения).

Для установления соединения и передачи данных между клиентами можно использовать функции `bind` для привязки сокета к определенному адресу и порту, `listen` для прослушивания входящих соединений, `accept` для принятия входящих соединений, а также функции `send` и `recv` для отправки и приема данных.

3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы было реализовано приложение для обмена текстовыми сообщениями между клиентами по локальной сети с использованием сокетов. Результат работы программы показан на рисунке 3.1.

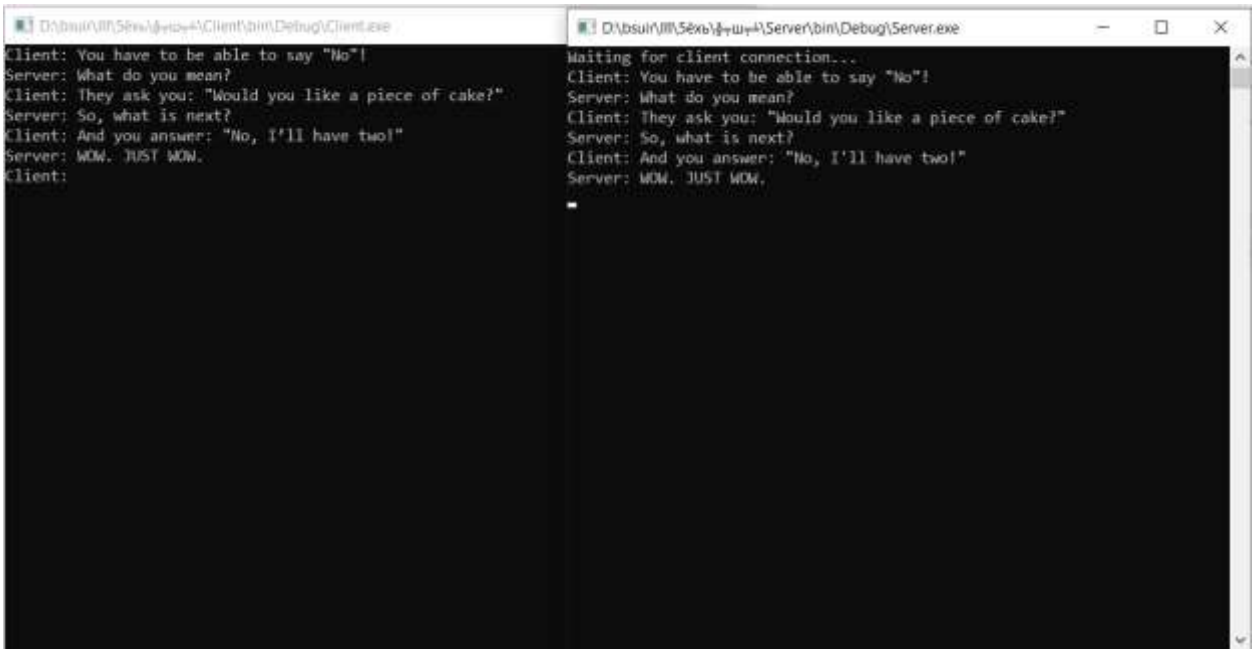

Рисунок 3.1 – Результат работы программы

ВЫВОДЫ

В ходе выполнения данной лабораторной работы были изучены средства обмена данными Windows. Изучены средства обмена данными и совместного доступа. Реализовано приложение для обмена текстовыми сообщениями между клиентами по локальной сети с использованием сокетов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Щупак Ю. Win32 API. Разработка приложений для Windows. – СПб: Питер, 2008. – 592 с.: ип.

[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

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 = WSAStartup(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 = WSAStartup(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;

    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;
}
```