# 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](#header-n405)  
 [一、实验目的](#header-n414)  
 [二、实验内容](#header-n419)  
 [三、实验步骤及实验结果](#header-n424)  
 [DES加密](#header-n425)  
 [加密大致流程](#header-n426)  
 [密钥生成](#header-n430)  
 [轮加密](#header-n433)  
 [TCP通信](#header-n441)  
 [通信模型图](#header-n442)  
 [TCP通信原理](#header-n445)  
 [代码](#header-n447)  
 [初始设置](#header-n448)  
 [socket的创建 socket()](#header-n451)  
 [server绑定套接字和地址 bind()](#header-n453)  
 [server生成初始密钥](#header-n455)  
 [server进行监听 listen()](#header-n459)  
 [client发起连接请求 connect()](#header-n461)  
 [server端受连接请求 accept()](#header-n464)  
 [消息的加密与解密](#header-n470)  
 [client加工初始密钥](#header-n473)  
 [client发送/接受数据](#header-n477)  
 [server转发报文](#header-n485)  
 [程序的退出](#header-n502)  
 [四、实验遇到的问题及其解决方法](#header-n517)  
 [五、实验结论](#header-n572)

## 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。

（正文采用宋体小四，间距20磅）

## 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。

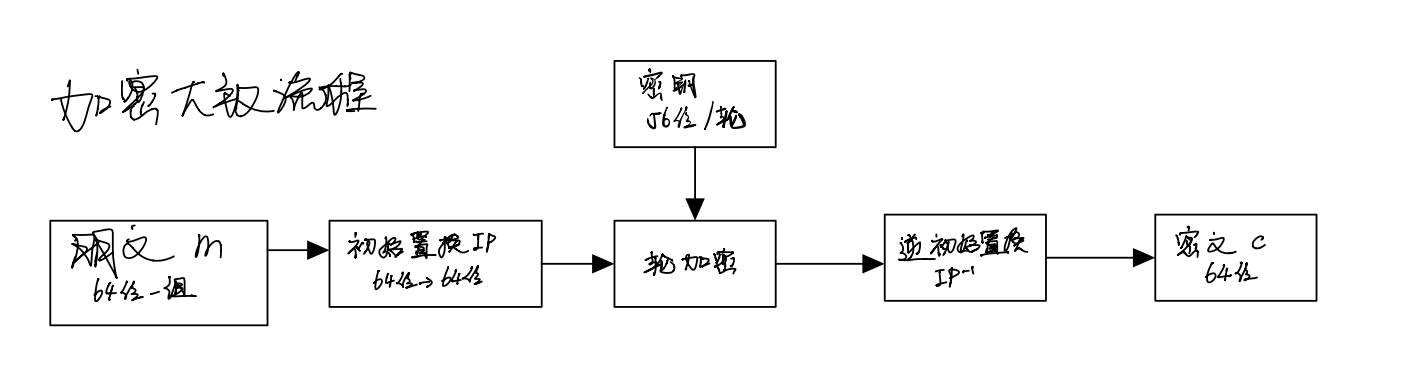
① 能够在了解 DES 算法原理的基础上，编程实现对字符串的 DES 加密解密操作。 ② 能够在了解 TCP 和 Linux 平台下的 Socket 运行原理的基础上，编程实现简单的 TCP 通信，为简化编程细节，不要求实现一对多通讯。 ③ 将上述两部分结合到一起，编程实现通信内容事先通过 DES 加密的 TCP 聊天程序， 要求双方事先互通密钥，在发送方通过该密钥加密，然后由接收方解密，保证在网络上传输 的信息的保密性。

