

网络安全技术实验报告

学院 网安学院

年级 大三

班级 信息安全单学位班

学号 1811494

姓名 刘旭萌

手机号 17320205058

网络安全技术实验报告

一、实验目的

二、实验内容

三、实验步骤及实验结果

DES加密

加密大致流程

密钥生成

轮加密

TCP通信

通信模型图

TCP通信原理

代码

初始设置

socket的创建 socket()

server绑定套接字和地址 bind()

server生成初始密钥

server进行监听 listen()

client发起连接请求 connect()

server端受连接请求 accept()

消息的加密与解密

client加工初始密钥

client发送/接受数据

server转发报文

程序的退出

四、实验遇到的问题及其解决方法

五、实验结论

一、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 或Windows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。

二、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。

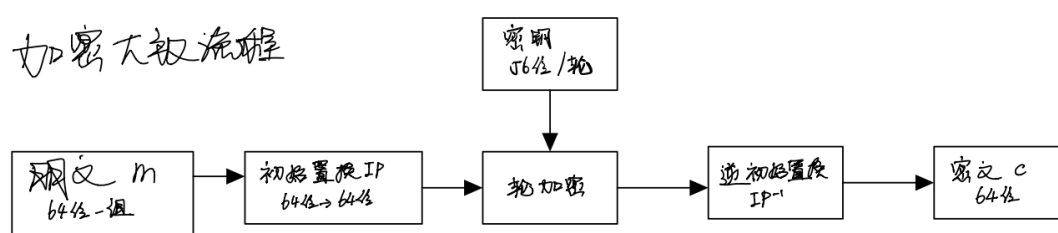
① 能够在了解 DES 算法原理的基础上，编程实现对字符串的 DES 加密解密操作。② 能够在了解 TCP 和 Linux 平台下的 Socket 运行原理的基础上，编程实现简单的 TCP 通信，为简化编程细节，不要求实现一对多通讯。③ 将上述两部分结合到一起，编程实现通信内容事先通过 DES 加密的 TCP 聊天程序，要求双方事先互通密钥，在发送方通过该密钥加密，然后由接收方解密，保证在网络上传输的信息的保密性。

三、实验步骤及实验结果

DES加密

加密大致流程

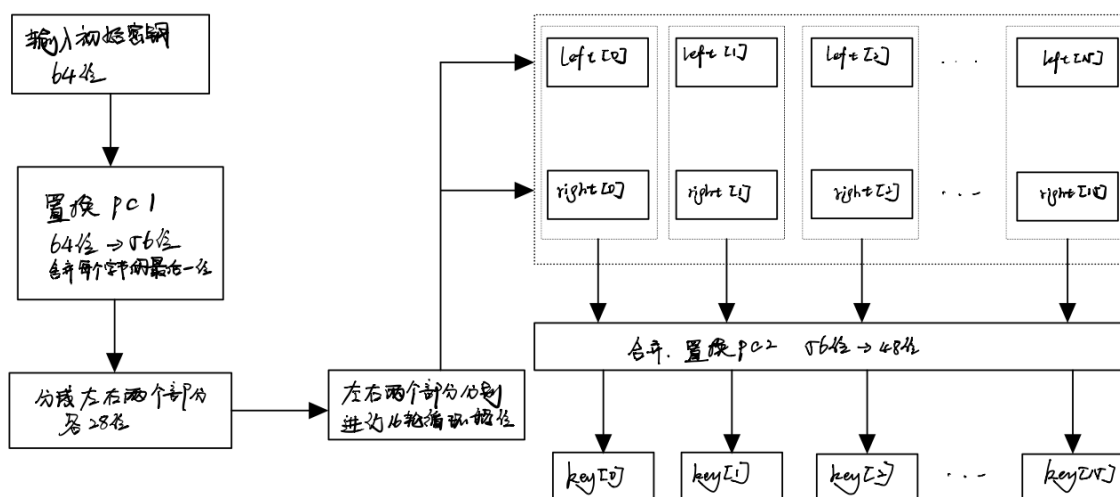
(解密流程相似，只是密钥使用顺序相反)



```
1 void round(BYTE m[8], BYTE key_final[16][6], BYTE afterIP_1[8])
2 {
3     //m_为明文, key_final为生成的十六组密钥, afterIP用于存储密文
4     BYTE m[8] = {};
5     convIP(m_, m); //初始置换IP
6     .... //轮加密
7
8     convIP_1(afterCombine, afterIP_1); //逆初始置换
9 }
```

