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

ОТЧЁТ

к лабораторной работе №7

на тему

Средства обмена данными (Windows). Изучение и использованием средств обмена данными и совместного доступа

Выполнил: студент группы 153503

Киселёва Елизавета Андреевна

Проверил: Гриценко Никита Юрьевич

Минск 2023

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc151681757)

[2 Краткие теоретические сведения 4](#_Toc151681758)

[3 Результаты выполнения лабораторной работы 7](#_Toc151681759)

[Выводы 9](#_Toc151681760)

[Список использованных источников 10](#_Toc151681761)

[Приложение А (обязательное) Листинг кода 11](#_Toc151681762)

## 1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения лабораторной работы является изучение и использование средств обмена данными и совместного доступа, создание приложения для обмена текстовыми сообщениями между клиентами по локальной сети с использованием сокетов.

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Операционная система *Windows* предоставляет механизмы для упрощения обмена данными и обмена данными между приложениями. В совокупности действия, включённые этими механизмами, называются межпроцессными коммуникациями (*IPC*). Некоторые формы МПК облегчают разделение труда между несколькими специализированными процессами. Другие формы *IPC* облегчают разделение труда между компьютерами в сети.

Как правило, приложения могут использовать *IPC*, классифицированные как клиенты или серверы. Клиент — это приложение или процесс, запрашивающий службу из какого-либо другого приложения или процесса. Сервер — это приложение или процесс, который отвечает на запрос клиента. Многие приложения действуют как клиент, так и сервер, в зависимости от ситуации. Например, приложение текстовой обработки может выступать в качестве клиента при запросе сводной таблицы производственных затрат из приложения электронной таблицы, выступающего в качестве сервера. Приложение электронной таблицы, в свою очередь, может выступать в качестве клиента при запросе последних уровней инвентаризации из приложения автоматического управления запасами.

После того как вы решите, что приложение будет использовать *IPC*, необходимо решить, какой из доступных методов *IPC* использовать. Вполне вероятно, что приложение будет использовать несколько механизмов *IPC*. Ответы на эти вопросы определяют, может ли приложение воспользоваться одним или несколькими механизмами *IPC*.

*Windows* поддерживает следующие механизмы *IPC*:

1 Буфер обмена выступает в качестве центрального хранилища для обмена данными между приложениями. Когда пользователь выполняет операцию вырезания или копирования в приложении, приложение помещает выбранные данные в буфер обмена в одном или нескольких стандартных, или определяемых приложением форматах. Затем любое другое приложение может получить данные из буфера обмена, выбирая из доступных форматов, которые оно понимает.

2 *COM*. Приложения, использующие *OLE*, управляют составными документами, то есть документами, состоящими из данных из различных приложений. *OLE* предоставляет службы, которые упрощают вызов другим приложениям для редактирования данных.

3 Копирование данных позволяет приложению отправлять сведения другому приложению с помощью сообщения *WM\_COPYDATA*. Этот метод требует взаимодействия между отправляющим и принимающим приложением. Принимающее приложение должно знать формат информации и иметь возможность идентифицировать отправителя. Отправляющее приложение не может изменять память, на которую ссылаются никакие указатели.

4 *DDE* — это протокол, позволяющий приложениям обмениваться данными в различных форматах. Приложения могут использовать *DDE* для одноразового обмена данными или для текущих обменов, в которых приложения обновляются друг друга по мере получения новых данных.

5 Сопоставление файлов позволяет процессу обрабатывать содержимое файла как блок памяти в адресном пространстве процесса. Процесс может использовать простые операции указателя для проверки и изменения содержимого файла. Если два или более процессов обращаются к одному сопоставлению файлов, каждый процесс получает указатель на память в своём адресном пространстве, который он может использовать для чтения или изменения содержимого файла. Процессы должны использовать объект синхронизации, например, семафор, чтобы предотвратить повреждение данных в многозадачности.