## 三、实验步骤及实验结果

### DES加密

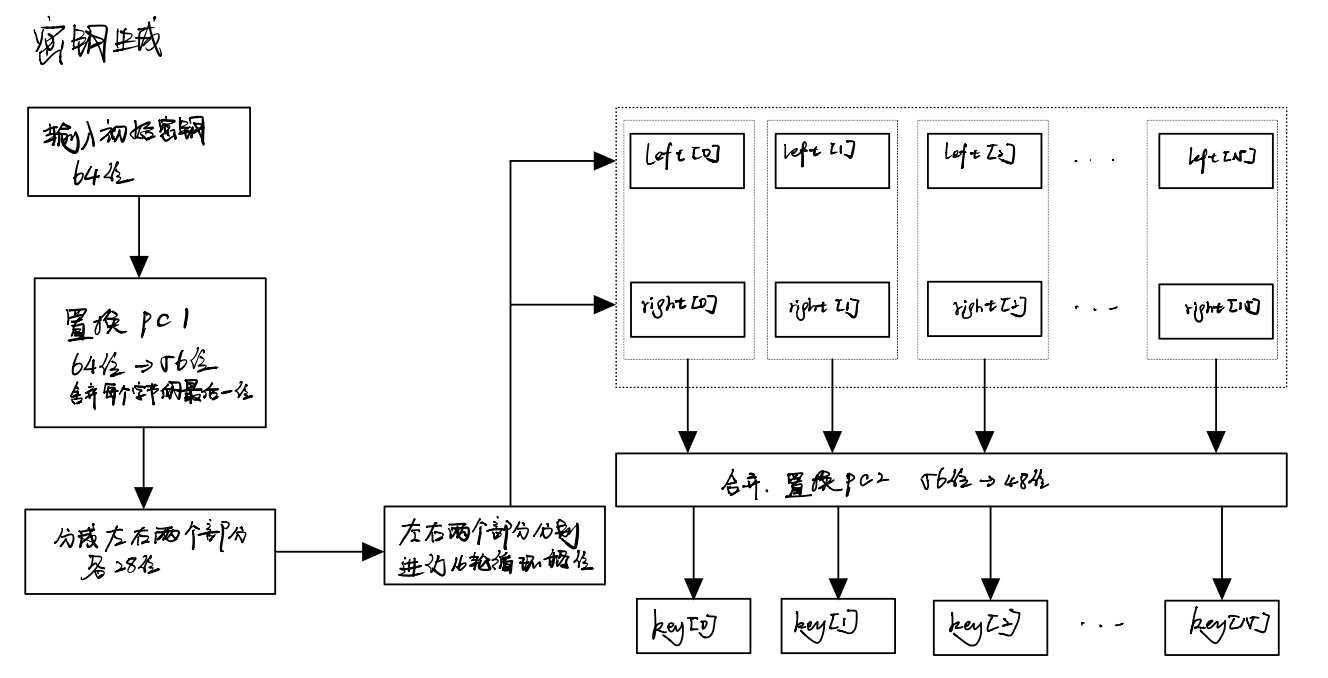
#### 加密大致流程

**（解密流程相似，只是密钥使用顺序相反）**



void round(BYTE m\_[8], BYTE key\_final[16][6], BYTE afterIP\_1[8])  
{  
 //m\_为明文，key\_final为生成的十六组密钥，afterIP用于存储密文  
 BYTE m[8] = {};  
 convIP(m\_, m);//初始置换IP  
 ....//轮加密  
   
 convIP\_1(afterCombine, afterIP\_1);//逆初始置换  
}

#### 密钥生成



//密钥产生  
void getkeys(BYTE key[8], BYTE key\_final[16][6])  
{  
 BYTE keys[7] = {};//存储经过PC1置换后的56位中间结果  
 for (int i = 0; i < 56; i++)  
 convert(key, keys, i, PC\_1);//PC1置换  
 bitset<56> key\_round[17];  
 //初始化全为0  
 for (int i = 0; i < 16; i++)key\_round[i].reset();  
 //信息复制，将BYTE数组格式转为bitset格式，便于后续处理  
 for (int i = 0; i < 7; i++)  
 {  
 for (int j = 0; j < 8; j++)  
 {  
 if (getbit(keys[i], j) == 1)  
 key\_round[0].set(i \* 8 + j);//该位设为1  
 }  
 }  
 for (int i = 0; i < 16; i++)//28位一组循环移位操作  
 {  
 bitset<56> left\_mov = key\_round[i] >> MOV[i];//注意大小端序问题，实际是左移  
 for (int j = 0; j < MOV[i]; j++)  
 {  
 left\_mov.set(28 - MOV[i] + j, key\_round[i][j]);  
 left\_mov.set(56- MOV[i] + j, key\_round[i][28 + j]);  
 }  
 key\_round[i+1] = left\_mov;  
 }  
  
