



南開大學
Nankai University

网络空间安全学院
网络安全技术实验报告

实验一：基于 DES 加密的 TCP 聊天
程序

学院：网安学院

年级：2021 级

班级：信息安全一班

学号：2111408

姓名：周钰宸

手机号：13212168838

2024 年 3 月 29 日

目录

1 实验目标	3
2 实验原理	3
2.1 DES	3
2.2 TCP	5
3 实验内容	6
3.1 规范习惯	6
3.2 DES	6
3.3 TCP	7
3.3.1 Server	7
3.3.2 Client	7
3.4 HeadquarterAndAgent	7
3.4.1 特工介绍	8
3.4.2 Special Operations	10
3.4.3 其它功能	10
4 实验步骤	11
4.1 实验模块搭建	11
4.2 DES	17
4.2.1 PaddingAndUnpadding	17
4.2.2 KeyGeneration	18
4.2.3 Encryption	19
4.2.4 Decryption	23
4.3 TCP	24
4.3.1 Server	25
4.3.2 Client	31
4.4 AgentTheme	32
4.4.1 Special Operations	32
4.4.2 其它功能	34
5 实验结果	37
5.1 DES 正确性验证	37
5.2 实验结果展示	38
6 实验遇到的问题及其解决方法	41
6.1 Visual Studio Code	41
6.1.1 规范化模块搭建	41
6.1.2 编译连接问题	41
6.2 DES	42
6.2.1 数据存储	42
6.2.2 矩阵与 vector 存储时间与空间复杂度高	43

6.2.3 填充问题	43
6.3 TCP Channel	43
6.3.1 进程结束不释放资源问题	43
6.3.2 进程通信问题	45
7 实验结论	45

1 实验目标

DES(Data Encryption Standard) 算法是一种用 56 位密钥来加密 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。

本次实验的目的如下：

1. 理解 DES 加解密的原理。
2. 理解 TCP 协议的工作原理。
3. 掌握 linux 下基于 socket 的编程方法。

最终达到的要求为：

1. 利用 socket 编写一个 TCP 聊天程序。
2. 通信内容经过 DES 加密于解密。

本次实验，我按照上述实验要求，并自主创新（花式整活 bushi），完成了一个以特工电影为主题的极其中二但又十分合理的 TCP 聊天加密程序。

2 实验原理

2.1 DES

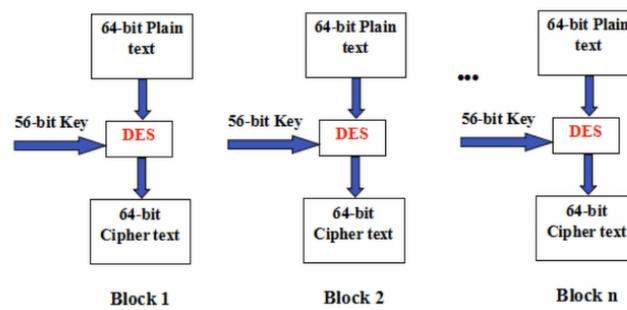


图 2.1: DES 对称加密算法

DES 是应用最广泛的对称加密算法之一，如图2.1所示。它主要用于计算机网络通信、电子资金传送系统以及保护用户文件。

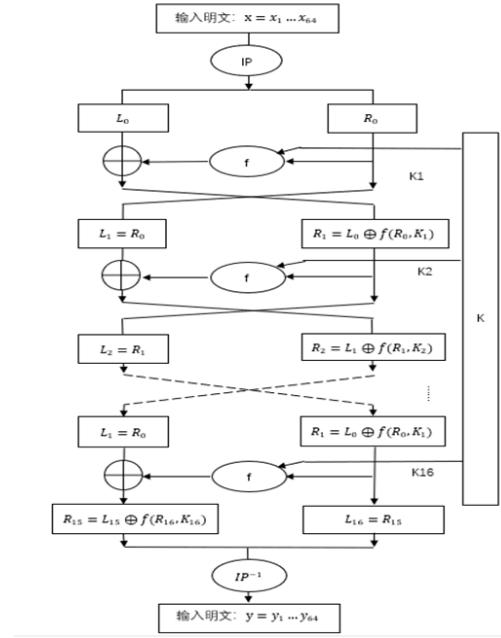


图 2.2: DES 流程 1

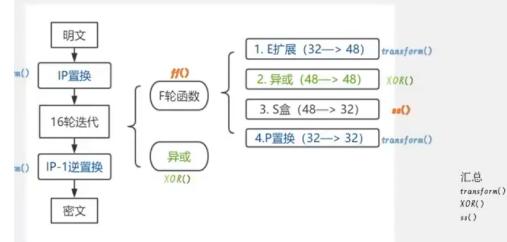


图 2.3: DES 流程 2

如图2.2和2.3所示，DES 明文分组长度为 64bit(不足 64bit 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。初始密钥长度为 64bit，但其中有 8bit 奇偶校验位，因此有效密钥长度是 56bit，子密钥生成算法产生 16 个 48bit 的子密钥，在 16 圈迭代中使用。解密与加密采用相同的算法，并且所使用的密钥也相同，只是各个子密钥的使用顺序不同。

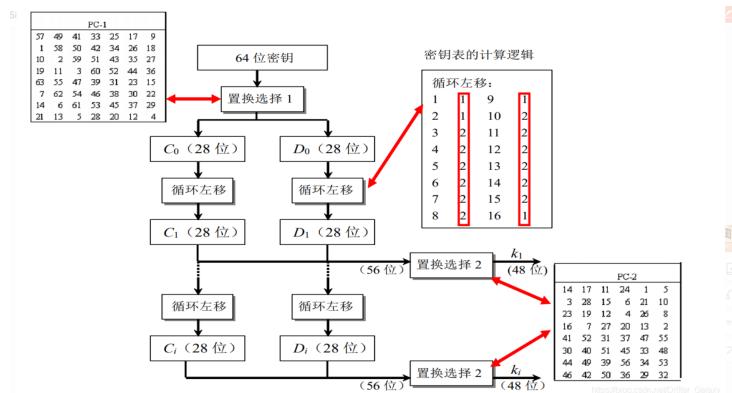


图 2.4: DES 的密钥生成

如图2.4所示，其中 64bit 初始密钥经过置换选择 PC-1、循环左移运算 LS、置换选择 PC-2，产生

16 轮迭代所用到的子密钥 k。初始密钥的第 8、16、24、32、40、48、56、64 位是奇偶校验位，其余 56 位为有效位。

DES 算法的全部细节都是公开的，其安全性完全依赖于密钥的保密。算法包括初始置换、逆初始置换、16 轮迭代以及子密钥生成算法。

2.2 TCP

TCP 是一种传输控制协议，它是一种端对端协议。使用 TCP 通讯的两台主机必须连接在一起，每一端都要为通话做好准备。

套接字 (Socket) 是一个中间软件抽象层，作为一组接口存在。本次实验就是基于 TCP 协议的流式套接字 (SOCK_STREAM)，提供面向连接的、可靠的数据传输服务，可保证无差错、无重复且按发送顺序提交给接收方。

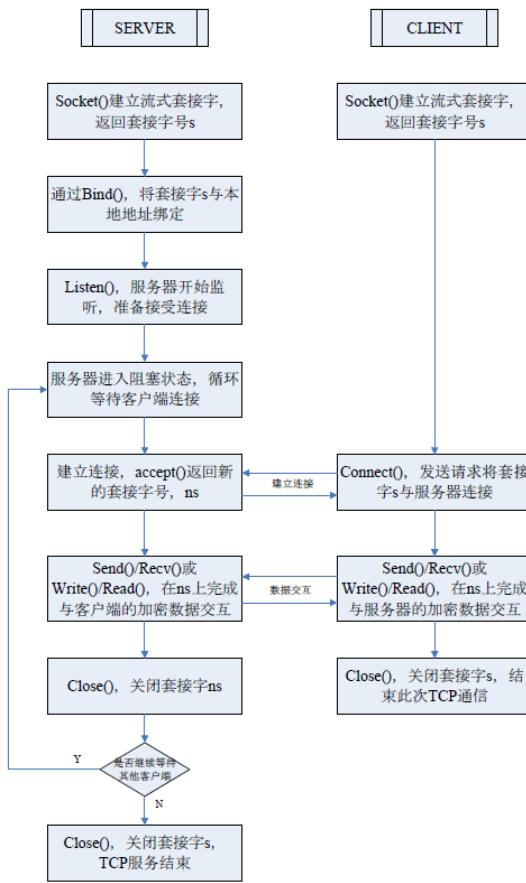


图 2.5: TCP 通信流程图

如2.5所示，具体 Socket 编程步骤为：

1. 服务器端：

- 加载套接字库，创建套接字（使用 `socket()` 函数）。
- 将套接字绑定到一个 IP 地址和端口上（使用 `bind()` 函数）。
- 将套接字设置为监听模式，等待连接请求（使用 `listen()` 函数）。

