

网络安全技术

实 验 报 告

学 院 网安学院
年 级 2021 级
班 级 信安一班
学 号 2113662
姓 名 张丛
手机号 18086215842

2024 年 3 月 12 日

目录

一、实验目标	1
二、实验内容	1
三、实验步骤	1
四、实验遇到的问题及其解决方法	18
五、实验结论	19

一、实验目的

- 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。

二、实验内容

- 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。

三、实验步骤及实验结果

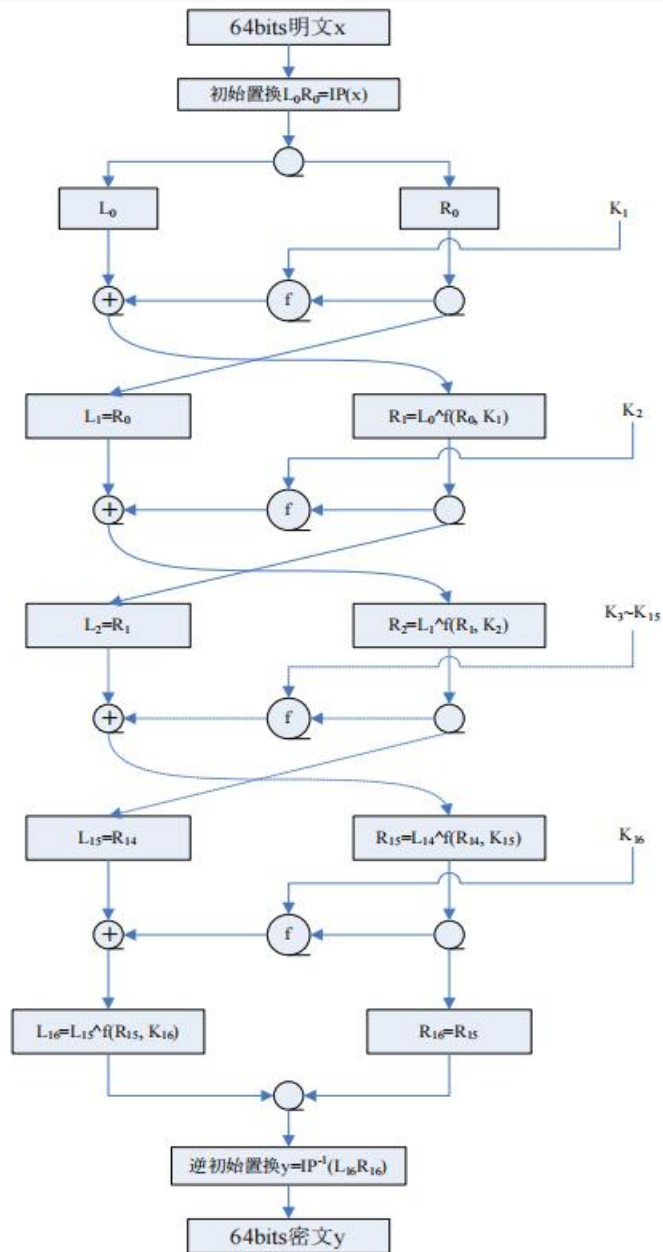
(1) DES

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。

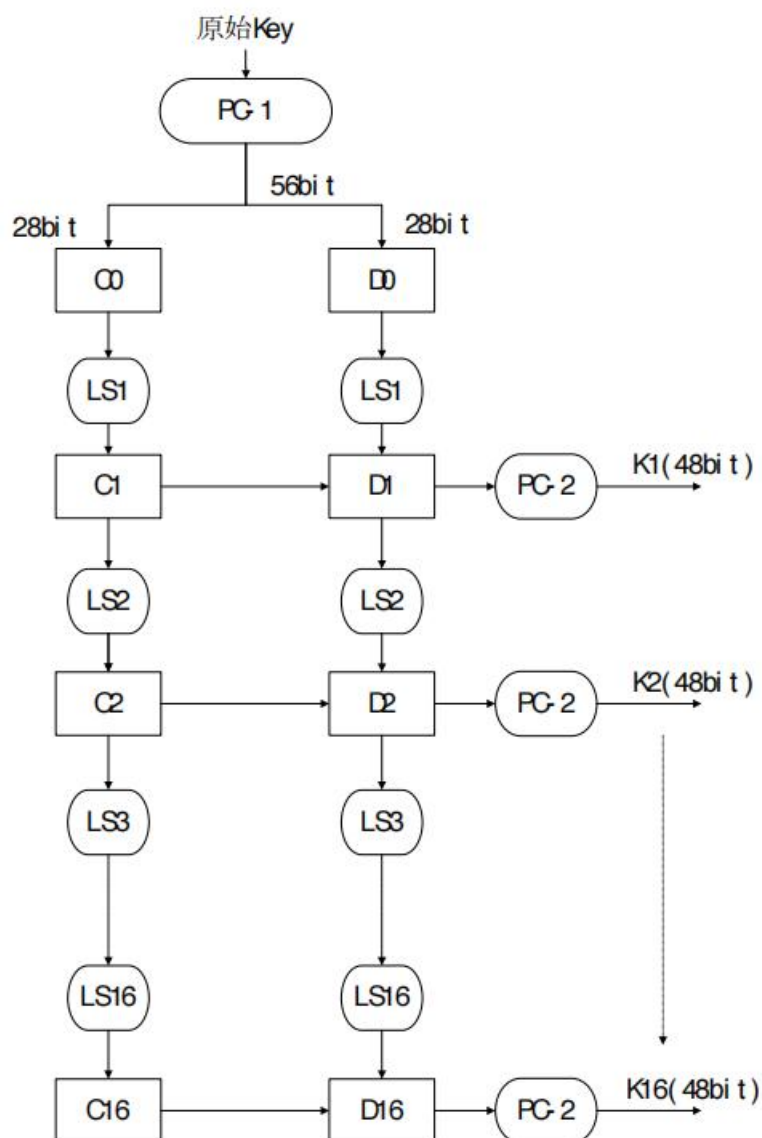
DES 明文分组长度为 64bit (不足 64 bit 的部分用 0 补齐)，密文分组长度也是 64bit。加密过程要经过 16 圈迭代。