 //PC\_2置换，把bitset转会BYTE，便于观察中间结果  
 for(int i=0;i<16;i++)  
 for (int j = 0; j < 48; j++)  
 {  
 write(key\_final[i][j/8], j%8, key\_round[i + 1][PC\_2[j] - 1]);  
 }  
}

#### 轮加密

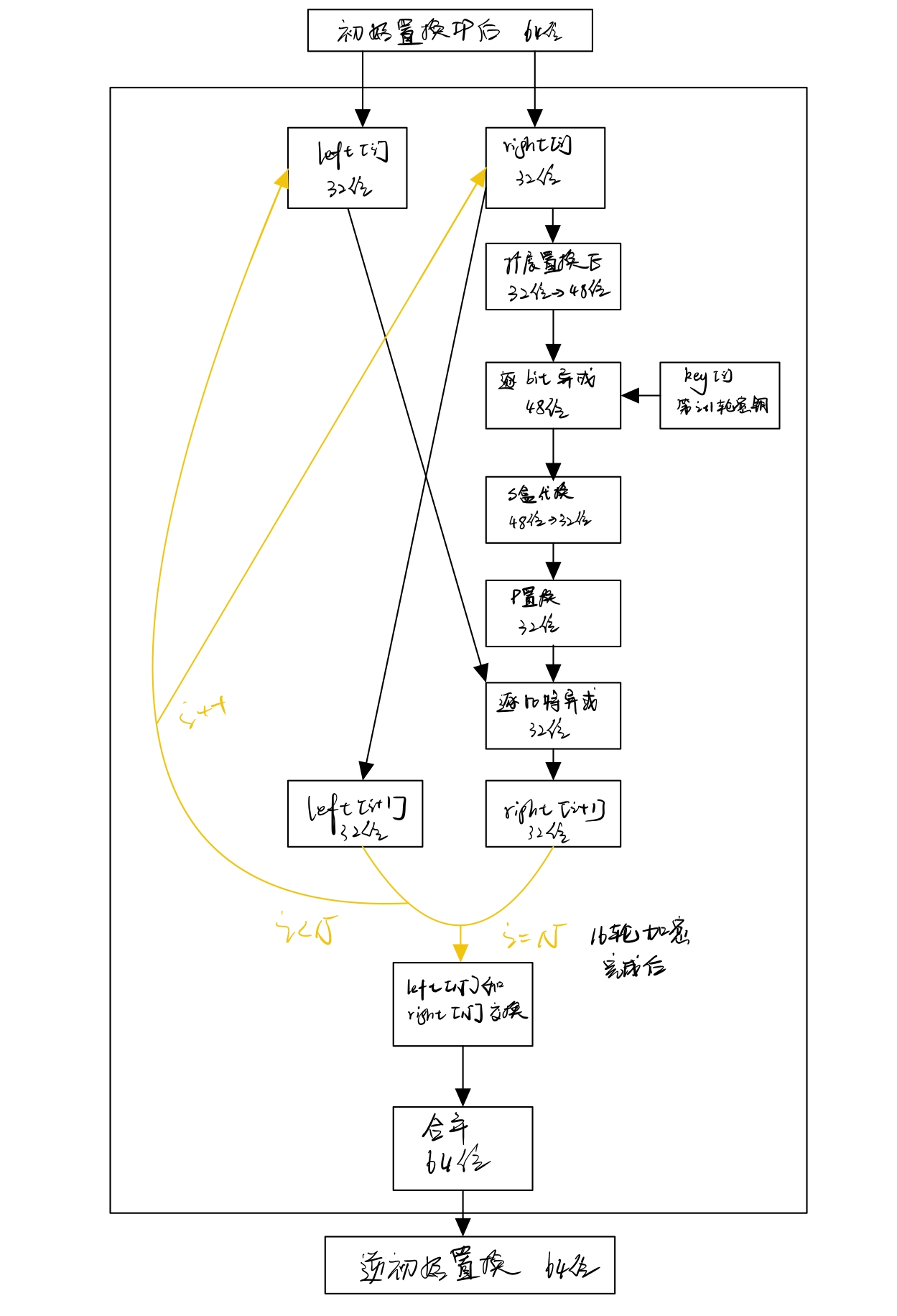


image-20210406161921364

void round(BYTE m\_[8], BYTE key\_final[16][6], BYTE afterIP\_1[8])  
{  
 BYTE m[8] = {};  
 ...;//初始置换IP  
   
