



南開大學
Nankai University

南 開 大 學

網 安 學 院

网络安全技术实验报告

第一次作业

沙璇 1911562

年级：2019 级

专业：信息安全

提交日期：2022/3/31

2022 年 3 月 31 日

摘要

基于 DES 加密的 TCP 通讯协议

关键字: DES , TCP

目录

| | |
|------------------|----|
| 一、 实验目的 | 1 |
| 二、 实验要求 | 1 |
| 三、 实验内容 | 1 |
| (一) DES 加密 | 1 |
| 1. 加密流程 | 1 |
| 2. 具体代码 | 4 |
| (二) DES 解密 | 7 |
| 1. 解密流程 | 7 |
| 2. 具体代码 | 7 |
| (三) TCP sever 端 | 10 |
| (四) TCP client 端 | 12 |
| 四、 实验结果 | 13 |
| 五、 总结 | 14 |

一、 实验目的

- a. 理解 DES 加解密原理。
- b. 理解 TCP 协议的工作原理。
- c. 掌握 linux 下基于 socket 的编程方法。

二、 实验要求

- a. 利用 socket 编写一个 TCP 聊天程序。
- b. 通信内容经过 DES 加密与解密

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

- a. 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。
- b. 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。
- c. 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、 实验内容

本次实验使用 C++ 语言编写，在 win11 中的 vs2010 运行。

实验利用 socket 接口实现了基于 TCP 协议通信的服务器 server 和客户端 client，双方可以进行聊天，在聊天的同时发送的信息经过 DES 算法加密，并在任意一方接收到加密后信息后再进行解密。

本次实验涉及到的源文件为：

1.server.cpp：可使用 socket 通信的服务器代码，其中含可加解密信息的 DES 算法实现。其中，start 函数生成子密钥，encode 函数用来加密，decode 函数解密的同时用来输出明文信息。

2.client.cpp：可使用 socket 通信的客户端代码，其中含可加解密信息的 DES 算法实现。逻辑和 server 类似。

(一) DES 加密

1. 加密流程

1. 所需参数

key：8 个字节共 64 位的工作密钥

data：8 个字节共 64 位的需要被加密或被解密的数据

mode：DES 工作方式，加密或者解密

2. 初始置换

DES 算法使用 64 位的密钥 key 将 64 位的明文输入块变为 64 位的密文输出块，并把输出块分为 L0、R0 两部分，每部分均为 32 位。初始置换规则如下图1所示

```

58,50,42,34,26,18,10,2,
60,52,44,36,28,20,12,4,
62,54,46,38,30,22,14,6,
64,56,48,40,32,24,16,8,
57,49,41,33,25,17, 9,1,
59,51,43,35,27,19,11,3,
61,53,45,37,29,21,13,5,
63,55,47,39,31,23,15,7,

```

图 1: caption

3. 加密处理—迭代过程

经过初始置换后，进行 16 轮完全相同的运算，在运算过程中数据与密钥结合。

函数 f 的输出经过一个异或运算，和左半部分结合形成新的右半部分，原来的右半部分成为新的左半部分。每轮迭代的过程可以表示如下：

$$L_n = R(n-1);$$

$$R_n = L(n-1) \oplus f(R_{n-1}, K_{n-1})$$

3.1 函数 f

函数 f 由四步运算构成：密钥置换 (K_n 的生成, $n=0 \sim 16$)；扩展置换；S-盒代替；P-盒置换。

3.1.1 密钥置换—子密钥生成

DES 算法由 64 位密钥产生 16 轮的 48 位子密钥。在每一轮的迭代过程中，使用不同的子密钥。

a、把密钥的奇偶校验位忽略不参与计算，即每个字节的第 8 位，将 64 位密钥降至 56 位，然后根据选择置换 PC-1 将这 56 位分成两块 C_0 (28 位) 和 D_0 (28 位)；

b、将 C_0 和 D_0 进行循环左移变化 (注：每轮循环左移的位数由轮数决定)，变换后生成 C_1 和 D_1 ，然后 C_1 和 D_1 合并，并通过选择置换 PC-2 生成子密钥 K_1 (48 位)；

c、 C_1 和 D_1 在次经过循环左移变换，生成 C_2 和 D_2 ，然后 C_2 和 D_2 合并，通过选择置换 PC-2 生成密钥 K_2 (48 位)；

d、以此类推，得到 K_{16} (48 位)。但是最后一轮的左右两部分不交换，而是直接合并在一起 $R_{16}L_{16}$ ，作为逆置换的输入块。其中循环左移的位数一共是循环左移 16 次，其中第一次、第二次、第九次、第十六次是循环左移一位，其他都是左移两位。

