

Socket聊天程序编写

本次实验进行了一个简单多人聊天服务器与客户端的设计和实现。

协议设计

消息类型

- 客户端和服务器之间通过字符串消息进行通信。

语法规义

- 客户端发送的消息是纯文本聊天消息。
- 服务器充当中介，接收客户端的消息并广播给所有其他客户端。
- 客户端接收到的消息显示在终端上。

时序

- 服务器和客户端之间通过TCP连接进行通信。
- 客户端连接到服务器后，创建一个接收消息的线程，该线程负责接收服务器发送的消息。
- 客户端和服务器之间的通信是异步的，可以同时接收和发送消息。

各模块功能

服务器

- 创建服务器套接字，绑定到指定的端口，并监听客户端连接请求。
- 当客户端连接到服务器时，为每个客户端创建一个新的线程，该线程用于接收客户端消息。
- 接收到客户端消息后，将消息广播给所有其他连接到服务器的客户端。

客户端

- 创建客户端套接字，并连接到指定的服务器IP地址和端口。
- 创建一个接收消息的线程，该线程用于接收服务器发送的消息。
- 在终端上接收用户输入，发送消息给服务器。

程序运行说明

服务器运行

1. 在服务器代码中指定要监听的IP地址、端口号 `PORT`。
2. 定义用户线程，内容包括信息的接收与广播。

```
DWORD WINAPI ThreadFunction(LPVOID lpParameter)//线程函数
{
    int receByt = 0;
    char RecvBuf[BufSize]; //接收缓冲区
    char SendBuf[BufSize]; //发送缓冲区
    //char exitBuf[5];
```

```

//SOCKET sock = *((SOCKET*)lpParameter);

//循环接收信息
while (true)
{
    int num = (int)lpParameter; //当前连接的索引
    Sleep(100); //延时100ms
    //receByt = recv(sock, RecvBuf, sizeof(RecvBuf), 0);
    receByt = recv(clientSockets[num], RecvBuf, sizeof(RecvBuf), 0); //接收信息

    if (receByt > 0) //接收成功
    {
        //创建时间戳，记录当前通讯时间
        auto currentTime = chrono::system_clock::now();
        time_t timestamp = chrono::system_clock::to_time_t(currentTime);
        tm localTime;
        localtime_s(&localTime, &timestamp);
        char timeStr[50];
        strftime(timeStr, sizeof(timeStr), "(%Y/%m/%d %H:%M:%S)",
        &localTime); // 格式化时间
        SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN |
        FOREGROUND_INTENSITY);
        cout << "Client " << clientSockets[num] << ": " << RecvBuf << " "
        << timeStr << endl;
        SetConsoleTextAttribute(hConsole, FOREGROUND_RED |
        FOREGROUND_GREEN | FOREGROUND_BLUE);
        sprintf_s(SendBuf, sizeof(SendBuf), "%s From %d %s ", RecvBuf,
        clientSockets[num], timeStr); // 格式化发送信息
        for (int i = 0; i < MaxClient; i++) //将消息同步到除发送者的所有聊天窗口
        {
            if (condition[i] == 1 && i != num)
            {
                send(clientSockets[i], SendBuf, sizeof(SendBuf), 0); //发送信息
            }
        }
    }
    else //接收失败
    {
        if (WSAGetLastError() == 10054) //客户端主动关闭连接
        {
            //创建时间戳，记录当前通讯时间
            auto currentTime = chrono::system_clock::now();
            time_t timestamp =
            chrono::system_clock::to_time_t(currentTime);
            tm localTime;
            localtime_s(&localTime, &timestamp);
            char timeStr[50];
            strftime(timeStr, sizeof(timeStr), "%Y/%m/%d %H:%M:%S",
            &localTime); // 格式化时间
            cout << "Client " << clientSockets[num] << " exited at " <<
            timeStr << endl;
            closesocket(clientSockets[num]);
            current_connect_count--;
            condition[num] = 0;
        }
    }
}

```

```

        cout << "Current user: " << current_connect_count << endl;
        return 0;
    }
    else
    {
        cout << "Failed to receive, Error:" << WSAGetLastError() <<
endl;
        break;
    }
}
}
}
}
}