6 *Mailslots*. Почтовые слои обеспечивают односторонний обмен данными. Любой процесс, создающий *mailslot*, является сервером *mailslot*. Другие процессы, называемые клиентами *mailslot*, отправляют сообщения на сервер *mailslot*, записывая сообщения в его *mailslot*. Входящие сообщения всегда добавляются к почтовому слою. *Mailslot* сохраняет сообщения до тех пор, пока сервер *mailslot* не прочитает их. Процесс может быть, как сервером *mailslot*, так и клиентом *mailslot*, поэтому двусторонняя связь возможна с использованием нескольких почтовых слотов.

7 Каналы. Существует два типа каналов для двусторонней связи: анонимные и именованные каналы. Анонимные каналы позволяют связанным процессам передавать информацию друг другу. Как правило, анонимный канал используется для перенаправления стандартных входных или выходных данных дочернего процесса, чтобы он смог обмениваться данными с родительским процессом. Именованные каналы используются для передачи данных между процессами, которые не являются связанными, и между процессами на разных компьютерах. Как правило, процесс сервера с именованным каналом создаёт именованный канал с известным именем или именем, которое должно быть передано клиентам.

8 *RPC* позволяет приложениям вызывать функции удалённо. Таким образом, *RPC* делает *IPC* так же просто, как вызов функции. *RPC* работает между процессами на одном компьютере или на разных компьютерах в сети.

9 Сокеты *Windows* — это не зависящий от протокола интерфейс. Она использует возможности связи базовых протоколов. В *Windows Sockets* 2 дескриптор сокета можно дополнительно использовать в качестве дескриптора файла со стандартными функциями файлового ввода-вывода. [1]

## 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ   РАБОТЫ

В ходе выполнения лабораторной работы было разработано приложение сетевого локального чата между клиентом и сервером.

При запуске приложения открывается два консольных окна. Первое из них – сервер (рисунок 3.1), второе – клиент (рисунок 3.2).

