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 гр. 153503

Звягинцева В.А.

Проверил:

Гриценко Н. Ю.

Минск 2023

СОДЕРЖАНИЕ

1 Постановка задачи3

2 Краткие теоретические сведения4

3 Результаты выполнения лабораторной работы7

Вывод9

Список использованных источников10

Приложение А11

**1 ПОСТАНОВКА ЗАДАЧИ**

Целью выполнения лабораторной работы является изучение и использование средств обмена данными и совместного доступа, а также разработка сетевого локального чата между клиентом и сервером 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 открывается два консольных окна. Сервер (рисунок 3.1) и клиент (рисунок 3.2).

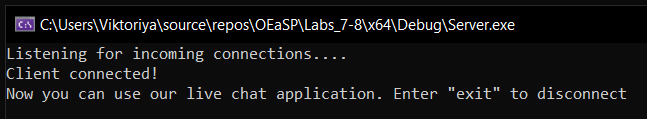


Рисунок 3.1. – Окно сервера

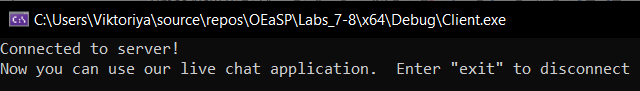


Рисунок 3.2. – Окно клиента

Сервер и клиент могут обмениваться сообщениями. На рисунке 3.3 показано, что сервер отправил сообщение клиенту и получил ответ.

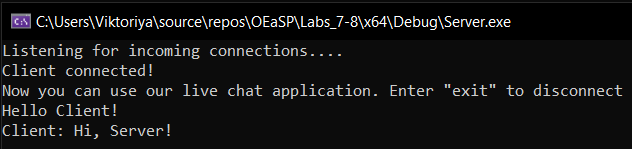


Рисунок 3.3 – Работа сервера

На рисунке 3.4 показано, что клиент получил сообщение от сервера и отправил ответ.

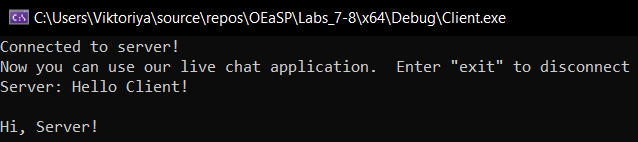


Рисунок 3.4 – Работа клиента

**ВЫВОД**

В результате выполнения лабораторной работы были изучены и использованы средства обмена данными и совместного доступа, а также разработан сетевой локальный чат между клиентом и сервером с использованием сокетов.

**СПИСОК ИСПОЛЬЗУЕМЫХ 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**

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#include <iostream>

#include <cstdio>

#include <cstring>

#include <winsock2.h>

#pragma comment(lib, "WS2\_32.lib")

using namespace std;

DWORD WINAPI serverReceive(LPVOID lpParam) { //Получение данных от клиента

char buffer[1024] = { 0 }; //Буфер для данных

SOCKET client = \*(SOCKET\*)lpParam; //Сокет для клиента

while (true) { //Цикл работы сервера

if (recv(client, buffer, sizeof(buffer), 0) == SOCKET\_ERROR) {

//Если не удалось получить данные буфера, сообщить об ошибке и выйти

cout << "recv function failed with error " << WSAGetLastError() << endl;

return -1;

}

if (strcmp(buffer, "exit\n") == 0) { //Если клиент отсоединился

cout << "Client Disconnected." << endl;

break;

}

cout << "Client: " << buffer << endl; //Иначе вывести сообщение от клиента из буфера

memset(buffer, 0, sizeof(buffer)); //Очистить буфер

}

return 1;

}

DWORD WINAPI serverSend(LPVOID lpParam) { //Отправка данных клиенту

char buffer[1024] = { 0 };

SOCKET client = \*(SOCKET\*)lpParam;

while (true) {

fgets(buffer, 1024, stdin);

if (send(client, buffer, sizeof(buffer), 0) == SOCKET\_ERROR) {

cout << "send failed with error " << WSAGetLastError() << endl;

return -1;

}

if (strcmp(buffer, "exit\n") == 0) {

cout << "Thank you for using the application" << endl;

break;

}

}

return 1;

}

int main() {

WSADATA WSAData; //Данные

SOCKET server, client; //Сокеты сервера и клиента

SOCKADDR\_IN serverAddr, clientAddr; //Адреса сокетов

WSAStartup(MAKEWORD(2, 0), &WSAData);

server = socket(AF\_INET, SOCK\_STREAM, 0); //Создали сервер

if (server == INVALID\_SOCKET) {

cout << "Socket creation failed with error:" << WSAGetLastError() << endl;

return -1;