- 当请求到来时，接受连接请求，返回一个新的通信套接字（使用 accept() 函数）。
- 使用通信套接字与客户端进行数据交换。

2. 客户端：

- 创建一个流式套接字（使用 socket() 函数）。
- 填写服务器的地址和端口信息（使用 connect() 函数）。
- 使用 send() 函数发送数据给服务器。
- 使用 recv() 函数接收服务器返回的数据。

3 实验内容

首先我说明本次实验的主题——特工电影。

这是我在看到本次实验要求将聊天内容加密后第一个想到的创意，也是十分兴奋地投入实践。最终整体实验完成度较高，除了利用 socket 编程和 DES 加解密外，还实现了多进程双全工通信，一次连接多次通信以及一些和特工主题相吻合的功能。具体而言会介绍本次实验的主要工作。

3.1 规范习惯

由于本次实验是在 Linux 下进行编程，与上学期的计算机网路课程的 Windows 下的 Winsock 库略有不同。除此之外，为了锻炼自己，我舍弃了之前的 Visual Studio，而是全程使用 VSCode 作为 IDE 编程，这个过程中加深了我对编译器连接器等的熟悉。

同时本次实验，我严格要求自己，采用了非常规范的工程编程习惯。极为清晰的框架增强了我的代码的可读性和可维护性。这一点也会在具体实验步骤中展开描述。



图 3.6: 本次实验规范编程框架

3.2 DES

在 DES 的实现中，为了尽可能优化代码的时间复杂度，本次实验的 DES 代码全部采用了 C++ 底层的位运算实现。并且代码逻辑模块清晰，可读性较强。具体的模块分为：

1. **Padding 和 Unpadding:** 由于 DES 算法对通信的明文有着 64bit 的要求，因此本次实验对少于 64 位的明文进行加密时，会首先对其末尾添加长度为填充长度的 bit。同样地，在解密的最后一步，进行过 IP 逆置换后，也会通过 unpadding 消除填充的长度，恢复原来的内容。
2. **KeyGeneration:** 分为 16 轮的置换选择，循环左移等。从 64 位的初始密钥，最终生成 16 个 48 位的轮密钥。
3. **Encryption:** 分为 IP 置换，16 轮的 F 函数迭代（拓展 + 异或 +S 盒 +P 置换），IP 逆置换。
4. **Decryption:** 同样分为 IP 置换，16 轮的 F 函数（拓展 + 异或 +S 盒 +P 置换），IP 逆置换。不过 16 轮密钥的使用顺序与 encryption 颠倒。

3.3 TCP

本次实验的 TCP 通讯可以直接使用流式套接字 socket 接口，大体流程框架如图2.5所示，进行了依次实现。

3.3.1 Server

1. **一次登录多次通讯:** Server 端实现了一次通信后的多次通信。具体而言就是在一个 Client 端的客户下线后，能够继续循环监听下一个客户的连接请求并继续保持通讯。不过也有一定的时长限制，超过 5 分钟的等待没有新的连接到来就会自动结束。同时也可以在任意时间结束。
2. **显示连接的 Client 的信息:** 在连接后可以显示登陆的 Client 的 IP 地址，端口和用户名等信息。
3. **正常功能:** 包括初始化网络环境，绑定 socket，监听连接，发送和接收消息等。
4. **全面的错误处理:** 对于各种连接过程的异常情况都进行了处理。
5. **正常退出:** 通过输入 OVER 大写进行退出。

3.3.2 Client

1. **自主输入 IP 地址:** 在向服务器端发起连接前，可以通过选择默认 IP 或者手动输入 IP 的方式进行配置。
2. **正常功能:** 包括初始化网络环境，绑定 socket，监听连接，发送和接收消息等。
3. **全面的错误处理:** 对于各种连接过程的异常情况都进行了处理。
4. **正常退出:** 通过输入 OVER 大写进行退出。

3.4 HeadquarterAndAgent

由于我的主题的特殊性，本次实验实现了诸多相关的有趣功能。这些都是仿照真实的特工对话场景，中二之中又带着一些合理 hh。

3.4.1 特工介绍

我首先围绕了结合著名度和个人喜好挑选了五位真实影视剧存在的特工。他们分别是：

1. 来自电影 007 系列：詹姆斯·邦德 (James Bond)，代号：007。



图 3.7: 007

2. 来自电影谍影重重 (The Bourne Identity) 系列：杰森·伯恩 (Jason Bourne)，真名是大卫·韦伯 (David Webb)。

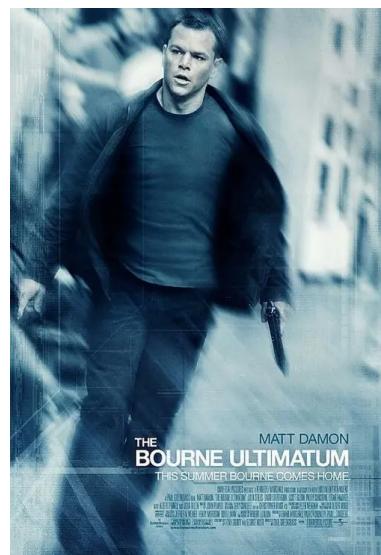


图 3.8: Jason Bourne

3. 来自美国漫威影业 (Marvel Studios) 的复仇者联盟 (Avengers) 系列：黑寡妇 (Black Widow)，真名为娜塔莎·罗曼诺夫 (Natasha Romanoff)。



图 3.9: Black Widow

4. 来自电影碟中谍 (Mission Impossible) 系列: 伊森·亨特 (Ethan Hunt), 这里是个小彩蛋, 由于没有找到该特工的代号, 直接将 Ethan Hunt 作为了代号, 演员真名汤姆·克鲁斯 (Tom Curise) 作为其特工“真名”。

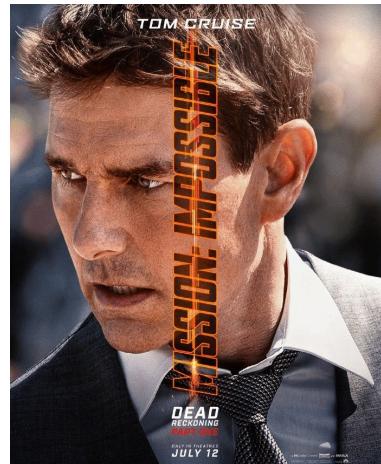


图 3.10: Ethan Hunt

5. 来自电影憨豆特工 (Johnny English) 系列: 强尼·英格力 (Johnny English), 这里同样也是小彩蛋, 直接使用了我们最爱的憨豆先生演员真名罗温·艾金森 (Rowan Atkinson) 作为其特工“真名”。



图 3.11: Johnny English

3.4.2 Special Operations

在我的设定中，特工位于秘密藏身处 (SecretHideout) 中与总部 (Headquarter) 进行对话，总部的长官 (Commander) 取自经典的 007 系列电影的长官 M。故事发生在二者的对话过程中，突然总部和秘密藏身处的系统检测到了敌人的窃听攻击，立马汇报给了长官，将由长官来决定采取的具体应对措施。一共有三种应对措施，其中前两者具有独特的行动代号：

- **Silent Guardian Operation:** 顾名思义，该代号行动意味者将会通信双方继续保持通讯，不过需要将所有通信内容基于加密算法加密。不过加密密钥需要提前商量好，因为总部没有资金实现非对称加密传输密钥。（实际上是我懒得做了hhh。）
- **Phantom Hook Operation:** "Phantom" 一词暗示着一种虚假或虚幻的东西，而"Hook" 则意味着诱饵或诱饵。对应着通信双方的行为就是继续保持正常通话，而不通过加密。假装对敌人的窃听未知，这样可以让敌人放松警惕 (let the guard down)。然后通过透露一些虚假的信息，如下次行动的错误地点，来作为诱饵迷惑敌人。
- **Abort and Destroy:** 顾名思义，总部在检测到敌人的监听攻击，为了防止过多的消息泄露，采用终止并销毁的方式结束通信，以除后患。

3.4.3 其它功能

除此之外，本次实验还在一些比较特殊的地方加了一些和特工主题相关的有趣功能。具体而言：

1. **身份验证**：在进入秘密藏身处和总部前，会对特工和长官的身份进行验证。其中特工必须输入某个记录在案的特工真实姓名才可以进入系统。这一点符合实际。因为特工的代号可能会暴露给敌人，但是特工的真实身份往往是高度机密，只有极少数重要的高层人员和特工本人才知道。因此可以作为身份验证的合理要求。
2. **错误处理**：不论是特工所在的秘密藏身处还是指挥官所在的总部，当出现通信问题如 socket 错误，网络不稳定等，都会由指挥官决定后续处理，可以选择重启系统通信或者是终止任务并防止潜在的敌人攻击。
3. **总部多负载通信**：由于总部需要管辖内部的多个特工，因此需要长期在线，不能轻易断电。因此实现了一个特工下线后自动等待下一个特工连入的连续通信功能。同时如果长期没有特工连入，总部也会自动下线也节省能源。
4. **虚假身份检测**：正如上一个功能所述，由于总部可以连续对多个特工对话，并且有望实现多特工同时通信（本次实验还没实现）。因此在特工同时通信时，如果在一个已经实名通信的用户还未退出时，另一个同样身份的特工进入，那么后者很有可能是敌人假扮的虚假特工。此时系统会检测在多个特工通信时，重复出现的相同身份特工。由总部通知秘密藏身处，并解除其通信。不过解除通信时候不会捅破窗户纸，秘密藏身处系统会假装只是信号紊乱导致的通信结束。同样目的还是让敌人放松警惕 (let the guard down)，然后快速派遣特工抓捕在秘密藏身处附近的敌人。
5. **代号匹配**：系统自动记录并匹配特工的代号，用于身份验证和通信。