密钥生成

密钥生成

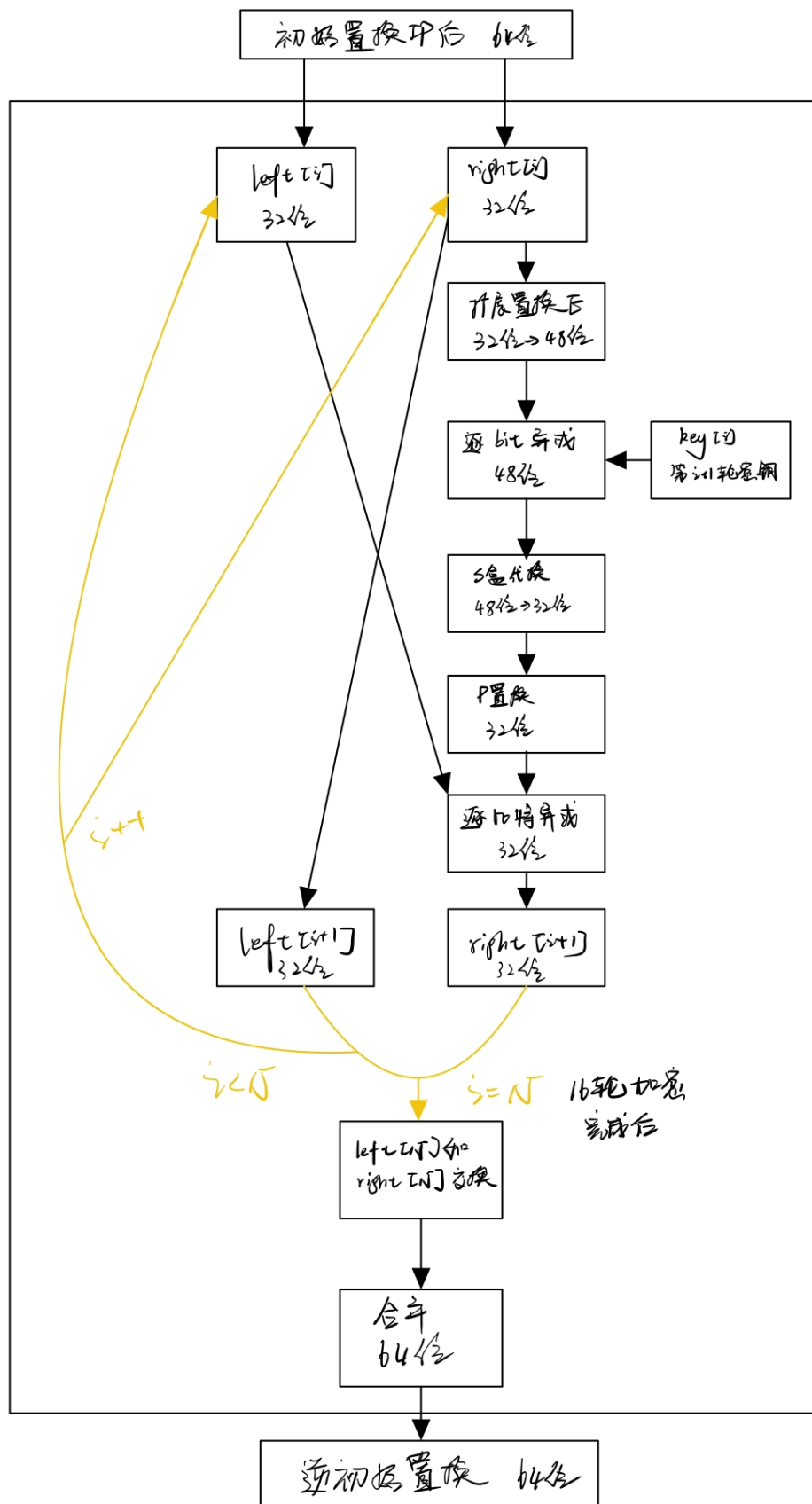


```

2 void getkeys(BYTE key[8], BYTE key_final[16][6])
3 {
4     BYTE keys[7] = {}; //存储经过PC1置换后的56位中间结果
5     for (int i = 0; i < 56; i++)
6         convert(key, keys, i, PC_1); //PC1置换
7     bitset<56> key_round[17];
8     //初始化全为0
9     for (int i = 0; i < 16; i++) key_round[i].reset();
10    //信息复制, 将BYTE数组格式转为bitset格式, 便于后续处理
11    for (int i = 0; i < 7; i++)
12    {
13        for (int j = 0; j < 8; j++)
14        {
15            if (getbit(keys[i], j) == 1)
16                key_round[0].set(i * 8 + j); //该位设为1
17        }
18    }
19    for (int i = 0; i < 16; i++) //28位一组循环移位操作
20    {
21        bitset<56> left_mov = key_round[i] >> MOV[i]; //注意大小端序问题, 实际是
左移
22        for (int j = 0; j < MOV[i]; j++)
23        {
24            left_mov.set(28 - MOV[i] + j, key_round[i][j]);
25            left_mov.set(56 - MOV[i] + j, key_round[i][28 + j]);
26        }
27        key_round[i+1] = left_mov;
28    }
29
30    //PC_2置换, 把bitset转成BYTE, 便于观察中间结果
31    for(int i=0;i<16;i++)
32        for (int j = 0; j < 48; j++)
33        {
34            write(key_final[i][j/8], j%8, key_round[i + 1][PC_2[j] - 1]);
35        }
36    }