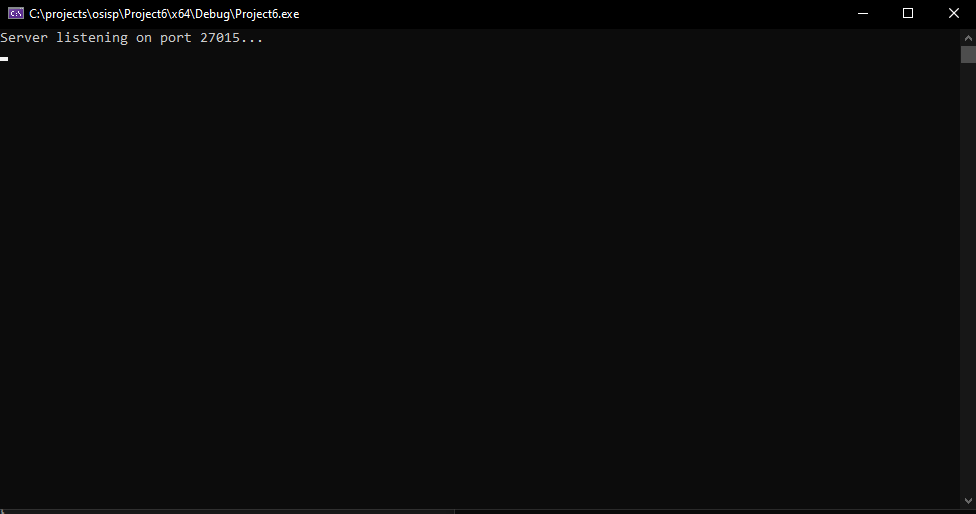
****

Рисунок 3.1 – Окно сервера

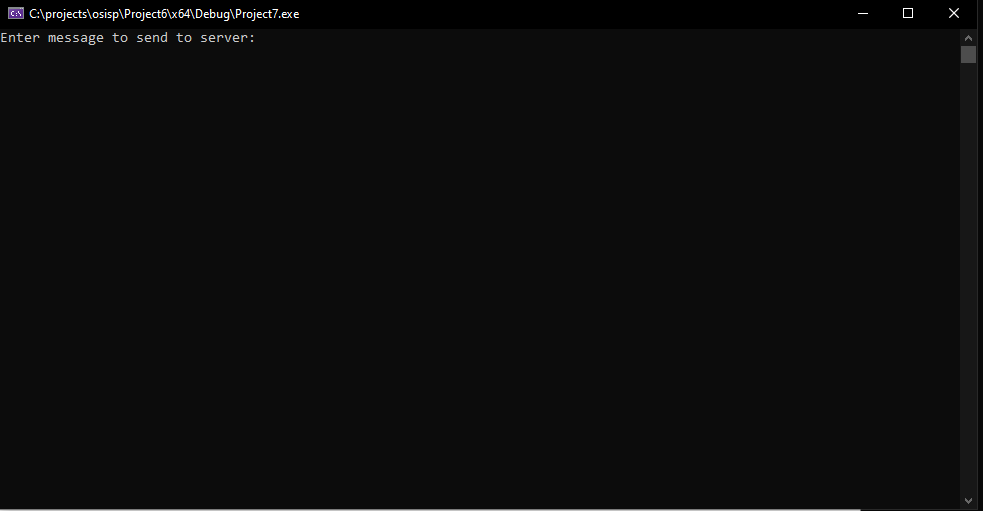


Рисунок 3.2 – Окно клиента

Сервер и клиент имеют возможность обмена сообщениями. Сервер получил сообщение от клиента и отправил ответ (рисунок 3.3).

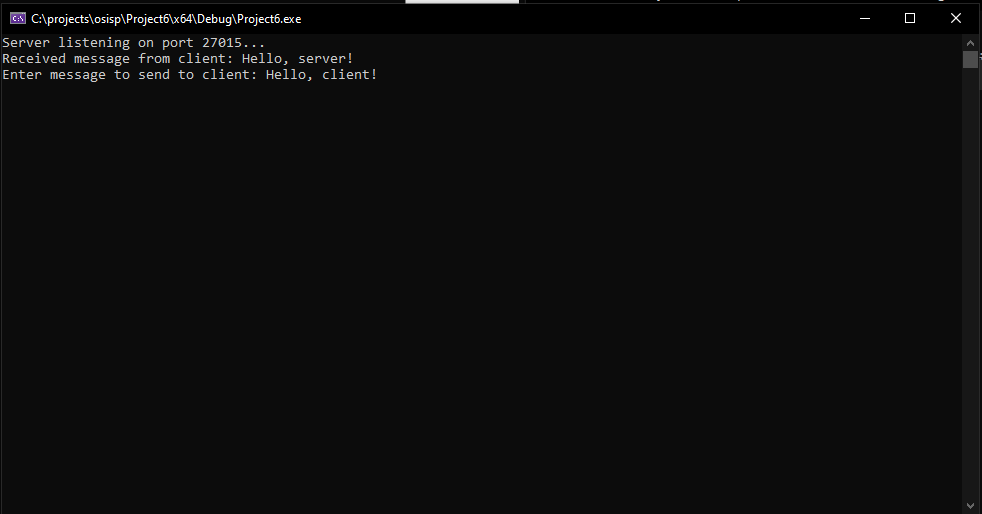
****

Рисунок 3.3 – Работа сервера

Клиент отправил сообщение серверу и получил ответ (рисунок 3.4).