3.1.2 密钥置换选择 1—PC-1(子密钥的生成)

操作对象是 64 位密钥

64 位密钥降至 56 位密钥不是说将每个字节的第八位删除，而是通过缩小选择换位表 1 (置换选择表 1) 的变换变成 56 位。如图2所示

```

57,49,41,33,25,17,9,1,
58,50,42,34,26,18,10,2,
59,51,43,35,27,19,11,3,
60,52,44,36,63,55,47,39,
31,23,15,7,62,54,46,38,
30,22,14,6,61,53,45,37,
29,21,13,5,28,20,12,4

```

图 2: caption

56 位密钥分成 C_0 和 D_0 ：

$$C_0(28 \text{ 位}) = K_{57}K_{49}K_{41} \dots K_{44}K_{36}$$

$$D_0(28 \text{ 位}) = K_{63}K_{55}K_{47} \dots K_{12}K_4$$

C1 和 D1 再次经过循环左移变换, 生成 C2 和 D2, C2 和 D2 合并, 通过 PC-2 生成子密钥 K2。以此类推, 得到子密钥 K1 K16。需要注意其中循环左移的位数。

3.1.2 扩展置换 E(E 位选择表)

通过扩展置换 E, 数据的右半部分 R_n 从 32 位扩展到 48 位。扩展置换改变了位的次序, 重复了某些位。

扩展置换的目的: a、产生与密钥相同长度的数据以进行异或运算, R_0 是 32 位, 子密钥是 48 位, 所以 R_0 要先进行扩展置换之后与子密钥进行异或运算; b、提供更长的结果, 使得在替代运算时能够进行压缩。

3.1.3 S-盒代替 (功能表 S 盒)

R_n 扩展置换之后与子密钥 K_n 异或以后的结果作为输入块进行 S 盒代替运算

功能是把 48 位数据变为 32 位数据

代替运算由 8 个不同的代替盒 (S 盒) 完成。每个 S-盒有 6 位输入, 4 位输出。

所以 48 位的输入块被分成 8 个 6 位的分组, 每一个分组对应一个 S-盒代替操作。经过 S-盒代替, 形成 8 个 4 位分组结果。

3.1.4 P-盒置换

S-盒代替运算, 每一盒得到 4 位, 8 盒共得到 32 位输出。这 32 位输出作为 P 盒置换的输入块。

P 盒置换将每一位输入位映射到输出位。任何一位都不能被映射两次, 也不能被略去。

经过 P-盒置换的结果与最初 64 位分组的左半部分异或, 然后左右两部分交换, 开始下一轮迭代。

4. 逆置换

将初始置换进行 16 次的迭代, 即进行 16 层的加密变换, 这个运算过程我们暂时称为函数 f 。得到 L16 和 R16, 将此作为输入块, 进行逆置换得到最终的密文输出块。逆置换是初始置换的逆运算。从初始置换规则中可以看到, 原始数据的第 1 位置换到了第 40 位, 第 2 位置换到了第 8 位。则逆置换就是将第 40 位置换到第 1 位, 第 8 位置换到第 2 位。