```

轮加密



```

1 void round(BYTE m[8], BYTE key_final[16][6], BYTE afterIP_1[8])
2 {
3     BYTE m[8] = {};
4     ...; //初始置换IP
5
6     //轮加密过程
7     BYTE left[4], right[4];
8     for (int i = 0; i < 4; i++) //分成左右两部分
  
```

```

9      {
10         left[i] = m[i];
11         right[i] = m[i+4];
12     }
13     for (int i = 0; i < 16; i++)//十六轮加密
14     {
15         BYTE afterE[6] = {}; //扩展置换后的结果
16         //扩展置换E
17         convertE(right, afterE);
18         //与密钥进行逐bit异或
19         BYTE afterXOR[6] = {};
20         xor_(key_final[i], afterE, afterXOR);
21         //S盒代换
22         BYTE afterS[4] = {};
23         convertS(afterXOR, afterS);
24         //P置换
25         BYTE afterP[4] = {};
26         convertP(afterS, afterP);
27         BYTE afterXOR2[4] = {};
28         for (int i = 0; i < 4; i++)//逐比特异或
29         {
30             afterXOR2[i] = left[i] ^ afterP[i];
31         }
32         //生成下一轮的left和right
33         for (int i = 0; i < 4; i++) left[i] = right[i];
34         for (int i = 0; i < 4; i++) right[i] = afterXOR2[i];
35         int ix = 0;
36     }
37
38     //左右交换
39     BYTE temp[4] = {};
40     for (int i = 0; i < 4; i++)
41     {
42         temp[i] = left[i];
43         left[i] = right[i];
44         right[i] = temp[i];
45     }
46     //合并
47     BYTE afterCombine[8];
48     for (int i = 0; i < 4; i++)
49     {
50         afterCombine[i] = left[i];
51         afterCombine[i + 4] = right[i];
52     }
53     ...//逆初始置换
54 }

```

其中，为简化各个转换过程，使用了一个转换函数

```

1 //做对应转换
2 void convert(u_char a[], u_char b[], int i, int conv[])//conv规则中的第i位完成
  转换写入到b中
3 {
4     int index_bit_b = i;
5     tuple_index_u_char_b = bit2u_char(index_bit_b);//i=(i/u_char_)*8+
      (i%u_char_)
6     int index_bit_a = conv[index_bit_b] - 1;//找到对应位置
7     tuple_index_u_char_a = bit2u_char(index_bit_a);
8     int temp = getbit(a[tuple_index_u_char_a.u_char_], index_u_char_a.bit_);//提取
      出第i位
9     write(b[tuple_index_u_char_b.u_char_], index_u_char_b.bit_, temp);//写入b中
10 }

