网络安全技术

实验报告

学　　院　网安学院
年　　级　2020
班　　级　密码科学与技术
学　　号　2011428
姓　　名　王天行
手机号　18856929728

2023 年 3 月 12 日

目录

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 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 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

1.DES 加密

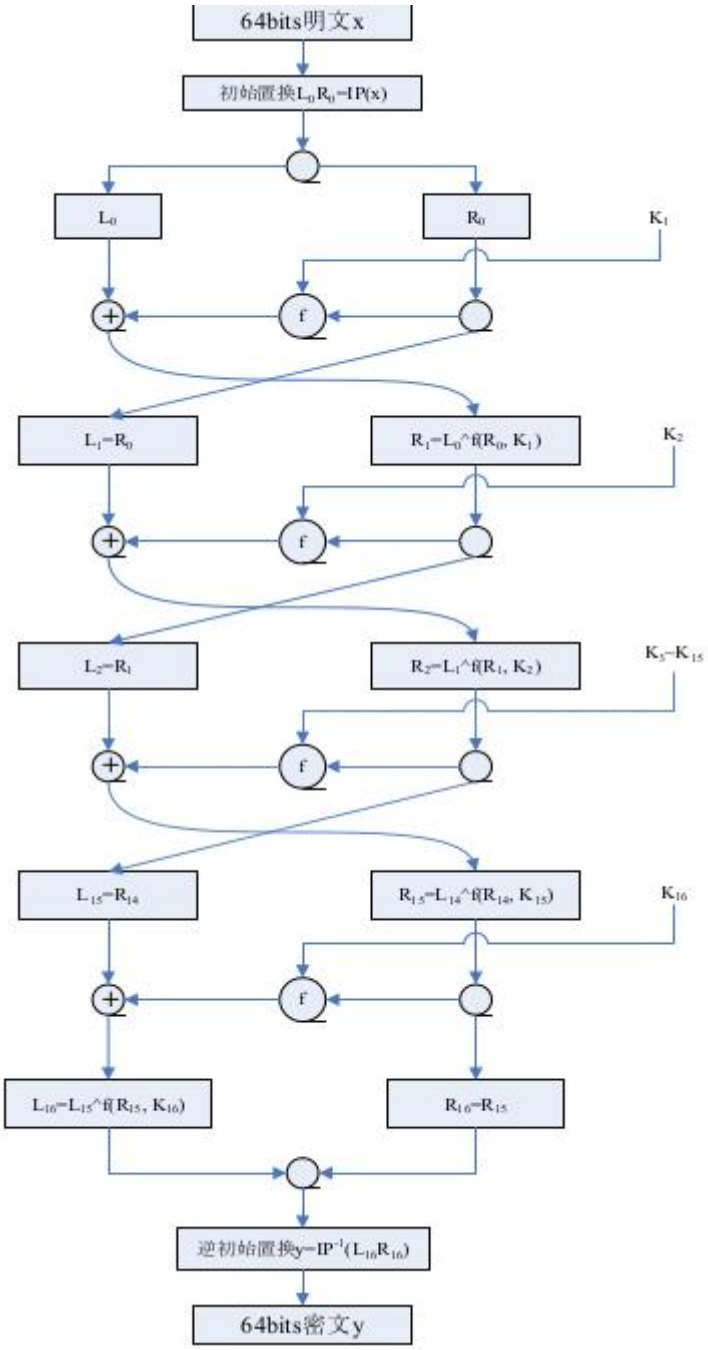
DES 类

```
class CDesOperator {
private:
    ULONG32 firstKey[2];//用于生成16个子秘钥
    ULONG32 subKey[16][2];//用与保存生成的16个子秘钥,并参与加密运算
    bool makeFirstKey(ULONG32* orgKey);//生成初始秘钥
    bool oneStepOfMakeSubKey(ULONG32* left, ULONG32* right, INT32 number);//16轮迭代生成子秘钥的每一轮运算 参数三为生成的子秘钥序号
    bool oneStepOfMakeData(ULONG32* left, ULONG32* right, INT32 number);//16轮迭代加密或解密的每一轮运算 参数三为子秘钥的选取
    bool handleData(ULONG32* data, ULONG8 choice);//用于完成一次独立的加密或解密过程
public:
    CDesOperator() {
        for (int i = 0; i < 16; i++) {
            for (int j = 0; j < 2; j++) {
                subKey[i][j] = 0;
            }
        }
        for (int i = 0; i < 2; i++) {
            firstKey[i] = 0;
        }
    }
    //加密 参数依次为  明文  明文长度  密文 密文长度  秘钥 秘钥长度
    bool Encry(char* pPlaintext, int nPlaintextLength, char* pCipherBuffer, int& nCipherBufferLength, char* pKey, int nKeyLength);
    //解密 参数依次为 密文 密文长度 明文 明文长度  秘钥  密钥长度
    bool Decry(char* pCipher, int nCipherBufferLength, char* pPlaintextBuffer, int& nPlaintextBufferLength, char* pKey, int nKeyLength);
};
```