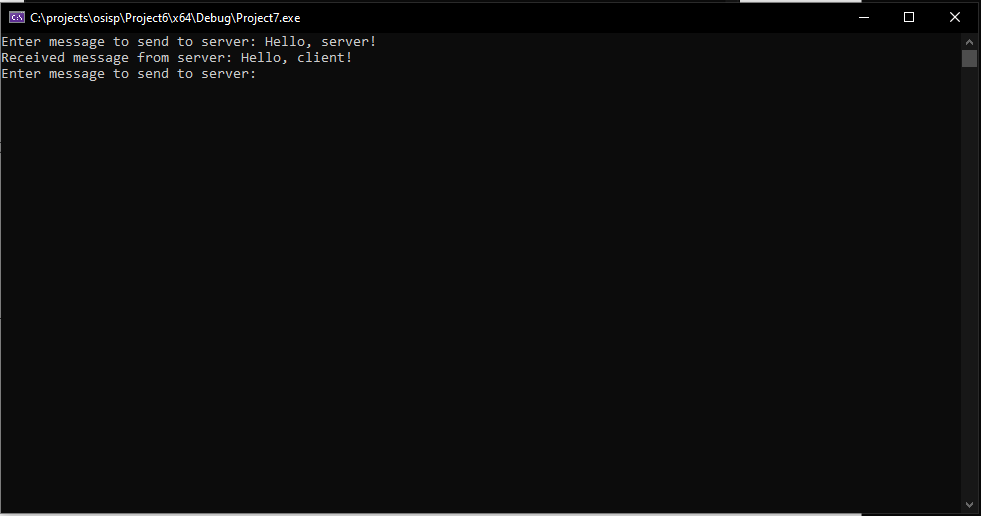


Рисунок 3.4 – Работа клиента

## ВЫВОДЫ

В ходе выполнения данной лабораторной работы были изучены и использованы средства обмена данными и совместного доступа, а также было разработано приложение для обмена текстовыми сообщениями между клиентами по локальной сети с использованием сокетов.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Взаимодействие между процессами [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/ipc/interprocess-communications.

## ПРИЛОЖЕНИЕ А

## (обязательное)

## Листинг кода

**Server.cpp**

#include <iostream>

#include <string>

#define WIN32\_LEAN\_AND\_MEAN

#include <windows.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#pragma comment(lib, "Ws2\_32.lib")

#define DEFAULT\_BUFLEN 512

#define DEFAULT\_PORT "27015"

int main() {

WSADATA wsaData;

int iResult;

SOCKET ListenSocket = INVALID\_SOCKET;

SOCKET ClientSocket = INVALID\_SOCKET;

struct addrinfo\* result = NULL;

struct addrinfo hints;

int recvbuflen = DEFAULT\_BUFLEN;

char recvbuf[DEFAULT\_BUFLEN];

iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);

if (iResult != 0) {

std::cerr << "WSAStartup failed with error: " << iResult << std::endl;

return 1;

}

ZeroMemory(&hints, sizeof(hints));

hints.ai\_family = AF\_INET;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_protocol = IPPROTO\_TCP;

hints.ai\_flags = AI\_PASSIVE;

iResult = getaddrinfo(NULL, DEFAULT\_PORT, &hints, &result);

if (iResult != 0) {

std::cerr << "getaddrinfo failed with error: " << iResult << std::endl;

WSACleanup();

return 1;

}

ListenSocket = socket(result->ai\_family, result->ai\_socktype, result->ai\_protocol);

if (ListenSocket == INVALID\_SOCKET) {

std::cerr << "socket failed with error: " << WSAGetLastError() << std::endl;

freeaddrinfo(result);

WSACleanup();

return 1;

}

iResult = bind(ListenSocket, result->ai\_addr, static\_cast<int>(result->ai\_addrlen));

if (iResult == SOCKET\_ERROR) {

std::cerr << "bind failed with error: " << WSAGetLastError() << std::endl;

freeaddrinfo(result);

closesocket(ListenSocket);

WSACleanup();

return 1;

}

freeaddrinfo(result);

iResult = listen(ListenSocket, SOMAXCONN);

if (iResult == SOCKET\_ERROR) {

std::cerr << "listen failed with error: " << WSAGetLastError() << std::endl;

closesocket(ListenSocket);

WSACleanup();

return 1;

}

std::cout << "Server listening on port " << DEFAULT\_PORT << "...\n";

ClientSocket = accept(ListenSocket, NULL, NULL);

if (ClientSocket == INVALID\_SOCKET) {

std::cerr << "accept failed with error: " << WSAGetLastError() << std::endl;

closesocket(ListenSocket);

WSACleanup();

return 1;

}

closesocket(ListenSocket);

do {

iResult = recv(ClientSocket, recvbuf, recvbuflen, 0);

if (iResult > 0) {

recvbuf[iResult] = '\0'; // Null-terminate the received data

std::cout << "Received message from client: " << recvbuf << std::endl;