```

转换函数调用样例

```

1 //初始置换
2 void convIP(u_char a[], u_char b[])//将a进行IP置换，结果存入b中
3 {
4     for (int i = 0; i < 64; i++)
5     {
6         convert(a, b, i, IP);
7     }
8 }

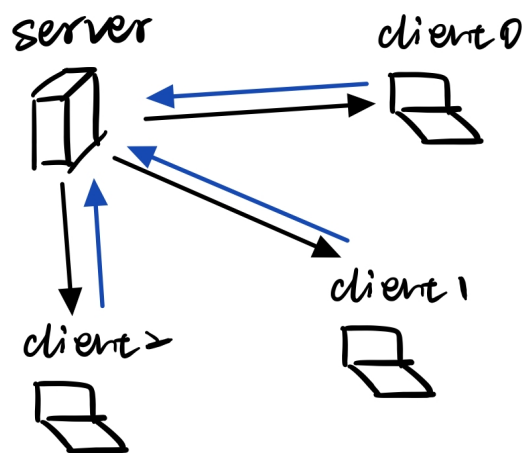
```

TCP通信

通信模型图

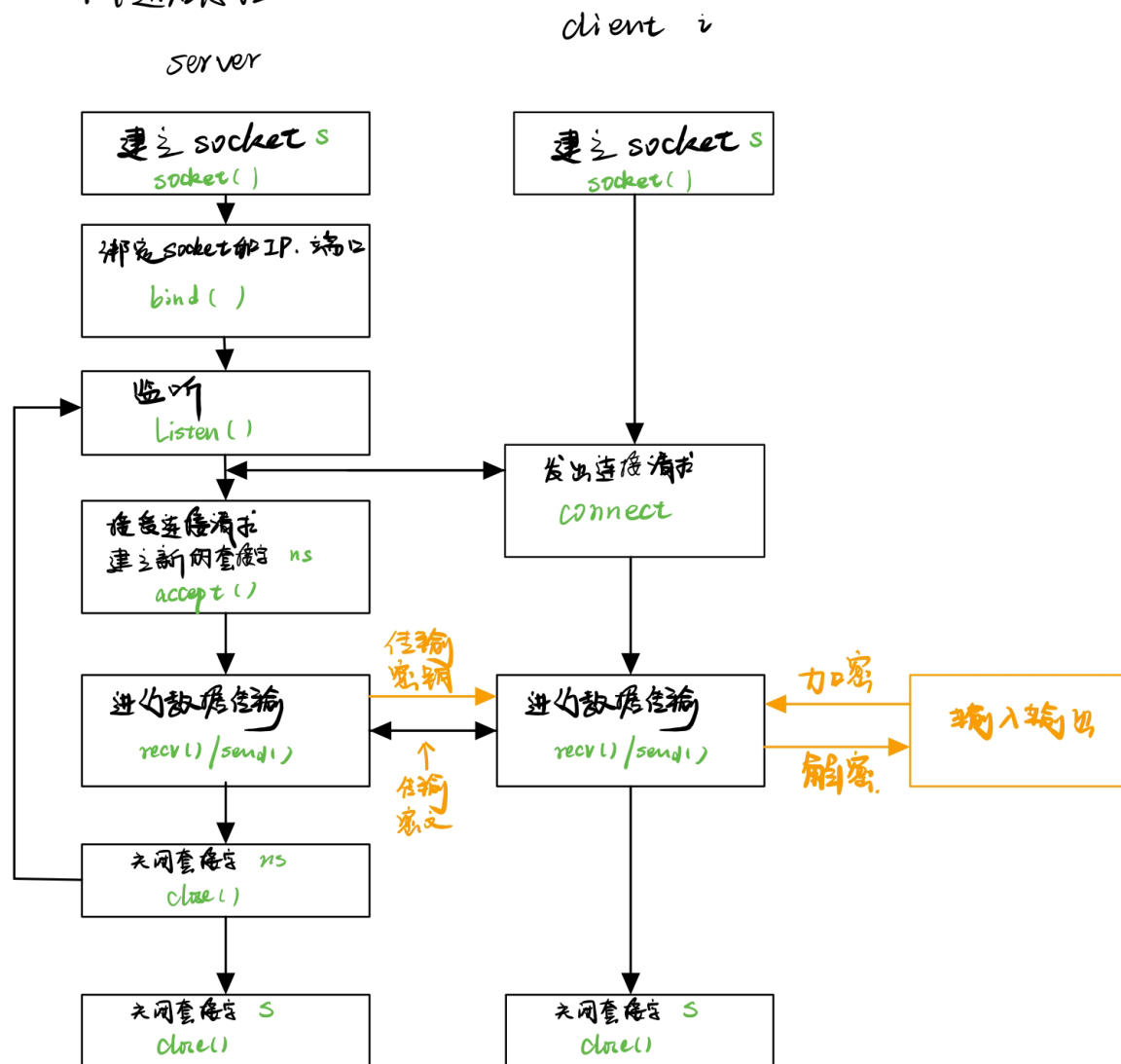
本次实验为了满足多对多通信，采用C/S模型进行设计，利用server对消息进行转发以实现多个client端之间的通信