其中 handleData 用来执行一次完整的加密或解密操作，oneStepOfMakeData 用来实现 16 轮加密或解密迭代中的每一轮除去初始置换和逆初始置换的中间操作，MakeFirstKey 用来利用用户输入的初始密钥，来形成 16 个迭代用到的子密钥，oneStepOfMakeKey 用来形成 16 个密钥中的每一个子密钥，Encry 用来对某一段字符加密，Decry 用来对某一段密文解密。

加密

流程图如下


64bits明文x
初始置换L_0R_0=IP(x)
L_0
R_0
K_1
f
$L_1=R_0$
$R_1=L_0$^f(R_0, K_1)
K_2
$L_2=R_1$
$R_2=L_1$^f(R_1, K_2)
K_3–K_{15}
$L_{15}=R_{14}$
$R_{15}=L_{14}$^f(R_{14}, K_{15})
K_{16}
$L_{16}=L_{15}$^f(R_{15}, K_{16})
$R_{16}=R_{15}$
逆初始置换y=IP^{-1}($L_{16}R_{16}$)
64bits密文y

```cpp
bool CDesOperator::Encry(char* pPlaintext, int nPlaintextLength, char* pCipherBuffer, int& nCipherBufferLength, char* pKey, int nKeyLength) {
    if (nKeyLength != 8) { return false; }
    makeFirstKey((ULONG32*)pKey);
    int length = ((nPlaintextLength + 7) / 8) * 2;      //length×4后肯定比原文长而且为64的整数倍
    if (nCipherBufferLength < length * 4) {                 //密文不够长
        nCipherBufferLength = length * 4;
    }
    memset(pCipherBuffer, 0, nCipherBufferLength);//密文置0
    ULONG32* output = (ULONG32*)pCipherBuffer;
    ULONG32* source;
    if (nPlaintextLength != sizeof(ULONG32) * length)     //确保明文64位对齐
    {
        source = new ULONG32[length];
        memset(source, 0, sizeof(ULONG32) * length);
        memcpy(source, pPlaintext, nPlaintextLength);
    }
    else
    {
        source = (ULONG32*)pPlaintext;
    }
    ULONG32 msg[2] = { 0,0 };
    for (int i = 0; i < (length / 2); i++) {                                    //每64位为一次加密
        msg[0] = source[2 * i];                 //64位为两个long，2位进行一次加密，下同
        msg[1] = source[2 * i + 1];
        handleData(msg, (ULONG8)0);                                 //加密
        output[2 * i] = msg[0];                                     //得到密文
        output[2 * i + 1] = msg[1];
    }
    if (pPlaintext != (char*)source)
    {
        delete[]source;
    }
    return true;
}
```

解密

```cpp
bool CDesOperator::Decry(char* pCipher, int nCipherBufferLength, char* pPlaintextBuffer, int& nPlaintextBufferLength, char* pKey, int nKeyLength) {
    if (nCipherBufferLength % 8 != 0)
    {
        return false;
    }
    if (nPlaintextBufferLength < nCipherBufferLength)           //与加密过程类似,不必多说
    {
        nPlaintextBufferLength = nCipherBufferLength;
        return false;
    }
    if (nKeyLength != 8)
    {
        return false;
    }
    makeFirstKey(_ergKey (ULONG32*)pKey);
    memset(_Dst pPlaintextBuffer, _Val 0, _Size nPlaintextBufferLength);
    ULONG32* pSouce = (ULONG32*)pCipher;
    ULONG32* pDest = (ULONG32*)pPlaintextBuffer;
    ULONG32 gp_msg[2] = { 0,0 };
    for (int i = 0; i < (nCipherBufferLength / 8); i++)
    {
        gp_msg[0] = pSouce[2 * i];
        gp_msg[1] = pSouce[2 * i + 1];
        handleData(left gp_msg, choice (ULONG8)1);
        pDest[2 * i] = gp_msg[0];
        pDest[2 * i + 1] = gp_msg[1];
    }
    return true;
}
```