// Get user input and send to the client

std::cout << "Enter message to send to client: ";

std::cin.getline(recvbuf, sizeof(recvbuf));

iResult = send(ClientSocket, recvbuf, strlen(recvbuf), 0);

if (iResult == SOCKET\_ERROR) {

std::cerr << "send failed with error: " << WSAGetLastError() << std::endl;

closesocket(ClientSocket);

WSACleanup();

return 1;

}

}

else if (iResult == 0) {

std::cout << "Connection closed by client\n";

break;

}

else {

std::cerr << "recv failed with error: " << WSAGetLastError() << std::endl;

closesocket(ClientSocket);

WSACleanup();

return 1;

}

} while (true);

iResult = shutdown(ClientSocket, SD\_SEND);

if (iResult == SOCKET\_ERROR) {

std::cerr << "shutdown failed with error: " << WSAGetLastError() << std::endl;

closesocket(ClientSocket);

WSACleanup();

return 1;

}

closesocket(ClientSocket);

WSACleanup();

return 0;

}

**Client.cpp**

#include <iostream>

#include <string>

#define WIN32\_LEAN\_AND\_MEAN

#include <windows.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#pragma comment(lib, "Ws2\_32.lib")

#define DEFAULT\_BUFLEN 512

#define DEFAULT\_PORT "27015"

int main(int argc, char\*\* argv) {

WSADATA wsaData;

int iResult;

SOCKET ConnectSocket = INVALID\_SOCKET;

struct addrinfo\* result = NULL, \* ptr = NULL, hints;

char sendbuf[DEFAULT\_BUFLEN];

char recvbuf[DEFAULT\_BUFLEN];

int recvbuflen = DEFAULT\_BUFLEN;

if (argc != 2) {

std::cerr << "usage: " << argv[0] << " <server IP address>\n";

return 1;

}

iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);

if (iResult != 0) {

std::cerr << "WSAStartup failed with error: " << iResult << std::endl;

return 1;

}

ZeroMemory(&hints, sizeof(hints));

hints.ai\_family = AF\_UNSPEC;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_protocol = IPPROTO\_TCP;

iResult = getaddrinfo(argv[1], DEFAULT\_PORT, &hints, &result);

if (iResult != 0) {

std::cerr << "getaddrinfo failed with error: " << iResult << std::endl;

WSACleanup();

return 1;

}

for (ptr = result; ptr != NULL; ptr = ptr->ai\_next) {

ConnectSocket = socket(ptr->ai\_family, ptr->ai\_socktype, ptr->ai\_protocol);

if (ConnectSocket == INVALID\_SOCKET) {

std::cerr << "socket failed with error: " << WSAGetLastError() << std::endl;

WSACleanup();

return 1;

}

iResult = connect(ConnectSocket, ptr->ai\_addr, static\_cast<int>(ptr->ai\_addrlen));

if (iResult == SOCKET\_ERROR) {

closesocket(ConnectSocket);

ConnectSocket = INVALID\_SOCKET;

continue;

}

break;

}

freeaddrinfo(result);

if (ConnectSocket == INVALID\_SOCKET) {

std::cerr << "Unable to connect to server! Error code: " << WSAGetLastError() << std::endl;

WSACleanup();

return 1;

}

do {

// Get user input and send to the server

std::cout << "Enter message to send to server: ";

std::cin.getline(sendbuf, sizeof(sendbuf));

iResult = send(ConnectSocket, sendbuf, strlen(sendbuf), 0);

if (iResult == SOCKET\_ERROR) {

std::cerr << "send failed with error: " << WSAGetLastError() << std::endl;

closesocket(ConnectSocket);

WSACleanup();

return 1;

}

// Receive message from the server

iResult = recv(ConnectSocket, recvbuf, recvbuflen, 0);

if (iResult > 0) {

recvbuf[iResult] = '\0'; // Null-terminate the received data

std::cout << "Received message from server: " << recvbuf << std::endl;

}

else if (iResult == 0) {

std::cout << "Connection closed by server\n";

break;

}

else {

std::cerr << "recv failed with error: " << WSAGetLastError() << std::endl;

closesocket(ConnectSocket);

WSACleanup();

return 1;

}

} while (true);

closesocket(ConnectSocket);

WSACleanup();

return 0;

}