C/S 模型



TCP通信原理

TCP通信原理



代码

初始设置

```

1 #include <winsock2.h> // windows socket 编程头文件
2 #pragma comment(lib, "ws2_32.lib") // 链接 ws2_32.lib 库文件到此项目中

```

```

1 // 加载 socket 库
2 WSADATA wsaData;
3 // MAKEWORD(2, 2), 成功返回 0
4 if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
5 {
6     cout << "socket 初始化失败" << endl;
7     return 0;
}

```

socket 的创建 socket()

```

1 // 创建一个 socket, 并将该 socket 绑定到一个特定的传输层
2 sockSer = socket(AF_INET, SOCK_STREAM, 0); // 地址类型 (ipv4), 服务类型 (流式套接字)

```

server 绑定套接字和地址 bind()

```

1 //初始化地址，client端地址初始化方法相同
2 addrSer.sin_addr.s_addr = inet_addr("127.0.0.1");//本次由于只在内网中测试，使用私有地址
3 addrSer.sin_family = AF_INET;
4 addrSer.sin_port = htons(PORT);
5
6 //绑定
7 if (bind(sockSer, (SOCKADDR*)&addrSer, sizeof(SOCKADDR)) == -1)
8     cout << "bind error" << endl;//强制类型转换，SOCKADDR_IN方便赋值 SOCKADDR方便传输

```

server生成初始密钥

本次加密统一采用**64位**的二进制串作为初始密钥

```

1 string initial_key()//server生成64位初始密钥
2 {
3     string key="";
4     for (int i = 0; i < 8; i++)
5     {
6         key += rand() % 128;
7     }
8     return key;
9 }

```

```

1 string key = initial_key();

```

server进行监听 listen()

```

1 listen(sockSer, 5);//等待队列最大长度为5

```

client发起连接请求 connect()

```

1 sockCli = connect(sockClient, (SOCKADDR*)&addrSer, sizeof(SOCKADDR));

```

待server端accept连接请求后，client端会处理server端分配的id和密钥

server端受连接请求 accept()

```

1 SOCKET sockConn[CLIENTNUM];//和多个client连接

```

server端接受连接后，向client发送**clientID**和**会话密钥**


```

1  sockConn[i] = accept(sockSer, (SOCKADDR*)&addrCli, &len); //失败
   sockConn=INVALID_SOCKET
2  if (sockConn[i] != INVALID_SOCKET)
3  {
4      cond++; //人数加一
5      string buf = "你的id是: ";
6      buf+= 48 + i; //简化, 最多九个人
7      send(sockConn[i], buf.data(), 50, 0);
8      send(sockConn[i], key.data(), 50, 0);
9      cout << "clients " << i << " have connected" << endl;
10 }

```

为了便于实验观察, 这里client会将密钥打印出来



```

Server
listening
clients 0 have connected
clients 1 have connected

Client
请输入端口号: 1
connecting
connected
你的id是: 0
密钥:  29 23 3E 04 61 6C 56 2E

Client
请输入端口号: 2
connecting
connected
你的id是: 1
密钥:  29 23 3E 04 61 6C 56 2E

```

消息的加密与解密

每次加密解密只能处理64位信息, 所以需要**分组处理**

```

1  char* msg_en(char* a, u_char key_final[16][6], char * cipher) {
2      int len = strlen(a);
3      u_char temp[8], tempcipher[8];
4      int index = 0;
5      for (int i = 0; i < len; i+=8) //分组处理, 每组64位8个字节
6      {
7          memset(temp, 0, sizeof(temp));
8          memset(tempcipher, 0, sizeof(tempcipher));
9          for (int j = 0; j < 8; j++)
10         {
11             temp[j] = (u_char)a[i + j];
12         }
13         round(temp, key_final, tempcipher);
14         for (int j = 0; j < 8; j++)
15         {
16             cipher[index++] = tempcipher[j];
17         }
18     }
19     return cipher;
20 }

