|  |  |  |
| --- | --- | --- |
| Министерство науки и высшего образования  Российской Федерации | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования | | |
| «Новосибирский государственный технический университет» | | |
|  | | |
| Теоретической и прикладной математики | | |
|  | | |
| Лабораторная работа № 3 | | |
| по дисциплине «Операционные системы, среды и оболочки» | | |
|  | | |
| **Разработка приложения интерактивной переписки** | | |
|  | | |
|  | Факультет: | ПМИ |
| Группа: | ПМИ-02 |
| Вариант: | 6 |
| Студент: | Сидоров Даниил, |
|  | Дюков Богдан |
| Преподаватель: | Кобылянский Валерий Григорьевич, |
|  | Филиппова Елена Владимировна  . |
|
|  |  |
|  | | |
| Новосибирск | | |
| 2022 | | |

1. **Цель работы**

Изучить основные принципы разработки многопользовательских приложе-ний, построенных на основе технологии клиент-сервер с использованием стека протоколов TCP/IP. С помощью API-интерфейса реализовать простой чат.

1. **Ход работы**

Программы сервера и клиента написаны на C++ с использованием Winsock.

Алгоритм работы программы сервера:

1) Инициализация библиотеки

2) Создание сокета

3) Привязка сокета к конкретному адресу

4) Переход в режим «прослушивания»

5) Бесконечная обработка запросов на установление соединения

6) Создание для каждого соединения нового потока «ClientThread»

Все подключения обрабатываются в отдельных потоках, реализованных через WINAPI.

При получении любого сообщения (кроме команды выхода из чата) от клиента, сервер пересылает это сообщение всем прочим участникам.

Если сервер переполнен и какой-то клиент пытается к нему подключиться, то сервер уведомляет его об этом и разрывает соединение.

При получении сообщения “Exit” от клиента сервер разрывает с ним соединение и уведомляет всех участников о его отключении.

С целью обеспечения безопасности данных сервера, таких как массив сокетов подключенных пользователей, массив информации об адресах этих сокетов и массив имён пользователей, для потоков был использован Мьютекс.

Алгоритм работы программы клиента:

1) Инициализация библиотеки

2) Создание сокета

3) Считывание из консоли IP-адреса и порта сервера

4) Установление соединения с сервером

6) Если сервер запрашивает имя, то ввод и отправка имени

7) Многопоточная работа с сервером

Пользователь может одновременно отправлять и получать сообщения. Помимо главного потока его процесс создает ещё 2 потока: поток на отправку и на приём сообщений с сервера.

**Описание функций, которые выполняются потоком**

Chat – ожидает данные от клиента, отсылает полученные от клиента сообщения или информацию о его выходе другим пользователям чата.

Send – ожидает нажатие ENTER, после которого появляется приглашение на ввод сообщения. После ввода сообщения оно отправляется на сервер.

Receive – ожидает данные с сервера, а после получения они выводятся на экран.

**Для работы с потоками использованы следующие функции:**

CreateThread – Создание потока, выполняющего указанную функцию.

WaitForSingleObject – Ожидание завершения потока.

CloseHandle – Закрытие потока.

ReleaseMutex(mutex) – Разблокировка Мьютекса.

**Схема взаимодействия сервера с клиентами**

Передача серверу запроса на соединение

Чтение данных, введенных пользователем

Пересылка данных серверу

Прием данных от сервера

Отправка сообщения об окончании приема данных

**Клиент**

Ожидание запроса на соединение от клиента

Установление соединения с клиентом

Прием данных от клиентов

Пересылка данных клиентам

Разрыв соединения с клиентом

**Сервер**

Передача серверу запроса на соединение

Чтение данных, введенных пользователем

Пересылка данных серверу

Прием данных от сервера

Отправка сообщения об окончании приема данных

**Клиент**

Рис.3.1 Схема взаимодействия Chat-сервера с клиентами

1. **Код Сервера**

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

#include <WinSock2.h>

#include <stdio.h>

#include <iostream>

#include <sstream>

#include <string>

using namespace std;

// Максимальный размер имени пользователя и сообщения (и от клиента и от сервера)

const int maximumNameSize = 100;

const int maximumMessageSize = 1000;

// Максимальное число пользователей на сервере

const int maxAmount = 100;

// Текущее число пользователей

int curAmount = 0;

// Хранение сокетов пользователей, информации об адресах сокетов и имён пользователей

SOCKET sockets[maxAmount];

