

南开大学

网络安全技术实验报告

基于DES加密的TCP聊天程序



学院：网络空间安全学院

年级：21级

班级：信安2班

学号：2113997

姓名：齐明杰

2024年3月13日

目录

- 1 实验目的
- 2 实验内容
- 3 实验步骤及实验结果
 - 3.1 构建项目
 - 3.2 DES加解密模块
 - 3.2.1 子密钥生成
 - 3.2.2 Feistel函数
 - 3.2.3 加/解密流程
 - 3.2.4 对外接口
 - 3.3 TCP客户端
 - 3.4 TCP服务端
 - 3.5 运行方法及示例
- 4 实验遇到的问题及其解决方法
- 5 实验结论

1 实验目的

DES(Data Encryption Standard)算法是一种用 56 位有效密钥来加密 64 位数据的对称分组加密算法，该算法流程清晰，已经得到了广泛的应用，算是应用密码学中较为基础的加密算法。TCP(传输控制协议)是一种面向链接的、可靠的传输层协议。TCP协议在网络层IP 协议的基础上，向应用层用户进程提供可靠的、全双工的数据流传输。

本章训练的目的如下。

- ①理解 DES 加解密原理。
- ②理解 TCP 协议的工作原理。
- ③掌握 linux 下基于 socket 的编程方法

本章训练的要求如下。

- ①利用 socket 编写一个 TCP 聊天程序。
- ②通信内容经过 DES 加密与解密，

2 实验内容

在 Linux 或Windows平台下，实现基于 DES 加密的 TCP 通信，具体要求如下：

- ①能够在了解 DES 算法原理的基础上，编程实现对字符串的 DES 加密解密操作。
- ②能够在了解 TCP 和 Linux 或Windows平台下的 Socket运行原理的基础上,编程实现简单的TCP通信，为简化编程细节，不要求实现一对多通讯。
- ③将上述两部分结合到一起,编程实现通信内容事先通过 DES 加密的 TCP 聊天程序,要求双方事先互通密钥，在发送方通过该密钥加密，然后由接收方解密，保证在网络上传输的信息的保密性。

3 实验步骤及实验结果

3.1 构建项目

- 实验环境: Windows 10 x64 专业版
- IDE: Visual Studio 2022 community
- 编程语言: C++

3.2 DES加解密模块

本实验实现了DES算法，涉及到密钥生成、数据加密、数据解密等步骤。DES算法的核心包括初始置换、16轮Feistel网络、最终置换，以及子密钥的生成和应用。

我在des.h中声明了DES类，声明需要的函数和相关接口：

```
1  #pragma once
2
3  #include <vector>
4  #include <bitset>
5  #include <cstdint>
6  #include <string>
7
8  class DES {
9      static const uint8_t IP[64];           // 第一轮置换
10     static const uint8_t FP[64];           // 最后一轮置换
11     static const uint8_t E_box[48];        // 拓展运算E盒
12     static const uint8_t P_box[32];        // 置换运算P盒
13     static const uint8_t S_box[8][4][16];  // 8个S盒
14     static const uint8_t PC_1[56];         // PC-1置换
15     static const uint8_t PC_2[48];        // 密钥压缩置换表
16     static const uint8_t shift[16];       // 每轮左移的位数
17
18 public:
19     DES(const std::string& key);           // 构造函数，需要64位密钥作为参数
20     std::vector<uint8_t> encrypt(const std::vector<uint8_t>& plaintext);
21     std::vector<uint8_t> decrypt(const std::vector<uint8_t>& ciphertext);
22     static std::vector<uint8_t> strToVec(const std::string& input) {
23         return std::vector<uint8_t>(input.begin(), input.end());
24     }
25     static std::string vecToStr(const std::vector<uint8_t>& input) {
26         return std::string(input.begin(), input.end());
27     }
28
29 private:
30     enum MODE { ENCRYPT, DECRYPT };
31     std::bitset<48> subKeys[16];          // 存储生成的16轮子密钥
```

```

32     std::bitset<64> execute(const std::bitset<64>& data, int mode); // 加
    密/解密
33     void genSubKeys(const std::bitset<64>& key); // 生成16轮子密钥
34     std::bitset<32> f(const std::bitset<32>& R, const std::bitset<48>& K);
    // f函数
35     template<size_t N> // 循环左移
36     std::bitset<N> leftRotate(const std::bitset<N>& bits, int shift);
37     template<size_t N>
38     uint8_t get(const std::bitset<N>& b, size_t pos) { return b[N - 1 -
pos]; }
39     template<size_t N>
40     void set(std::bitset<N>& b, size_t pos, size_t value) { b[N - 1 - pos]
= value; }
41     std::vector<uint8_t> pad(const std::vector<uint8_t>& data, size_t
blockSize); // PKCS#7填充
42     std::vector<uint8_t> unpad(const std::vector<uint8_t>& data, size_t
blockSize); // PKCS#7去填充
43 };
44

```

使用了C++的 `bitset` 和 `vector` 来存储数据。

3.2.1 子密钥生成

密钥生成步骤涉及到PC-1置换减少密钥从64位到56位，去除了8位奇偶校验位。然后，根据每一轮的需要进行左移操作，再通过PC-2压缩置换得到48位的子密钥。这一过程共重复16轮，生成16个子密钥用于后续的加密解密过程。

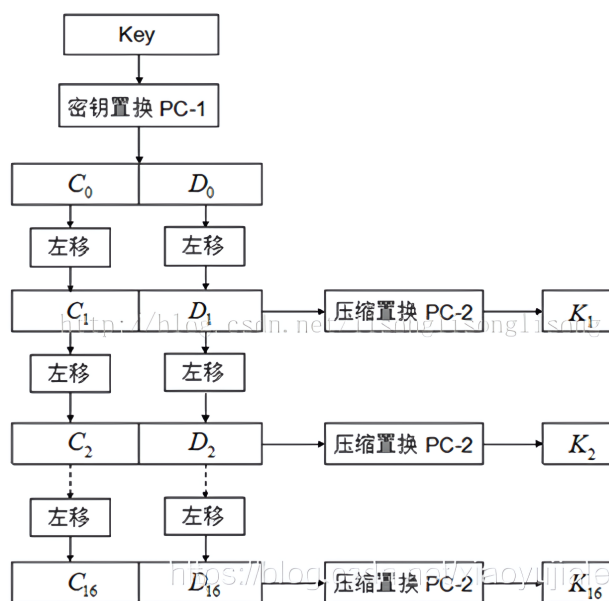


图1 DES密钥生成示意图

```

1 void DES::genSubKeys(const std::bitset<64>& key) {
2     std::bitset<56> realKey;