```

client加工初始密钥

为了简化执行流程，加解密的**16轮密钥**在**通话开始前生成**

函数 `getkeys` 已在上一节 `des` 部分中详细解释

```
1 //生成密钥
2 getkeys((u_char*)recvBuf, key_final); //加密密钥
3 for (int i = 0; i < 16; i++)
4     for (int j = 0; j < 6; j++)
5         key_final[15 - i][j] = key_final[i][j]; //解密密钥
```

client发送/接受数据

为了实现**全双工通信**，即双方可以自由指定每一轮通信的发送方和接收方，使用**多线程**变成处理**接收和转发过程**

发送消息

```
1 DWORD WINAPI handlerRequest1(LPVOID lparam)
2 {
3     while (1)
4     {
5         char sendBuf[BUF_SIZE] = {};
6         char buffer[BUF_SIZE] = {};
7         SOCKET socketClient = (SOCKET)(LPVOID)lparam;
8         cin.getline(buffer, 2048, '\n');
9         if (buffer[0]) //如果检测到了键盘输入
10        {
11            sendBuf[0] = ID + 48; //消息格式封装
12            msg_form m = string_to_msg(sendBuf);
13            if (!strcmp(m.msg, "quit") || !strcmp(buffer, "quit")) //不是程序
14                //退出指令
15                { //退出程序
16                    sendBuf[1] = ID + 48;
17                    strcat(sendBuf, buffer);
18                    send(socketClient, sendBuf, 2048, 0);
19                    cond = 1;
20                    return 0;
21                }
22            //提取消息内容
23            sendBuf[1] = buffer[0];
24            int i;
25            for (i = 1; buffer[i] != 0; i++)
26                buffer[i - 1] = buffer[i];
27            buffer[i - 1] = 0;
28            char buf[BUF_SIZE] = {};
29            msg_en(buffer, key_final, buf); //对消息内容进行加密
30            strcat(sendBuf, buf); //拼接报头（发送方和接收方id）
31            send(socketClient, sendBuf, 2048, 0); //发送
32        }
33        Sleep(200);
34    }
35    return 0;
36 }
```

接收消息

```
1 DWORD WINAPI handlerRequest2(LPVOID lparam)
```

```

2  {
3      while (1)
4      {
5          char recvBuf[BUF_SIZE] = {};
6          SOCKET socketClient = (SOCKET)(LPVOID)lparam;
7          recv(socketClient, recvBuf, 2048, 0);
8          if (recvBuf[0])
9              {/*如果有接收到的内容*/
10                 cout << recvBuf[0] << ": ";
11                 int i;
12                 for (i = 2; recvBuf[i] != 0; i++)
13                     recvBuf[i - 2] = recvBuf[i];/*去掉报头*/
14                 recvBuf[i - 2] = 0;
15                 char buf[BUF_SIZE] = {};
16                 msg_de(recvBuf, key_final_, buf);/*对消息进行解密*/
17                 cout << buf << endl;/*打印明文*/
18                 int k = 1;
19             }
20             sleep(200);
21     }
22     return 0;
23 }

```

主函数

```

1  while(1)
2  {
3      hThread1 = ::CreateThread(NULL, NULL, handlerRequest1,
4      LPVOID(sockClient), 0, &dwThreadId1);
5      hThread2 = ::CreateThread(NULL, NULL, handlerRequest2,
6      LPVOID(sockClient), 0, &dwThreadId2);
7      WaitForSingleObject(hThread1, 200);/*由于线程函数中有while(1)，该函数只需要创
8      建一个线程*/
9      WaitForSingleObject(hThread2, 200);
10     /*CloseHandle(hThread2);
11     CloseHandle(hThread1);*/
12     if (cond) break;
13 }

```

server转发报文

收到的message格式为，所以第一位是**发送方ID**，第二位是**接收方ID**

```

1  class msg_form
2  {
3  public:
4
5      char from_name;/*发送方标号*/
6      char to_name;/*接收方*/
7      char msg[BUF_SIZE];/*消息内容*/
8      ...
9  }

