Министерство науки и высшего образования

Российской федерации

Федеральное государственное бюджетное

Образовательное учреждение высшего образования

«Новосибирский Государственный Технический Университет»

Кафедра теоретической и прикладной информатики

Лабораторная работа №3

«Разработка приложения интерактивной переписки»

Факультет: прикладной математики и информатики

Группа: ПМИ-12

Бригада: 1

Студенты: Михайловский М.А.

Швадченко А.В.

Преподаватели: Кобылянский В.Г.

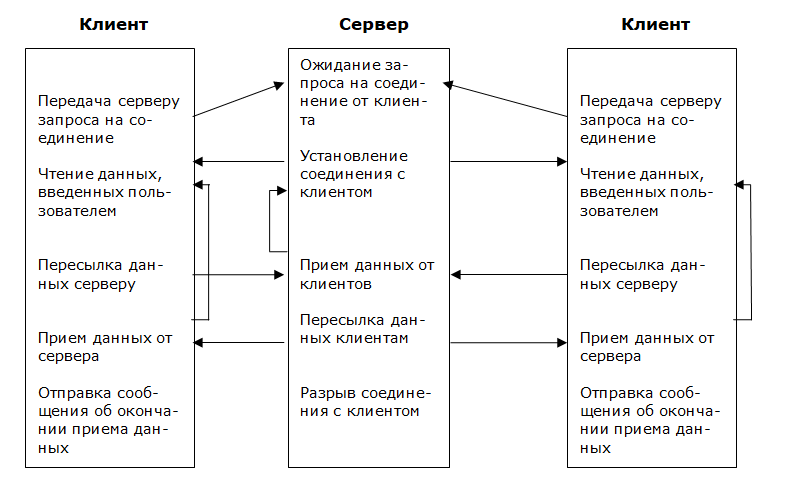
Филиппова Е.В.

Новосибирск, 2023

1. Цель работы:

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

1. Выполнение работы:
   1. Схема взаимодействия сервера с клиентами



* 1. Текст программы

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

#define \_CRT\_SECURE\_NO\_WARNINGS

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#include <WinSock2.h>

#include <stdio.h>

#include <iostream>

#include <sstream>

#include <string>

using namespace std;

const int N = 256;

const int M = 1000;

const int maxAmount = 5; //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 chat(LPVOID clientSocket)

{

int retVal;

char szReq[M];

SOCKET clientSock;

int cur;

int i, j;

char szResp[M];

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

while (true)

{

//получить 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(szResp, names[cur]);

strcat(szResp, " left the 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(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(szResp, names[cur]);

strcat(szResp, ": ");

strcat(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()

{

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(2001);

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 started 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 connection accepted 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], "admin\_shut"))

{

char array[] = "";

const char\* szResp = "Server shutdown";

printf("Server shutdown\n");

//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(NewClient, "New client, ip: ");

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

strcat(NewClient, "; Name: ");

strcat(NewClient, names[curAmount]);

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

curAmount++;

//рассылаем всем клиентам сообщение о новом пользователе, кроме него самого

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, "Sorry, too much people on the line", N, 0);

closesocket(clientSock);

printf("Connection closed\n");

}

//чат

DWORD threadID;

CreateThread(NULL, NULL, chat, &clientSock, NULL, &threadID);

}

}

closesocket(servSock);

WSACleanup();

return 0;

}

* 1. Клиент

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

#define \_CRT\_SECURE\_NO\_WARNINGS

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#include <WinSock2.h>

#include <stdio.h>

#include <conio.h>

#include <iostream>

#include <string>

#include <cstdio>

using namespace std;

const int M = 1000;

const int N = 256;

bool fin;

DWORD WINAPI receive(LPVOID clientSocket)

{

int retVal;

SOCKET clientSock;

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

char Resp[M];

//сообщение от сервера

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

//если это сообщение закрытия сервера

if (!strcmp(Resp, "Server shutdown"))

{

printf("Server shutdown\n");

fin = true;

return 0;

}

//если сервер переполнен

if (!strcmp(Resp, "Sorry, too much people on the line"))

{

printf("Sorry, too much people on the line\n");

fin = true;

return 0;

}

//если клиент все еще работает

if (!fin)

{

if (retVal == SOCKET\_ERROR)

{

//ошибка, завершение работы

retVal = 0;

printf("Unable to recv\n");

fin = true;

return 0;

}

else

{

//напечатать сообщение сервера

printf("%s\n", Resp);

}

}

return 1;

}

DWORD WINAPI send(LPVOID clientSocket)

{

int retVal;

char pBuf[M];

SOCKET clientSock;

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

gets\_s(pBuf);

//если введенная строка - закрытие клиента

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

{

fin = true;

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

return 0;

}

else

{

//отправить сообщение серверу

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

if (retVal == SOCKET\_ERROR)