算法包括：初始置换 IP、逆初始置换 IP⁻¹、16 轮迭代以及子密钥生成算法。

DES 流程如下：



生成 16 轮子密钥流程:



(2) TCP

TCP, 即传输控制协议(Transmission Control Protocol), 是一种端对端的协议。

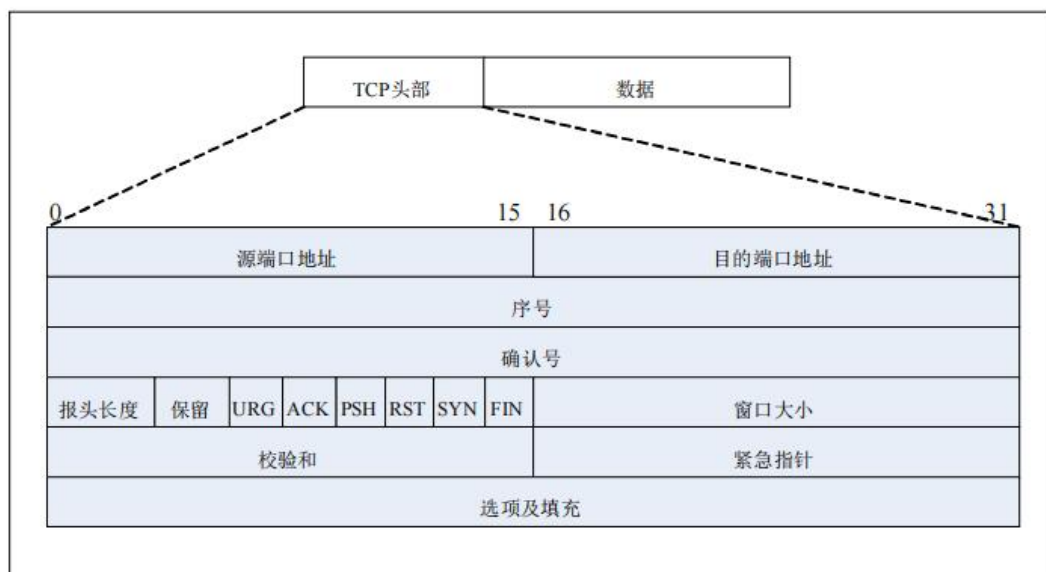
要用 TCP 协议与另一台计算机通信, 两台机之间必须连接在一起, 每一端都为通话做好准备。