密钥生成

生成初始秘钥（makeFirstKey）

```
bool CDesOperator::makeFirstKey(ULONG32* orgKey) {  //输入的orgKey为64位 输出为firstKey[2]
    ULONG32 tempKey[2] = { 0 };
    ULONG32* pFirstKey = (ULONG32*)firstKey;
    ULONG32* pTempKey = (ULONG32*)tempKey;
    memset(_Dst:(ULONG8*)firstKey, _Val:0, _Size:sizeof(firstKey));
    memcpy(_Dst:(ULONG8*)&tempKey, _Src:(ULONG8*)orgKey, _Size:8);
    memset(_Dst:(ULONG8*)subKey, _Val:0, _Size:sizeof(subKey));
    int  j = 0;
    for (j = 0; j < 28; j++) {
        if (keyleft[j] > 32) {
            if (pTempKey[1] & pc_by_bit[keyleft[j] - 1]) {
                pFirstKey[0] |= pc_by_bit[j];
            }
        }
        else {
            if (pTempKey[0] & pc_by_bit[keyleft[j] - 1]) {
                pFirstKey[0] |= pc_by_bit[j];
            }
        }
        if (keyright[j] > 32) {
            if (pTempKey[1] & pc_by_bit[keyright[j] - 1]) {
                pFirstKey[1] |= pc_by_bit[j];
            }
        }
        else {
            if (pTempKey[0] & pc_by_bit[keyright[j] - 1]) {
                pFirstKey[1] |= pc_by_bit[j];
            }
        }
    }
    for (j = 0; j < 16; j++) {
        oneStepOfMakeSubKey(left:&pFirstKey[0], right:&pFirstKey[1], number:j);            //
    }
    return true;
}
```

生成 16 个密钥中的每一个子密钥（oneStepOfMakeSubKey）

```
bool CDesOperator::oneStepOfMakeSubKey(ULONG32* left, ULONG32* right, INT32 number) {
    ULONG32 tempKey[2] = { 0,0 };
    ULONG32* pTempKey = (ULONG32*)tempKey;
    ULONG32* pSubKey = (ULONG32*)subKey[number];
    ULONG32 helpData[3] = { 0x0,0x80000000,0xc0000000 };//辅助数据,通过与上它们可以得到相应数据的最高位,待会有奇用
    pTempKey[0] = *left & helpData[lefttable[number]];
    pTempKey[1] = *right & helpData[lefttable[number]];
    if (lefttable[number] == 1) {                          //注意要达到循环左移的效果,没有相应的操作,只有先将其最高位保存下来，在想办法将其移到低位上去
        pTempKey[0] >>= 27;                                //具体实现;
        pTempKey[1] >>= 27;                                    //与0xc0000000(110000000...高位为1)等数据相与得到高位(其他位被置0)
    }
    else {                                                                  //相与后的数据右移 将高位移到低位   由于只有28位 移26或27位即可，不必移30或31位
        pTempKey[0] >>= 26;                                //至此 最高位被移到了最低位(相对于28位) 接下来与左移的数据相或即可
        pTempKey[1] >>= 26;
    }
    pTempKey[0] &= 0xfffffff0;     //本来只有28位却必须用32位存储,直接将低4位清0即可
    pTempKey[1] &= 0xfffffff0;
    *left <<= lefttable[number];                                        // 左移
    *right <<= lefttable[number];
    *left |= pTempKey[0];                                               //相或
    *right |= pTempKey[1];
    pTempKey[0] = 0;
    pTempKey[1] = 0;                                                        //至此循环左移结束 接下来56-->48

    int j = 0;
    for (; j < 48; j++) {
        if (j < 24) {
            if (*left & pc_by_bit[keychoose[j] - 1]) {
                pSubKey[0] |= pc_by_bit[j];
            }
        }
        else {
            if (*right & pc_by_bit[keychoose[j] - 28]) {
                pSubKey[1] |= pc_by_bit[j - 24];
            }
        }
    }
    return true;
}
```