```

需要提取接收方ID，并顶向传送消息

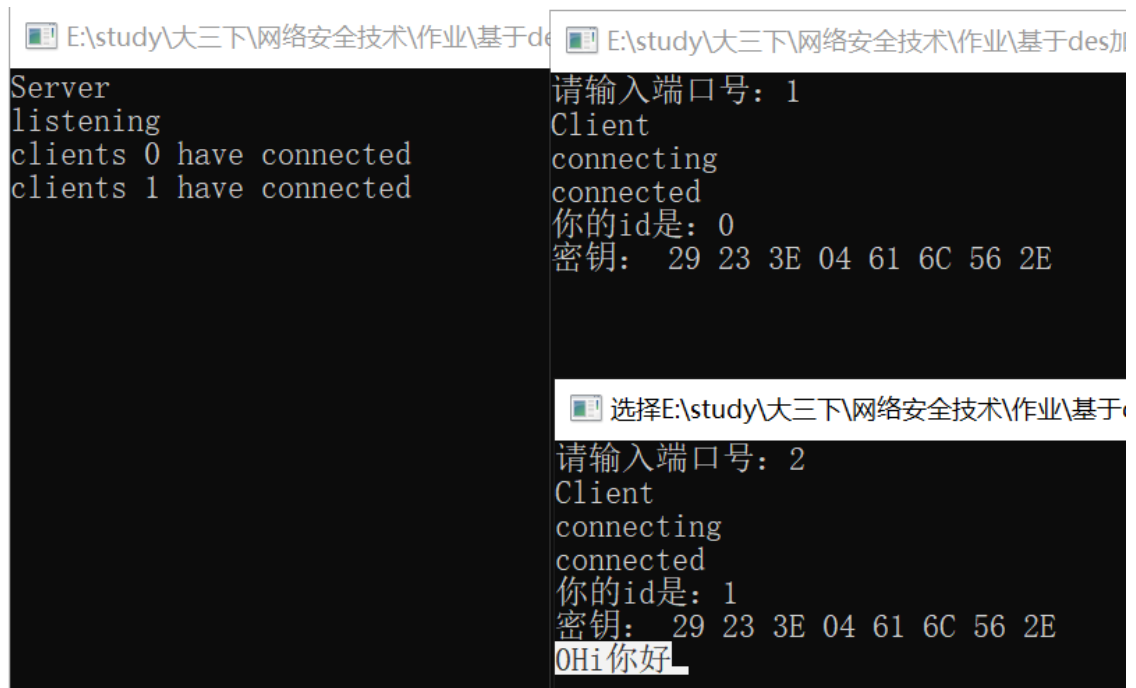
```

1 char recvBuf[BUF_SIZE] = {};
2 recv(sockettemp, recvBuf, 1000, 0);
3 if (recvBuf[0])
4 {
5     cout << recvBuf << endl;
6     msg_form m = string_to_msg(recvBuf);
7     int id = m.to_name - 48;
8     send(sockConn[id], recvBuf, 50, 0);
9     ...
10 }

```

通信过程样例：

1. 输入目的clientid和消息内容



```

Server
listening
clients 0 have connected
clients 1 have connected

Client
请输入端口号: 1
connecting
connected
你的id是: 0
密钥: 29 23 3E 04 61 6C 56 2E

Client
请输入端口号: 2
connecting
connected
你的id是: 1
密钥: 29 23 3E 04 61 6C 56 2E
0Hi你好

```

2. server收到消息第一位为发送方ID，第二位为目的方ID，后面为消息密文，可以看到，消息已经成功加密并以密文形式传输

```

Server
listening
clients 0 have connected
clients 1 have connected
10?鴉?

```

3. client0收到消息并进行解密

```
请输入端口号: 1
Client
connecting
connected
你的id是: 0
密钥: 29 23 3E 04 61 6C 56 2E
1: Hi你好
```

```
请输入端口号: 2
Client
connecting
connected
你的id是: 1
密钥: 29 23 3E 04 61 6C 56 2E
0Hi你好
```