4 实验步骤

4.1 实验模块搭建

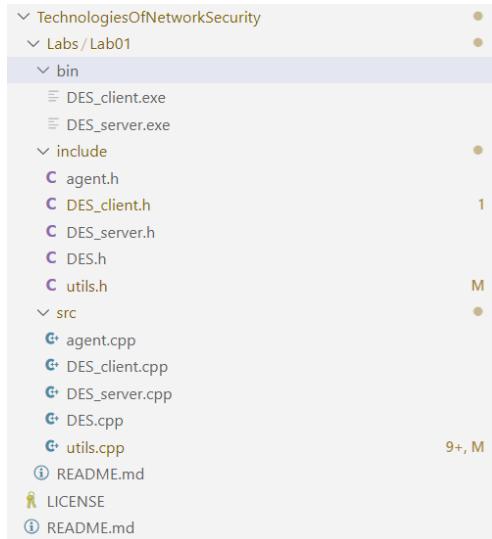


图 4.12: 本次实验模块讲解

如图4.12所示，本次实验搭建了一个相对非常完善且规整的代码框架，具有较高的可读性和可维护性。具体而言：

1. **bin:** 存在最终的可执行文件即 DES_Server.exe 和 DES_Client.exe。

- 服务器端：DES_Server.exe
- 客户端：DES_Client.exe

2. **include:** 包含了一系列进行常量声明的头文件。

- **agent.h:** 包含定义的 Agent 类，用于后续操作存储 Agent 对象。会在 Server 端也就是总部端通过 vector 进行存储，方便多特工同时通信时进行管理（但暂未实现多特工同时通信）。

```

1 #include <iostream>
2 #include <limits>
3 #include <string .h>
4 #include <netinet/in .h>
5 using namespace std;
6 /**
7 * Agent class
8 * This class is used to store the information of the agent
9 * including the agent's name, code name, and socket
10 */
11 class Agent
12 {
13 private:
14     string agentName;
15     string agentCodeName;

```

```

16     int agentSocket;
17
18 public:
19     Agent();
20     Agent(string name, string codeName, int socket);
21     ~Agent() {};
22     string getAgentName();
23     void setAgentName(string name);
24     string getAgentCodeName();
25     void setAgentCodeName(string codeName);
26     int getAgentSocket();
27     void setAgentSocket(int socket);
28 };

```

- **DES.h:** 定义了将要使用的 DES 算法的工具类 **DESUtil**。其中包含了用于加解密和密钥生成的各个常量，以及一些工具函数。都通过了相对解耦的接口实现，增强了代码的可维护性和鲁棒性。

```

1  typedef int INT32;
2  #include <cstdint>
3  #include <bitset>
4  #include <cstring>
5  #include <iostream>
6  #include <vector>
7  #include <assert.h>
8  using namespace std;
9  #define SUCCESS 1
10 #define FAILURE 0
11 class DESUtils
12 {
13     public:
14         const uint8_t ip[64] = {58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20,
15             12, 4,
16                 62, 54, 46, 38, 30, 22, 14, 6, 64, 56, 48, 40, 32, 24,
17                     16, 8,
18                     57, 49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35, 27, 19, 11,
19                         3,
20                         61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23,
21                             15, 7};
22     /**
23      * 其它具体DES算法所需的Box等常量，这里省略了展示
24      * 加密与解密: const uint8_t ip_reverse[64], const uint8_t SBox[8][64], const
25          uint8_t EBox[48], const uint8_t PBox[32]
26      * 密钥生成: const uint8_t PC1[56], const uint8_t shiftBits[16], const uint8_t
27          PC2[48]
28      */
29     // 56 bits × 16 stored in 16 Rounds
30     uint64_t roundKeys[16];
31     // Generate Key for DES, using the first 56 bits of the input key

```

```

26    INT32 permutationKey(uint64_t *key, int mode);
27    INT32 shiftKey(uint32_t *key, int round);
28    // Formal DES Algorithm for 16 rounds
29    INT32 padding(vector<uint8_t> &data);
30    INT32 unpadding(vector<uint8_t> &data);
31    INT32 initialPermutation(uint64_t *data);
32    INT32 initialPermutationReverse(uint64_t *data);
33    INT32 expansionBox(uint32_t *data, uint64_t &output);
34    INT32 extractionBox(uint64_t *data, uint32_t &output);
35    INT32 permutation(uint32_t *data);
36    INT32 roundHandler(uint32_t *leftData, uint32_t *rightData, uint64_t *roundKey);
37 public:
38    // Constructor
39    DESUtils();
40    // Destructor
41    ~DESUtils();
42    // Encrypt the plaintext
43    void encrypt(const char *plaintext, char *ciphertext);
44    // Decrypt the ciphertext
45    INT32 genKey(uint64_t *key);
46    void decrypt(const char *ciphertext, char *plaintext);
47 };

```

3. utils.h: 主要定义了一些辅助的工具函数和工具常量。其中比较重要的有：

- string timeNow(): 用于**显示当前时间来进行展示**。
- void signalHandler(int sig): 用于进程之间的通信，在聊天需要结束时，由一个进程通过信号通知另一个进程。由该函数进行处理。
- ssize_t TotalRecv(int s, void *buf, size_t len, int flags): 借鉴了实验手册中的代码，用于在**网络不稳定时完整地就接收发送的内容**。这是因为 TCP 在某些特殊情况下（例如链路质量变化），recv() 一次并不能返回对方发送的全部数据，需要多次调用 recv() 才能获得全部数据，使用该函数是为了确保整个数据块可以被顺利接收。
- void PhantomHook(int role, Agent agent): 用于**不加密的公开信道交流**。通过 Agent 参数传输特工代号和 socket 等，通过 role 分为特工和总部两个不同的身份进行不同的控制流。
- void SilentGuardian(int role, Agent agent, DESUtils des): 用于**DES 算法加密的公开信道交流**。**密钥通过硬编码提前约定**。通过 Agent 参数传输特工代号和 socket 等，通过 role 分为特工和总部两个不同的身份进行不同的控制流。通过 DESUtils 参数传输对应的加密工具类。

```

1 #include <iostream>
2 #include <set>
3 #include <sstream>
4 #include <iomanip>
5 #include <sys/socket.h>
6 #include <chrono>
7 #include <unistd.h>

```

```

8 #include <signal.h>
9 #include <stdlib.h>
10 #include "agent.h"
11 #include "DES.h"
12 using namespace std;
13 enum
14 {
15     M16 = 0,
16     AGENT = 1
17 };
18 void printBits64(uint64_t num);
19 void printBits32(uint32_t num);
20 string stringBits64(uint64_t num);
21 string stringBits32(uint32_t num);
22 string timeNow(); // Get current time to
                    display
23 void signalHandler(int sig); // Handle signal between
                                process to control them
24 ssize_t TotalRecv(int s, void *buf, size_t len, int flags); // Receive data from
                    socket, in case of partial data out of unstable communication
25 void PhantomHook(int role, Agent agent); // Special Operation:
                    Communication between Agent and Headquarter without DES encryption to lure the
                    enemy
26 void SilentGuardian(int role, Agent agent, DESUtils des); // Special Operation:
                    Communication between Agent and Headquarter with DES encryption to protect the
                    secret

```

4. **DES_Server.h:** 其中包含了最重要程序部分之一, TCP 通信的服务器端即 Headquarter 端。这里重点提到了一些有意思或者重要的常量, 有一部分是小彩蛋哦:

- **char *key = "YCZhouNB":** 由于 DES 密钥暂时无法通过非对称加密算法传输 (没有实现), 因此采用硬编码的形式提前写入。**这里选择的是我的名字缩写 ZhouYuChen。正好 8 个 char64 位, 想不稻吧。-(小小自恋一下 hhh)-**
- define MAX_CONNECTION 5: **这里为了总部遭受 DDos 攻击**, 因此设定最大连接数为 5。避免同时多个特工连接总部导致总部瘫痪。
- define SERVER_PORT 8007: 端口取自著名电影 **007 系列**。
- map<string, string> agentConfidential: 存储特工真名和代号的对应关系。
- void emergencyResponse(): 对错误处理的统一处理函数。
- void handle_sigint(int sig): **用于控制主进程监听的随时结束, 并且释放对应的资源。详见**

```

1 #include <iostream>
2 #include <limits>
3 #include <string.h>
4 #include <mutex>
5 #include <map>
6 #include <signal.h>