TCP 协议主要用来处理数据流, 可以正确处理乱序的数据包。TCP 协议甚至还允许存在丢失的或者损坏的数据包, 最终它可以再次得到这些数据包。

TCP 协议每发送一个数据包将会收到一个确认信息。这种发送/应答模式可以使对方知道你是否收到了数据，从而保证了可靠性。

TCP 协议之所以是全双工的就是因为“捎带确认”信息，它允许双方同时发送数据，这是通过在当前的数据包中携带以前收到的数据的确认信息的方式实现的。

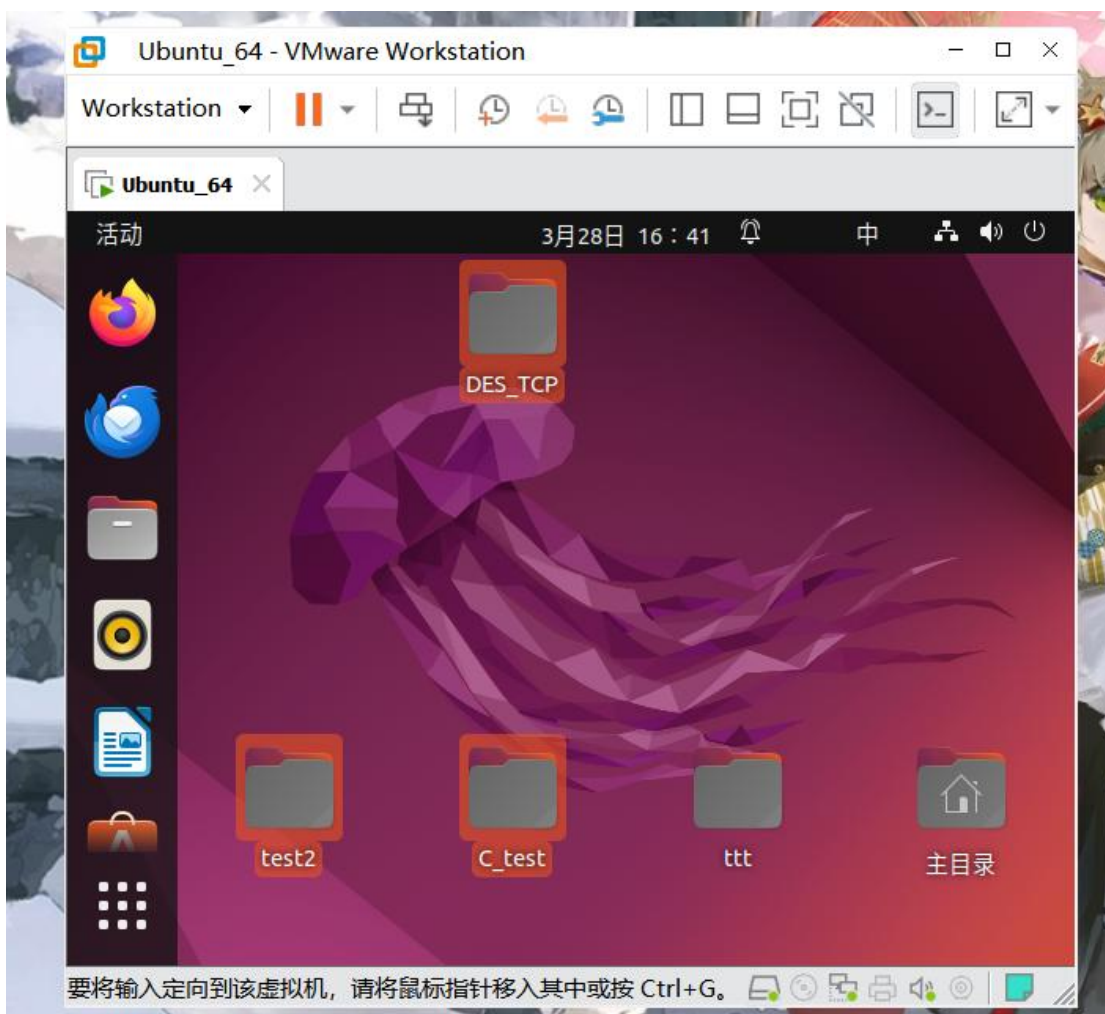
TCP 数据包格式:



(3) linux 虚拟机安装

虚拟机软件：VMware

Linux 发行版: Ubuntu_2022



(4) socket 编程

套接字（Socket）是应用层与 TCP/IP 协议族通信的中间软件抽象层，它是一组接口，它把复杂的 TCP/IP 协议族隐藏在 Socket 接口的背后，通过 Socket 函数调用去传输数据以符合指定的协议。

