|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра теоретической и прикладной информатики | | |
| Лабораторная работа № 3 | | |
| по дисциплине «Компьютерные сети» | | |
| **Разработка приложения интерактивной переписки** | | |
|  | | |
|  | Бригада 11 | Затолоцкая юлия |
| Группа ПМ-91 | константинова анастасия |
|  |  |
|  |  |
| Преподаватели | Сивак Мария Алексеевна |
|  | Кобылянский Валерий Григорьевич |
| Дата | 02.11.2021 |
|  |  |
| Новосибирск, 2021 | | |

1. **Цель работы**

Изучить основные принципы разработки многопользовательских приложений, построенных на основе технологии клиент-сервер с использованием стека протоколов TCP/IP.

1. **Задание**

С помощью API-интерфейса реализовать простой чат. Каждая бригада должна написать chat-сервер и chat-клиента. Сервер должен поддерживать соединение сразу от нескольких клиентов. Обмен между клиентами осуществляется через сервер. При получении сообщения от какого-либо клиента, сервер дублирует его на своем экране и оповещает всех подсоединенных клиентов, отправляя каждому из них данное со-общение. При подсоединении нового клиента к chat-серверу, сервер оповещает каж-дого клиента о новом пользователе, посылая им его IP-адрес и имя.

1. **Код программы**

**Сервер:**

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#define \_CRT\_SECURE\_NO\_WARNINGS

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

#include <WinSock2.h>

#include <stdio.h>

#include <iostream>

#include <sstream>

#include <string>

#include <thread>

using namespace std;

const int N = 100; // Количество пользователей

const int M = 1000; // Размер запроса/ответа

const int maxAmount = 20; //макс число пользователей на сервере

SOCKET servSock;

int curAmount; //текущее число пользователей

//информация о пользователях

SOCKET clSockets[maxAmount];

SOCKADDR\_IN clSADDR[maxAmount];

USHORT ports[maxAmount];

char names[maxAmount + 1][N];

DWORD WINAPI chatClient(LPVOID clientSocket)

{

int retVal;

char szReq[M];

SOCKET clientSock;

int cur;

int i, j;

char szResp[M];

clientSock = \*((SOCKET\*)clientSocket);

while (true)

{

//получаем сообщение от клиента

retVal = recv(clientSock, szReq, M, 0);

if (retVal == SOCKET\_ERROR)

{

printf("Unable to recv\n");

closesocket(clientSock);

printf("Connection closed\n");

return SOCKET\_ERROR;

}

else

{

if (retVal >= M) retVal = M - 1;

szReq[retVal] = '\0';

}

printf("Data received\n");

SOCKADDR\_IN sin;

for (i = 0; i < curAmount; i++)

{

if (clSockets[i] == clientSock)

sin = clSADDR[i];

}

cur = 0;

//поиск имени пользователя

while (ports[cur] != sin.sin\_port)

cur++;

//

//если пользователь решил выйти

if (!strcmp(szReq, "s"))

{

//оповестить всех других пользователей о выходе пользователя

szResp[0] = '\0';

strcat\_s(szResp, names[cur]);

strcat\_s(szResp, " leaved chat");

for (i = 0; i < curAmount; i++)

{

if (clSockets[i] != clientSock) retVal = send(clSockets[i], szResp, M, 0);

}

if (retVal == SOCKET\_ERROR)

{

printf("Unable to send\n");

return SOCKET\_ERROR;

}

printf("Client disconnected\n");

closesocket(clientSock);

printf("Connection closed\n");

//удаляем информацию о пользователе

for (j = cur; j < curAmount; j++)

{

clSockets[j] = clSockets[j + 1];

clSADDR[j] = clSADDR[j + 1];

ports[j] = ports[j + 1];

strcpy\_s(names[j], names[j + 1]);

}

clSockets[curAmount - 1] = SOCKET\_ERROR;

curAmount--;

printf("Current amount of clients: %i\n", curAmount);

return SOCKET\_ERROR;

}

//если принятое сообщение - не пустая строка

if (szReq[0] != '\0')

{

//выводим имя пользователя и его сообщение

printf("%s: %s\n", names[cur], szReq);

szResp[0] = '\0';

strcat\_s(szResp, names[cur]);

strcat\_s(szResp, ": ");

strcat\_s(szResp, szReq);

printf("Sending response from server\n");

//и отсылаем всем пользователям, кроме него самого

for (i = 0; i < curAmount; i++)

{

if (clSockets[i] != clientSock)

retVal = send(clSockets[i], szResp, M, 0);

}

if (retVal == SOCKET\_ERROR)

{

printf("Unable to send\n");

return SOCKET\_ERROR;

}

}

}

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

WORD sockVer;