流程图如图3所示

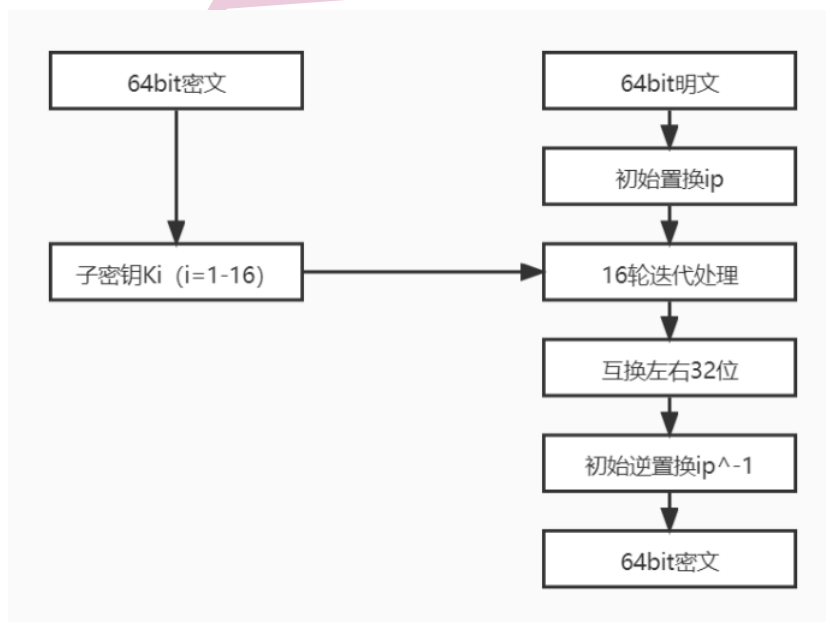


图 3: Caption

2. 具体代码

将明文进行分组，以 8 个字母为一组，最后一组不足 8 个则补 0。并将每一组 8 个字母根据 ASCII 码转为 $8 \times 8 = 64$ 位的二进制数字

des

```
1 void encode(char* a){
2
3     strcpy(cleartext,a);
4     //CBC模式下的加密
5     i = 0; //将明文每8个字符作为一个分组，共有n个分组
6     n = 0;
7     while (cleartext[i] != '\0')
8     {
9         n++;
10        i++;
11    }
12    k = n % 8;
13    n = (n - 1) / 8 + 1;
14
15    for (l = 0; l < n; l++)
16    {
17        if (l == (n - 1) && k != 0)
18            { //将每个分组的8个字符放到数组group中
19                for (i = 0; i < k; i++)
20                    group[i] = cleartext[i + (8 *
21                        l)];
22                for (i = k; i < 8; i++)
23                    group[i] = ' '; //后面的用空格
24                    补充位数
25            }
26        else
27            for (i = 0; i < 8; i++)
28                group[i] = cleartext[i + (8 *
29                    l)];
30        //将得到的明文转化成二进制数储存到数组text中
31        for (i = 0; i < 8; i++)
32        {
33            int a[8] = { 0,0,0,0,0,0,0,0 };
34            m = group[i];
35            for (j = 0; m != 0; j++)
36            {
37                a[j] = m % 2;
38                m = m / 2;
39            }
40            for (j = 0; j < 8; j++)
41                text[(i * 8) + j] = a[7 - j];
42        }
43    }
```

对将每一组进行 DES 算法加密变换，具体步骤如下：

des

```

1 //前一分组的密文异或当前分组
2 for (i = 0; i < 64; i++)
3     text[i] = iter_result[1][i] ^ text[i];
4
5 for (i = 0; i < 64; i++) //初始换位
6     IP_replace[i] = text[IP[i] - 1];
7
8 for (i = 0; i < 32; i++) //左右两部分各32位
9 {
10     L0[i] = IP_replace[i];
11     R0[i] = IP_replace[i + 32];
12 }
13
14 //十六次迭代
15
16 for (t = 0; t < 16; t++)
17 {
18     for (i = 0; i < 48; i++)
19         RE0[i] = R0[E[i] - 1]; //右半部分32位通过E表扩展成48位
20
21     for (i = 0; i < 48; i++) //RE0与K异或
22         RK[i] = RE0[i] ^ K[t][i];
23
24     for (i = 0; i < 8; i++) //将R和K异或运算的结果通过S位移表
25     {
26         r[i] = RK[(i * 6) + 0] * 2 + RK[(i * 6) + 5];
27         c[i] = RK[(i * 6) + 1] * 8 + RK[(i * 6) + 2] * 4 + RK[(i * 6)
28             + 3] * 2 + RK[(i * 6) + 4];
29     }
30     RKS[0] = S1[r[0]][c[0]];
31     RKS[1] = S2[r[1]][c[1]];
32     RKS[2] = S3[r[2]][c[2]];
33     RKS[3] = S4[r[3]][c[3]];
34     RKS[4] = S5[r[4]][c[4]];
35     RKS[5] = S6[r[5]][c[5]];
36     RKS[6] = S7[r[6]][c[6]];
37     RKS[7] = S8[r[7]][c[7]];
38
39     for (i = 0; i < 8; i++) //结果转成32位二进制存在SP中
40     {
41         int b[4] = { 0, 0, 0, 0 };
42         m = RKS[i];
43         for (j = 3; m != 0; j--)
44             b[j] = m % 2;