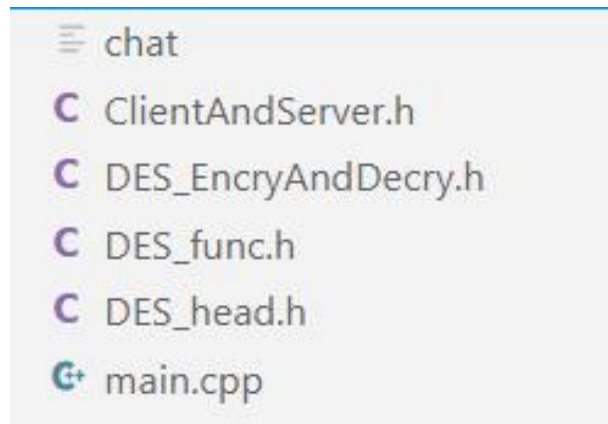
套接字存在于通信区域中，一个套接字只能与同一区域内的套接字交换数据。

本次实验使用的是流式套接字。

Linux 下的常用套接字函数就不在此赘述。

(5) 代码实现

代码结构：



共包含四个头文件和一个 `cpp` 文件。

其中 `DES_head.h` 主要是定义了 `CDesOperate` 类和一些关键的数据结构等等。

`DES_func.h` 包含 DES 加解密流程的关键函数的实现，如子密钥生成、F 函数等等。

`DES_EncryAndDecry.h` 是 DES 对通信内容进行加解密的实现。

`ClientAndServer.h` 是实现 TCP 聊天功能的模块。

主函数如下：


```

int main() {
    while(1) {
        cout << "Client or server? (s/c/q)" << endl;
        string client_or_server;
        cin >> client_or_server;
        if (client_or_server == "s") {
            if(server() != 0){
                break;
            }
        }
        else if (client_or_server == "c") {
            if(client() != 0){
                break;
            }
        }
        else if(client_or_server == "q"){
            cout << "Successfully quit the program!" << endl;
            break;
        }
        else {
            cout << "Error input! Please type s for server, c for client." << endl;
        }
    }
    return 0;
}

```

实现了一个简单的客户端-服务器选择器,通过命令行提示用户选择是作为客户端还是服务器,或者退出程序。

主函数中 client()和 server()这两个函数的实现:

```

1.  int client()
2.  { printf("Please enter the server address: ");
3.      char strIpAddr[16];
4.      cin >> strIpAddr; // 输入 IP 地址
5.
6.      int nConnectSocket, nLength; // 套接字描述符和长度
7.      struct sockaddr_in sDestAddr; // 目标地址
8.
9.      // 创建套接字
10.     if ((nConnectSocket = socket(AF_INET, SOCK_STREAM, 0)) < 0)
11.     {
12.         perror("Socket"); // 如果套接字创建失败,则打印错误消息
13.         exit(errno);
14.     }
15.
16.     // 初始化目标地址结构
17.     sDestAddr.sin_family = AF_INET; // 设置地址族为 IPv4

```

```

18.     sDestAddr.sin_port = htons(tcp_port); // 设置端口号
19.     sDestAddr.sin_addr.s_addr = inet_addr(strIpAddr); // 设置 IP 地址
20.
21.     // 连接到服务器
22.     if (connect(nConnectSocket, (struct sockaddr *) &sDestAddr, sizeof(sDestAddr)) !=
        0)
23.     {
24.         perror("Connect "); // 如果连接失败, 则打印错误消息
25.         exit(errno); // 退出程序并返回错误代码
26.     }
27.     else
28.     {
29.         printf("Connect Success! \nBegin to chat...\n"); // 打印连接成功的消息
30.         SecretChat(nConnectSocket, strIpAddr, "benbenmi"); // 开始密聊
31.     }
32.
33.     close(nConnectSocket); // 关闭套接字
34.     return 0;
35. }