```

```
7 #include <sys/socket.h> // Provides system calls and data structures for sockets
8 #include <netinet/in.h> // Provides data structures for Internet address family
9 #include <arpa/inet.h> // Provides IP address conversion functions
10 #include <unistd.h> // Provides general system call functions
11 #include <netdb.h> // Provides domain name resolution functions
12 #include <sys/wait.h> // Provides waitpid() function
13 #include "utils.h" // utils.h is already includes DES.h and agent.h
14 using namespace std;
15 /**
16  * In case the M16 headquarter is under a DDoS attack
17  * There will be a maximum of 5 connection requests
18 */
19 #define MAX_CONNECTION 5
20 #define SERVER_PORT 8007
21
22 map<string, string> agentConfidential;
23 string agentNames[] = {"James Bond", "David Webb", "Natasha Romanoff", "Tom Cruise",
24 "Rowan Atkinson"};
25 string agentPasswords[] = {"007", "Jason Bourne", "Black Widow", "Ethan Hunt",
26 "Johnny English"};
27
28 /* DES parameters */
29 DESUtils des;
30 char *key = "YCZhouNB"; // Hardcoded key
31 uint64_t numKey; // Key in hexadecimal
32 char cipheredtext[64];
33
34 /* Socket parameters */
35 string commanderName;
36 int sListenSocket;
37 int sAcceptSocket;
38 sockaddr_in serverAddr;
39 sockaddr_in agentAddr;
40 int instruction;
41 char agentCodeName[64];
42 // vector to store the agents
43 vector<Agent> agents;
44 pid_t nPid;
45 // Mutex for the agents
46 mutex agentsMtx;
47 // Timeout for the socket
48 struct timeval timeout;
49
50 // Handling emergency for the command center
51 void emergencyResponse();
52 // Handling the signal of Ctrl+C to resolve the Parent Process holding socket WHILE
53 // dying issue
54 void handle_sigint(int sig);
```

5. **DES_Client.h:** 也包含了最重要程序部分之一，TCP 通信的客户端即 Agent 端。这里重点提到了一些有意思或者重要的常量：

- define SERVER_PORT 8007
- char *key = "YCZhouNB"
- string serverIP = "192.168.126.128" 与 int choice: 实现采用默认 IP 地址或者手动输入 IP 地址的形式连接客户端。

```

1 #include <iostream>
2 #include <limits>
3 #include <string.h>
4 #include <chrono>
5 #include <sys/socket.h> // Provides system calls and data structures for sockets
6 #include <netinet/in.h> // Provides data structures for Internet address family
7 #include <arpa/inet.h> // Provides IP address conversion functions
8 #include <unistd.h> // Provides general system call functions
9 #include <netdb.h> // Provides domain name resolution functions
10 #include "utils.h" // utils.h is already includes DES.h and agent.h
11 using namespace std;
12 #define SERVER_PORT 8007
13 #define BUFFERSIZE 1024
14
15 /* DES parameters */
16 DESUtils des;
17 char *key = "YCZhouNB";
18 uint64_t numKey;
19 char decryptedtext[64];
20
21 /* socket parameters */
22 string agentName;
23 string agentCodeName;
24 string serverIP = "192.168.126.128";
25 int agentSocket;
26 struct sockaddr_in serverAddr;
27
28 int choice; // choice for the user to select the default IP address or input the IP
29   address manually
30 char strSocketBuffer[BUFFERSIZE];

```

6. **src:** 包含了一系列对 include 中头文件中声明的函数进行的实现的 cpp 文件。详细实现见下面具体部分，这里只简单列出各个文件的含义和做了什么：

- **agent.cpp:** 一些 agent 类的构造函数，采用公访私的方式操作对象成员变量等函数实现。
- **DES.cpp:** 对 DES 算法的各个函数接口进行了实现，具体实现详见后面。
- **utils.cpp:** 主要对各个调试使用的函数接口，以及最重要的几个包括 **totalRecv**, **Silent-Guardian** 和 **PhantomHook** 在内的 DES 算法应用的 TCP 通信函数的实现。

- **DES_Server.cpp**: 对服务器端也就是 Headquarter 一侧的 main 函数之中的一些初始化，TCP 通信初始化等的实现。
- **DES_Cilent.cpp**: 对客户端也就是 Client 一侧的 main 函数之中的一些初始化，TCP 通信初始化等的实现。

4.2 DES

4.2.1 PaddingAndUnpadding

由于 DES 算法需要对明文有 64bit 的要求，因此在不足 64bit 的情况下，必要时要进行填充操作。具体而言，就是在其数据后填充 padding 长度，以两位十六进制形式。直到填充到 64 的整数倍。

Unpadding 时，只要不是一些 data 为空等的特殊情况，就先获取 Padding 后在数据尾部留存的填充长度，然后对应通过一个 resize 函数直接对 vector 的 data 操作，去除末尾的 padLength 长即可。

代码如下：

```

1 INT32 DESUtils::padding(vector<uint8_t> &data)
2 {
3     /**
4      * @brief Padding the data to be a multiple of 64 bits (8 bytes)
5      * @param data: the data to be padded, modified in place
6      * @return: the result of padding
7      */
8     size_t blockSize = 8; // Block size in bytes (64 bits)
9     size_t padLength = blockSize - (data.size() % blockSize);
10
11    // If the data size is already a multiple of the block size, add a new block of
12    // padding
13    if (padLength == 8)
14        padLength = 0;
15    for (size_t i = 0; i < padLength; i++)
16        data.push_back(static_cast<uint8_t>(padLength));
17    return SUCCESS;
18 }
19 INT32 DESUtils::unpadding(vector<uint8_t> &data)
20 {
21     /**
22      * @brief Unpadding the data to be a multiple of 64 bits (8 bytes)
23      * @param data: the data to be unpadded, modified in place
24      * @return: the result of unpadding
25      */
26     size_t blockSize = 8; // Block size in bytes (64 bits)
27     if (data.size() == 0)
28         // If the data is empty, there is no need to unpad
29         return SUCCESS;
30     size_t padLength = data.back();
31     if (padLength > blockSize || padLength > data.size())

```

```

31     // If the pad length is greater than the block size , return the data without
32     // unpadding
33     return SUCCESS;
34     data.resize(data.size() - padLength);
35     return SUCCESS;
}

```

4.2.2 KeyGeneration

- 位移 ShiftKey: 这里为了尽可能地优化时间复杂度，采用了位运算的形式。具体而言就是根据 shiftBits 这个常量 Box 来前后拼接，以此实现循环位移的效果。并根据不同的轮数 round 移动不同位数。

```

1 INT32 DESUtils::shiftKey(uint32_t *key, int round)
2 {
3     /**
4      * @brief Shift the leftKey or the rightKey based on different round
5      * param key: the key to be shifted
6      * param round: one of 16
7      */
8     *key = (*key << shiftBits[round - 1]) | (*key >> (28 - shiftBits[round - 1]));
9     *key &= 0xFFFFFFFF;
10    return SUCCESS;
11 }

```

- 置换 PermutationKey: 通过 mode 参数控制是使用 PC1 还是 PC2，具体也是通过位运算来优化。只需要将所需要的位移动到结尾，异或 0x1，再挪回去即可实现。

```

1 INT32 DESUtils::permutationKey(uint64_t *key, int mode)
2 {
3     /**
4      * @brief Permute the key
5      * param key: the key to be permuted
6      * param mode: 1 for using PC1, 2 for using PC2
7      */
8     uint64_t outputKey = 0;
9     switch (mode)
10    {
11        case 1:
12            // Permutate the key using PC1(64bits to 56bits)
13            for (int i = 0; i < 56; i++)
14                outputKey |= ((*key >> (64 - PC1[i])) & 0x1) << (55 - i);
15            break;
16        case 2:
17            // Choose 48bits from 56bits using PC2
18            for (int i = 0; i < 48; i++)
19                outputKey |= ((*key >> (56 - PC2[i])) & 0x1) << (47 - i);
20            break;
}

```

```

21     }
22     *key = outputKey;
23     return SUCCESS;
24 }
```

3. 具体密钥生成算法 `genKey`: 通过最开始的 `uint64_t` 的 64 位 key, 经过最开始的 PC1 后取出左右部分, 进行 16 轮的操作。其中通过调用已经实现好的接口 `permutationKey` 进行置换, `shiftKey` 进行位移。最终结果保存在 `roundKeys` 中。

```

1 INT32 DESUtils::genKey(uint64_t *key)
2 {
3     /**
4      * @brief Generate 16 subkeys for DES
5      */
6     uint64_t tempKey = *key;
7     // First step: Permute the key using PC1 from 64bits to 56bits
8     permutationKey(&tempKey, 1);
9     // Second step: Generate 16 subkeys
10    uint32_t leftKey = tempKey >> 28;
11    uint32_t rightKey = tempKey & 0xffffffff;
12    for (int i = 1; i <= 16; i++)
13    {
14        shiftKey(&leftKey, i);
15        shiftKey(&rightKey, i);
16        uint64_t subKey = (uint64_t(leftKey) << 28) | rightKey;
17        permutationKey(&subKey, 2);
18        subKey <= 16; // Shift the subKey 48 bits to the high 48 bits
19        roundKeys[i - 1] = subKey;
20    }
21    return SUCCESS;
22 }
```