 //轮加密过程  
 BYTE left[4], right[4];  
 for (int i = 0; i < 4; i++)//分成左右两部分  
 {  
 left[i] = m[i];  
 right[i] = m[i+4];  
 }  
 for (int i = 0; i < 16; i++)//十六轮加密  
 {  
 BYTE afterE[6] = {};//扩展置换后的结果  
 //扩展置换E  
 convertE(right,afterE);  
 //与密钥进行逐bit异或  
 BYTE afterXOR[6] = {};  
 xor\_(key\_final[i], afterE, afterXOR);  
 //S盒代换  
 BYTE afterS[4] = {};  
 convertS(afterXOR, afterS);  
 //P置换  
 BYTE afterP[4] = {};  
 convertP(afterS, afterP);  
 BYTE afterXOR2[4] = {};  
 for (int i = 0; i < 4; i++)//逐比特异或  
 {  
 afterXOR2[i] = left[i] ^ afterP[i];  
 }  
 //生成下一轮的left和right  
 for (int i = 0; i < 4; i++)left[i] = right[i];  
 for (int i = 0; i < 4; i++)right[i] = afterXOR2[i];  
 int ix = 0;  
 }  
  
 //左右交换  
 BYTE temp[4] = {};  
 for (int i = 0; i < 4; i++)  
 {  
 temp[i] = left[i];  
 left[i] = right[i];  
 right[i] = temp[i];  
 }  
 //合并  
 BYTE afterCombine[8];  
 for (int i = 0; i < 4; i++)  
 {  
 afterCombine[i] = left[i];  
 afterCombine[i + 4] = right[i];  
 }  
 ...//逆初始置换  
}

其中，为简化各个转换过程，使用了一个转换函数

//做对应转换  
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  
{  
 int index\_bit\_b = i;  
 tuple\_ index\_u\_char\_b = bit2u\_char(index\_bit\_b);//i=(i/u\_char\_)\*8+(i%u\_char\_)  
 int index\_bit\_a = conv[index\_bit\_b] - 1;//找到对应位置  
 tuple\_ index\_u\_char\_a = bit2u\_char(index\_bit\_a);  
 int temp = getbit(a[index\_u\_char\_a.u\_char\_], index\_u\_char\_a.bit\_);//提取出第i位  
 write(b[index\_u\_char\_b.u\_char\_], index\_u\_char\_b.bit\_, temp);//写入b中  
}

转换函数调用样例

//初始置换  
void convIP(u\_char a[], u\_char b[])//将a进行IP置换，结果存入b中  
{  
 for (int i = 0; i < 64; i++)  
 {  
 convert(a, b, i, IP);  
 }  
}