```

```

1.  int server()
2.  {
3.      int nListenSocket, nAcceptSocket;
4.      struct sockaddr_in sLocalAddr, sRemoteAddr;
5.      socklen_t nLength = sizeof(sRemoteAddr); // 声明一个变量用于存储远程地址结构体的长度
6.
7.      // 创建监听套接字
8.      if ((nListenSocket = socket(AF_INET, SOCK_STREAM, 0)) == -1)
9.      {
10.         perror("socket"); // 如果创建套接字失败, 则打印错误消息
11.         exit(1);
12.     }
13.
14.     // 初始化本地地址结构
15.     memset(&sLocalAddr, 0, sizeof(sLocalAddr)); // 清空结构体
16.     sLocalAddr.sin_family = AF_INET; // 设置地址族为 IPv4
17.     sLocalAddr.sin_port = htons(tcp_port); // 设置端口号
18.     //sLocalAddr.sin_addr.s_addr = htonl(INADDR_ANY); // 设置 IP 地址为本地任意可用地址
19.
20.     char server_ip[INET_ADDRSTRLEN];
21.     printf("Enter IP address: ");
22.     scanf("%s", server_ip);
23.

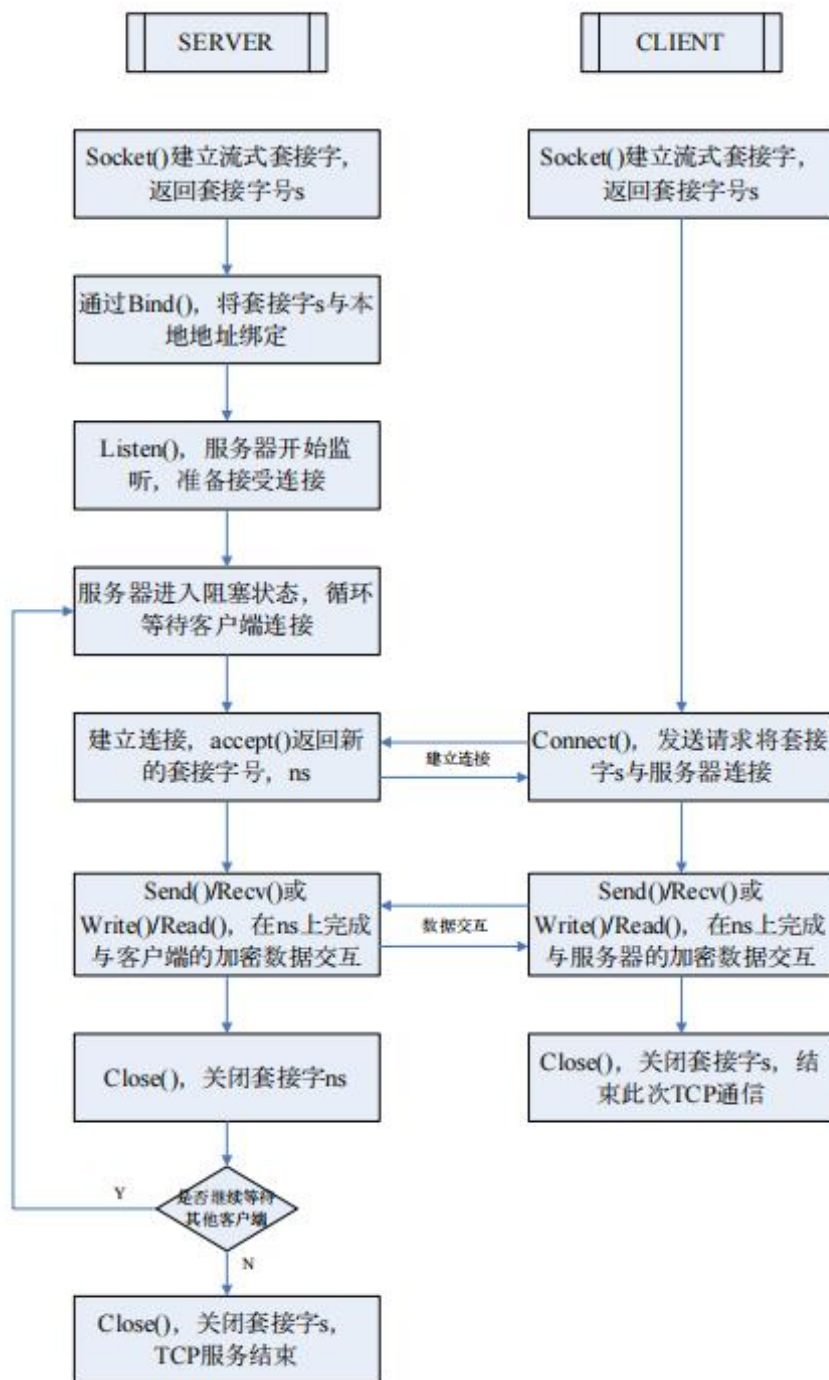
```

```

24.     if(inet_pton(AF_INET, server_ip, &sLocalAddr.sin_addr) <= 0) {
25.         perror("inet_pton");
26.         exit(1);
27.     }
28.
29.     // 将套接字与本地地址绑定
30.     if (bind(nListenSocket, (struct sockaddr *) &sLocalAddr, sizeof(struct sockaddr))
        == -1)
31.     {
32.         perror("bind"); // 如果绑定失败, 则打印错误消息
33.         exit(1); // 退出程序并返回错误代码
34.     }
35.     // 开始监听套接字
36.     if (listen(nListenSocket, 5) == -1)
37.     {
38.         perror("listen"); // 如果监听失败, 则打印错误消息
39.         exit(1); // 退出程序并返回错误代码
40.     }
41.     printf("Listening.....");
42.
43.     // 接受连接请求, 并返回新的套接字用于通信
44.     nAcceptSocket = accept(nListenSocket, (struct sockaddr *) &sRemoteAddr, &nLength);
45.     close(nListenSocket); // 关闭监听套接字
46.
47.     // 打印连接信息
48.     printf("server: got connection from %s, port %d, socket %d\n",
49.         inet_ntoa(sRemoteAddr.sin_addr), ntohs(sRemoteAddr.sin_port), nAcceptSocket)
        ;
50.     // 开始密聊
51.     SecretChat(nAcceptSocket, inet_ntoa(sRemoteAddr.sin_addr), "benbenmi");
52.
53.     close(nAcceptSocket); // 关闭通信套接字
54.     return 0;
55. }