```

```

3   for (int i = 0; i < 56; i++) {
4       set(realKey, i, get(key, PC_1[i] - 1));
5   }
6   std::bitset<28> left(realKey.to_ullong() >> 28);
7   std::bitset<28> right(realKey.to_ullong() & 0xFFFFFFFF);
8   for (int i = 0; i < 16; i++) {
9       left = leftRotate(left, shift[i]);
10      right = leftRotate(right, shift[i]);
11      std::bitset<56> combined(left.to_string() + right.to_string());
12      for (int j = 0; j < 48; j++) {
13          set(subKeys[i], j, get(combined, PC_2[j] - 1));
14      }
15  }
16 }
17
18 template<size_t N>
19 std::bitset<N> DES::leftRotate(const std::bitset<N>& bits, int shift) {
20     return (bits << shift) | (bits >> (N - shift));
21 }

```

首先，使用PC-1置换表（置换选择1），将64位原始密钥压缩为56位。这一过程中，原始密钥中的某些位被舍弃，主要是为了奇偶校验位的去除。接下来，将这56位密钥分割成两个28位的部分，分别对这两部分进行循环左移。循环左移的位数根据DES算法的规定在不同的轮次有不同的值，这通过一个叫 `shift` 的数组来指定。

3.2.2 Feistel函数

Feistel网络是DES算法的核心，它包括扩展置换、S盒置换、P盒置换和异或操作。每一轮Feistel网络都会使用一个48位的子密钥，和数据的右半部分进行操作，然后将操作的结果和左半部分进行异或操作，最后交换左右两半，进入下一轮。

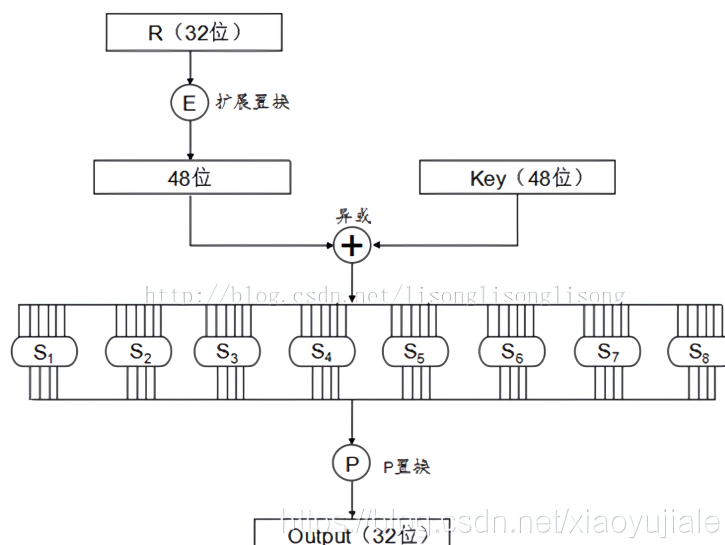


图2 f函数示意图

```

1  std::bitset<32> DES::f(const std::bitset<32>& R, const std::bitset<48>& K)
   {
2      std::bitset<48> block;
3      for (int i = 0; i < 48; i++) {
4          set(block, i, get(R, E_box[i] - 1));
5      }
6      block ^= K;
7      std::bitset<32> result;
8      for (int i = 0; i < 8; i++) {
9          int row = (get(block, i * 6 + 5) << 1) + get(block, i * 6);
10         int col = (get(block, i * 6 + 4) << 3) + (get(block, i * 6 + 3) <<
11         2) + (get(block, i * 6 + 2) << 1) + get(block, i * 6 + 1);
12         uint8_t sBoxVal = S_box[i][row][col];
13         for (int k = 0; k < 4; k++) {
14             set(result, i * 4 + k, (sBoxVal >> (3 - k)) & 1);
15         }
16     }
17     std::bitset<32> output;
18     for (int i = 0; i < 32; i++) {
19         set(output, i, get(result, P_box[i] - 1));
20     }
21     return output;
22 }

```

1. **扩展置换**：首先，函数接收32位的数据块 **R** 和48位的子密钥 **K** 作为输入。数据块 **R** 通过E盒（扩展盒）进行扩展置换，从32位扩展到48位。这一过程使用 **E_box** 数组定义的置换规则，增加了数据的长度，为接下来的异或操作做准备。
2. **与子密钥进行异或**：扩展后的数据块与子密钥 **K** 进行异或操作，异或操作是加密过程中引入密钥的主要方式之一，用于混合数据和密钥，增加破解的难度。
3. **S盒置换**：异或结果被分为8个6位的部分，每部分通过一个S盒进行置换。S盒（替代盒）是固定的非线性替代过程，每个S盒将6位输入映射为4位输出。这一步骤是DES算法中引入复杂性和非线性的关键部分。通过行和列的计算选取S盒中的值，行由边缘位决定，列由中间4位决定。
4. **P盒置换**：所有S盒的输出合并成一个32位的块，然后通过P盒（置换盒）进行另一次置换，这也是一个固定的置换过程，旨在进一步混淆数据。P盒置换通过 **P_box** 数组定义的置换规则执行。
5. **返回值**：最后，**f** 函数输出32位经过上述一系列变换的数据块。这个输出将会在Feistel网络中与另一半数据块进行异或操作。