4.2.3 Encryption

1. 初始置换 IP: 也是通过位运算来实现性能优化。实现起来也相对简单, 其中用到了常量 ip 盒。还是挪至最低为通过异或 0x1 取出值再挪动回去。

```

1 INT32 DESUtils::initialPermutation(uint64_t *data)
2 {
3     /**
4      * @brief The first step of DES encryption using IP, before 16 rounds
5      * param data: the data to be permuted
6      * return: the result of permutation
7      */
8     uint64_t output = 0;
9     for (int i = 0; i < 64; i++)
10        output |= (*data >> (64 - ip[i]) & 0x1) << (63 - i);
11     *data = output;
```

```

12     return SUCCESS;
13 }
```

2. 拓展盒 **expansionBox**: 同样用了位运算优化性能。使用 EBox 常量进行位移即可，位移原理类似这里不再赘述。通过 EBox 中的 48 个 32 位数将高 48 进行成功拓展，最后通过 static_cast 转换使用 uint64_t 存储。

```

1 INT32 DESUtils::expansionBox(uint32_t *data, uint64_t &output)
2 {
3     /**
4      * @brief Expansion Box used for roundHandler to expand the data from 32bits to
5      *        48bits
6      * @param data: right data with 32bits waiting to be expanded in 48bits
7      * @param output: the expanded data saved in 64 bits
8      * @return: the result of the expansion
9      */
10    for (int i = 0; i < 48; i++)
11    {
12        output |= (static_cast<uint64_t>((*data >> (32 - EBox[i])) & 0x1) << (63 -
13            i));
14    }
15    return SUCCESS;
16 }
```

3. S 盒压缩：同样使用位运算实现，不过原理较为复杂。S-Box 是 DES 算法中的一个重要部分，它用于将 48 位的数据压缩回 32 位。

在函数中，首先进行一个循环，循环次数为 8，因为 DES 算法中有 8 个 S-Box。在每次循环中，首先从 data 中提取 6 位数据，存储在 subDataBits 中。提取的方法是先将 data 右移 $(16+6 \times i)$ 位，然后与 0x3F（即二进制的 111111）进行位与操作，这样就可以得到 data 的 6 位子数据。但是最开始提取的是最低的 6 位，因此下一步要用最后一个 SBox。

因此接下来逻辑是 subDataBits 作为索引，从 SBox[7 - i] 中查找对应的 4 位数据。这 4 位数据再左移 $(4 \times i)$ 位，然后与 output 进行位或操作，将结果存储回 output 中。具体代码如下：

```

1 INT32 DESUtils::extractionBox(uint64_t *data, uint32_t &output)
2 {
3     /**
4      * @brief Extraction Box used for roundHandler to extract data from 48bits back
5      *        to 32bits
6      * param data: the xored data with 48bits waiting to be extracted to 32bits
7      * return: the result of the extraction
8      */
9    for (int i = 0; i < 8; i++)
10    {
11        bitset<6> subDataBits((*data >> (16 + 6 * i)) & 0x3F); // Store 6 bits in a
12            bitset
13        output |= static_cast<uint32_t>((SBox[7 - i][subDataBits.to_ulong()] | 0x0)
14            << (4 * i));
```

```

12     }
13     return SUCCESS;
14 }
```

4. 置换 permutation: 轮函数的最后一步，进行 32 位的置换。同样通过位运算实现，逻辑类似不再赘述。

```

1 INT32 DESUtils::permutation(uint32_t *data)
2 {
3     /**
4      * @brief 4st and the last step of the roundHandler
5      * param data: the data to be permuted
6      * return: the result of the permutation
7      */
8     uint32_t output = 0;
9     for (int i = 0; i < 32; i++)
10        output |= ((*data >> (32 - PBox[i])) & 0x1) << (31 - i);
11     *data = output;
12     return SUCCESS;
13 }
```

5. 轮函数 roundHandler: 十六轮的轮函数，通过依次调用 expansionBox，与 roundKey 异或，extractionBox 和 permutation 实现一轮的处理。最后交换对应的左右部分，并进行异或。

```

1 INT32 DESUtils::roundHandler(uint32_t *leftData, uint32_t *rightData, uint64_t
2     *roundKey)
3 {
4     /**
5      * @brief Handler for 16 rounds of encryption for DES
6      * param leftData:Li-1 for each round 32 bits
7      * param rightData:Ri-1 for each round 32 bits
8      * param roundKey: one of 16 roundKey for the current round
9      * return: the Li and Ri that will be facilitated for next round
10     */
11     // f function for this round
12     uint64_t expandedData = 0;
13     expansionBox(rightData, expandedData);
14     expandedData ^= *roundKey;
15     uint32_t extractedData = 0;
16     extractionBox(&expandedData, extractedData);
17     permutation(&extractedData);
18     uint32_t tempLeftData = *leftData;
19     *leftData = *rightData;
20     *rightData = tempLeftData ^ extractedData;
21     return SUCCESS;
22 }
```

6. 逆置换 initialPermutationReverse: DES 加密的最后一步，同样用了位运算。逻辑也类似 IP

置换，不过是使用了常量 ip_reverse 这个 Box，不再重复赘述。

```

1 INT32 DESUtils::initialPermutationReverse(uint64_t *data)
2 {
3     /**
4      * @brief The last step of DES encryption using IP^-1, after 16 rounds
5      * param data: the data to be permuted
6      * return: the result of permutation
7      */
8     uint64_t output = 0;
9     for (int i = 0; i < 64; i++)
10        output |= (*data >> (64 - ip_reverse[i]) & 0x1) << (63 - i);
11     *data = output;
12     return SUCCESS;
13 }
```

7. 加密函数 encrypt：这里是正式的 DES 加密函数。通过读入 char 类型的 plaintext 并转换为 char 类型的 ciphertext。主要分为以下几步：

- 将明文按照 64 位分组，并转换为 vector<uint8_t>
- 调用 padding(plaintextVec) 进行填充
- 依次取出每个 64 位 data，然后开始调用 initialPermutation，16 轮的 roundHandler，以及最后的 initialPermutationReverse。**加密的主体部分。**
- 最后通过位移形式将 64 位的 bit 恢复为 char 类型的密文。

代码如下：

```

1 void DESUtils::encrypt(const char *plaintext, char *ciphertext)
2 {
3     /**
4      * @brief Encrypt the plaintext using DES
5      * param plaintext: the plaintext to be encrypted
6      * param key: the key for encryption
7      * param ciphertext: the result of encryption
8      */
9     // Divide the plaintext into 64bits in a group
10    // Convert the plaintext from a char array to a vector of uint8_t for further
11    // processing
12    vector<uint8_t> plaintextVec(plaintext, plaintext + strlen(plaintext));
13    // Apply padding to the plaintext vector to ensure its size is a multiple of the
14    // block size
15    padding(plaintextVec);
16    int length = plaintextVec.size();
17    for (int j = 0; j < length; j += 8)
18    {
19        uint64_t data = 0;
20        int blockLength = (j + 8 < length) ? 8 : length - j; // Determine the length
21        // of the current block
22        initialPermutation(data);
23        for (int i = 0; i < 16; i++)
24            data = roundHandler(data, key[i]);
25        initialPermutationReverse(data);
26        for (int i = 0; i < 8; i++)
27            ciphertext[j + i] = (char)(data >> (63 - i) & 0x1);
28    }
29 }
```

```

19     for (int i = 0; i < blockLength * 8; i++)
20         data |= ((uint64_t)(plaintextVec[j + i / 8] >> (7 - (i % 8))) & 0x1) <<
21             (63 - i);
22
23     uint64_t tempData = 0;
24     // IP
25     initialPermutation(&data);
26     uint32_t leftData = data >> 32;
27     uint32_t rightData = data & 0xffffffff;
28     for (int i = 0; i < 16; i++)
29     {
30         // 16 rounds
31         roundHandler(&leftData, &rightData, &roundKeys[i]);
32         if (i == 15)
33             swap(leftData, rightData);
34         tempData = ((uint64_t)leftData << 32) | rightData;
35     }
36     data = tempData;
37     // IP^-1
38     initialPermutationReverse(&data);
39     // Convert the 64-bit data back to char array
40     for (int i = 0; i < blockLength; i++)
41         ciphertext[j + i] = (char)((data >> (56 - 8 * i)) & 0xFF);
42     return;
43 }
```

4.2.4 Decryption

DES 的解密部分与加密部分几乎完全一致。只有两点不同：

- 最开始不进行 unpading，在整体解密结束后进行 unpading。
- 采用的密钥为相反顺序，也就是第一次先使用第十六轮密钥，第 16 次使用第一轮密钥。