```

3. 编译并运行服务器代码。

4. 服务器将开始监听指定端口，等待客户端连接请求。接收到请求后为其创建一个单独的线程。

```

while (true)
{
    if (current_connect_count < MaxClient)
    {
        int num = check();
        int addrLen = sizeof(SOCKADDR);
        clientSockets[num] = accept(serverSocket,
(sockaddr*)&clientAddr[num], &addrLen); //接收客户端请求
        if (clientSockets[num] == SOCKET_ERROR) //错误处理
        {
            perror("Client failed! \n");
            closesocket(serverSocket);
            WSACleanup();
            exit(EXIT_FAILURE);
        }
        condition[num] = 1; //连接位置1表示占用
        current_connect_count++; //当前连接数加1
        //创建时间戳，记录当前通讯时间
        auto currentTime = chrono::system_clock::now();
        time_t timestamp = chrono::system_clock::to_time_t(currentTime);
        tm localTime;
        localtime_s(&localTime, &timestamp);
        char timeStr[50];
        strftime(timeStr, sizeof(timeStr), "%Y/%m/%d %H:%M:%S", &localTime);
        // 格式化时间
        cout << "Client " << clientSockets[num] << " connected at " <<
timeStr << endl;
        cout << "Current user: " << current_connect_count << endl;
        HANDLE Thread = CreateThread(NULL, 0,
(LPTHREAD_START_ROUTINE)ThreadFunction, (LPVOID)num, 0, NULL); //为当前用户创建
线程
        if (Thread == NULL) //线程创建失败
        {
            perror("Thread failed!\n");
            exit(EXIT_FAILURE);
        }
    }
}

```

```

        else
        {
            closeHandle(Thread);
        }
    }
    else
    {
        cout << "Server is busy now..." << endl;
    }
}

```

客户端运行

1. 在客户端代码中指定服务器的IP地址和端口号。
2. 启动服务器后启动客户端，客户端将连接到服务器。
3. 在终端上，用户可以输入聊天消息，按回车键发送消息。

```

while (true)
{
    SetConsoleTextAttribute(hConsole, FOREGROUND_BLUE |
    FOREGROUND_INTENSITY);
    cout << ">>";
    cin.getline(buf, sizeof(buf));
    if (strcmp(buf, "logout") == 0) //输入exit退出
    {
        break;
    }
    send(clientSocket, buf, sizeof(buf), 0); //发送消息
    SetConsoleTextAttribute(hConsole, FOREGROUND_RED | FOREGROUND_GREEN |
    FOREGROUND_BLUE);
}

```

5. 客户端通过线程接收到服务器广播的消息后会显示在终端上。

```

DWORD WINAPI recvThread() //接收消息线程
{
    while (true)
    {
        char buffer[BufSize] = {}; //接收数据缓冲区
        if (recv(clientSocket, buffer, sizeof(buffer), 0) > 0)
        {
            SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN |
            FOREGROUND_INTENSITY);
            cout << buffer << endl;
            SetConsoleTextAttribute(hConsole, FOREGROUND_BLUE |
            FOREGROUND_INTENSITY);
            cout << ">>";
        }
        else if (recv(clientSocket, buffer, sizeof(buffer), 0) < 0)
        {