SOCKADDR\_IN clSADDR[maxAmount];

char names[maxAmount + 1][maximumNameSize];

HANDLE mutex;

// Послать информацию конкретного пользователя (его сообщение или его выход из чата) другим пользователям

DWORD NotifyUsers(int cur, char messageToOtherUsers[])

{

int retVal = 0;

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

{

if (i != cur)

{

retVal = send(sockets[i], messageToOtherUsers, maximumMessageSize, 0);

}

}

if (retVal == SOCKET\_ERROR)

{

return SOCKET\_ERROR;

}

return 0;

}

// Функция, которая будет выполняться потоком, параметр функции типа LPVOID - это указатель на любой тип

DWORD WINAPI Chat(LPVOID clientSocket)

{

// Массив содержит какую то информацию о клиенте (его имя + сообщение, информацию о его выходе и т. д.), которая

// посылается другим пользователям чата

char messageToOtherUsers[maximumMessageSize];

int retVal;

char szReq[maximumMessageSize];

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

while (true)

{

int cur;

// Поиск информации об адресе сокета пользователя

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

{

if (sockets[cur] == clientSock)

{

break;

}

}

// Пытаемся получить данные от клиента, клиент должен отправить какое-то сообщение, которое запишется в szReq

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

// Если пользователь выходит из чата

if (!strcmp(szReq, "Exit\n") || retVal == SOCKET\_ERROR)

{

// Выводим соответствующие сообщения на экран сервера

printf("Client disconnected\n");

printf("Connection closed\n");

if (retVal == SOCKET\_ERROR)

{

printf("Unable to recv\n");

}

// Сконструируем сообщение для остальных пользователей

messageToOtherUsers[0] = '\0';

strcat(messageToOtherUsers, names[cur]);

strcat(messageToOtherUsers, " left the chat\n");

// Отправим сообщение о выходе пользователя другим пользователям

if (curAmount > 1 && NotifyUsers(cur, messageToOtherUsers) == SOCKET\_ERROR)

{

printf("Unable to send\n");

}

// Блокируем мьютекс для обработки клиентских данных

WaitForSingleObject(mutex, INFINITE);

// Удалим всю информацию о пользователе

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

{

sockets[j] = sockets[j + 1];

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

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

}

// Очищаем освободившееся место

sockets[curAmount - 1] = SOCKET\_ERROR;

curAmount--;

// Разблокируем мьютекс

ReleaseMutex(mutex);

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

closesocket(clientSock);

return SOCKET\_ERROR;

}

printf("Data received\n");

// Отсылаем полученное от пользователя сообщение (не пустое) всем остальным пользователям (и выводим его на экран сервера)

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

{

// Выводим сообщение на экран сервера

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

// Сконструируем сообщение для остальных пользователей

messageToOtherUsers[0] = '\0';

strcat(messageToOtherUsers, names[cur]);

strcat(messageToOtherUsers, ": ");

strcat(messageToOtherUsers, szReq);

// Отправим сообщение от пользователя другим пользователям

if (NotifyUsers(cur, messageToOtherUsers) == SOCKET\_ERROR)

{

printf("Unable to send\n");

closesocket(clientSock);

return SOCKET\_ERROR;

}

}

}

}

int main()

{

WSADATA wsaData;

// Загрузка (инициализация) библиотеки

if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)

{

printf("Error WinSock version initializaion\n");

WSACleanup();

system("pause");

return SOCKET\_ERROR;

}

// Создаем сокет

SOCKET 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 (int i = 0; i < maxAmount; i++)

{

sockets[i] = SOCKET\_ERROR;

}

// Заполнение информации об адресе сокета

SOCKADDR\_IN sin;

sin.sin\_family = PF\_INET;

sin.sin\_port = htons(2006);

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

// Привязка сокета к конкретному адресу (параметры: дескриптор сокета, указатель на структуру sin, размер структуры)

if (bind(servSock, (LPSOCKADDR)&sin, sizeof(sin)) == SOCKET\_ERROR)

{

printf("Unable to bind\n");

WSACleanup();

system("pause");

return SOCKET\_ERROR;

}

char host[maximumNameSize]; // Буфер для ip-адреса хоста

char hostName[1024]; // Буфер для имени хоста

// Сначала получаем имя хоста, после получаем ip хоста по его имени

if (gethostname(hostName, 1024) == 0)

{

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

}

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

while (true)

{

// Пытаемся начать слушать сокет

if (listen(servSock, 10) == SOCKET\_ERROR)