### 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

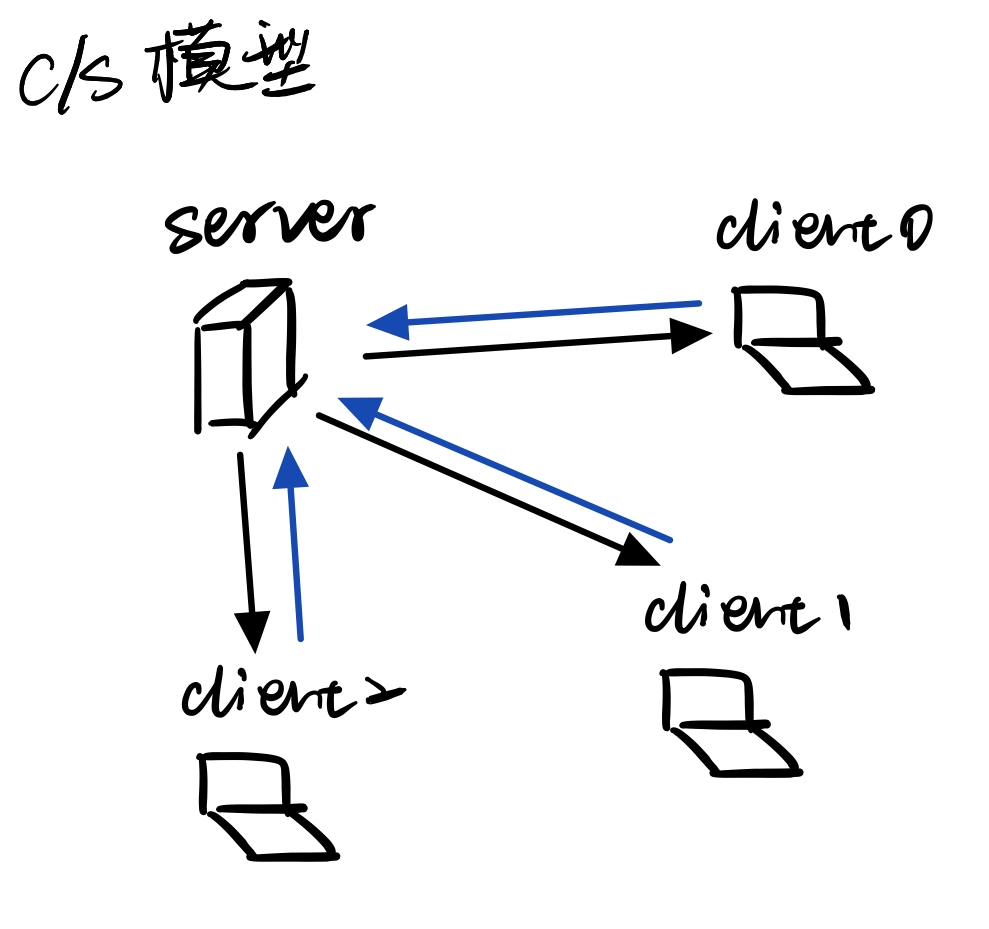
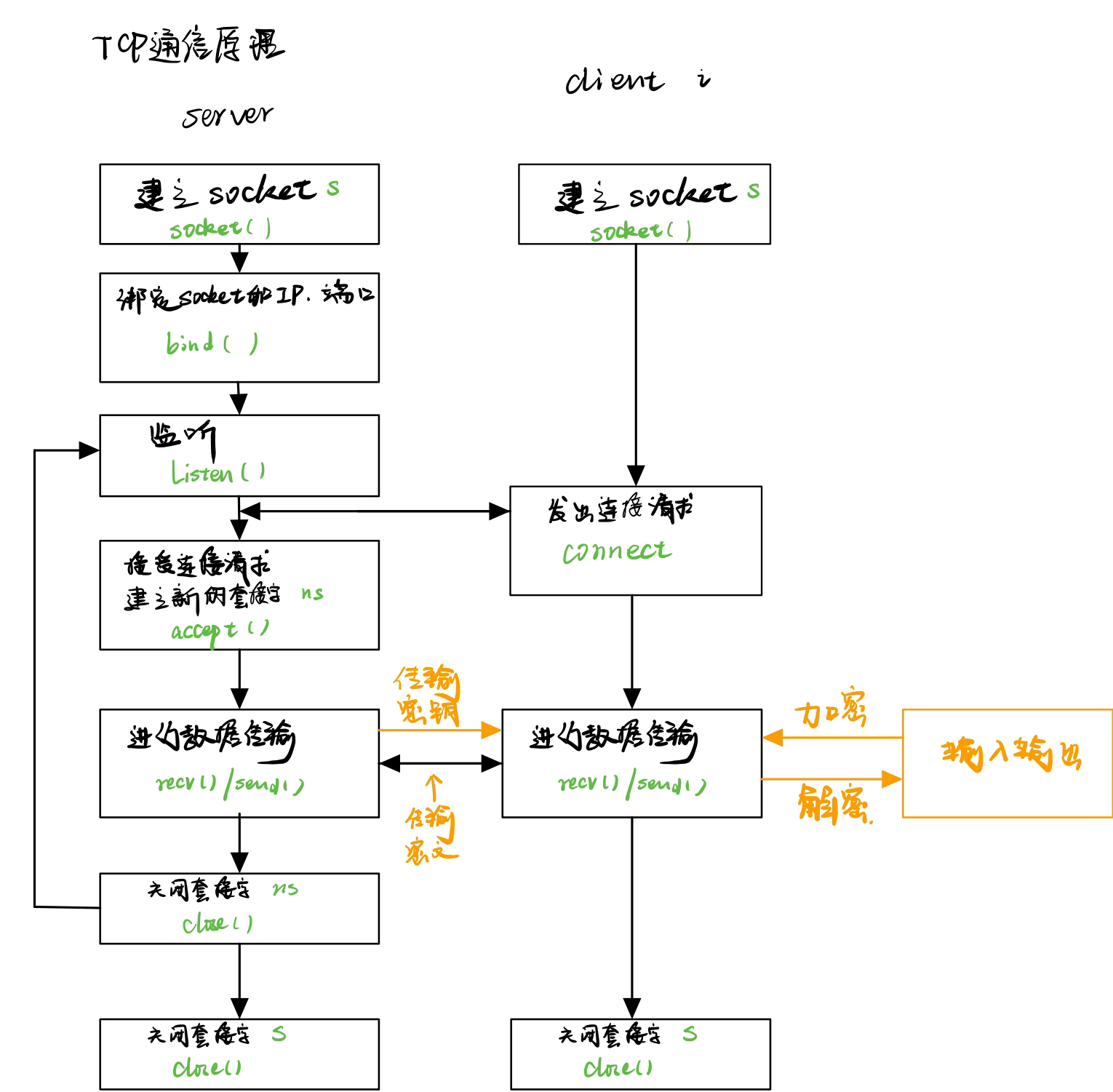