```

总的来说, 这两个函数实现了下面的通信流程:



client 和 server 都有一个共同的 SecretChat 聊天函数, 在这里面收发消息并使用 DES 加密:

```

1. void SecretChat(int nSock, char *pRemoteName, const char *pKey)
2. {
3.     CDesOperate cDes; // 创建 DES 操作对象
4.     // 检查密钥长度是否为 8

```

```

5.         if (strlen(pKey) != 8)
6.         {
7.             printf("key length must be 8\n");
8.             return;
9.         }
10.        pid_t nPid;
11.        nPid = fork(); // 创建子进程
12.        if (nPid != 0) // 父进程
13.        {
14.            while (1)
15.            {
16.                char strSocketBuffer[max_msg_len]; // 套接字数据缓冲区
17.                bzero(&strSocketBuffer, max_msg_len); // 清空缓冲区
18.
19.                int nLength = 0; // 接收数据长度
20.                nLength = TotalRecv(nSock, strSocketBuffer, max_msg_len, 0); // 接收数据
21.                if (nLength != max_msg_len) // 接收到数据长度不等于缓冲区大小时, 跳出循环
22.                {
23.                    break;
24.                }
25.                else
26.                {
27.                    string strDecryBuffer; // 解密后的数据字符串
28.                    // 解密接收到的数据
29.                    cDes.decry_operation(string(strSocketBuffer), string(pKey), strDecryBu
30.                    ffer);
31.                    // 打印接收到的消息
32.                    printf("receive message from <%s>: %s\n", pRemoteName, strDecryBuffer.
33.                    c_str());
34.
35.                    // 检查是否收到退出命令
36.                    if (strncmp("quit", strDecryBuffer.c_str(), 4) == 0)
37.                    {
38.                        printf("Quit! \n");
39.                        break;
40.                    }
41.                }
42.            }
43.        }
44.        else // 子进程
45.        {
46.            while (1)
47.            {
48.                char strStdinBuffer[max_msg_len]; // 标准输入缓冲区

```

```

47.         bzero(&strStdinBuffer, max_msg_len); // 清空缓冲区
48.
49.         while(strStdinBuffer[0]==0)
50.         {
51.             // 从标准输入读取数据
52.             if (fgets(strStdinBuffer, max_msg_len, stdin) == NULL)
53.             {
54.                 continue;
55.             }
56.         }
57.         char strEncryBuffer[max_msg_len]; // 加密后的数据缓冲区
58.         int nLen = max_msg_len; // 加密后数据的长度
59.
60.         // 将 char 数组转换为 string
61.         string strStdin(strStdinBuffer);
62.         string key(pKey);
63.
64.         // 加密数据
65.         string encryptedData;
66.         cDes.encry_operation(strStdinBuffer, string(pKey), encryptedData);
67.
68.         // 将 string 转换为 char* 数组
69.         strncpy(strEncryBuffer, encryptedData.c_str(), max_msg_len);
70.         //printf("after DES: %s\n", strEncryBuffer);
71.
72.         // 发送加密后的数据到套接字
73.         if (send(nSock, strEncryBuffer, nLen, 0) != nLen)
74.         {
75.             perror("send");
76.         }
77.         else
78.         {
79.             // 检查是否发送退出命令
80.             if (strncmp("quit", strStdinBuffer, 4) == 0)
81.             {
82.                 printf("Quit! \n");
83.                 break;
84.             }
        }
    }
}
}
}
}

```

知识点：

Linux 是一种多用户、多进程的操作系统。每个进程都有一个唯一的进程标

标识符，操作系统通过对机器资源进行时间共享，并发的运行许多进程。

在 Linux 中，程序员可以使用 `fork()` 函数创建新进程，它可以与父进程完全并发的运行。

`fork()` 函数不接受任何参数，并返回一个 `int` 值。当它被调用时，创建出的子进程除了拥有自己的进程标识符以外，其余特征，例如数据段，堆栈段，代码段等和其父进程完全相同。

`fork()` 函数向子进程返回 0，向父进程返回子进程的进程标识符，该标识符是一个非零的 `int` 值。

SecretChat 聊天函数创建了 `CDesOperate` 对象，然后调用了加解密函数 `encry_operation` 和 `decry_operation`，使得通信内容安全传输。