轮加密（oneStepOfMakeData）

```
bool CDesOperator::oneStepOfMakeData(ULONG32* left, ULONG32* right, INT32 number) {
    ULONG32 oldRight = *right;
    ULONG8 useBySBox[8] = { 0 };
    ULONG32 exdesP[2] = { 0 };  //用于存放拓展后的数据
     //32---->48
    int j = 0;
    for (; j < 48; j++) {                                   //只对right做拓展
        if (j < 24)
        {
            if (*right & pc_by_bit[des_E[j] - 1])
            {
                exdesP[0] |= pc_by_bit[j];
            }
        }
        else
        {
            if (*right & pc_by_bit[des_E[j] - 1])
            {
                exdesP[1] |= pc_by_bit[j - 24];
            }
        }

    }
    for (j = 0; j < 2; j++) {
        exdesP[j] ^= subKey[number][j]; //子秘钥参与的异或运算
    }
    //48------>32
    exdesP[1] >>= 8;       //24位存放再32的,所以左移8位到最低位
    useBySBox[7] = (ULONG8)(exdesP[1] & 0x0000003fL);         //与上00000...00111111  低6位全为1
    exdesP[1] >>= 6;                                          //左移6位
    useBySBox[6] = (ULONG8)(exdesP[1] & 0x0000003fL);
    exdesP[1] >>= 6;
    useBySBox[5] = (ULONG8)(exdesP[1] & 0x0000003fL);
    exdesP[1] >>= 6;
    useBySBox[4] = (ULONG8)(exdesP[1] & 0x0000003fL);
```

```
    exdesP[0] >>= 8;
    useBySBox[3] = (ULONG8)(exdesP[0] & 0x0000003fL);
    exdesP[0] >>= 6;
    useBySBox[2] = (ULONG8)(exdesP[0] & 0x0000003fL);
    exdesP[0] >>= 6;
    useBySBox[1] = (ULONG8)(exdesP[0] & 0x0000003fL);
    exdesP[0] >>= 6;
    useBySBox[0] = (ULONG8)(exdesP[0] & 0x0000003fL);
    exdesP[0] = 0;
    exdesP[1] = 0;                                          //至此数据被分为8组,每组6位(尽管必须用8位存储)
    *right = 0;
    for (j = 0; j < 7; j++) {                                       //查SBox表  6位变4位     即48---->32
        *right |= des_S[j][useBySBox[j]];
        *right <<= 4;
    }
    *right |= des_S[j][useBySBox[j]];
    ULONG32 tempData = 0;
    for (j = 0; j < 32; j++)                                        //不必多说 换位
    {
        if (*right & pc_by_bit[des_P[j] - 1])
        {
            tempData |= pc_by_bit[j];
        }
    }
    *right = tempData;

    *right ^= *left;                        //传的是指针 ,用于迭代
    *left = oldRight;
    return true;
}
```