image-20210330201611866

#### TCP通信原理



#### 代码

##### 初始设置

#include <WinSock2.h>//windows socket 编程头文件  
#pragma comment(lib,"ws2\_32.lib")//链接ws2\_32.lib库文件到此项目中﻿

//加载socket库  
WSADATA wsaData;  
//MAKEWORD(2.2),成功返回0  
if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)  
{  
 cout << "socket初始化失败" << endl;  
 return 0;

##### socket的创建 socket()

//创建一个socket，并将该socket绑定到一个特定的传输层  
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()

//初始化地址，client端地址初始化方法相同  
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  
addrSer.sin\_family = AF\_INET;  
addrSer.sin\_port = htons(PORT);  
  
//绑定  
if (bind(sockSer, (SOCKADDR\*)&addrSer, sizeof(SOCKADDR)) == -1)  
 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

string initial\_key()//server生成64位初始密钥  
{  
 string key="";  
 for (int i = 0; i < 8; i++)  
 {  
 key += rand() % 128;  
 }  
 return key;  
}

string key = initial\_key();

##### server进行监听 listen()

listen(sockSer, 5);//等待队列最大长度为5

##### client发起连接请求 connect()

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()

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**

sockConn[i] = accept(sockSer, (SOCKADDR\*)&addrCli, &len);//失败sockConn=INVALID\_SOCKET  
if (sockConn[i] != INVALID\_SOCKET)  
{  
 cond++;//人数加一  
 string buf = "你的id是：";  
 buf+= 48 + i;//简化，最多九个人  
 send(sockConn[i], buf.data(), 50, 0);  
 send(sockConn[i], key.data(), 50, 0);  
 cout << "clients "<<i<<" have connected" << endl;  
}