代码如下：

```

1 void DESUtils::decrypt(const char *ciphertext, char *plaintext)
2 {
3     /**
4      * @brief Decrypt the ciphertext using DES
5      *        The only difference between encryption and decryption is the order of
6      *        the roundKeys
7      * param ciphertext: the ciphertext to be decrypted
8      * param key: the key for decryption
9      * param plaintext: the result of decryption
10     */
11     // Divide the ciphertext into 64bits in a group
12     // Convert the ciphertext from a char array to a vector of uint8_t for further
13     // processing
```

```
12     vector<uint8_t> ciphertextVec(ciphertext, ciphertext + strlen(ciphertext));
13     int length = ciphertextVec.size();
14     for (int j = 0; j < length; j += 8)
15     {
16         uint64_t data = 0;
17         int blockLength = (j + 8 < length) ? 8 : length - j; // Determine the length
18         // of the current block
19         for (int i = 0; i < blockLength * 8; i++)
20             data |= ((uint64_t)(ciphertext[j + i / 8] >> (7 - (i % 8))) & 0x1) << (63
21             - i);
22         uint64_t tempData = 0;
23         // IP
24         initialPermutation(&data);
25         uint32_t leftData = data >> 32;
26         uint32_t rightData = data & 0xffffffff;
27         for (int i = 0; i < 16; i++)
28         {
29             // 16 rounds
30             roundHandler(&leftData, &rightData, &roundKeys[15 - i]);
31             if (i == 15)
32                 swap(leftData, rightData);
33             tempData = ((uint64_t)leftData << 32) | rightData;
34         }
35         data = tempData;
36         // IP^-1
37         initialPermutationReverse(&data);
38         // Convert the 64-bit data back to char array
39         for (int i = 0; i < blockLength; i++)
40             plaintext[j + i] = (char)((data >> (56 - 8 * i)) & 0xFF);
41     }
42     // Unpadding the plaintext
43     vector<uint8_t> plaintextVec(plaintext, plaintext + strlen(plaintext));
44     unpadding(plaintextVec);
45     for (int i = 0; i < plaintextVec.size(); i++)
46         plaintext[i] = plaintextVec[i];
47     plaintext[plaintextVec.size()] = '\0';
48     return;
49 }
```

4.3 TCP

TCP 部分主要负责程序的主体框架，包括初始化，socket 绑定连接等。

4.3.1 Server

1. 处理异常情况接口函数 **emergencyResponse**: 主要提供 Commander 在各种异常情况下的处理选择。

```
1 void emergencyResponse()
2 {
3     /**
4      * This is a function to handle emergency for the command center
5      * @return void
6      */
7     string str_time = timeNow();
8     cout << "<Headquarter::System @" + str_time + "# Message>:We must have been
9         compromised!" << endl;
10    cout << "<Headquarter::System @" + str_time + "# Message>:Commander, please
11        give us further instructions!" << endl;
12    cout << "1:Abort the system and destroy all files, NOW!" << endl;
13    cout << "2:Reboot the system." << endl;
14    while (true)
15    {
16        cin >> instruction;
17        if (cin.fail())
18        {
19            cin.clear();
20            cin.ignore(numeric_limits<streamsize>::max(), '\n');
21            cout << "<Headquarter::System @" + str_time + "# Message>:Invalid
22                input. Please enter a number." << endl;
23            continue;
24        }
25        switch (instruction)
26        {
27            case 1:
28                cout << "<Headquarter::System @" + str_time + "# Message>:System
29                    aborted and files all in ash!" << endl;
30                exit(0);
31                break;
32            case 2:
33                cout << "<Headquarter::System @" + str_time + "# Message>:Yes,
34                    Commander!" << endl;
35                return;
36                break;
37            default:
38                cout << "<Headquarter::System @" + str_time + "# Message>:Sorry, sir! I
39                    am not following you." << endl;
40                break;
41        }
42    }
43 }
```

2. main 函数主体：接下来全都在 main 函数中。

- 初始化：首先是对一些变量进行初始化，这里 `signal(SIGINT, handle_sigint)` 是在处理特殊情况用于紧急停止，作用详见 6.3.1。

另外 `agentConfidential` 存储特工代号和真名的对应关系，这里也是进行初始化。同时是进行 16 轮密钥的生成等。

```

1 // Initializations
2 signal(SIGINT, handle_sigint);
3 timeout.tv_sec = 300; // After 300 seconds, the socket will be closed
4 automatically
5 timeout.tv_usec = 0;
6 numKey = 0;
7 for (int i = 0; i < 8; ++i)
8 {
9     numKey <= 8;
10    numKey |= static_cast<unsigned char>(key[i]);
11 }
12 des.genKey(&numKey);
13 memset(cipheredtext, 0, 64);
14 memset(agentCodeName, 0, 64);
15 // Agent's confidential information
16 for (int i = 0; i < sizeof(agentNames) / sizeof(agentNames[0]); i++)
17     agentConfidential[agentPasswords[i]] = agentNames[i];

```

- TCP 流程：下面是正式的 TCP 流程，依次进行 `socket` 创建，`socket` 超时设定，`bind` 绑定 `socket`，`listen` 监听连接请求。其中任何一步出现问题都可以回到最初的 REBOOT 位置重开。通过调用 `emergencyResponse` 接口。

```

1 REBOOT:
2     string str_time1 = timeNow();
3     cout << "<Headquarter::System @ " + str_time1 + "# Message>:Welcome back ,
4         Commander M!" << endl;
5     cout << "-----Commander M-----" << endl;
6     cout << "<Headquarter::System @ " + str_time1 + "# Message>:System rebooting..." 
7         << endl;
8     cout << "-----Creating Socket-----" << endl;
9     // Create a socket
10    sListenSocket = socket(AF_INET, SOCK_STREAM, 0);
11    if (sListenSocket < 0)
12    {
13        cout << "<Headquarter::System @ " + str_time1 + "# Message>:Oops! There is a
14            fatal error in creating socket!" << endl;
15        emergencyResponse();
16        // If the preceding function returns, the system will be rebooted
17        goto REBOOT;
18    }
19    cout << "<Headquarter::System @ " + str_time1 + "# Message>:Successfully
20        creating socket!" << endl;

```

```

17     if (setsockopt(sListenSocket, SOL_SOCKET, SO_RCVTIMEO, (char *)&timeout,
18         sizeof(timeout)) < 0)
19     {
20         cout << "<Headquarter::System @ " + str_time1 + "# Message>:Oops! There is a
21             fatal error in setting socket!" << endl;
22         emergencyResponse();
23         // If the preceding function returns, the system will be rebooted
24         goto REBOOT;
25     }
26
27     if (setsockopt(sListenSocket, SOL_SOCKET, SO_SNDDTIMEO, (char *)&timeout,
28         sizeof(timeout)) < 0)
29     {
30         cout << "<Headquarter::System @ " + str_time1 + "# Message>:Oops! There is a
31             fatal error in setting socket!" << endl;
32         emergencyResponse();
33         // If the preceding function returns, the system will be rebooted
34         goto REBOOT;
35     }
36
37     // Bind the socket to the address
38     cout << "-----Binding Socket-----" << endl;
39     serverAddr.sin_family = AF_INET;
40     serverAddr.sin_port = htons(SERVER_PORT);
41     serverAddr.sin_addr.s_addr = htonl(INADDR_ANY);
42     if (bind(sListenSocket, (sockaddr *)&serverAddr, sizeof(struct sockaddr)) < 0)
43     {
44         cout << "<Headquarter::System @ " + str_time1 + "# Message>:Oops! There is a
45             fatal error in binding socket!" << endl;
46         emergencyResponse();
47         // If the preceding function returns, the system will be rebooted
48         goto REBOOT;
49     }
50     string str_time2 = timeNow();
51     cout << "<Headquarter::System @ " + str_time2 + "# Message>:Successfully binding
52         socket!" << endl;
53
54     // Listen for agents to connect
55     cout << "-----Listening for Agents-----" << endl;
56     if (listen(sListenSocket, MAX_CONNECTION) < 0)
57     {
58         cout << "<Headquarter::System @ " + str_time2 + "# Message>:Oops! There is a
59             fatal error in listening for agents!" << endl;
60         emergencyResponse();
61         // If the preceding function returns, the system will be rebooted
62         goto REBOOT;
63     }