2. 基于 TCP 的聊天功能模块设计实现

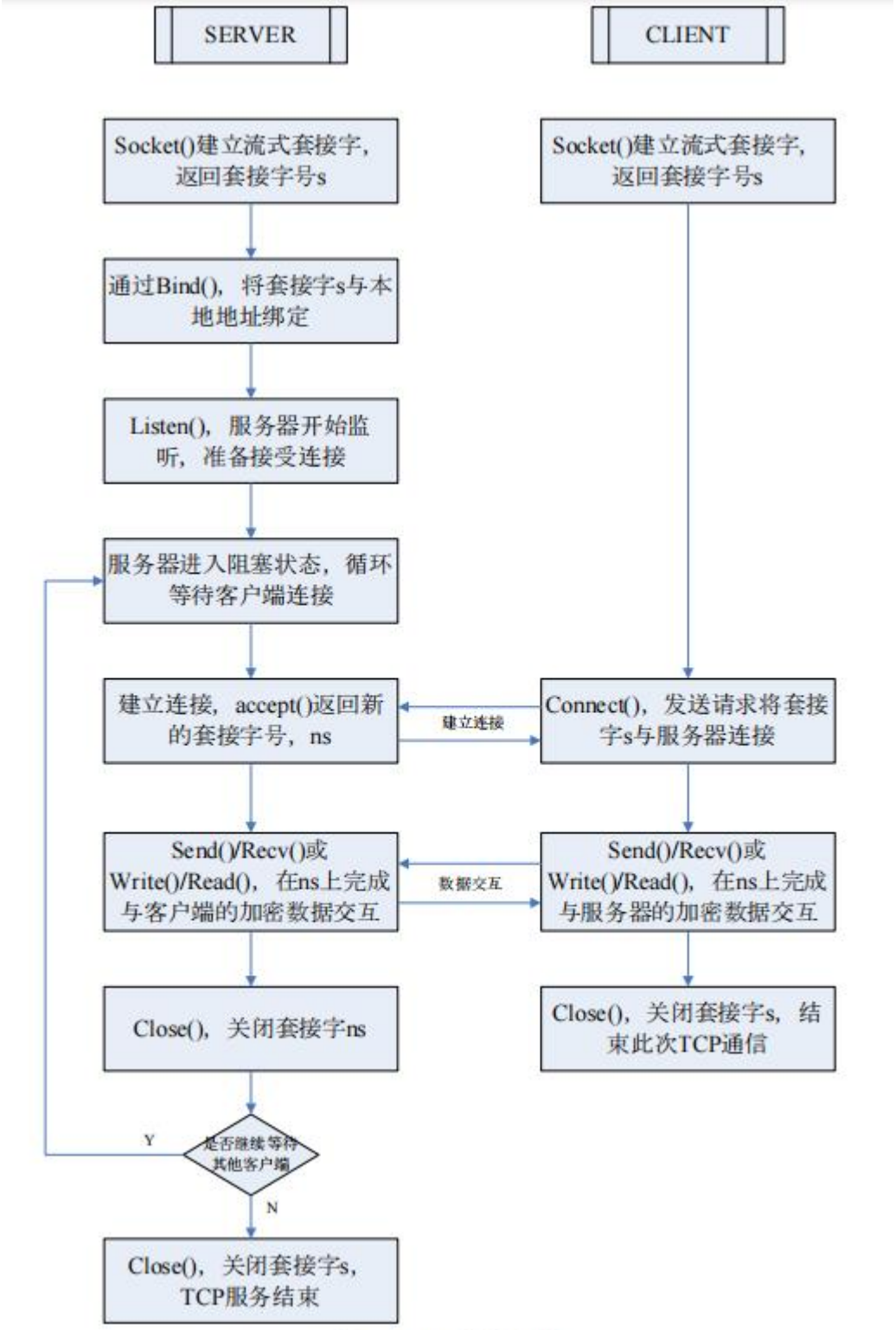


图 2-4 TCP 通信流程图

建立连接

客户端

```
if (strStdinBuffer[0] == 'c' || strStdinBuffer[0] == 'C')
{//be a client
    char strIpAddr[16];
    printf("Please input the server address:\r\n");
    cin >> strIpAddr;
    int nConnectSocket;
    struct sockaddr_in sDestAddr;
    if ((nConnectSocket = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        perror("Socket");
        exit(errno);
    }
    bzero(&sDestAddr, sizeof(sDestAddr));
    sDestAddr.sin_family = AF_INET;
    sDestAddr.sin_port = htons(SERVERPORT);
    sDestAddr.sin_addr.s_addr = inet_addr(strIpAddr);
    /* 连接服务器 */
    if (connect(nConnectSocket, (struct sockaddr*)&sDestAddr, sizeof(sDestAddr)) != 0)
    {
        perror("Connect ");
        exit(errno);
    }
    else
    {
        printf("Connect Success!  \nBegin to chat...\n");
        safeChat(nConnectSocket, strIpAddr, "iloveyou");
    }
    close(nConnectSocket);
}
```

服务端

```
{//be a server
    int nListenSocket, nAcceptSocket;
    socklen_t nLength = 0;
    struct sockaddr_in sLocalAddr, sRemoteAddr;
    if ((nListenSocket = socket(PF_INET, SOCK_STREAM, 0)) == -1)
    {
        perror("socket");
        exit(1);
    }

    bzero(&sLocalAddr, sizeof(sLocalAddr));
    sLocalAddr.sin_family = PF_INET;
    sLocalAddr.sin_port = htons(SERVERPORT);
    sLocalAddr.sin_addr.s_addr = INADDR_ANY;

    if (bind(nListenSocket, (struct sockaddr*)&sLocalAddr, sizeof(struct sockaddr)) == -1)
    {
        perror("bind");
        exit(1);
    }
    if (listen(nListenSocket, 5) == -1)
    {
        perror("listen");
        exit(1);
    }
    printf("Listening...\n");
    nLength = sizeof(struct sockaddr);
    if ((nAcceptSocket = accept(nListenSocket, (struct sockaddr*)&sRemoteAddr, &nLength)) == -1)
    {
        perror("accept");
        exit(errno);
    }
    else
    {
        close(nListenSocket);
        printf("server: got connection from %s, port %d, socket %d\n", inet_ntoa(sRemoteAddr.sin_addr), ntohs(sRemoteAddr.sin_port), nAcceptS
        safeChat(nAcceptSocket, inet_ntoa(sRemoteAddr.sin_addr), "iloveyou");
        close(nAcceptSocket);
```