{

printf("Unable to send\n");

WSACleanup();

system("pause");

return 0;

}

}

return 1;

}

int main()

{

WSADATA wsaData;

int retVal = 0;

fin = false;

char name[N];

int host;

WORD ver = MAKEWORD(2, 2);

WSAStartup(ver, (LPWSADATA)&wsaData);

LPHOSTENT hostEnt;

hostEnt = gethostbyname("localhost");

if (!hostEnt)

{

cout << "unable to collect receivehostbyname" << endl;

WSACleanup();

return 1;

}

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

if (clientSock == SOCKET\_ERROR)

{

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

WSACleanup();

system("pause");

return 1;

}

string ip;

cout << "ip>";

cin >> ip;

cin.ignore();

cout << "host>";

cin >> host;

SOCKADDR\_IN serverInfo;

serverInfo.sin\_family = PF\_INET;

serverInfo.sin\_addr.S\_un.S\_addr = inet\_addr(ip.c\_str());

serverInfo.sin\_port = htons(host);

retVal = connect(clientSock, (LPSOCKADDR)&serverInfo, sizeof(serverInfo));

if (retVal == SOCKET\_ERROR)

{

printf("Unable to connect\n");

WSACleanup();

system("pause");

return 1;

}

printf("Connection made successfully\n");

printf("Enter your name: ");

scanf("%s", name);

//отправить имя серверу

retVal = send(clientSock, name, N, 0);

if (retVal == SOCKET\_ERROR)

{

cout << "unable to send" << endl;

WSACleanup();

return 1;

}

printf("Enter 's' to finish chat\n");

//пока не завершена работа клиента/сервера

while (!fin)

{

DWORD threadID;

CreateThread(NULL, NULL, send, &clientSock, NULL, &threadID);

CreateThread(NULL, NULL, receive, &clientSock, NULL, &threadID);

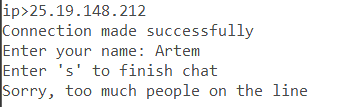
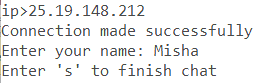
}

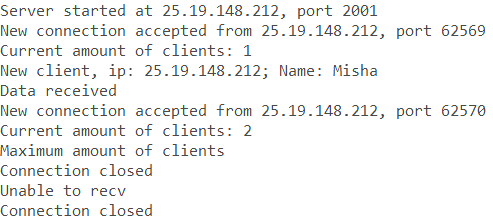
closesocket(clientSock);

WSACleanup();

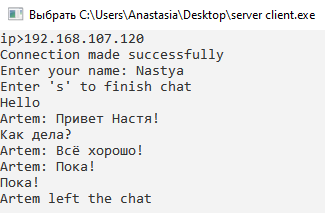
return 0;

}

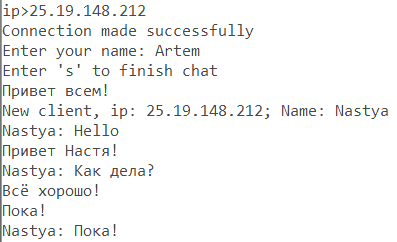
* 1. Тестирование
     1. Достижение максимума пользователей
        1. Первый клиент  
           
        2. Второй клиент
        3. Сервер

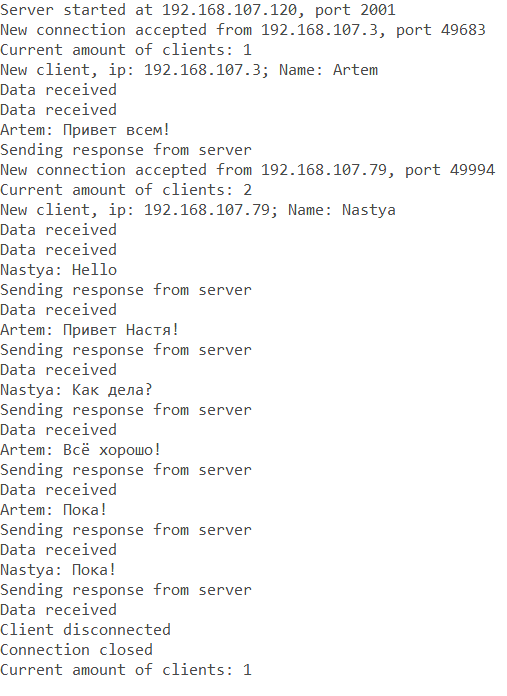


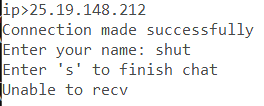
* + 1. Стандартная работа чата
       1. Первый клиент



* + - 1. Второй клиент



* + - 1. Сервер  
         
    1. Закрытие сервера
       1. Клиент с именем shut

* + - 1. Сервер