WSADATA wsaData;

int retVal;

int i;

curAmount = 0;

sockVer = MAKEWORD(2, 2);

WSAStartup(sockVer, &wsaData);

//Создаем сокет

servSock = socket(PF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

if (servSock == INVALID\_SOCKET)

{

printf("Unable to create socket\n");

WSACleanup();

system("pause");

return SOCKET\_ERROR;

}

for (i = 0; i < maxAmount; i++)

{

clSockets[i] = SOCKET\_ERROR;

}

SOCKADDR\_IN sin;

sin.sin\_family = PF\_INET;

sin.sin\_port = htons(2011);

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

retVal = bind(servSock, (LPSOCKADDR)&sin, sizeof(sin));

if (retVal == SOCKET\_ERROR)

{

printf("Unable to bind\n");

WSACleanup();

system("pause");

return SOCKET\_ERROR;

}

char host[N];

char HostName[1024]; //создаем буфер для имени хоста

if (!gethostname(HostName, 1024)) //получаем имя хоста

{

if (LPHOSTENT lphost = gethostbyname(HostName)) //получаем IP сервера

strcpy(host, inet\_ntoa(\*((in\_addr\*)lphost->h\_addr\_list[0])));

//преобразуем переменную типа LPIN\_ADDR в DWORD

}

printf("Server is working at %s, port %d\n", host, htons(sin.sin\_port));

while (true)

{

//Пытаемся начать слушать сокет

retVal = listen(servSock, 10);

if (retVal == SOCKET\_ERROR)

{

printf("Unable to listen\n");

WSACleanup();

system("pause");

return SOCKET\_ERROR;

}

//Ждем клиента

SOCKET clientSock;

SOCKADDR\_IN from;

int fromlen = sizeof(from);

clientSock = accept(servSock, (struct sockaddr\*)&from, &fromlen);

if (clientSock == INVALID\_SOCKET)

{

printf("Unable to accept\n");

WSACleanup();

system("pause");

return SOCKET\_ERROR;

}

//установлено новое соединение

printf("New client from %s, port %d\n", inet\_ntoa(from.sin\_addr),

htons(from.sin\_port));

printf("Current amount of clients: %i\n", curAmount + 1);

char NewClient[M];

//получаем имя нового клиента

retVal = recv(clientSock, names[curAmount], N, 0);

//Пытаемся получить данные от клиента

if (retVal == SOCKET\_ERROR)

{

printf("Unable to recv\n");

system("pause");

return SOCKET\_ERROR;

}

//если имя - команда на закрытие сервера

if (!strcmp(names[curAmount], "stop"))

{

char\* szResp = (char\*)"Server closed.";

//посылаем сообщение о закрытии всем клиентам и закрываем сокеты

for (i = 0; i < curAmount; i++)

{

retVal = send(clSockets[i], szResp, M, 0);

closesocket(clSockets[i]);

}

retVal = send(clientSock, szResp, M, 0);

closesocket(clientSock);

break;

}

else

{

//если не достиг максимум пользователей

if (curAmount < maxAmount)

{

//сохраняем и выводим информацию о текущем пользователе

ports[curAmount] = from.sin\_port;

clSockets[curAmount] = clientSock;

clSADDR[curAmount] = from;

NewClient[0] = '\0';

strcat\_s(NewClient, names[curAmount]);

strcat\_s(NewClient, " join to chat! IP: ");

strcat\_s(NewClient, inet\_ntoa(from.sin\_addr));

printf\_s("%s\n", NewClient);

curAmount++;

//рассылаем всем клиентам сообщение о новом пользователе

for (i = 0; i < curAmount; i++)

if (clSockets[i] != clientSock) retVal = send(clSockets[i], NewClient, M, 0);

}

else

{

//если достигнут максимум, отправляем пользователю сообщение о достижении максимума и закрываем сокет

printf("Maximum amount of clients\n");

retVal = send(clientSock, "FULL.", N, 0);

closesocket(clientSock);

printf("Connection closed\n");

}

DWORD threadName;

CreateThread(NULL, NULL, chatClient, &clientSock, NULL, &threadName);

}

}

closesocket(servSock);

WSACleanup();

return 0;

}

**Клиент:**

using System;

using System.Text;

using System.Net.Sockets;

namespace client3

{

class Program

{

static void recvMessage(Socket client)

{

try

{

Byte[] data = new Byte[1000];

client.Receive(data, SocketFlags.None);

int i = 0;

for (; data[i] != 0;)

i++;

String responseData = Encoding.ASCII.GetString(data, 0, i);

//печатаем то, что получили

Console.WriteLine("Received: {0}", responseData);

}

catch (ArgumentNullException expt)