聊天功能

```
void safeChat(int nSock, char* pRemoteName, char* pKey)
{
    CDesOperator cDes;
    if (strlen(pKey) != 8)
    {
        printf("Key length error");
        return;
    }
    pid_t nPid;
    nPid = fork();
    if (nPid != 0)                    //子线程用于接受信息
    {
        while (1)
        {
            bzero(&strSocketBuffer, BUFFERSIZE);
            int nLength = 0;
            nLength = totalRecv(nSock, strSocketBuffer, BUFFERSIZE, 0);
            if (nLength != BUFFERSIZE)
            {
                break;
            }
            else
            {
                int nLen = BUFFERSIZE;
                cDes.Decry(strSocketBuffer, BUFFERSIZE, strDecryBuffer, nLen, pKey, 8);
                strDecryBuffer[BUFFERSIZE - 1] = 0;
                if (strDecryBuffer[0] != 0 && strDecryBuffer[0] != '\n')
                {
                    printf("Receive message form <%s>: %s\n", pRemoteName, strDecryBuffer);
                    if (0 == memcmp("quit", strDecryBuffer, 4))
                    {
                        printf("Quit!\n");
                        break;
                    }
                }
            }
        }
    }
```

```
else                                                                    //父线程发送消息
{
    while (1)
    {
        bzero(&strStdinBuffer, BUFFERSIZE);
        while (strStdinBuffer[0] == 0)
        {
            if (fgets(strStdinBuffer, BUFFERSIZE, stdin) == NULL)                    //读取一行
            {
                continue;
            }
        }
        int nLen = BUFFERSIZE;
        cDes.Encry(strStdinBuffer, BUFFERSIZE, strEncryBuffer, nLen, pKey, 8);
        if (send(nSock, strEncryBuffer, BUFFERSIZE, 0) != BUFFERSIZE)
        {
            perror("send");
        }
        else
        {
            if (0 == memcmp("quit", strStdinBuffer, 4))
            {
                printf("Quit!\n");
                break;
            }
        }
    }
}
```

3. 实验结果

客户端

```
wtx@ubuntu:/mnt/hgfs/Lab/des-tcp/code$ ./chat
Client or Server?
c
Please input the server address:
127.0.0.1
Connect Success!
Begin to chat...
hi
Receive message from <127.0.0.1>: hi

quit
Quit!
```

服务器端

```
Client or Server?
s
Listening...
server: got connection from 127.0.0.1, port 59038, socket 4
Receive message from <127.0.0.1>: hi

hi
Receive message from <127.0.0.1>: quit
Quit!
```

四、实验遇到的问题及其解决方法

1.所使用的头文件

#include <netdb.h> 文件如其名，包括结构 hostent（主机环境），获得主机的信息的几个函数（gethostbyname）。似乎这个就是定义主机的各项环境，例如 hostname 等等

#include <unistd.h> 提供通用的文件、目录、程序及进程操作的函数

#include <errno.h> 提供错误号 errno 的定义，用于错误处理

#include <stdlib.h> 某些结构体定义和宏定义，如 EXIT_FAILURE、EXIT_SUCCESS 等

#include <sys/types.h> 数据类型定义

#include <netinet/in.h> 定义数据结构 sockaddr_in

#include <sys/socket.h> 提供 socket 函数及数据结构

#include <sys/wait.h> 提供进程等待

#include <arpa/inet.h> 提供 IP 地址转换函数

#include <sys/ioctl.h> 提供对 I/O 控制的函数

五、实验结论

1.DES

了解了 DES 的加密流程，和以前课程实现的 AES 加密进行对照学习，比较了两个加密算法的安全性。

2. TCP

在之前代码的基础上加入线程处理，使得通讯双方能够同时收、发信息