```

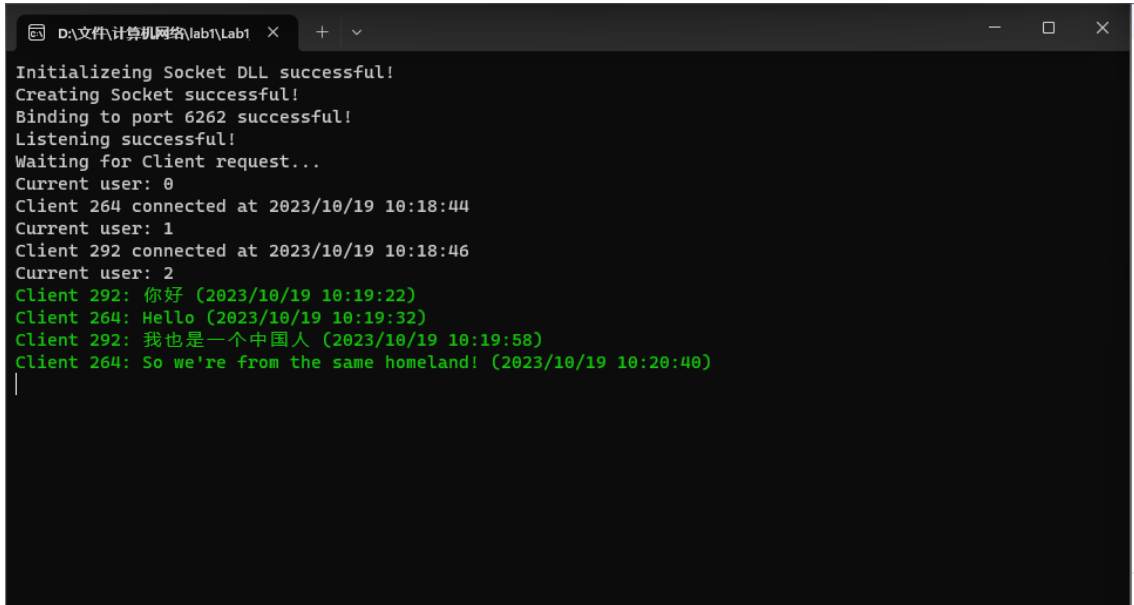
```

        cout << "Connection lost!" << endl;
        break;
    }
}
sleep(100); //延时100ms
return 0;
}

```

运行结果

- 服务器将接收来自客户端的消息，并广播给所有连接的客户端。

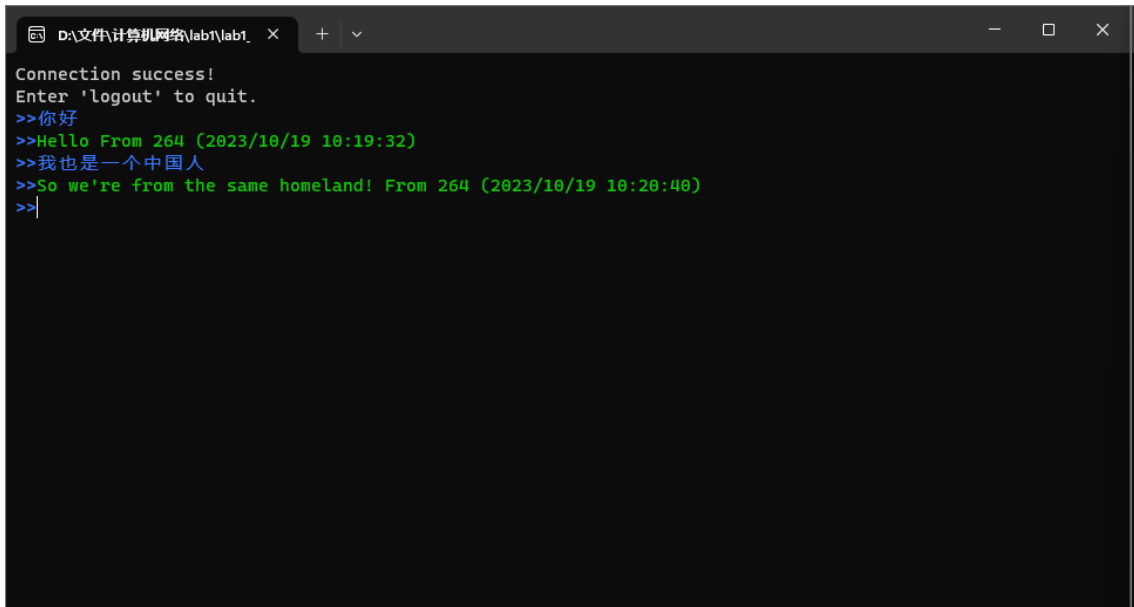


```

D:\文件\计算机网络\lab1\Lab1
Initializeing Socket DLL successful!
Creating Socket successful!
Binding to port 6262 successful!
Listening successful!
Waiting for Client request...
Current user: 0
Client 264 connected at 2023/10/19 10:18:44
Current user: 1
Client 292 connected at 2023/10/19 10:18:46
Current user: 2
Client 292: 你好 (2023/10/19 10:19:22)
Client 264: Hello (2023/10/19 10:19:32)
Client 292: 我也是一个中国人 (2023/10/19 10:19:58)
Client 264: So we're from the same homeland! (2023/10/19 10:20:40)
|