```

```

45         m = m / 2;
46     }
47     for (j = 0; j < 4; j++)
48         SP[j + (i * 4)] = b[j];
49 }
50
51 for (i = 0; i < 32; i++) //二进制结果再经过一个P盒换位
52     RKSP[i] = SP[P[i] - 1];
53
54 for (i = 0; i < 32; i++) //与上一次迭代得到的左边异或运算，得到本次迭
    代的右边
55     Ri[i] = L0[i] ^ RKSP[i];
56
57 for (i = 0; i < 32; i++)
58 {
59     L0[i] = R0[i];
60     R0[i] = Ri[i];
61 }
62 }
63
64 //一个左右32位交换
65 for (i = 0; i < 32; i++)
66     Li[i] = R0[i];
67 for (i = 0; i < 32; i++)
68     R0[i] = L0[i];
69 for (i = 0; i < 32; i++)
70     L0[i] = Li[i];
71
72 for (i = 0; i < 32; i++) //左右两部分合起来存到iter_replace中
73     iter_replace[i] = L0[i];
74     for (i = 32; i < 64; i++)
75         iter_replace[i] = R0[i - 32];
76
77 for (i = 0; i < 64; i++) //进行初始换位的逆过程
78     iter_result[l + 1][IP[i] - 1] = iter_replace[i];
79
80 for (i = 0; i < 64; i++)
81     result[l][i] = iter_result[l + 1][i];
82 }
83 for (j = 0; j < n; j++) //把二进制result转成十进制存到H中
84 for (i = 0; i < 16; i++)
85     H[i + (j * 16)] = result[j][0 + (i * 4)] * 8 + result[j][1 + (i * 4)]
        * 4 + result[j][2 + (i * 4)] * 2 + result[j][3 + (i * 4)];
86
87 for (i = 0; i < n * 16; i++)
88 {
89     if (H[i] < 10)
90         secretText[i] = H[i] + 48;

```



```

91     else if (H[i] == 10)
92         secretText[i] = 'A';
93     else if (H[i] == 11)
94         secretText[i] = 'B';
95     else if (H[i] == 12)
96         secretText[i] = 'C';
97     else if (H[i] == 13)
98         secretText[i] = 'D';
99     else if (H[i] == 14)
100         secretText[i] = 'E';
101     else if (H[i] == 15)
102         secretText[i] = 'F';
103     }
104     for (i = 1 * 16; i < 208; i++)
105         secretText[i] = '\0'; // 注意数组越界
106 }

```

(二) DES 解密

1. 解密流程

DES 解密流程与加密流程基本一致，仍为先进行初始置换，最后进行逆置换，中间 16 轮利用 16 个密钥的迭代加密，唯一不同的地方就是所生成的 16 个密钥的使用顺序，加密运算与解密运算的密钥使用顺序正好相反。

2. 具体代码

des

```

1 //解密程序
2 void decode(char* b){
3     strcpy(secretText,b);
4     for (i = 0; i < 208; i++)
5         H[i] = 0;
6
7     for (i = 0; secretText[i] != '\0'; i++) //十六进制密文转化成十进制存
        放在H中
8     {
9         if (secretText[i] >= '0' && secretText[i] <= '9')
10             H[i] = secretText[i] - '0';
11         else if (secretText[i] >= 'A' && secretText[i] <= 'F')
12             H[i] = secretText[i] - 'A' + 10;
13         else if (secretText[i] >= 'a' && secretText[i] <= 'f')
14             H[i] = secretText[i] - 'a' + 10;
15
16     }
17     n = i; //密文中共有n个字符
18     for (i = 0; i < n; i++) //将十进制密文转化成二进制存放在数组C中
19     {