3.2.3 加/解密流程

由于DES加密和解密的流程相同，仅仅是密钥使用的顺序不同，因此我将其合并成一个函数，通过参数来控制其是否为加密/解密：

```

1  std::bitset<64> DES::execute(const std::bitset<64>& data, int mode) {
2      std::bitset<64> block;
3      for (int i = 0; i < 64; i++) {

```

```

4         set(block, i, get(data, IP[i] - 1));
5     }
6     std::bitset<32> left(block.to_ullong() >> 32);
7     std::bitset<32> right(block.to_ullong() & 0xFFFFFFFF);
8     for (int round = 0; round < 16; round++) {
9         std::bitset<32> rightExpanded = f(right, subKeys[mode == DECRYPT ?
15 - round : round]);
10        std::bitset<32> temp = left ^ rightExpanded;
11        left = right;
12        right = temp;
13        if (round == 15) {
14            std::swap(left, right);
15        }
16    }
17    block = (std::bitset<64>(left.to_ullong()) << 32) | std::bitset<64>
(right.to_ullong());
18    std::bitset<64> result;
19    for (int i = 0; i < 64; i++) {
20        set(result, i, get(block, FP[i] - 1));
21    }
22    return result;
23 }

```

- **加密过程**：首先对明文进行**初始置换(IP)**，然后进入**16轮Feistel网络**，每一轮使用一个子密钥进行处理。16轮完成后，进行**最终置换(FP)**，得到密文。
- **解密过程**：解密过程与加密过程相似，但子密钥的使用顺序相反，即从第16个子密钥开始使用，直到第1个子密钥。

3.2.4 对外接口

考虑到填充以及TCP传输数据的问题(见后文)，我采用了 `std::vector<uint8_t>` 作为接口，同时加入 `PKCS#7` 填充方案，作为接口：

```

1 std::vector<uint8_t> DES::encrypt(const std::vector<uint8_t>& plaintext) {
2     auto padText = pad(plaintext, 8);
3     std::vector<uint8_t> ciphertext;
4     for (size_t i = 0; i < padText.size(); i += 8) {
5         std::bitset<64> block;
6         for (int j = 0; j < 64; ++j) {
7             set(block, j, (padText[i + j / 8] >> (7 - (j % 8))) & 0x01);
8         }
9         auto encBlock = execute(block, ENCRYPT);
10        for (int j = 0; j < 8; ++j) {
11            ciphertext.push_back(static_cast<uint8_t>((encBlock >> (56 - 8
12 * j)).to_ullong() & 0xFF));
13        }
14    }
15 }

```



```

14     return ciphertext;
15 }
16
17 std::vector<uint8_t> DES::decrypt(const std::vector<uint8_t>& ciphertext)
18 {
19     std::vector<uint8_t> decText;
20     for (size_t i = 0; i < ciphertext.size(); i += 8) {
21         std::bitset<64> block;
22         for (int j = 0; j < 64; ++j) {
23             set(block, j, (ciphertext[i + j / 8] >> (7 - (j % 8))) &
24                 0x01);
25         }
26         auto decBlock = execute(block, DECRYPT);
27         for (int j = 0; j < 8; ++j) {
28             decText.push_back(static_cast<uint8_t>((decBlock >> (56 - 8 *
29                 j)).to_ullong() & 0xFF));
30         }
31     }
32     return unpad(decText, 8);
33 }

```

函数将数据分割为以64bit为单位的块，分块加密，并自动填充/去填充，实现了任意长度的数据进行加密，解密同理。

密钥则采用输入16进制数作为接口，例如：

```

1 DES* des = new DES("133457799BBCDFF1"); //传入密钥

```

3.3 TCP客户端

客户端则负责与用户交互，发送和接收消息。本次客户端我采用C++ MFC可视化编程实现。

功能概述：

- 在上方输入服务端IP，服务端口，然后输入自己的用户名，点击连接服务器即可连接服务端。
- 聊天区会显示服务端发送来的信息，解析出用户名和消息后，拼接上当前时间进行显示。
- 点击退出，即可离开聊天区，此时服务器和还在聊天区的客户会收到离开信息。
- 在下面编辑框输入后，点击发送即可发送信息到服务器(不会直接显示在聊天区)。

支持：

- 多开客户端，在断连后重新连接服务器(之前的聊天内容不会清空)。
- 多线程管理，主线程负责控制用户和界面的交互，按钮点击事件等，子线程负责接受服务器的消息并转交给主线程打印处理。

界面设计：



部分重要函数变量声明如下：

```

1  afx_msg void OnBnClickedButtonExit(); // 退出按钮
2  afx_msg void OnBnClickedButtonSend(); // 发送按钮
3  afx_msg void OnBnClickedButtonConnect(); // 连接按钮
4  static constexpr UINT BufferSize = 1024; // 缓冲区大小
5  virtual void OnClose(); // 重写关闭窗口函数
6  SOCKET SockClient = INVALID_SOCKET; // 客户端套接字
7  HANDLE hThread = NULL; // 线程句柄
8  CString UserName; // 用户名
9  CString key; // 密钥
10 DES* des; // DES加密解密对象
11 void PrintMsg(const CString& Name, const CString& strMsg); // 打印消息
12 static DWORD WINAPI ReceiveMessages(LPVOID pParam); // 接收消息线程函数
13 LRESULT OnUpdateChatMsg(WPARAM wParam, LPARAM lParam); // 更新聊天消息

```

其中的几个关键函数：

1. `OnBnClickedButtonExit()` : 退出客户端按钮点击事件

功能：当用户点击退出按钮时，此函数会被触发。它会首先确认用户真的想要退出，然后关闭与服务器的套接字连接、终止消息接收线程，释放Winsock资源，并关闭聊天窗口。

2. `OnBnClickedButtonSend()` : 发送消息按钮点击事件

功能：此函数处理用户的消息发送请求。它首先检查消息内容的有效性，然后调用DES加密消息并发送消息到服务器。如果发送失败，它会提醒用户，并允许用户重新设置连接的参数。

3. `OnBnClickedButtonConnect()` : 连接服务器按钮点击事件

功能：此函数处理用户的连接请求。它首先初始化Winsock、获取IP、端口和用户名，然后尝试与服务器建立连接。一旦连接成功，它会发送用户名给服务器并启动一个新线程来接收服务器的消息。

