

Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра теоретической прикладной информатики

Лабораторная работа № 3
по дисциплине «Операционные системы, среды и оболочки»

РАЗРАБОТКА ПРИЛОЖЕНИЯ ИНТЕРАКТИВНОЙ ПЕРЕПИСКИ

Факультет:	ПМИ
Группа:	ПМ-92
Бригада:	8
Студенты:	Иванов В., Кутузов И.
Преподаватель:	Кобылянский В. Г.

Новосибирск

2021

Цель работы

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

Ход работы

С помощью API-интерфейса реализовать простой чат. 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 сообщение. При подсоединении нового клиента к chat-серверу, сервер оповещает каждого клиента о новом пользователе, посылая им его IP-адрес и имя.

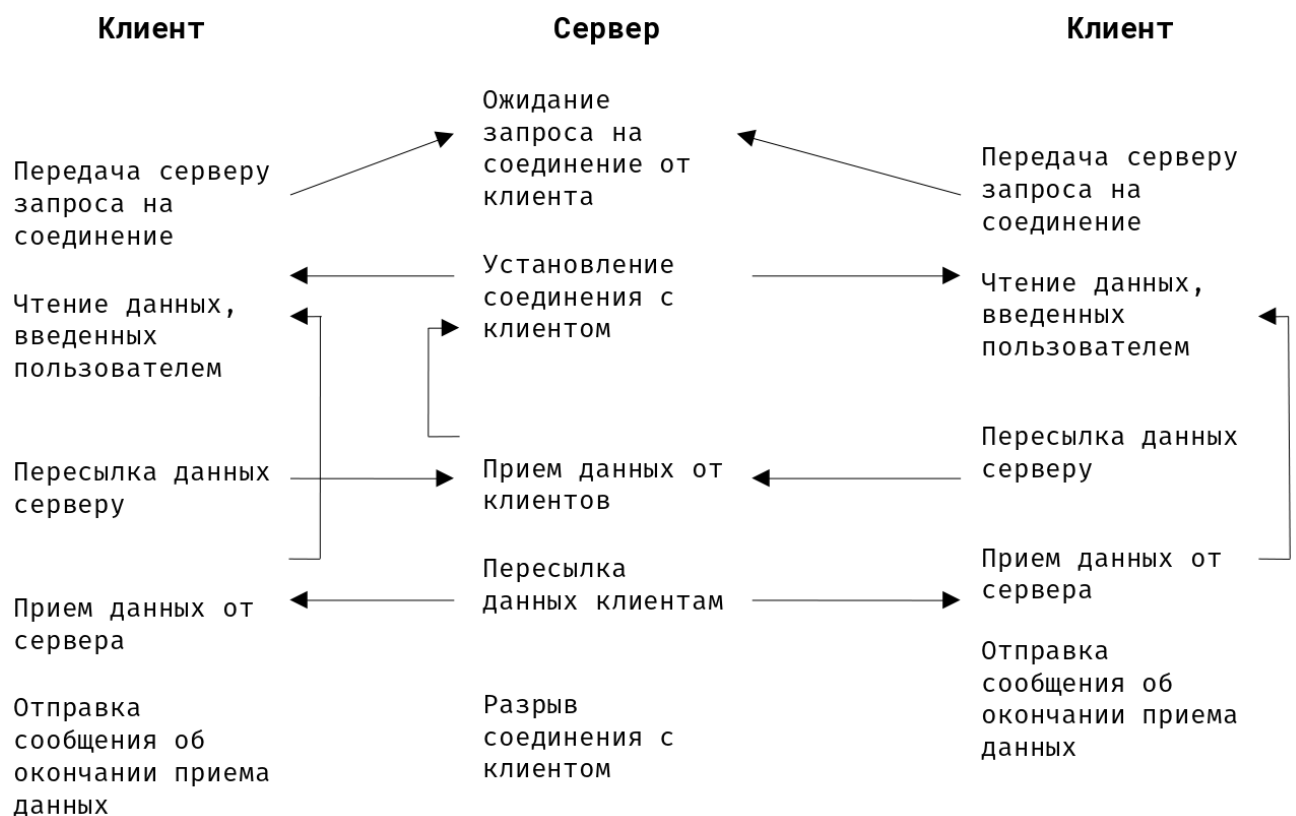


Схема взаимодействия чат-сервера с клиентами

Код программ

client.cpp

```
#pragma comment (lib,"Ws2_32.lib")
#include <WinSock2.h>
#include <ws2tcpip.h>
#include <stdio.h>
#include <conio.h>
```

```

#include <iostream>
#include <string>
using namespace std;
bool offline; // client status flag
const int U = 256; // username length
const int M = 1000; // message length

DWORD WINAPI receive(LPVOID clientSocket)
{
    int retVal; // return value for error check
    char Resp[M]; // response

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

    // message from the server
    retVal = recv(clientSock, Resp, M, 0);

    // if the server is down
    if (!strcmp(Resp, "Server shutdown."))
    {
        cout << "Server shutdown." << endl;
        offline = true;
        return 0;
    }

    // if the server is full
    if (!strcmp(Resp, "Server is full, please try again later."))
    {
        cout << "Server is full, please try again later." << endl;
        offline = true;
        return 0;
    }

    // if the client is still working
    if (!offline)
    {
        if (retVal == SOCKET_ERROR)
        {
            retVal = 0;
            cout << "Error: Unable to receive message!" << endl;
            offline = true;
            return 0;
        }
        else
        {
            // print server message
            cout << Resp << endl;
        }
    }
    return 1;
}

DWORD WINAPI send(LPVOID clientSocket)
{
    int retVal;
    char Buf[M]; // buffer array

```

```

SOCKET clientSock;
clientSock = *((SOCKET*)clientSocket);
gets_s(Buf);

// if the user entered /q (quit)