5. 多个用户之间通讯示例

[illegible]

```
E:\study\大三下\网络安全技术\作业\基于des加密的tcp聊天\client\Debug\client.exe
Server
请输入聊天室人数:
3
listening
clients 0 have connected
clients 1 have connected
clients 2 have connected
21 U则o□
21 U则o□
20 U则o□

E:\study\大三下\网络安全技术\作业\基于des加密的tcp聊天\client\Debug\client.exe
Client
请输入端口号: 1
connecting
connected
你的id是: 0
密钥: 44 04 1A 48 1D 72 2A 49
2: Hi

E:\study\大三下\网络安全技术\作业\基于des加密的tcp聊天\client\Debug\client.exe
Client
请输入端口号: 2
connecting
connected
你的id是: 1
密钥: 44 04 1A 48 1D 72 2A 49
2: Hi
2: Hi

E:\study\大三下\网络安全技术\作业\基于des加密的tcp聊天\client\Debug\client.exe
Client
请输入端口号: 3
connecting
connected
你的id是: 2
密钥: 44 04 1A 48 1D 72 2A 49
1Hi
1Hi
0Hi
```

客户端输入 quit，向server发送quit消息后主动退出

server端收到client发送的 `quit` 指令判断该client退出，当所有client都断开连接时退出程序，关闭socket

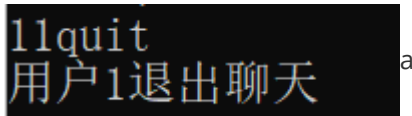
```
1  if (!strcmp(m.msg, "quit"))
2  {
3      cout << "用户" << (int)lparam << "退出聊天" << endl;
4      cond--; if (!cond) return 0;
5  }
```

关闭socket

```
1  closesocket(sockSer);
2  WSACleanup();
```

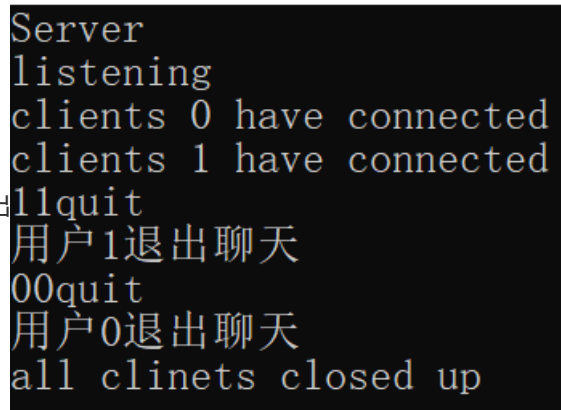
例如，

1. client1发送 `quit` ,发送后客户端退出，server显示该用户断开连接



```
1quit
用户1退出聊天
```

2. client0发送 `quit` ,所有连接断开，server退出



```
Server
listening
clients 0 have connected
clients 1 have connected
1quit
用户1退出聊天
00quit
用户0退出聊天
all clinets closed up
```

四、实验遇到的问题及其解决方法

1. server端和client端不能相互通信
 - 经检查发现是server和client设置了同一IP地址和端口
2. 加解密结果部分正确部分错误
 - 发现是5盒数据部分有误
3. 当双方通信时，输入函数cin会造成阻塞，未获取输入时不能打印接收到的消息
 - 使用 `cin.getline`
4. 使用多线程时风扇声音很大
 - 合理使用 `sleep()` 函数，减少没有必要的收发函数执行次数
5. 主线程执行完之后直接结束程序
 - 使用函数 `waitForSingleObject(hThread1, 200)`; 等待子进程执行完毕 (return) 后再继续执行主函数
6. 发现工程文件大小明显大于其它程序
 - 经查，为了使用类型 `BYTE` 调用了头文件 `windows.h`，引入了非常多的无用内容，调整后改为使用 `unsigned char` 可以达到相同的效果
7. 生成密钥的过程中，左移部分移位方向与期望不同

- bitset自身规定的大小端序与我自己定义的不同，所以移位方向需要相反

```
1 | bitset<56> left_mov = key_round[i] >> MOV[i]; //注意大小端序问题，实际是左移
```

8. 返回的字符串内容返回时正确而再次使用时发生变化

- 函数栈帧回收，所以不应该使用指针类型作为返回值，而是作为参数传入函数进行赋值

9. 报错 xxx重复定义

- 注意不要重复引用头文件
- 在头文件首部添加 `#pragma once`

10. 只能传输8个字节的消息

- des加密解密需要进行分组处理，每组的大小是64位，不足的分组补0

11. 使用rand每次生成的密钥都相同

- 添加种子 `srand((int)time(0));`

五、实验结论

- 细致地学习了des加解密的各个细节
- 学习了socket通信编程
- 学习了多线程编程