{

printf("Unable to listen\n");

WSACleanup();

system("pause");

return SOCKET\_ERROR;

}

// Информация об адресе клиентского сокета

SOCKADDR\_IN from;

int fromlen = sizeof(from);

// Программа останавливается и ждет запроса на установление соединения от клиентов

// Как только появился запрос, программа идет дальше

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

if (clientSock == INVALID\_SOCKET)

{

printf("Unable to accept\n");

WSACleanup();

system("pause");

return SOCKET\_ERROR;

}

// Новое соединение установлено, в консоли сервера выводим ip-адрес клиента, порт клиента и текущее число подключенных клиентов

printf("New connection accepted from %s, port %d\n", inet\_ntoa(from.sin\_addr), htons(from.sin\_port));

printf("Number of users in the chat: %i\n", curAmount + 1);

// Пытаемся получить данные от клиента, клиент должен отправить свое имя

// Функция возвращает 0 в случае успешной передачи, иначе -1 (например клиент завершил программу и оборвал соединение)

if (recv(clientSock, names[curAmount], maximumNameSize, 0) == SOCKET\_ERROR)

{

printf("Unable to recv\n");

WSACleanup();

system("pause");

return SOCKET\_ERROR;

}

mutex = CreateMutex(NULL, FALSE, NULL);

// Если не достигнут максимум одновременно подключенных к серверу пользователей

if (curAmount < maxAmount)

{

// Массив содержит какую то информацию о клиенте(его имя + сообщение, информацию о его выходе и т. д.), которая

// посылается другим пользователям чата

char messageToOtherUsers[maximumMessageSize];

// Заполняем данный массив

messageToOtherUsers[0] = '\0';

strcat(messageToOtherUsers, "New client, ip: ");

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

strcat(messageToOtherUsers, "; Name: ");

strcat(messageToOtherUsers, names[curAmount]);

strcat(messageToOtherUsers, "\n");

// Выводим содержимое messageToOtherUsers на экран сервера

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

if (NotifyUsers(curAmount + 1, messageToOtherUsers) == SOCKET\_ERROR)

{

printf("Unable to send\n");

WSACleanup();

system("pause");

// Разблокируем мьютекс

ReleaseMutex(mutex);

return SOCKET\_ERROR;

}

// Блокируем мьютекс для обработки клиентских данных

WaitForSingleObject(mutex, INFINITE);

// Сохраняем информацию о текущем пользователе

sockets[curAmount] = clientSock;

clSADDR[curAmount] = from;

curAmount++;

// Разблокируем мьютекс

ReleaseMutex(mutex);

}

else

{

// Если достигнут максимум, оповещаем пользователя, пытавшегося подключиться к серверу, о невозможности подключиться

// и завершаем использование данного клиентского сокета

int retVal = send(clientSock, "Sorry, the chat is full", maximumNameSize, 0);

closesocket(clientSock);

clientSock = -1;

// Выводим соответствующую информацию на экран сервера

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

printf("Connection closed\n");

}

DWORD threadID;

// Созданный поток должен определить начальный адрес кода, с которого новый поток должен исполняться.

// Как правило, начальный адрес - это название функции, определенной в коде программы (Chat). Эта функция получает единственный

// параметр и возвращает значение типа DWORD

if(clientSock != -1)

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

}

// Закрываем серверный сокет

closesocket(servSock);

WSACleanup();

return 0;

}

1. **Код Клиента**

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

#include <WinSock2.h>

#include <stdio.h>

#include <conio.h>

#include <iostream>

#include <string>

using namespace std;

// Максимальная длина имени пользователя и отправленного сообщения (или клиентом, или сервером)

const int maximumNameSize = 100;

const int maximumMessageSize = 1000;

HANDLE mutex;

// Имя пользователя

char name[maximumNameSize];

// Функции отправки сообщений, которая будет выполняться потоком, параметр функции типа LPVOID - это указатель на любой тип

DWORD WINAPI Send(LPVOID clientSocket)

{

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

while (true)

{

char message[maximumMessageSize];

// Ожидаем ввод ENTER

while ((\_getch()) != '\r');

WaitForSingleObject(mutex, INFINITE);

printf\_s(name);

printf\_s(": ");

// Считываем сообщение

fgets(message, maximumMessageSize, stdin);

ReleaseMutex(mutex);

// Отправка полученного сообщения

if (send(clientSock, message, maximumMessageSize, 0) == SOCKET\_ERROR)

{

printf("Unable to send\n");