```

```

20         int he[4] = { 0,0,0,0 };
21         for (j = 3; H[i] != 0; j--)
22         {
23             he[j] = H[i] % 2;
24             H[i] = H[i] / 2;
25         }
26         for (j = 0; j < 4; j++)
27             C[j + (i * 4)] = he[j];
28     }
29
30     k = n / 16;
31     for (l = 0; l < k; l++)
32     {
33         for (i = 0; i < 64; i++) //将每个分组对应的64位二进制密文放到
34             iter_result[l + 1][i] = C[i + (l * 64)];
35
36         //对每个text进行DES解密
37
38         for (i = 0; i < 64; i++) //进行初始换位
39             IP_replace[i] = iter_result[l + 1][IP[i] - 1];
40
41         for (i = 0; i < 32; i++) //分成左右两部分，各32位
42         {
43             L0[i] = IP_replace[i];
44             R0[i] = IP_replace[i + 32];
45         }
46
47         //十六次迭代
48         for (t = 0; t < 16; t++)
49         {
50             for (i = 0; i < 48; i++) //将右半部分通过扩展换位表E
51                 RE0[i] = R0[E[i] - 1];
52             for (i = 0; i < 48; i++) //RE0异或K
53                 RK[i] = RE0[i] ^ K[15 - t][i];
54
55             for (i = 0; i < 8; i++) //用S表变换R异或K的结果
56             {
57                 r[i] = RK[(i * 6) + 0] * 2 + RK[(i * 6) + 5];
58                 c[i] = RK[(i * 6) + 1] * 8 + RK[(i * 6) + 2]
59                     * 4 + RK[(i * 6) + 3] * 2 + RK[(i * 6) +
60                     4];
61             }
62
63             RKS[0] = S1[r[0]][c[0]];
64             RKS[1] = S2[r[1]][c[1]];
65             RKS[2] = S3[r[2]][c[2]];

```

```
64     RKS[3] = S4[r[3]][c[3]];
65     RKS[4] = S5[r[4]][c[4]];
66     RKS[5] = S6[r[5]][c[5]];
67     RKS[6] = S7[r[6]][c[6]];
68     RKS[7] = S8[r[7]][c[7]];
69
70     for (i = 0; i < 8; i++) //把结果转成32位二进制储存在
                               数组SP中
71     {
72         int b[4] = { 0,0,0,0 };
73         m = RKS[i];
74         for (j = 3; m != 0; j--)
75         {
76             b[j] = m % 2;
77             m = m / 2;
78         }
79         for (j = 0; j < 4; j++)
80             SP[j + (i * 4)] = b[j];
81     }
82
83     for (i = 0; i < 32; i++) //将二进制结果再经过一个P盒
                               换位
84         RKSP[i] = SP[P[i] - 1];
85
86     for (i = 0; i < 32; i++) //与前一次的左部异或运算，得
                               到本次迭代的右部
87         Ri[i] = L0[i] ^ RKSP[i];
88
89     for (i = 0; i < 32; i++)
90     {
91         L0[i] = R0[i];
92         R0[i] = Ri[i];
93     }
94 }
95
96 //一个左右32位交换
97
98 for (i = 0; i < 32; i++)
99     Li[i] = R0[i];
100 for (i = 0; i < 32; i++)
101     R0[i] = L0[i];
102 for (i = 0; i < 32; i++)
103     L0[i] = Li[i];
104
105 //初始换位的逆过程
106
107 for (i = 0; i < 32; i++) //把左右两部分合起来存到iter_replace
                               中
```

```

108         iter_replace[i] = L0[i];
109     for (i = 32; i < 64; i++)
110         iter_replace[i] = R0[i - 32];
111
112     for (i = 0; i < 64; i++) //进行初始换位的逆过程
113         text[IP[i] - 1] = iter_replace[i];
114
115     for (i = 0; i < 64; i++) //前一分组的密文异或当前分组所得明文
116         //的二进制放到result中
117         result[1][i] = iter_result[1][i] ^ text[i];
118
119 }
120
121 for (i = 0; i < (n / 16); i++) //将二进制转成十进制
122     for (j = 0; j < 8; j++)
123         M[i][j] = result[i][(j * 8) + 0] * 128 + result[i][(j
124             * 8) + 1] * 64 + result[i][(j * 8) + 2] * 32 +
125             result[i][(j * 8) + 3] * 16 + result[i][(j * 8) +
126             4] * 8 + result[i][(j * 8) + 5] * 4 + result[i]
127             [(j * 8) + 6] * 2 + result[i][(j * 8) + 7];
128
129     printf("Client: ");
130     for (i = 0; i < (n / 16); i++)
131         for (j = 0; j < 8; j++)
132             printf("%c", M[i][j]);
133     printf("\n");
134 }