为了便于实验观察，这里client会将密钥打印出来

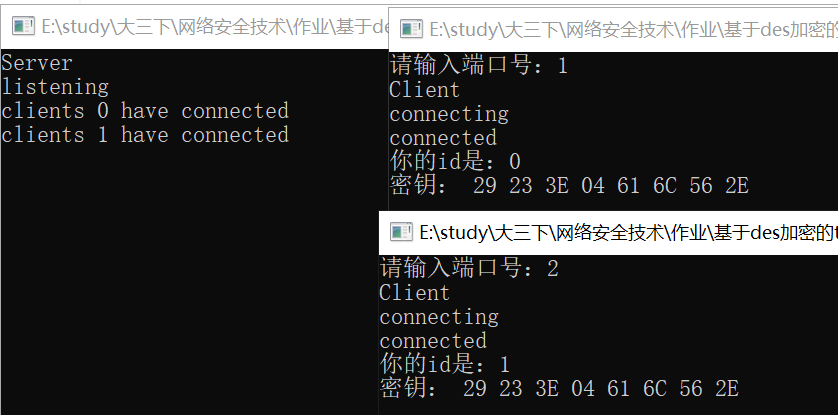


image-20210330205801454

##### 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**

char\* msg\_en(char\* a, u\_char key\_final[16][6],char \* cipher) {  
 int len = strlen(a);  
 u\_char temp[8],tempcipher[8];  
 int index = 0;  
 for (int i = 0; i < len; i+=8)//分组处理，每组64位8个字节  
 {  
 memset(temp, 0, sizeof(temp));  
 memset(tempcipher, 0, sizeof(tempcipher));  
 for (int j = 0; j < 8; j++)  
 {  
 temp[j] = (u\_char)a[i + j];  
 }  
 round(temp, key\_final, tempcipher);  
 for (int j = 0; j < 8; j++)  
 {  
 cipher[index++] = tempcipher[j];  
 }  
 }  
 return cipher;  
}

##### 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

//生成密钥  
getkeys((u\_char\*)recvBuf, key\_final); //加密密钥  
for (int i = 0; i < 16; i++)  
 for (int j = 0; j < 6; j++)  
 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**

DWORD WINAPI handlerRequest1(LPVOID lparam)  
{  
 while (1)  
 {  
 char sendBuf[BUF\_SIZE] = {};  
 char buffer[BUF\_SIZE] = {};  
 SOCKET socketClient = (SOCKET)(LPVOID)lparam;  
 cin.getline(buffer, 2048, '\n');  
 if (buffer[0])//如果检测到了键盘输入  
 {  
 sendBuf[0] = ID + 48;//消息格式封装  
 msg\_form m = string\_to\_msg(sendBuf);  
 if (!strcmp(m.msg, "quit") || !strcmp(buffer, "quit"))//不是程序退出指令  
 {//退出程序  
 sendBuf[1] = ID + 48;  
 strcat(sendBuf, buffer);  
 send(socketClient, sendBuf, 2048, 0);  
 cond = 1;  
 return 0;  
 }  
 //提取消息内容  
 sendBuf[1] = buffer[0];  
 int i;  
 for (i = 1; buffer[i] != 0; i++)  
 buffer[i - 1] = buffer[i];  
 buffer[i - 1] = 0;  
 char buf[BUF\_SIZE] = {};  
 msg\_en(buffer, key\_final,buf);//对消息内容进行加密  
 strcat(sendBuf, buf);//拼接报头（发送方和接收方id）  
 send(socketClient, sendBuf, 2048, 0);//发送  
 }  
 Sleep(200);  
 }  
 return 0;  
}

**接收消息**

DWORD WINAPI handlerRequest2(LPVOID lparam)  
{  
 while (1)  
 {  
 char recvBuf[BUF\_SIZE] = {};  
 SOCKET socketClient = (SOCKET)(LPVOID)lparam;  
 recv(socketClient, recvBuf, 2048, 0);  
 if (recvBuf[0])  
 {//如果有接收到的内容  
 cout << recvBuf[0] << ": ";  
 int i;  
 for (i = 2; recvBuf[i] != 0; i++)  
 recvBuf[i - 2] = recvBuf[i];//去掉报头  
 recvBuf[i - 2] = 0;  
 char buf[BUF\_SIZE] = {};  
 msg\_de(recvBuf, key\_final\_, buf);//对消息进行解密  
 cout << buf << endl;//打印明文  
 int k = 1;  
 }  
 Sleep(200);  
 }  
 return 0;  
}

**主函数**

while(1)  
{  
 hThread1 = ::CreateThread(NULL, NULL, handlerRequest1, LPVOID(sockClient), 0, &dwThreadId1);  
 hThread2 = ::CreateThread(NULL, NULL, handlerRequest2, LPVOID(sockClient), 0, &dwThreadId2);  
 WaitForSingleObject(hThread1, 200);//由于线程函数中有while(1)，该函数只需要创建一个线程  
 WaitForSingleObject(hThread2, 200);  
 /\*CloseHandle(hThread2);  
 CloseHandle(hThread1);\*/  
 if (cond) break;  
}