加解密函数如下：

```

// DES 加密操作
int CDesOperate::encry_operation(string raw_text, string key, string& encry_res) {
    // 检查密钥长度是否为 8
    if (key.size() != 8) {
        cout << "error: key size isn't 8!" << endl;
        return -1;
    }

    // 将密钥转换为二进制形式
    vector<bool> bin_key = key_str2bool(key);

    // 生成子密钥
    vector<vector<bool>> > sub_key = gen_subkey(bin_key);

    // 将原始文本转换为二进制形式
    vector<vector<bool>> > bin_raw_text = encry_str2bool(raw_text);

    // 存储加密后的二进制密文
    vector<vector<bool>> > bin_code_text;

    // 对每个文本块进行加密
    for (int i = 0; i < bin_raw_text.size(); i++) {
        vector<bool> temp_res = encry_process(bin_raw_text[i], sub_key);
        bin_code_text.push_back(temp_res);
    }
}

```



```

// 解密操作函数
int CDesOperate::decry_operation(string cipherText, string key, string& decry_res) {
    // 检查密钥长度是否为8
    if (key.size() != 8) {
        cout << "error: key size isn't 8!" << endl;
        return -1;
    }

    // 将密钥字符串转换为布尔向量
    vector<bool> bin_key = key_str2bool(key);

    // 生成子密钥
    vector< vector<bool> > sub_key = gen_subkey(bin_key);

    // 将密文字符串转换为布尔向量
    vector< vector<bool> > bin_code_text = decry_str2bool(cipherText);

    // 存储解密后的布尔向量
    vector< vector<bool> > bin_raw_text;

    // 解密过程
    for (int i = 0; i < bin_code_text.size(); i++) {
        vector<bool> temp_res = decry_process(bin_code_text[i], sub_key);
        bin_raw_text.push_back(temp_res);
    }

    // 将解密后的布尔向量转换为字符串
    for (int i = 0; i < bin_raw_text.size(); i++) {
        string temp_res = "";
        int letter_assign = 0;
        while (letter_assign < 64) {
            // 截取每8位二进制作为一个字符
            vector<bool> cuttedBin(bin_raw_text[i].begin() + letter_assign,
                                   bin_raw_text[i].begin() + letter_assign + 8);
        }
    }
}

```

主要是实现了对文本块的加解密，实现过程中需要调用具体的 DES 加解密

函数：

即：

```

// 步骤 1: 初始置换 IP
vector<bool> temp_step1 = init_replacement_
vector<bool> step1_l, step1_r;
vector<bool> temp_l, temp_r;
vector<bool> step3;

// 将初始置换结果分为左右两部分
for (int i = 0; i < 32; i++) {
    temp_l.push_back(temp_step1[i]);
    temp_r.push_back(temp_step1[i + 32]);
}
step1_l = temp_l;
step1_r = temp_r;

// 步骤 2: 16 次迭代
// 每次迭代: l_i = r_(i-1), r_i = f(r_(i-1))
for (int i = 0; i < 16; i++) {
    // 计算 F 函数的结果
    vector<bool> f_func_res = f_func(step1_
    // 计算 XOR 结果
    vector<bool> xor_res = XOR(f_func_res,
    step1_l = step1_r;
    step1_r = xor_res;
}

// 步骤 3: 逆初始置换 IP
step3 = step1_l;
step3.insert(step3.end(), step1_r.begin(),

```

以及:

```

// 解密处理函数
vector<bool> CDesOperate:: decry_proce
    // 步骤1: 初始置换 IP
    vector<bool> temp_step1 = init_repl
    vector<bool> step1_l, step1_r;
    vector<bool> temp_l, temp_r;
    vector<bool> step3;

    // 将初始置换后的向量分成左右两半
    for (int i = 0; i < 32; i++) {
        temp_l.push_back(temp_step1[i]);
        temp_r.push_back(temp_step1[i]);
    }
    step1_l = temp_l;
    step1_r = temp_r;

    // 步骤2: 16轮迭代
    // 每轮迭代: r_i = l_(i-1), l_i = r_(i-1)
    for (int i = 0; i < 16; i++) {
        // F 函数的结果
        vector<bool> f_func_res = f_func(step1_l, step1_r);
        vector<bool> xor_res = XOR(f_func_res, step1_r);
        step1_r = step1_l;
        step1_l = xor_res;
    }

    // 步骤3: 逆初始置换 IP
    step3 = step1_l;
    step3.insert(step3.end(), step1_r.begin(), step1_r.end());
    step3 = init_replacement_IP(step3);
    return step3;

```

这两个函数才算真正具体的 DES 加解密过程。

事实上每次的 DES 加解密函数又调用了 DES_func.h 中的函数，即子密钥生成、F 函数的实现等等，而这些函数又进一步调用了 DES_head.h 中的数据结构等等。