{

Console.WriteLine("ArgumentNullException: {0}", expt);

}

catch (SocketException expt)

{

Console.WriteLine("SocketException: {0}", expt);

}

Console.WriteLine("\n Press Enter to continue...");

Console.Read();

}

static void Main(string[] args)

{

try

{

Console.WriteLine("Enter ip of server:");

string localhost = Console.ReadLine();

Console.WriteLine("Enter port of server: ");

Int32 port = Convert.ToInt32(Console.ReadLine());

Socket client = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

client.Connect(localhost, port);

Console.WriteLine("Enter name:");

string message = Console.ReadLine(); // строка, которую пошлем серверу

Byte[] data = System.Text.Encoding.ASCII.GetBytes(message);

client.Send(data, data.Length, SocketFlags.None);

while (true)

{

message = Console.ReadLine(); // строка, которую пошлем серверу

data = System.Text.Encoding.ASCII.GetBytes(message);

Console.WriteLine("Sent: {0}", message);//печатаем то, что отправили

client.Send(data, data.Length, SocketFlags.None);

// буффер для приема сообщений

data = new Byte[1000];

System.Threading.Thread th\_add = new System.Threading.Thread(delegate () { recvMessage(client); });

th\_add.Start();

// строка для приема сообщений сервера

/\*client.Receive(data, SocketFlags.None);

int i = 0;

for (; data[i] != 0;)

i++;

String responseData = Encoding.ASCII.GetString(data, 0, i);

//печатаем то, что получили

Console.WriteLine("Received: {0}", responseData);\*/

}

// закрываем соединение

client.Close();

}

catch (ArgumentNullException expt)

{

Console.WriteLine("ArgumentNullException: {0}", expt);

}

catch (SocketException expt)

{

Console.WriteLine("SocketException: {0}", expt);

}

Console.WriteLine("\n Press Enter to continue...");

Console.Read();

}

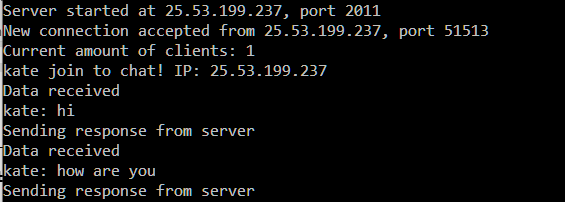
}

}

**Результаты тестирования**

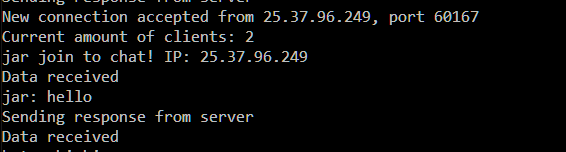
Запустим сервер. Подключим клента.

*Сервер*



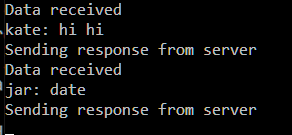
Запустим клиента на другом компьютере и подключимся к серверу.

*Сервер*



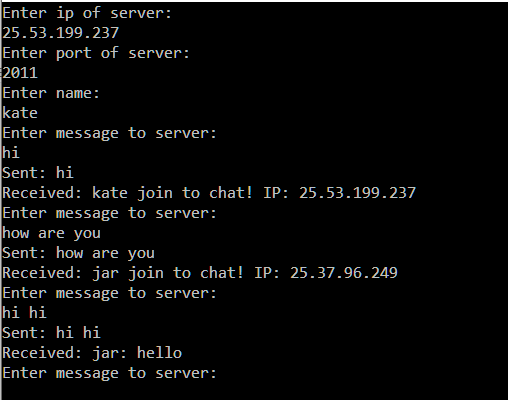
Сервер получил сообщение, ответ клиенту отправлен успешно.

Второй клиент в это время вводит сообщение. Отправим сообщение, чтобы убедиться, что буфер не был использован вторым потоком.

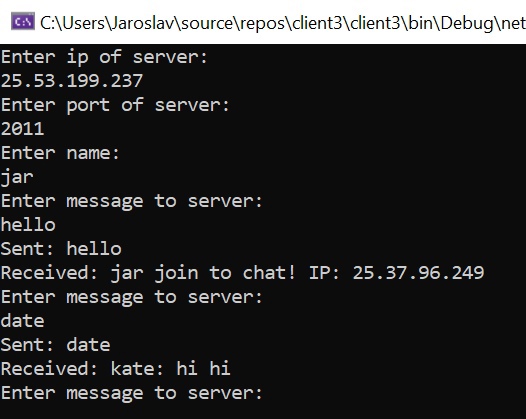


Приложение со стороны клиентов:

*Клиент 1*



*Клиент 2*



1. **Выводы**

Контрольные вопросы проработаны.