##### 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**

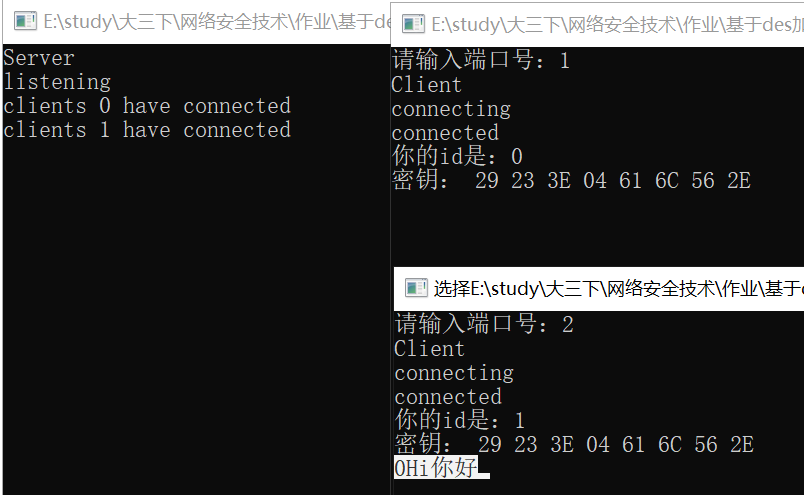
class msg\_form  
{  
public:  
  
 char from\_name;//发送方标号  
 char to\_name;//接收方  
 char msg[BUF\_SIZE];//消息内容  
 ...  
}

需要提取接收方ID，并顶向传送消息

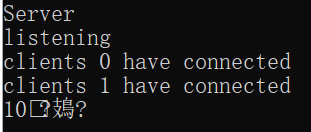
char recvBuf[BUF\_SIZE] = {};  
recv(sockettemp, recvBuf, 1000, 0);  
if (recvBuf[0])  
{  
 cout << recvBuf << endl;  
 msg\_form m = string\_to\_msg(recvBuf);  
 int id = m.to\_name - 48;  
 send(sockConn[id], recvBuf, 50, 0);  
 ...  
}

通信过程样例：

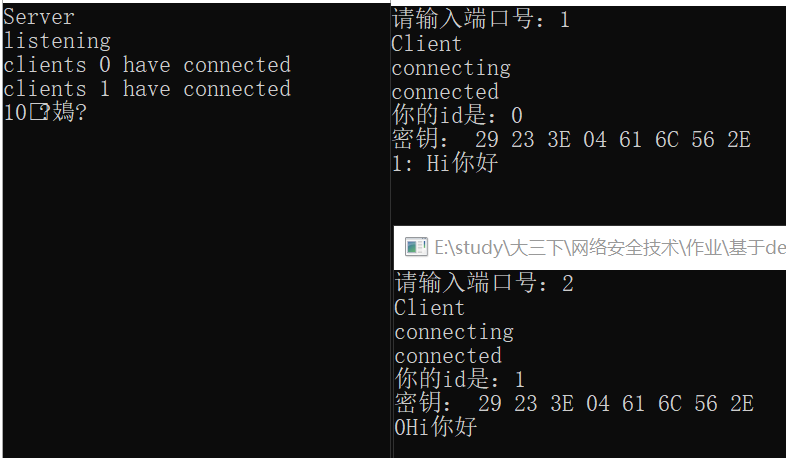
1. 输入目的clientid和消息内容

* 
* image-20210330205932982

1. 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**

* 

1. client0收到消息并进行解密

* 
* image-20210330210214286

##### 程序的退出

client端输入quit，向server发送quit消息后主动退出

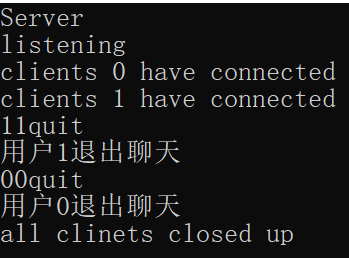
server端收到client发送的quit指令判断该client退出，当所有client都断开连接时退出程序，关闭socket

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

关闭socket

closesocket(sockSer);  
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fig:a
2. client0发送quit，所有连接断开，server退出

## 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serer和client设置了统一IP地址和端口
2. 加解密结果部分正确部分错误
   * 发现是S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
   * 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

## 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

备注：（正文采用宋体小四，间距20磅）

以上说明仅供参考。实验报告从这几部分展开，具体内容可自由发挥。如有雷同，均按零分处理。