if (!strcmp(Buf, "/q"))
{
    offline = true;
    retVal = send(clientSock, Buf, M, 0);
    return 0;
}
else
{
    // send message to the server
    retVal = send(clientSock, Buf, M, 0);

    if (retVal == SOCKET_ERROR)
    {
        cout << "Error: Unable to send message!" << endl;
        WSACleanup();
        system("pause");
        return 0;
    }
}
return 1;
}

int main()
{
    string ip;
    WSADATA wsaData;
    int retVal = 0;
    offline = false;
    char username[U];
    WORD ver = MAKEWORD(2, 2);

    WSAStartup(ver, (LPWSADATA)&wsaData);
    SOCKET clientSock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

    if (clientSock == SOCKET_ERROR)
    {
        cout << "Error: Unable to create socket!" << endl;
        WSACleanup();
        system("pause");
        return 1;
    }

    // reading IP
    cout << "IP > ";
    cin >> ip;
    cin.ignore();

    SOCKADDR_IN serverInfo;
    serverInfo.sin_family = AF_INET;
    serverInfo.sin_port = htons(2008);
    inet_pton(AF_INET, ip.c_str(), &serverInfo.sin_addr);

```

```

// connecting to the server
retVal = connect(clientSock, (LPSOCKADDR)&serverInfo, sizeof(serverInfo));

if (retVal == SOCKET_ERROR)
{
    cout << "Error: Unable to connect to the server!" << endl;
    WSACleanup();
    system("pause");
    return 1;
}

// reading username
cout << "Connected!" << endl;
cout << "Username: ";
cin >> username;

// send username to the server
retVal = send(clientSock, username, U, 0);

if (retVal == SOCKET_ERROR)
{
    cout << "Error: Unable to send username!" << endl;
    WSACleanup();
    return 1;
}

cout << "Welcome to the chat room. Use /q to exit." << endl;

// while the client/server is running
while (!offline)
{
    DWORD threadID;
    CreateThread(NULL, NULL, send, &clientSock, NULL, &threadID);
    CreateThread(NULL, NULL, receive, &clientSock, NULL, &threadID);
}

closesocket(clientSock);
WSACleanup();
return 0;
}

```

server.cpp

```

#define _WINSOCK_DEPRECATED_NO_WARNINGS
#pragma comment (lib,"Ws2_32.lib")
#include <WinSock2.h>
#include <stdio.h>
#include <iostream>
#include <sstream>
#include <string>
using namespace std;
const int U = 256; // username length
const int M = 1000; // message length
const int maxClients = 5; // maximum number of users
int curClients; // number of connected users
char usernames[maxClients + 1][U];