4. `ReceiveMessages(LPVOID pParam)` : 接收消息线程函数

功能: 此函数在单独的线程中运行, 不断地从服务器接收消息。一旦接收到消息, 进行DES解密, 使用OnUpdateChatMsg 函数将消息发送到主线程进行显示。

5. `OnUpdateChatMsg(WPARAM wParam, LPARAM lParam)` : 更新聊天消息函数

功能: 这是一个消息处理函数, 负责处理从 ReceiveMessages 线程发送来的消息, 并在聊天窗口中显示它们。

6. `PrintMsg(const CString& Name, const CString& strMsg)` : 打印消息函数

功能: 这个函数负责在客户端的用户界面上显示消息。它格式化并将消息追加到聊天窗口中。

7. `OnClose()` : 关闭窗口函数

功能: 与 OnBnClickedButtonExit() 类似, 当用户试图关闭聊天客户端窗口时, 此函数被触发。它确保所有的资源都被适当地释放, 并通知服务器客户端的退出。

其中, 最重要的三个函数分别是连接服务器, 发送消息以及接收服务器消息线程函数。

```
1 // 连接服务器
2 void CChatRoomDlg::OnBnClickedButtonConnect() {
3     // 判断是否已经连接
4     if (SockClient != INVALID_SOCKET) {
5         MessageBox("已经连接到服务器!");
6         return;
7     }
8
9     // 初始化 Winsock
10    WSADATA wsaData = { 0 };
11    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
12        MessageBox("初始化Winsock失败!");
13        return;
14    }
15
16    // 获取IP和端口
17    CString strIP, strPort;
18    GetDlgItemText(IDC_EDIT_PORT, strPort);
19    CIPAddressCtrl* pIP = (CIPAddressCtrl*)GetDlgItem(IDC_IPADDRESS);
20    {
21        BYTE nf1, nf2, nf3, nf4;
22        pIP->GetAddress(nf1, nf2, nf3, nf4);
23        strIP.Format("%d.%d.%d.%d", nf1, nf2, nf3, nf4);
24    }
25
26    // 获取DES密钥
27    GetDlgItemText(IDC_DES_KEY, key);
28    if (key.GetLength() == 16) {
```

```

29         des = new DES(key.GetBuffer());
30     }
31     else {
32         MessageBox("无效的DES密钥! (应为16个16进制数)");
33         WSACleanup();
34         return;
35     }
36
37     // 获取用户名
38     GetDlgItemText(IDC_EDIT_NAME, UserName);
39
40     // 判断上述信息合法
41     if (strIP.IsEmpty() || strPort.IsEmpty() || UserName.IsEmpty()) {
42         MessageBox("请填写完整信息! ");
43         WSACleanup();
44         return;
45     }
46
47     // 创建套接字
48     SockClient = socket(AF_INET, SOCK_STREAM, 0);
49     if (SockClient == INVALID_SOCKET) {
50         MessageBox("创建套接字失败! ");
51         WSACleanup();
52         return;
53     }
54
55     // 设置服务器地址
56     sockaddr_in serverAddr;
57     serverAddr.sin_family = AF_INET;
58     serverAddr.sin_port = htons(_ttoi(strPort));
59     if (inet_pton(AF_INET, CT2A(strIP.GetBuffer()), &
60 (serverAddr.sin_addr)) != 1) {
61         MessageBox("无效的IP地址! ");
62         WSACleanup();
63         return;
64     }
65
66     // 连接服务器
67     if (connect(SockClient, (SOCKADDR*)&serverAddr, sizeof(serverAddr)) ==
68 SOCKET_ERROR) {
69         MessageBox("连接服务器失败! ");
70         closesocket(SockClient);
71         SockClient = INVALID_SOCKET;
72         WSACleanup();
73         return;
74     }

```

```

73
74     // 发送用户名给服务器
75     std::vector<uint8_t> encName = des-
>encrypt(DES::strToVec(UserName.GetBuffer()));
76     send(SocketClient, reinterpret_cast<const char*>(encName.data()),
encName.size(), 0);
77     MessageBox("连接成功!");
78
79     // IP, 端口, 用户名, 不再可编辑
80     GetDlgItem(IDC_IPADDRESS)->EnableWindow(FALSE);
81     GetDlgItem(IDC_EDIT_PORT)->EnableWindow(FALSE);
82     GetDlgItem(IDC_EDIT_NAME)->EnableWindow(FALSE);
83
84     // 创建一个新线程来持续接收来自服务器的消息
85     hThread = CreateThread(NULL, 0, ReceiveMessages, this, 0, NULL);
86 }

```

- 检查是否已存在一个有效的套接字连接 (`SocketClient != INVALID_SOCKET`)，如果存在，显示消息框提示用户已经连接到服务器，避免重复连接。
- 通过 `WSAStartup` 调用初始化Winsock 2.2，这是在Windows平台上进行网络通信之前必需的步骤。如果初始化失败，将弹出消息框并返回。
- 获取用户界面中输入的服务器IP地址和端口号，以及用户输入的DES密钥和用户名。这里使用 `CIPAddressCtrl` 控件来获取IP地址，并将其转换为字符串格式。对于DES密钥，它还检查了密钥长度是否为16个16进制数，以确保密钥的有效性。
- 创建一个TCP套接字 (`SOCK_STREAM`)，并根据获取到的IP地址和端口号设置服务器的地址信息，尝试连接到服务器。如果连接失败，将关闭套接字并清理Winsock资源。

连接成功后，用户输入的用户名通过DES算法加密后发送给服务器。这里，`DES::encrypt` 和 `DES::strToVec` 方法被用来将用户名字符串转换为向量并进行加密。加密后的用户名作为客户端身份的一部分发送给服务器。