```

(三) TCP sever 端

具体代码如下

des

```

1 void main()
2 {
3     start();
4     //前面的这一部分是版本协商器
5     WORD wVersionRequested;
6     WSADATA wsaData;
7
8     int err;
9     wVersionRequested = MAKEWORD(1, 1);
10
11     err = WSAStartup(wVersionRequested, &wsaData);
12     if (err != 0) {
13
14
15         return;

```

```
16     }
17     if (LOBYTE(wsaData.wVersion) != 1 || HIBYTE(wsaData.wVersion) != 1)
18
19     {
20         WSACleanup();
21         return;
22     }
23
24     SOCKET server, cAddr; // 定义一个套接字
25     server = socket(AF_INET, SOCK_STREAM, 0); // 参数说明: IP家族协议、流式
        套接字、默认参数
26
27     SOCKADDR_IN sAddr; // 定义服务器的地址
28     sAddr.sin_family = AF_INET; // 协议
29     sAddr.sin_port = htons(1352); // 端口
30     sAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1"); // 服务端的IP地址
31
32     bind(server, (SOCKADDR*)&sAddr, sizeof(SOCKADDR)); // 将服务器的地址信
        息和套接字进行绑定
33
34     listen(server, 5); // 进行监听, 队列长度为5个字节
35
36     SOCKET sockConn; // 定义连接套接字
37     int len = sizeof(SOCKADDR);
38
39     cout << "waiting for client connection" << endl;
40     sockConn = accept(server, (SOCKADDR*)&cAddr, &len); // 接受请求
41     if (sockConn == INVALID_SOCKET)
42     {
43         cout << "ERROR: Connecting to client has failed." << endl;
44         return;
45     }
46     else
47     {
48         cout << "Client has successfully connected" << endl;
49     }
50
51     char sendbuf[256];
52     char recvbuf[256];
53     while (1)
54     {
55         cout << "Server: ";
56         cin >> sendbuf;
57         if (strcmp(sendbuf, "quit") == 0)
58         {
59             break;
60         }
61         encode(sendbuf);
```

```
62         send(sockConn, secretText, strlen(secretText)+1, 0); // 发送数据
63         recv(sockConn, recvbuf, 256, 0); // 接受数据
64         decode(recvbuf);
65     }
66     closesocket(server);
67     WSACleanup();
68 }
```