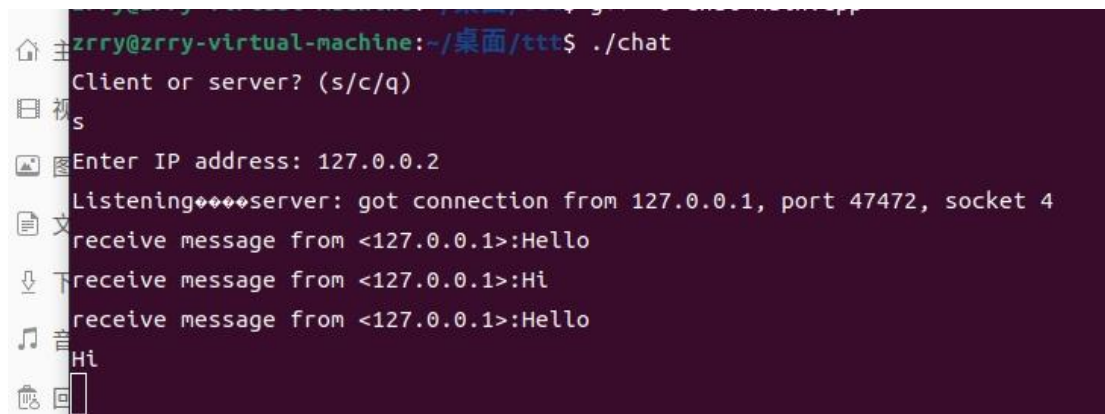
在实验文档中有这些内容的详细说明而且代码量太大故不在此赘述。

(6) 实验结果

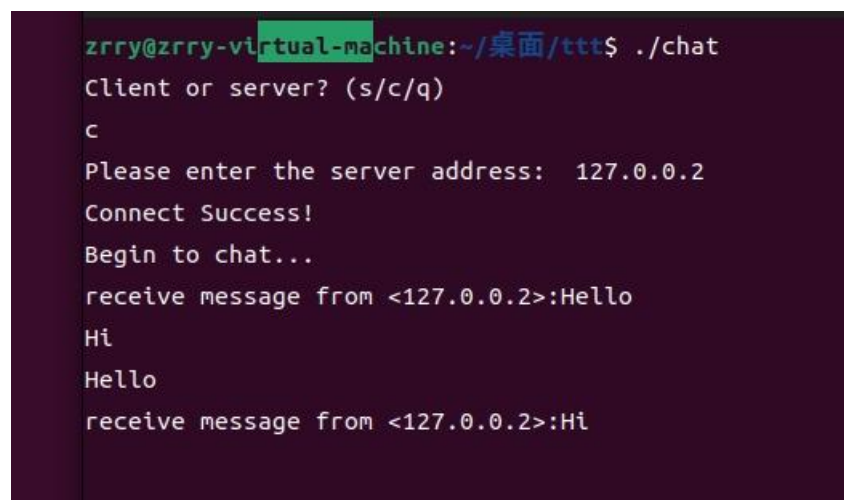
如下：

先启动 Server 端，Server 进行监听。

后启动 Client 端，输入 Server 地址，进行连接，然后进行互发消息。



```
zrry@zrry-virtual-machine:~/桌面/ttt$ ./chat
Client or server? (s/c/q)
s
Enter IP address: 127.0.0.2
Listeningserver: got connection from 127.0.0.1, port 47472, socket 4
receive message from <127.0.0.1>:Hello
receive message from <127.0.0.1>:Hi
receive message from <127.0.0.1>:Hello
Hi
```



```
zrry@zrry-virtual-machine:~/桌面/ttt$ ./chat
Client or server? (s/c/q)
c
Please enter the server address: 127.0.0.2
Connect Success!
Begin to chat...
receive message from <127.0.0.2>:Hello
Hi
Hello
receive message from <127.0.0.2>:Hi
```

四、实验遇到的问题及其解决方法

实验文档其实有详尽的代码样例，但有很多代码细节需要我们来实现。

在 DES 加解密和 TCP 数传输中，往往需要进行将文件数据转换为二进制数据的操作，而转化的过程中又往往涉及多种数据结构，令人感到繁琐，最终通过资料查询和工具使用来解决。

在 DES 实现中，因为代码环环相扣，有时在一个函数上出现错误，会导致结果错误甚至没有结果，只能慢慢排查，且多加验证。

五、实验结论

本次实验，我成功实现了一个基于 DES 加密和 TCP 协议的聊天程序。在这个程序中，我深入理解了 DES 加解密的原理和 TCP 协议的工作原理，并掌握了在 Linux 环境下基于 socket 的编程方法。

在实验过程中，我遇到了一些问题，比如在实现 DES 加解密过程中，需要处理大量的数据结构和调用多个函数，容易出现错误。另外，在调试过程中，由于代码的复杂性，有时候需要耐心地逐步排查错误，确保程序的正确性。

总的来说，本次实验让我深入了解了网络安全技术的一些基本原理和实践方法，为进一步学习和应用网络安全技术打下了良好的基础。