另外是多线程处理，另外一个线程用于接收服务器消息：

```

1 // 接收服务器消息线程函数
2 DWORD WINAPI CChatRoomDlg::ReceiveMessages(LPVOID pParam) {
3     CChatRoomDlg* pThis = reinterpret_cast<CChatRoomDlg*>(pParam);
4     std::vector<uint8_t> buffer(BUFFER_SIZE);
5     // 循环接收服务器消息
6     while (true) {
7         buffer.clear();
8         buffer.resize(BUFFER_SIZE);
9         // 约定：服务器发送的消息格式为：用户名:消息
10        int ret = recv(pThis->SocketClient, reinterpret_cast<char*>
(buffer.data()), BUFFER_SIZE, 0);
11        if (ret <= 0) {
12            pThis->MessageBox("与服务器断开连接!");

```

```

13         closesocket(pThis->SockClient);
14         pThis->SockClient = INVALID_SOCKET;
15         return 0;
16     }
17     buffer.resize(ret);
18     // 将接收到的消息解密并发送给主线程
19     std::string decmsg = DES::vecToStr(pThis->des->decrypt(buffer));
20     pThis->PostMessage(WM_UPDATE_CHAT_MSG, 0, (LPARAM)new
CString(decmsg.c_str()));
21     }
22     closesocket(pThis->SockClient);
23     pThis->SockClient = INVALID_SOCKET;
24     delete pThis->des;
25     return 0;
26 }

```

使用 `recv` 函数从 `SockClient` 套接字接收数据，数据被存储在 `buffer` 向量中。
`BUFFER_SIZE` 定义了接收缓冲区的大小。如果 `recv` 函数返回值小于等于0，表示连接被关闭或发生错误，此时会弹出消息框通知用户与服务器断开连接，并关闭套接字。

另外，接收到的数据是加密的，所以首先需要使用 `DES::decrypt` 方法对其进行解密。

发送消息函数则使用 `send` 来进行发送加密过的信息：

```

1 // 发送消息
2 void CChatRoomDlg::OnBnClickedButtonSend() {
3     .....
4     .....
5     // 加密消息
6     std::vector<uint8_t> encMsg = des-
>encrypt(DES::strToVec(strMsg.GetBuffer()));
7     if (send(SockClient, reinterpret_cast<const char*>(encMsg.data()),
encMsg.size(), 0) == SOCKET_ERROR) {
8         MessageBox("发送消息失败！");
9         isConnected = false;
10    }
11    .....
12    .....
13 }

```

3.4 TCP服务端

定义了服务器类。由于我实现了多人聊天，故定义了客户结构体，由服务器来进行维护：

```

1 // 定义客户结构体
2 struct Client {
3     SOCKET sock; // 套接字

```

```

4     string username;                // 用户名
5     Client(SOCKET sock = INVALID_SOCKET, string username = "$") :
    sock(sock), username(username) {}
6 };
7
8 // 聊天室服务器类
9 class ChatRoomServer {
10 public:
11     ChatRoomServer(UINT port, UINT client, string key);    // 构造函数
12     ~ChatRoomServer();    // 析构函数
13     void Start();    // 启动服务器
14     void Stop();    // 关闭服务器
15     void PrintInfo(const string& info);    // 输出日志
16
17 private:
18     // 定义服务器相关常量
19     UINT MAX_CLIENTS;    // 最大客户端数量
20     UINT PORT;    // 服务器端口
21     string KEY;    // 密钥
22     constexpr static UINT BUFFER_SIZE = 1024;    // 缓冲区大小
23
24     // 定义服务器相关变量
25     SOCKET SockServer = INVALID_SOCKET;    // 服务器套接字
26     Client* clients;    // 客户端数组
27     HANDLE* hThreads;    // 线程句柄, 每个客户端均有一个线程来处理
28     HANDLE hCommandThread;    // 服务器命令线程句柄
29     UINT hpointer = 0;    // 线程句柄数组的指针
30     sockaddr_in addrServer;    // 服务器地址
31     bool shouldRun = true;    // 用于标记服务器是否应继续运行
32     DES* des;    // DES加密解密对象
33
34     // 定义服务器相关函数
35     void InitWinSock();    // 初始化WinSock
36     int find_pos();    // 查找空闲的客户端存放位置
37     UINT Online_Count();    // 获取在线人数
38     static DWORD WINAPI ClientHandler(LPVOID pParam);    // 每个客户的线程函数
39     void BroadcastMessage(const string& msg);    // 将消息广播给所有客户端
40     string GetCurrTime();    // 获取当前时间
41     static DWORD WINAPI ListenForCommand(LPVOID pParam);    // 监听服务器命令
42 };

```

重要函数及其功能:

ChatRoomServer::InitWinSock(): 初始化WinSock

功能：这个函数用于启动WinSock 2.2版本。初始化过程中，如果发生错误，程序会输出错误信息并退出。

ChatRoomServer::Online_Count()：计算在线客户端数量

功能：遍历客户端数组，统计并返回有效套接字的数量，从而得知当前在线的客户端数量。

ChatRoomServer::Start()：启动服务器

功能：启动服务器的主循环，并准备接收来自客户端的连接。

- **创建服务器套接字：**首先使用 `socket()` 函数创建一个新的套接字。
- **绑定套接字：**使用 `bind()` 函数将新创建的套接字绑定到指定的IP地址和端口上。
- **开始监听：**通过 `listen()` 函数使服务器开始监听客户端的连接请求。
- **接受客户端连接：**使用 `accept()` 函数接受来自客户端的连接。对于每一个成功的连接，都会在服务器中为该客户端分配一个新的套接字。
- **创建客户端线程：**每当有新的客户端连接时，都会为这个客户端创建一个新的线程来处理它的消息。这确保了服务器能够并发地处理多个客户端。

ChatRoomServer::ClientHandler(LPVOID pParam)：处理客户端消息线程函数

功能：为每一个连接的客户端独立执行，处理来自客户端的消息，并与其他客户端进行交互。

- **获取用户名：**首先，这个函数从客户端接收其用户名。这是客户端首次与服务器交互的部分。获取的是加密后的用户名，需要调用DES模块来解密，
- **发送欢迎消息：**为新连接的客户端发送一个欢迎消息，并广播给所有其他在线的客户端。
- **消息循环：**函数接着进入一个循环，不断地接收来自客户端的消息并解密，并将其广播给其他客户端。
- **断开连接处理：**如果客户端发送了退出消息或者由于某种原因与服务器断开了连接，该函数会广播这个客户端的退出消息，然后关闭与该客户端的连接。