(四) TCP client 端

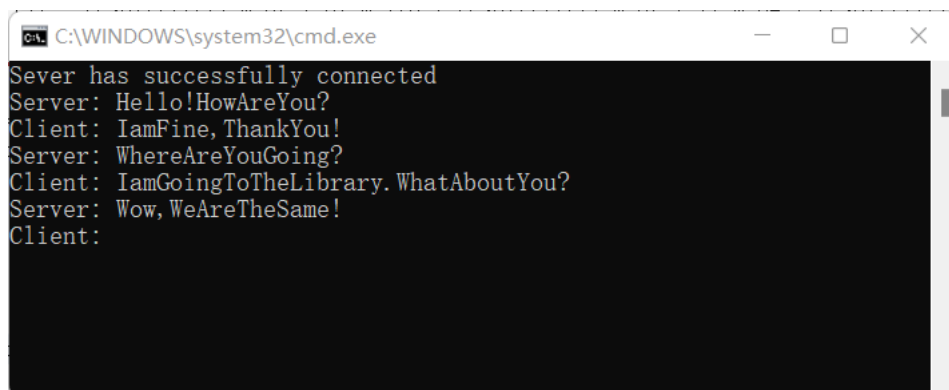
具体代码如下

```
des
1 void main()
2 {
3     start();
4     WORD wVersionRequested;
5     WSADATA wsaData;
6
7     int err;
8     wVersionRequested = MAKEWORD(1, 1);
9
10    err = WSAStartup(wVersionRequested, &wsaData);
11    if (err != 0) {
12
13
14        return;
15    }
16    if (LOBYTE(wsaData.wVersion) != 1 || HIBYTE(wsaData.wVersion) != 1)
17
18    {
19
20
21        WSACleanup();
22        return;
23
24    }
25    //
26    SOCKET client;
27    client = socket(AF_INET, SOCK_STREAM, 0);
28    SOCKADDR_IN sAddr;
29    sAddr.sin_family = AF_INET;
30    sAddr.sin_port = htons(1352);
31    sAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");
32
33    int res = connect(client, (SOCKADDR*)&sAddr, sizeof(SOCKADDR));
34    if (res != 0)
35    {
```

```
36         cout << "ERROR:Connecting to sever has failed." << endl;
37         return;
38     }
39     else
40     {
41         cout << "Sever has successfully connected" << endl;
42     }
43
44     char sendbuf[256];
45     char recvbuf[256];
46     while (1)
47     {
48         recv(client , recvbuf , 256,0);
49         decode(recvbuf);
50
51
52         cout << "Client: ";
53         cin >> sendbuf;
54         if (strcmp(sendbuf, "quit") == 0)
55         {
56             break;
57         }
58         encode(sendbuf);
59         send(client , secretText , strlen(secretText) + 1,0);
60     }
61     closesocket(client);
62
63     WSACleanup();
64 }
```

四、 实验结果

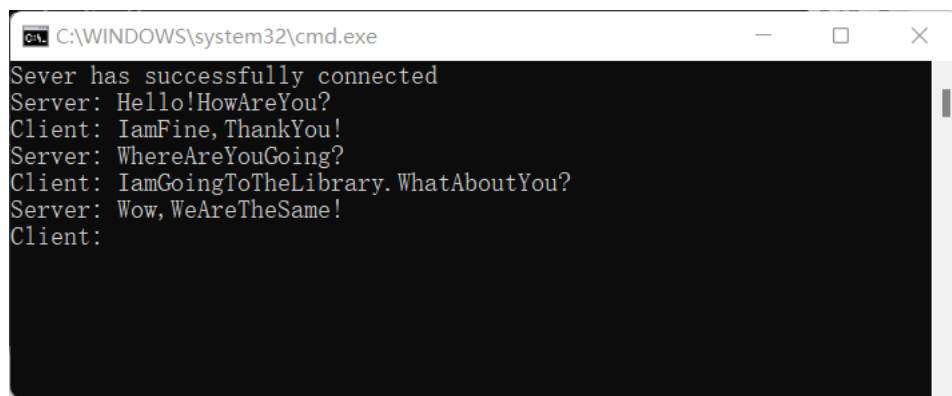
sever 端结果如图4所示



```
C:\WINDOWS\system32\cmd.exe
Sever has successfully connected
Server: Hello!HowAreYou?
Client: IamFine,ThankYou!
Server: WhereAreYouGoing?
Client: IamGoingToTheLibrary. WhatAboutYou?
Server: Wow, WeAreTheSame!
Client:
```

图 4: sever

client 端结果如图5所示



```
C:\WINDOWS\system32\cmd.exe
Sever has successfully connected
Server: Hello!HowAreYou?
Client: IamFine, ThankYou!
Server: WhereAreYouGoing?
Client: IamGoingToTheLibrary. WhatAboutYou?
Server: Wow, WeAreTheSame!
Client:
```

图 5: client

五、 总结

通过本次实验，我对 DES 算法加解密的基本流程有了进一步的了解，并巩固了通过 socket 实现 TCP 协议聊天程序的方法，了解了 TCP 协议的工作原理。