```

- 循环监听，处理连接请求并 accept：这部分主要有几处重点：

- accept 后更换使用新的 socket 即 sAcceptSocket。

- 超时设定：即若检测到 `accpet` 出错，并且错误原因 `errno== EWOULDBLOCK`，则立即终止监听，等待子进程结束后关闭程序。
- 通过 `fork` 实现多进程，但是这里父进程实际上会等待子进程结束再会重新开始监听，子进程会收集来源 Agent 的信息并记录在 vector 中。

```
1  while (true)
2  {
3      // Accept the connection from the agent
4      /* blablabla */
5      if ((sAcceptSocket = accept(sListenSocket, (sockaddr *)&agentAddr, (socklen_t *)
6          &agentAddr)) < 0)
7      {
8          if (errno == EWOULDBLOCK)
9          {
10             /**
11              * TIMEOUT
12              * If the interval between connections is too long
13              * The SYSTEM will wait for the last agent to leave
14              * And then the SYSTEM will go down to save energy
15              */
16             cout << "<Headquarter::System @ " + str_time3 + "# WARNING>:Too long
17                 intervals between connections." << endl;
18             cout << "<Headquarter::System @ " + str_time3 + "# WARNING>:The main
19                 system will go down." << endl;
20             int status;
21             waitpid(nPid, &status, 0);
22             cout << "-----SYSTEM SHUTDOWN-----" << endl;
23             close(sListenSocket);
24             exit(0);
25         }
26         else
27         {
28             // Other fatal errors except for the interval between connections
29             /* blablabla */
30         }
31     }
32     nPid = fork();
33     if (nPid != 0)
34     {
35         // Parent process, busy waiting for the agent to leave
36         int status;
37         waitpid(nPid, &status, 0);
38         continue;
39     }
40     else
41     {
42         // Child process, handle the connection
43         // For every agent, we will create a new process to handle the connection
44         close(sListenSocket);
```

```

42     string str_time4 = timeNow();
43     printf("<Headquarter::System @ %s # Message>:We got a agent from %s, port
44         %d, socket %d\n", str_time4.c_str(), inet_ntoa(agentAddr.sin_addr),
45         ntohs(agentAddr.sin_port), sAcceptSocket);
46     mutex mtx;
47     {
48         lock_guard<mutex> lock(mtx);
49         recv(sAcceptSocket, agentCodeName, 64, 0);
50         cout << "<Headquarter::System @ " + str_time4 + " # Message>:Agent
51             code name:" + agentCodeName + "." << endl;
52     }
53     // Push the agent into the vector
54     Agent agent(agentConfidential[agentCodeName], agentCodeName,
55                 sAcceptSocket);
56     {
57         lock_guard<mutex> lock(agentsMtx);
58         agents.push_back(agent);
59     }

```

- 敌人监听! 开始行动: 这里会假装收到了敌人的监听, 然后由长官负责开启不同的行动代号, 对应了 utils.cpp 中实现的不同行动。将在后面进行详细说明。

```

1      // Fake a emergency
2      sleep(3);
3      // Earshot warning
4      string str_time6 = timeNow();
5      cout << "<Headquarter::System @ " + str_time6 + " # WARNING>:The enemy is
6          potentially in earshot. Be careful!" << endl;
7      while (true)
8      {
9          string str_time7 = timeNow();
10         cout << "<Headquarter::System @ " + str_time7 + " # Message>:Please
11             give us further instructions!" << endl;
12         cout << "1:Let's GO DARK! Slient Guardian, ACTIVATE!" << endl;
13         cout << "2:Forget about their existence." << endl;
14         cout << "3:Terminate the connection! ASAP!" << endl;
15         cin >> instruction;
16         if (cin.fail())
17         {
18             cin.clear();
19             cin.ignore(numeric_limits<streamsize>::max(), '\n');
20             string str_time8 = timeNow();
21             cout << "<Headquarter::System @ " + str_time8 + " #
22                 Message>:Invalid input. Please enter a number." << endl;
23             continue;
24         }

```

```
25     case 1:
26         cout << "<Headquarter::System @ " + str_time9 + " # Message>:Yes,
27             sir!" << endl;
28         /**
29          * Secret Code Name: Silent Guardian
30          * This is a secret communicating channel for the headquarter and
31          * the agent
32          * In case there is enemy's interception, this will be activated
33          * to ensure the safety of the communication
34          */
35
36         SilentGuardian(M16, agent, des);
37         close(sAcceptSocket);
38         exit(0);
39         return 0;
40         break;
41
42     case 2:
43         cout << "<Headquarter::System @ " + str_time9 + " # Message>:Yes,
44             sir!" << endl;
45         /**
46          * Normally communicating without any encryption
47          * This could be a potential threat to the system
48          * But we can also use this as a trap to catch the enemy
49          * Like a bait to lure the enemy in
50          * And then we can take them down
51          * This is a risky move, but it could be a potential strategy
52          */
53         PhantomHook(M16, agent);
54         close(sAcceptSocket);
55         exit(0);
56         return 0;
57         break;
58
59     case 3:
60         cout << "<Headquarter::System @ " + str_time9 + " #
61             Message>:System aborted and files all in ash!" << endl;
62         /**
63          * In case the system is compromised
64          * We have to abort the system and destroy all files
65          * This is the last resort
66          */
67
68         des.encrypt("SYSTEM DOWN", cipheredtext);
69         send(sAcceptSocket, cipheredtext, 255, 0);
70         close(sAcceptSocket);
71         exit(0);
72         break;
73
74     default:
75         cout << "<Headquarter::System @ " + str_time9 + " #
76             Message>:Sorry, sir! I am not following you." << endl;
77         break;
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
```

```

68     }
69 }
```

4.3.2 Client

和 Server 端类似，这里也是先进行初始化等就不再赘述，先重点展示下 socket 的 TCP 处理流程：这部分主要负责客户端的创建 Socket，自主输入 IP 地址或者使用默认地址，connect 连接服务器端，最后连接成功后将 codename 发送给 Server。之后的行为和 Server 端类似，会接收总部发来的行动要求，然后根据不同的行动代号采取不同的行动，这里不再赘述。

```

1   cout << "-----Creating Socket-----" << endl;
2   // Create a socket
3   agentSocket = socket(AF_INET, SOCK_STREAM, 0);
4   // Test if the socket is created successfully
5   if (agentSocket < 0)
6   {
7       str_time = timeNow();
8       cout << "<SecretHideout::System @ " + str_time + " # Message>:Oops!There is a
      fatal error in creating socket!" << endl;
9       cout << "<SecretHideout::System @ " + str_time + " # Message>:Report this
      issue to the headquarter immediately!" << endl;
10      exit(0);
11  }
12  str_time = timeNow();
13  cout << "<SecretHideout::System @ " + str_time + " # Message>:Successfully
      creating socket!" << endl;
14  cout << "<SecretHideout::System @ " + str_time + " # Message>:Agent, please input
      the IP adress of the Headquarter in M16:" << endl;
15  while (true)
16  {
17      cout << "<SecretHideout::System @ " + str_time + " # Message>:1:Use the
      Default IP Address" << endl;
18      cout << "<SecretHideout::System @ " + str_time + " # Message>:2:Input the IP
      Address Manually" << endl;
19      cin >> choice;
20      if (cin.fail())
21      {
22          cin.clear();                                // Clear the error
23          flag
24          cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignore the rest
              of the input
25          str_time = timeNow();
26          cout << "<SecretHideout::System @ " + str_time + " # Message>:Invalid
              input. Please enter a number." << endl;
27          continue;
28      }
29      switch (choice)
30      {
```

```

30     case 1:
31         break;
32     case 2:
33         cout << "<SecretHideout::System @ " + str_time + " # Message>:Please
34             input the IP address of the Headquarter in M16:" << endl;
35         cin >> serverIP;
36         break;
37     default:
38         str_time = timeNow();
39         cout << "<SecretHideout::System @ " + str_time + " # Message>:Sorry, sir!
40             I am not following you." << endl;
41         continue;
42     }
43     break;
44 // Set the address of the server
45 serverAddr.sin_family = AF_INET;
46 serverAddr.sin_port = htons(SERVER_PORT);
47 serverAddr.sin_addr.s_addr = inet_addr(serverIP.c_str());
48
49 // Connect to the server
50 str_time = timeNow();
51 cout << "-----Connecting to the Headquarter-----" << endl;
52 if (connect(agentSocket, (struct sockaddr *)&serverAddr, sizeof(serverAddr)) < 0)
53 {
54     str_time = timeNow();
55     cout << "<SecretHideout::System @ " + str_time + " # Message>:Oops! There is a
56         fatal error in connecting to the Headquarter!" << endl;
57     cout << "<SecretHideout::System @ " + str_time + " # Message>:Report this
58         issue to the headquarter immediately!" << endl;
59     exit(0);
60 }
61 str_time = timeNow();
62 cout << "<SecretHideout::System @ " + str_time + " # Message>:Successfully
63     connected to the Headquarter!" << endl;
64
65 // Send the codename to the headquarter
66 send(agentSocket, agentCodeName.c_str(), strlen(agentCodeName.c_str()), 0);
67 /* 剩余部分和Server端类似，都是敌人入侵，然后展开行动 */

```

4.4 AgentTheme

4.4.1 Special Operations

- **Silent Guardian Operation:** 利用 fork 实现了多线程双全工通信。主要目的是进行 DES 加密的通信。程序进行初始化后通过 fork 分为两部分：
 - 父进程：负责接收信息并打印。通过调用 TotalRecv 进行接收。接收会进行 decrypt 等操作，