代码如下：

```
1 // 每个客户的线程函数
2 DWORD WINAPI ChatRoomServer::ClientHandler(LPVOID pParam) {
3     ChatRoomServer* pThis = reinterpret_cast<ChatRoomServer*>(pParam);
4     std::vector<uint8_t> buffer(BUFFER_SIZE);
5     int bytes;
6
7     // 获取当前客户端
8     Client* client = &pThis->clients[pThis->hpointer];
9
10    // 读取客户端的用户名
11    bytes = recv(client->sock, reinterpret_cast<char*>(buffer.data()),
12    BUFFER_SIZE, 0);
13    if (bytes <= 0) {
14        closesocket(client->sock);
```



```

14         return 0;
15     }
16     // 使用 std::vector<uint8_t> 截取实际接收的数据长度
17     buffer.resize(bytes);
18
19     // 解密
20     client->username = DES::vecToStr(pThis->des->decrypt(buffer));
21
22     // 发送欢迎消息
23     string welcomeMsg = "欢迎 " + client->username + " 加入聊天室!";
24     pThis->PrintInfo(client->username + " 加入聊天室.");
25     pThis->BroadcastMessage("系统消息:" + welcomeMsg);
26
27     // 循环接收客户端消息
28     while (true) {
29         buffer.clear();
30         buffer.resize(BUFFER_SIZE);
31         // 接收客户端信息, 无需用户名
32         bytes = recv(client->sock, reinterpret_cast<char*>(buffer.data()),
pThis->BUFFER_SIZE, 0);
33         // 解密
34         buffer.resize(bytes);
35         string decmsg = DES::vecToStr(pThis->des->decrypt(buffer));
36
37         // 客户端发送退出消息或异常断开连接
38         if (bytes <= 0 || decmsg == "exit") {
39             // 广播客户端退出消息
40             string exitMsg = client->username + " 已退出聊天室." + "(当前在
线人数: " + to_string(pThis->Online_Count() - 1) + ")";
41             pThis->PrintInfo(exitMsg);
42             pThis->BroadcastMessage("系统消息:" + exitMsg);
43
44             // 客户端断开连接
45             closesocket(client->sock);
46             client->sock = INVALID_SOCKET;
47             break;
48         }
49
50         // 正常广播消息 约定消息格式为 "用户名:消息内容"
51         string message = client->username + ":" + decmsg;
52         pThis->PrintInfo("正在广播来自 " + client->username + " 的消息: " +
decmsg);
53         pThis->BroadcastMessage(message);
54     }
55     return 0;
56 }

```

`ChatRoomServer::BroadcastMessage(const string& msg)`：广播消息

- 遍历客户端：遍历所有已连接的客户端。
- 发送消息：DES加密后使用 `send()` 函数将指定的消息发送给每一个在线的客户端。

代码如下：

```
1 // 广播消息给所有客户端
2 void ChatRoomServer::BroadcastMessage(const string& msg) {
3     // 先将string转换为vector<uint8_t>加密
4     std::vector<uint8_t> encmsg = des->encrypt(DES::strToVec(msg));
5     for (UINT i = 0; i < MAX_CLIENTS; i++) {
6         if (clients[i].sock != INVALID_SOCKET) {
7             // 发送数据
8             send(clients[i].sock, reinterpret_cast<const char*>
(encmsg.data()), encmsg.size(), 0);
9         }
10    }
11 }
```

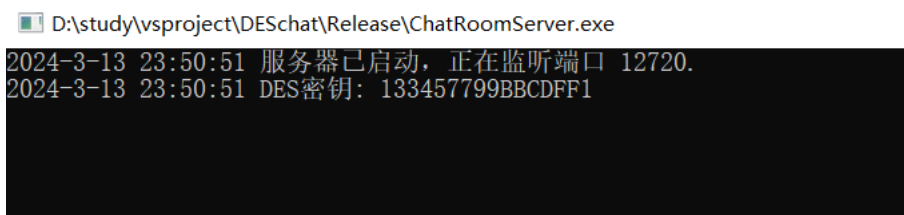
`ChatRoomServer::ListenForCommand(LPVOID pParam)`：处理服务器命令线程函数

功能：这个函数在一个单独的线程上运行，监听并处理来自服务器管理员在控制台上输入的命令。

- 命令循环：函数在一个循环中不断地监听控制台的输入。
- 处理 `exit` 命令：如果输入的命令是 `exit`，这个函数将修改服务器的状态变量使其停止运行，并关闭服务器套接字以使 `accept()` 函数返回并退出主循环。
- 处理 `count` 命令：如果输入的命令是 `count`，这个函数将计算并输出当前在线的客户端数量。