```

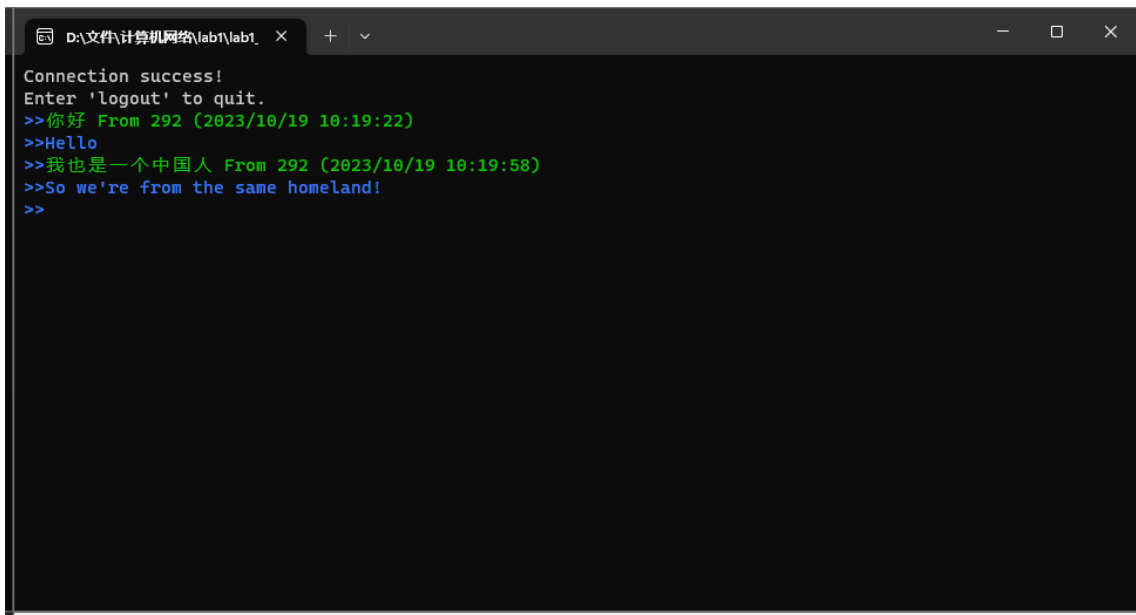
- 客户端会显示服务器广播的消息和用户发送的消息。其中自己发送的消息以蓝色、其他人发送的消息以绿色显示。



```

D:\文件\计算机网络\lab1\lab1
Connection success!
Enter 'logout' to quit.
>>你好
>>Hello From 264 (2023/10/19 10:19:32)
>>我也是一个中国人
>>So we're from the same homeland! From 264 (2023/10/19 10:20:40)
>>|

```



```
D:\文件\计算机网络\lab1\lab1_ x + v
Connection success!
Enter 'logout' to quit.
>>你好 From 292 (2023/10/19 10:19:22)
>>Hello
>>我是一个中国人 From 292 (2023/10/19 10:19:58)
>>So we're from the same homeland!
>>
```

结束运行

- 客户端可以通过输入 "logout" 来退出聊天程序。
- 服务器可以通过关闭终端来终止运行。

运行逻辑合理性

- 服务器支持多个客户端连接，每个客户端都有自己的线程，使得多人同时聊天成为可能。
- 服务器接收来自客户端的消息，并在终端上显示，实现了简单的聊天功能。
- 客户端能够连接到服务器，接收来自服务器的消息并发送消息，实现了基本的聊天客户端功能。