WSACleanup();

system("pause");

return -1;

}

else if (!strcmp(message, "Exit\n")) // Если пользователь хочет выйти

{

return 0;

}

}

}

// Функции принятия сообщений, которая будет выполняться потоком, параметр функции типа LPVOID - это указатель на любой тип

DWORD WINAPI Receive(LPVOID clientSocket)

{

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

while (true)

{

char message[maximumMessageSize];

// Получаем сообщение от сервера

int retVal = recv(clientSock, message, maximumMessageSize, 0);

WaitForSingleObject(mutex, INFINITE);

// Чат полон? Выводим это в консоли и прекращаем взаимодействие с сервером

if (!strcmp(message, "Sorry, the chat is full"))

{

printf("Sorry, the chat is full\n");

return 0;

}

else if (retVal == SOCKET\_ERROR) // Если не удалось получить сообщение

{

printf("Unable to recv\n");

ReleaseMutex(mutex);

return -1;

}

else if (retVal == 0) // При удачном завершении соединения

{

printf("You have logged out of the chat\n");

ReleaseMutex(mutex);

return 0;

}

else // Во всех остальных случаях просто выводим полученное сообщение на экран

{

printf("%s", message);

}

ReleaseMutex(mutex);

}

}

int main()

{

WSADATA wsaData;

// Загрузка (инициализация) библиотеки

if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)

{

printf("Error WinSock version initializaion\n");

WSACleanup();

system("pause");

return SOCKET\_ERROR;

}

// Создание сокет клиента

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

if (clientSock == SOCKET\_ERROR)

{

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

WSACleanup();

system("pause");

return SOCKET\_ERROR;

}

string ip;

cout << "ip>";

cin >> ip;

cin.ignore();

// Заполнение информации об адресе сокета

SOCKADDR\_IN serverInfo;

serverInfo.sin\_family = PF\_INET;

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

serverInfo.sin\_port = htons(2006);

// Установление соединения с сервером

if (connect(clientSock, (LPSOCKADDR)&serverInfo, sizeof(serverInfo)) == SOCKET\_ERROR)

{

printf("Unable to connect\n");

WSACleanup();

system("pause");

return SOCKET\_ERROR;

}

// Ввод имени пользователя

printf("Connection made successfully\n");

printf("Enter your name: ");

cin >> name;

cin.ignore();

// Отправка имени серверу

if (send(clientSock, name, maximumNameSize, 0) == SOCKET\_ERROR)

{

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

WSACleanup();

system("pause");

return SOCKET\_ERROR;

}

printf("\nPress Enter to send the message\n");

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

// Создаем мьютекс

mutex = CreateMutex(NULL, FALSE, NULL);

if (mutex == NULL)

{

printf\_s("Error mutex creation\n");

WSACleanup();

system("pause");

return SOCKET\_ERROR;

}

// Обрабатываем сообщения многопоточно

DWORD threadID;

HANDLE sendHandle = CreateThread(NULL, NULL, Send, &clientSock, NULL, &threadID);

if (sendHandle == NULL)

{

printf\_s("Error when creating the Send stream");

return SOCKET\_ERROR;

}

HANDLE recvHandle = CreateThread(NULL, NULL, Receive, &clientSock, NULL, &threadID);

if (recvHandle == NULL)

{

printf\_s("Error when creating the Receive stream");

return SOCKET\_ERROR;

}

// Ожидаем завершения работы с сервером

WaitForSingleObject(recvHandle, INFINITE);

CloseHandle(sendHandle);

CloseHandle(recvHandle);

// Завершаем работу

closesocket(clientSock);

WSACleanup();

return 0;

}

1. **Тесты**

|  |
| --- |
| Тест №1. Подключение двух пользователей чата и между ними происходит переписка |
| Экран сервера: |
| Экран пользователя “Даниил”: |
| Экран пользователя “Богдан”: |
| Тест №2. На сервере установлено ограничение на 1 человека |
| Экран сервера: |
| Экран пользователя “Богдан”: |
| Экран пользователя “Даниил”: |
| Тест №3. Отправка сообщения одним пользователем, пока другой пользователь набирает сообщение  За счет блокировки получения каких-либо сообщений от сервера во время набора текста, пользователь, набирающий текст, получает сообщение от другого пользователя только после отправки своего сообщения. |
| Экран сервера: |
| Экран пользователя “Богдан”: |
| Экран пользователя “Даниил” |

1. **Вывод**

Контрольные вопросы проработаны.