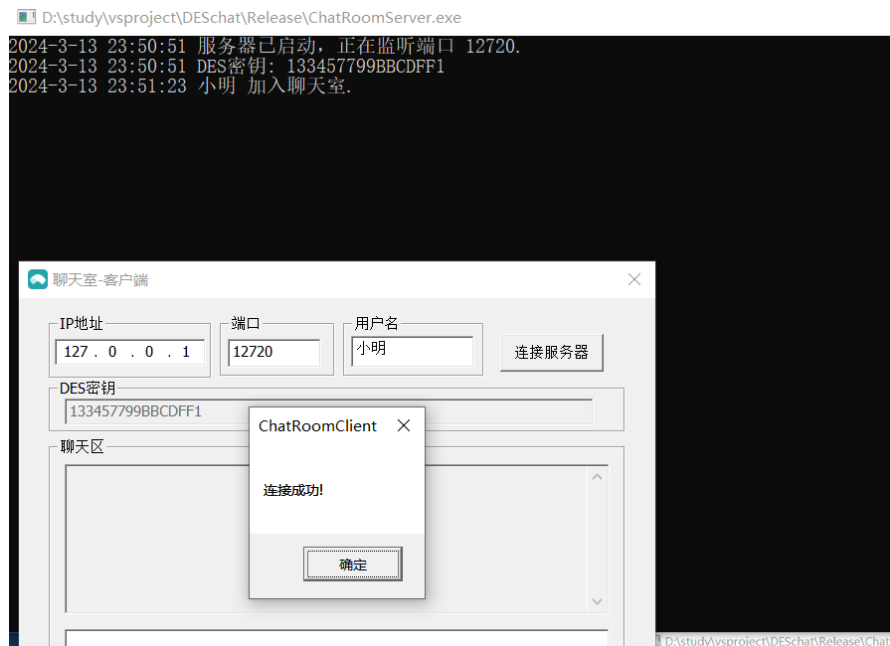
3.5 运行方法及示例

首先运行 `ChatRoomServer.exe`，显示端口以及DES密钥(ip为127.0.0.1)：

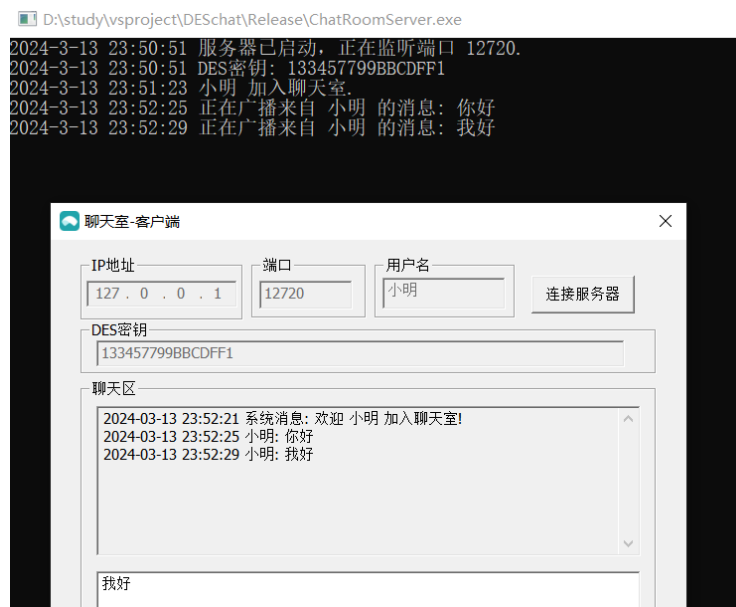


```
D:\study\vsproject\DESchat\Release\ChatRoomServer.exe
2024-3-13 23:50:51 服务器已启动，正在监听端口 12720.
2024-3-13 23:50:51 DES密钥: 133457799BBCDFF1
```

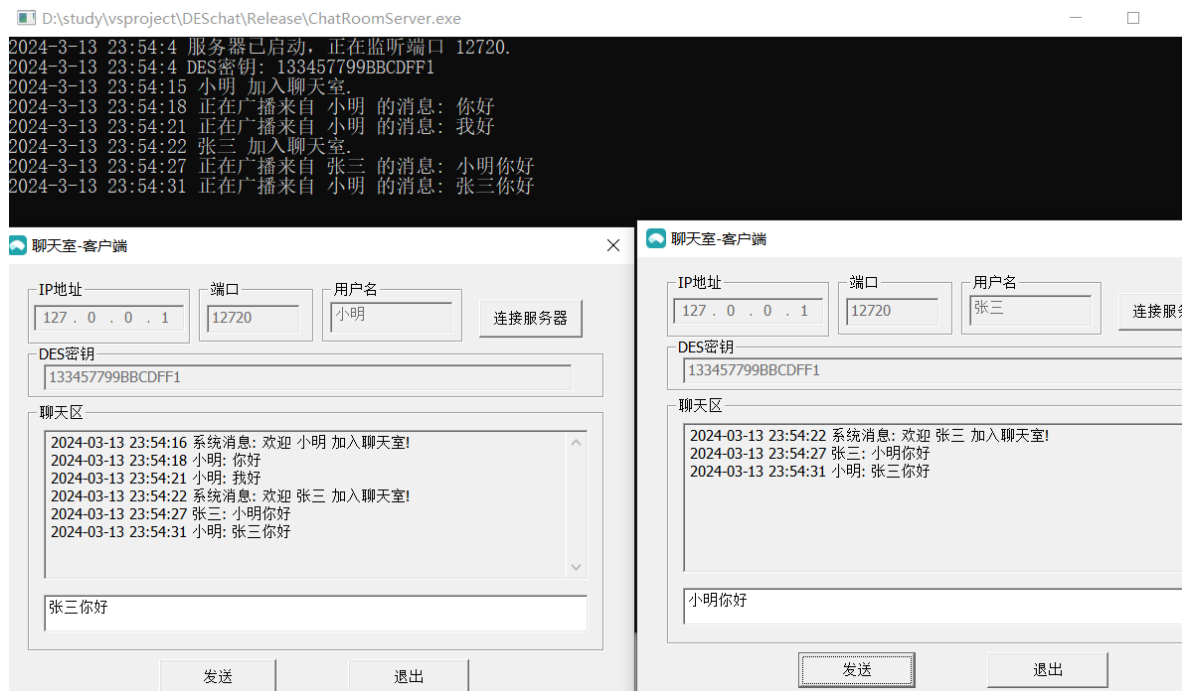
启动 `ChatRoomClient.exe`，输入任意用户名(DES密钥已固定，防止与服务器不同)，点击连接服务器：