```

```

SOCKET servSock;
SOCKET clSockets[maxClients];
SOCKADDR_IN clSADDR[maxClients];
USHORT ports[maxClients];

DWORD WINAPI chat(LPVOID clientSocket)
{
    int retVal; // return value for error check
    char Req[M]; // request
    char Resp[M]; // response
    int i, j, cur;

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

    while (true)
    {
        // message from the client
        retVal = recv(clientSock, Req, M, 0);

        if (retVal == SOCKET_ERROR)
        {
            cout << "Error: Unable to receive message!" << endl;
            closesocket(clientSock);
            cout << "Connection closed." << endl;
            return SOCKET_ERROR;
        }
        else
        {
            // if message is empty
            if (retVal >= M) retVal = M - 1;
            Req[retVal] = '\\0';
        }

        cout << "Data received." << endl;

        SOCKADDR_IN sin;
        for (i = 0; i < curClients; i++)
        {
            if (clSockets[i] == clientSock)
            {
                sin = clSADDR[i];
            }
        }

        cur = 0;
        while (ports[cur] != sin.sin_port) // username search
        {
            cur++;
        }

        if (!strcmp(Req, "/q")) // if the user entered /q (quit)
        {
            // notify all clients that the user is logged out
            Resp[0] = '\\0';
            strcat_s(Resp, usernames[cur]);
        }
    }
}

```

```

    strcat_s(Resp, " left the chat.");

    for (i = 0; i < curClients; i++)
    {
        if (clSockets[i] != clientSock) retVal = send(clSockets[i], Resp, M, 0);
    }

    if (retVal == SOCKET_ERROR)
    {
        cout << "Error: Unable to send message!" << endl;
        return SOCKET_ERROR;
    }

    cout << "Client disconnected." << endl;
    closesocket(clientSock);
    cout << "Connection closed." << endl;

    // delete user information
    for (j = cur; j < curClients; j++)
    {
        clSockets[j] = clSockets[j + 1];
        clSADDR[j] = clSADDR[j + 1];
        ports[j] = ports[j + 1];
        strcpy_s(usernames[j], usernames[j + 1]);
    }

    clSockets[curClients - 1] = SOCKET_ERROR;
    curClients--;
    cout << "Current online: " << curClients << endl; // print the updated number
of online users
    return SOCKET_ERROR;
}

if (Req[0] != '\0') // if the received message isn't an empty string
{
    // print username and message
    cout << usernames[cur] << ": " << Req << endl;
    Resp[0] = '\0';
    strcat_s(Resp, usernames[cur]);
    strcat_s(Resp, ": ");
    strcat_s(Resp, Req);

    cout << "Sending response from the server." << endl;

    // send message to other users
    for (i = 0; i < curClients; i++)
    {
        if (clSockets[i] != clientSock) retVal = send(clSockets[i], Resp, M, 0);
    }

    if (retVal == SOCKET_ERROR)
    {
        cout << "Error: Unable to send message!" << endl;
        return SOCKET_ERROR;
    }
}
}

```

```

}

int main()
{
    int retVal;
    int i;
    char NewClient[M];

    curClients = 0;

    WORD sockVer;
    WSADATA wsaData;
    sockVer = MAKEWORD(2, 2);
    WSASStartup(sockVer, &wsaData);

    // creating socket
    servSock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

    if (servSock == INVALID_SOCKET)
    {
        cout << "Error: Unable to create socket!" << endl;
        WSACleanup();
        system("pause");
        return SOCKET_ERROR;
    }

    for (i = 0; i < maxClients; i++)
    {
        clSockets[i] = SOCKET_ERROR;
    }

    SOCKADDR_IN sin;
    sin.sin_family = AF_INET;
    sin.sin_port = htons(2008);
    sin.sin_addr.s_addr = INADDR_ANY;

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

    if (retVal == SOCKET_ERROR)
    {
        cout << "Error: Unable to bind socket!" << endl;
        WSACleanup();
        system("pause");
        return SOCKET_ERROR;
    }

    cout << "Server started at port: " << htons(sin.sin_port) << endl;

    while(true)
    {
        // starting to listen
        retVal = listen(servSock, 10);

        if (retVal == SOCKET_ERROR)
        {
            cout << "Error: Cannot listen to socket!" << endl;