}

serverAddr.sin\_addr.s\_addr = INADDR\_ANY;

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(5555);

if (bind(server, (SOCKADDR\*)&serverAddr, sizeof(serverAddr)) == SOCKET\_ERROR) {

cout << "Bind function failed with error: " << WSAGetLastError() << endl;

return -1;

}

if (listen(server, 0) == SOCKET\_ERROR) { //Если не удалось получить запрос

cout << "Listen function failed with error:" << WSAGetLastError() << endl;

return -1;

}

cout << "Listening for incoming connections...." << endl;

char buffer[1024]; //Создать буфер для данных

int clientAddrSize = sizeof(clientAddr); //Инициализировать адерс клиента

if ((client = accept(server, (SOCKADDR\*)&clientAddr, &clientAddrSize)) != INVALID\_SOCKET) {

//Если соединение установлено

cout << "Client connected!" << endl;

cout << "Now you can use our live chat application. " << "Enter \"exit\" to disconnect" << endl;

DWORD tid; //Идентификатор

HANDLE t1 = CreateThread(NULL, 0, serverReceive, &client, 0, &tid); //Создание потока для получения данных

if (t1 == NULL) //Ошибка создания потока

cout << "Thread Creation Error: " << WSAGetLastError() << endl;

HANDLE t2 = CreateThread(NULL, 0, serverSend, &client, 0, &tid); //Создание потока для отправки данных

if (t2 == NULL)

cout << "Thread Creation Error: " << WSAGetLastError() << endl;

WaitForSingleObject(t1, INFINITE);

WaitForSingleObject(t2, INFINITE);

closesocket(client); //Закрыть сокет

if (closesocket(server) == SOCKET\_ERROR) { //Ошибка закрытия сокета

cout << "Close socket failed with error: " << WSAGetLastError() << endl;

return -1;

}

WSACleanup();

}

}

**client.cpp**

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#include <iostream>

#include <cstdio>

#include <cstring>

#include <winsock2.h>

#pragma comment(lib, "WS2\_32.lib")

using namespace std;

DWORD WINAPI clientReceive(LPVOID lpParam) { //Получение данных от сервера

char buffer[1024] = { 0 };

SOCKET server = \*(SOCKET\*)lpParam;

while (true) {

if (recv(server, buffer, sizeof(buffer), 0) == SOCKET\_ERROR) {

cout << "recv function failed with error: " << WSAGetLastError() << endl;

return -1;

}

if (strcmp(buffer, "exit\n") == 0) {

cout << "Server disconnected." << endl;

return 1;

}

cout << "Server: " << buffer << endl;

memset(buffer, 0, sizeof(buffer));

}

return 1;

}

DWORD WINAPI clientSend(LPVOID lpParam) { //Отправка данных на сервер

char buffer[1024] = { 0 };

SOCKET server = \*(SOCKET\*)lpParam;

while (true) {

fgets(buffer, 1024, stdin);

if (send(server, buffer, sizeof(buffer), 0) == SOCKET\_ERROR) {

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

return -1;

}

if (strcmp(buffer, "exit") == 0) {

cout << "Thank you for using the application" << endl;

break;

}

}

return 1;

}

int main() {

WSADATA WSAData;

SOCKET server;

SOCKADDR\_IN addr;

WSAStartup(MAKEWORD(2, 0), &WSAData);

if ((server = socket(AF\_INET, SOCK\_STREAM, 0)) == INVALID\_SOCKET) {

cout << "Socket creation failed with error: " << WSAGetLastError() << endl;

return -1;

}

addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1"); //коннект к серверу

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(5555); //порт

if (connect(server, (SOCKADDR\*)&addr, sizeof(addr)) == SOCKET\_ERROR) {

cout << "Server connection failed with error: " << WSAGetLastError() << endl;

return -1;

}

cout << "Connected to server!" << endl;

cout << "Now you can use our live chat application. " << " Enter \"exit\" to disconnect" << endl;

DWORD tid;

HANDLE t1 = CreateThread(NULL, 0, clientReceive, &server, 0, &tid);

if (t1 == NULL) cout << "Thread creation error: " << GetLastError();

HANDLE t2 = CreateThread(NULL, 0, clientSend, &server, 0, &tid);

if (t2 == NULL) cout << "Thread creation error: " << GetLastError();

WaitForSingleObject(t1, INFINITE);

WaitForSingleObject(t2, INFINITE);

closesocket(server);

WSACleanup();

}