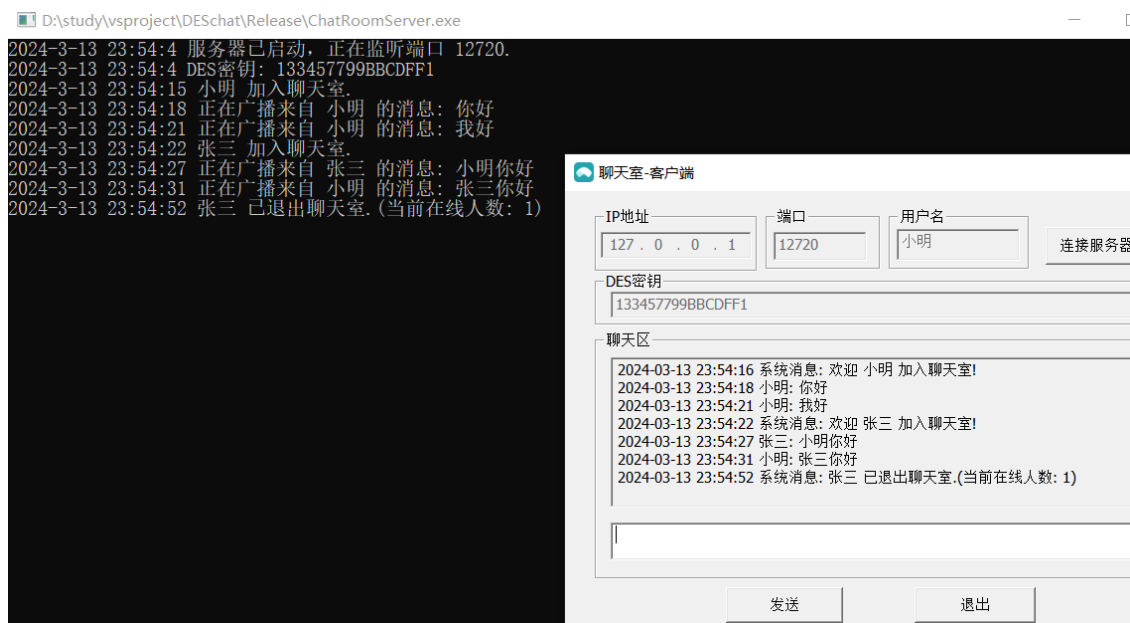
此时客户端可发送任意消息：



支持多人聊天，打开另一个客户端，进行连接后，两个人(或更多)接下来可以任意进行实时聊天了：

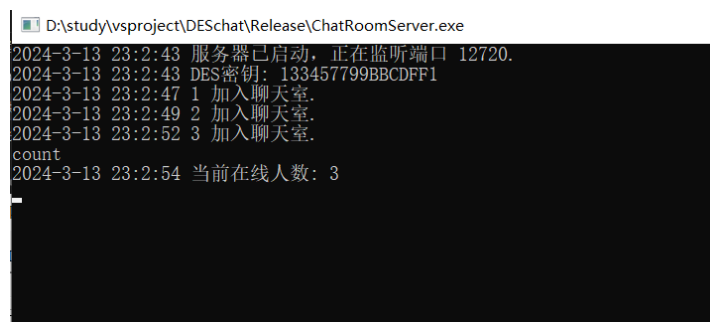


途中若有一人离开聊天，服务器将对其他所有人进行提示：



退出服务端，使用 `Ctrl+C` 即可，退出客户端，点击 `退出` 按钮即可。

另外，服务器可以通过输入 `count` 来查询当前在线的客户数量：



4 实验遇到的问题及其解决方法

- 加密数据长度并不总是块的整数倍

在DES加密中，数据的长度必须是64位的整数倍。为了解决这个问题，采用了PKCS#7填充方案对数据进行填充，确保数据长度符合要求：

```
1  std::vector<uint8_t> DES::pad(const std::vector<uint8_t>& data, size_t
    blockSize) {
2      std::vector<uint8_t> paddedData = data;
3      size_t padLength = blockSize - (data.size() % blockSize);
4      for (size_t i = 0; i < padLength; i++) {
5          paddedData.push_back(static_cast<uint8_t>(padLength));
6      }
7      return paddedData;
8  }
9
10 std::vector<uint8_t> DES::unpad(const std::vector<uint8_t>& data, size_t
    blockSize) {
11     if (data.empty()) return {};
12     size_t padLength = data.back();
13     if (padLength > data.size() || padLength > blockSize) return data;
14     return std::vector<uint8_t>(data.begin(), data.end() - padLength);
15 }
```

- 使用string作为接口导致解密数据出错

在原先我使用了string作为接口：

```
1  std::string DES::encrypt(const std::string& plaintext) {
2      .....
3      .....
4  }
5
6  std::string DES::decrypt(const std::string& ciphertext) {
7      .....
8      .....
9  }
```

但是结合到TCP数据传输，发现这样会导致数据解密失败出错。

原因：加密传输的数据，可能中间就含有`\0`，而string则以第一个`\0`作为结尾，将会导致丢失数据，并且由于数据不完整，无法进行解密直接会报错。

解决办法是使用`std::vector<uint8_t>`作为接口，这样无论数据流中间是否有`\0`，我们均可不丢失长度。

- bitset的操作问题

① std::bitset<size>的下标是从低位开始遍历的

由于DES的常量是**从高位开始**定义顺序的，而bitset按下标递增是**从低位到高位**的，因此我额外使用两个函数来辅助简化算法实现：

```
1 uint8_t get(const std::bitset<N>& b, size_t pos) { return b[N - 1 - pos]; }
2 void set(std::bitset<N>& b, size_t pos, size_t value) { b[N - 1 - pos] = value; }
```

② 循环左移问题

c++的<<操作符并不会进行循环左移，我们需要自己进行循环左移操作：

```
1 template<size_t N>
2 std::bitset<N> DES::leftRotate(const std::bitset<N>& bits, int shift) {
3     return (bits << shift) | (bits >> (N - shift));
4 }
```

循环左移的实现利用了std::bitset的特性，将位集向左移指定位数，**同时将左端溢出的位重新从右端插入**。这种操作保证了密钥的位结构在旋转过程中不丢失任何信息。

5 实验结论

通过本次实验，我们成功实现了一个基于DES加密算法的TCP聊天程序。此实验不仅加深了我们对DES算法的理解，包括其密钥生成、数据加解密过程等，也让我们熟悉了TCP协议的基本使用方法，如创建套接字、连接服务器、发送和接收消息等。