```



```

        WSACleanup();
        system("pause");
        return SOCKET_ERROR;
    }

    SOCKET clientSock;
    SOCKADDR_IN from;
    int fromlen = sizeof(from);

    // accepting the client's request
    clientSock = accept(servSock, (struct sockaddr*)&from, &fromlen);

    if (clientSock == INVALID_SOCKET)
    {
        cout << "Error: Unable to accept socket!" << endl;
        WSACleanup();
        system("pause");
        return SOCKET_ERROR;
    }

    cout << "New connection: " << inet_ntoa(from.sin_addr) << ":" <<
    htons(from.sin_port) << endl;
    cout << "Users: " << curClients + 1 << endl;

    // get username of a new client
    retVal = recv(clientSock, usernames[curClients], U, 0);

    // get client data
    if (retVal == SOCKET_ERROR)
    {
        cout << "Error: Unable to receive message!" << endl;
        system("pause");
        return SOCKET_ERROR;
    }

    // if username is /shutdown
    if (!strcmp(usernames[curClients], "/shutdown"))
    {
        // send "Server shutdown" to all clients and close sockets
        for (i = 0; i < curClients; i++)
        {
            retVal = send(clSockets[i], "Server shutdown.", M, 0);
            closesocket(clSockets[i]);
        }

        retVal = send(clientSock, "Server shutdown.", M, 0);
        closesocket(clientSock);
        break;
    }
    else
    {
        // if the maximum number of users isn't reached
        if (curClients < maxClients)
        {
            // save and print information about the current user
            ports[curClients] = from.sin_port;
            clSockets[curClients] = clientSock;
        }
    }
}

```

```

        clSADDR[curClients] = from;
        NewClient[0] = '\0';

        strcat_s(NewClient, "Client ");
        strcat_s(NewClient, usernames[curClients]);
        strcat_s(NewClient, " connected (IP: ");
        strcat_s(NewClient, inet_ntoa(from.sin_addr));
        strcat_s(NewClient, ")");
        cout << NewClient << endl;

        curClients++;

        // send a message to other clients about the new user
        for (i = 0; i < curClients; i++)
        {
            if (clSockets[i] != clientSock) retVal =
                send(clSockets[i], NewClient, M, 0);
        }
    }
    else
    {
        // if the maximum number of users is reached
        cout << "Max online." << endl;
        retVal = send(clientSock, "Server is full, please try again later.", U,
0);

        closesocket(clientSock);
        cout << "Connection closed." << endl;
    }

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


}

closesocket(servSock);
WSACleanup();
return 0;
}

```


Тестирование

Запустим сервер:

 Выбрать C:\Users\vv.ivanov.2019\source\repos\lr3_server\Debug\lr3_server.exe

Server started at port: 2008

Запустим первый клиент, введем IP-адрес и имя пользователя:

 Выбрать C:\Users\vv.ivanov.2019\source\repos\lr3_client\Debug\lr3_client.exe

```

IP > 172.17.1.18
Connected!
Username: Biba
Welcome to the chat room. Use /q to exit.

```

Теперь данный пользователь может отправлять сообщения в чат.
При этом реакция сервера следующая:

Выбрать C:\Users\w.v.ivanov.2019\source\repos\lr3_server\Debug\lr3_server.exe

```
Server started at port: 2008
New connection: 172.17.1.18:51633
Users: 1
```

Аналогично запустим второй клиент:

Выбрать C:\Users\w.v.ivanov.2019\source\repos\lr3_client\Debug\lr3_client.exe

```
IP > 172.17.1.18
Connected!
Username: Boba
Welcome to the chat room. Use /q to exit.
```

Первый клиент оповещается о подключении второго:

Выбрать C:\Users\w.v.ivanov.2019\source\repos\lr3_client\Debug\lr3_client.exe

```
IP > 172.17.1.18
Connected!
Username: Biba
Welcome to the chat room. Use /q to exit.
Client Boba connected (IP: 172.17.1.18)
```

Пользователи в реальном времени видят сообщения других пользователей. Имя отправителя отображаются для всех сообщений, кроме сообщений самого пользователя:

Выбрать C:\Users\w.v.ivanov.2019\source\repos\lr3_client\Debug\lr3_client.exe

```
IP > 172.17.1.18
Connected!
Username: Boba
Welcome to the chat room. Use /q to exit.
Сообщение 1
Biba: Сообщение 2
```

Выбрать C:\Users\w.v.ivanov.2019\source\repos\lr3_client\Debug\lr3_client.exe

```
IP > 172.17.1.18
Connected!
Username: Biba
Welcome to the chat room. Use /q to exit.
Client Boba connected (IP: 172.17.1.18)
Boba: Сообщение 1
Сообщение 2
```

Сервер при этом ведет историю обмена сообщениями:

Выбрать C:\Users\w.v.ivanov.2019\source\repos\lr3_server\Debug\lr3_server.exe

```
Server started at port: 2008
New connection: 172.17.1.18:58753
Users: 1
Client Biba connected (IP: 172.17.1.18)
Data received.
New connection: 172.17.1.18:58756
Users: 2
Client Boba connected (IP: 172.17.1.18)
Data received.
Data received.
Boba: Сообщение 1
Sending response from the server.
Data received.
Biba: Сообщение 2
Sending response from the server.
```