如果接收到的信息是”OVER”则结束通信并通知另一个进程结束。如果是”COMMUNICATION ERROR”则进行错误处理。

- 子进程：通过 `getline` 实现读取输入。加密后 `send`。同样检查是不是 “OVER”，如果是 OVER “通知另一个进程结束。

具体进程间通信问题详见 6.3.2。代码如下：

```
1     void SilentGuardian(int role, Agent agent, DESUtils des)
2 {
3     if (role == AGENT)
4     {
5         pid_t nPid;
6         nPid = fork();
7         if (nPid != 0)
8         {
9             /**
10              * Parent process
11              * In charge of the following:
12              * 1. Receive message from the headquarter
13              * 2. Decrypt the message
14              * 3. Output the message
15             */
16
17             while(true)
18             {
19                 nLength = TotalRecv(socket, strSocketBuffer, 255, 0);
20                 if (nLength == 255)
21                 {
22                     // Normal Case
23                     des.decrypt(strSocketBuffer, decryptedtext);
24                     decryptedtext[254] = 0;
25                     if (decryptedtext[0] != 0 && decryptedtext[0] != '\n')
26                     {
27                         cout << "<SecretHideout::Headquarter @" + str_time + "#"
28                         Message: " + decryptedtext << endl;
29                         /* 检查是不是“OVER”，如果是OVER“通知另一个进程结束 */
30                         /* 检查是不是“COMMUNICATION ERROR”，如果是同样结束 */
31                     }
32                 } else{
33                     /* blablabla 错误处理*/
34                 }
35             }
36         }
37         /**
38          * Child process
39          * In charge of the following:
40          * 1. Get the message from the agent
41          * 2. Encrypt the message
42     }
```

```

42     * 3. Send the message to the headquarter
43     */
44     while (true)
45     {
46         // Get the message from the agent
47         cin.getline(strStdinBuffer, 255);
48         int nLen = 255;
49         // Encrypt the message
50         des.encrypt(strStdinBuffer, encryedtext);
51         // Send the message to the agent
52         if (send(socket, encryedtext, sizeof(encryedtext), 0) != 255)
53         {
54             /* 网络问题，错误处理 */
55         } else{
56             // 正常情况
57             /* 检查是不是“OVER”，如果是OVER“通知另一个进程结束”*/
58             cout << "<Headquarter::System @ " + timeNow() + "# Message>:Message
59             Sent." << endl;
60         }
61     }
62 }
63 else if (role == M16)
64 {
65     /* blablabla 类似处理*/
66 }
67 return;
68 }
```

- **Phantom Hook Operation:** 这里不再展示，和 Silent Guardian 程序比较类似。只是开头对话不同，send 和 recv 前后没有通过 DES 加密而已。

```

1     cout << "<SecretHideout::System @ " + str_time + "# Message>:We're reading
2     you loud and clear." << endl;
3     cout << "<SecretHideout::System @ " + str_time + "# Message>:Phantom Hook is
4     activated.." << endl;
5     cout << "<SecretHideout::System @ " + str_time + "# Message>:You know what
6     to do....." << endl;
```

4.4.2 其它功能

除此之外，本次实验还在一些比较特殊的地方加了一些和特工主题相关的有趣功能。具体而言：

1. **身份验证：**通过在 Client 端对五个特工的真名要求输入，如果不能正确输入特工真名，就会驱除出去。

```

1     cout << "<SecretHideout::System @ " + str_time + "# Message>:Stop there
2     mate! Who are you?" << endl;
```

```

2     cout << "Well, I am:";
3     getline(cin, agentName);
4     /**
5      * Check the agent's identity
6      * If the agent is not one of the following, the agent is not allowed to enter
7      * Because the real name of the agent is TOP confidential, only the someone with
8      * the following name is trustworthy to allowed to enter
9      * small eastereggs here hhh, Tom Cruise and Rowan Atkinson are the real name of
10     * the actor of the spy in the movie
11     * Agent List:
12     * 1. James Bond, 007
13     * 2. David Webb, Jason Bourne
14     * 3. Natasha Romanoff, Black Widow
15     * 4. Tom Cruise, Ethan Hunt
16     * 5. Rowan Atkinson, Johnny English
17     */
18 // From the Movie: James Bond
19 if (agentName == "James Bond")
20     agentCodeName = "007";
21 /* blablabla */
22 else
23 {
24     // If the agent is not one of the following, he has to be a intruder
25     str_time = timeNow();
26     cout << "<SecretHideout::System @ " + str_time + " # Message>:Sorry, you are
27         not allowed to enter!" << endl;
28     cout << "<SecretHideout::System @ " + str_time + " # Message>:This is a
29         restricted area!" << endl;
30     cout << "<SecretHideout::System @ " + str_time + " # Message>:Run for your
31         life, outlaw!" << endl;
32     exit(0);
33 }

```

2. **错误处理:** TCP 部分的错误通过总部端的 emergencyResponse 实现。
3. **总部多负载通信:** 通过 **Server 端的 fork**, 将监听进程和为某个连接而来的 client 的独特进程分开处理, 但暂时还未实现多个特工同时与总部进行通信的功能。
4. **虚假身份检测:** 具体实现方法是在**总部 Server 端通过检测存有 Agent 类对象的 vector 是否有相同的 codeName 的对象存在**。若存在, 向客户端 (实际上为 SecretHideout 即秘密藏身处的系统) 发送一个”FAKE IDENTITY”用于提醒。

```

1 // Receive the code name of the agent
2 {
3     lock_guard<mutex> lock(agentsMtx);
4     for (auto &agent : agents)
5     {
6         if (agent.getAgentCodeName() == agentCodeName)
7         {

```

```

8      /**
9       * If the agent is already in the system
10      * That means the agent is a FAKE one
11      * TERMINATE THE CONNECTION
12      * Tell the secret hideout to disconnect the person too
13      */
14      string str_time5 = timeNow();
15      cout << "<Headquarter::System @ " + str_time5 + " # "
16      Message>:Agent " + agentCodeName + " is already in the
17      system." << endl;
18      // ENCRYPT THE MESSAGE so that the enemy cannot be aware of
19      the situation
20      des.encrypt("FAKE IDENTITY", cipheredtext);
21      send(sAcceptSocket, cipheredtext, 100, 0);
22      close(sAcceptSocket);
23      exit(0);
24  }
25 }
26 }
```

同时秘密藏身处该方会不打草惊蛇地断开连接，结束通话。

```

1  else if (strcmp(decryptedtext, "FAKE IDENTITY") == 0)
2  {
3      /**
4       * IF the Headquarter is telling back the current person using
5       * the secret hideout is ACTUALLY FAKE
6       * We do not distract the intruder, we need to let him assume that
7       * he is still safe here
8       * But we need to close the connection
9       * The MI6 will take care of the rest, sending someone to catch
10      the intruder near the hideout
11      */
12      cout << "<SecretHideout::System @ " + str_time + " # Message> " +
13      "I'm sorry, the Headquarter is not available right
14      now..." +
15      << endl;
16      cout << "-----Connection Closed-----" << endl;
17      close(agentSocket);
18      return 0;
19 }
```

5. 代号匹配：这个实现很简单，就是通过中不断的 map<string, string> agentConfidential 实现。

5 实验结果

5.1 DES 正确性验证

这里通过在 DES.cpp 中加入如下代码进行验证。

```
1 int main()
2 {
3     DESUtils des;
4     char *plaintext = "123456"; // 原始明文
5     char ciphertext[64]; // 存储加密后的密文
6     char decryptedtext[64]; // 存储解密后的文本
7
8     // 清零
9     memset(ciphertext, 0, sizeof(ciphertext));
10    memset(decryptedtext, 0, sizeof(decryptedtext));
11    uint64_t key = 0x133457799BBCDFF1;
12    des.genKey(&key);
13    // 输出16轮密钥
14    cout << "Round Keys: " << endl;
15    for (int i = 0; i < 16; i++)
16    {
17        cout << "Round " << i + 1 << ":" << hex << showbase << des.roundKeys[i] <<
18        endl;
19    }
20    // 加密
21    cout << "Encrypting..." << endl;
22    des.encrypt(plaintext, ciphertext);
23    cout << "plaintext: " << plaintext << endl;
24    for (int i = 0; i < strlen(plaintext); i++)
25    {
26        printf("%02x ", (unsigned char)plaintext[i]);
27    }
28    printf("\n");
29    cout << "ciphertext: " << ciphertext << endl;
30    for (int i = 0; i < strlen(ciphertext); i++)
31    {
32        printf("%02x ", (unsigned char)ciphertext[i]);
33    }
34    printf("\n");
35    // 解密
36    cout << "Decrypting..." << endl;
37    des.decrypt(ciphertext, decryptedtext);
38
39    cout << "strlen(decryptedtext): " << strlen(decryptedtext) << endl;
40    for (int i = 0; i < strlen(decryptedtext); i++)
41    {
42        printf("%02x ", (unsigned char)decryptedtext[i]);
43    }
44}
```

```
43     printf("\n");
44     // 检查解密后的文本是否与原始明文相同
45     if (strcmp(plaintext, decryptedtext) == 0)
46     {
47         std::cout << "DES encryption and decryption are correct!" << std::endl;
48     }
49     else
50     {
51         std::cout << "Error in DES encryption or decryption." << std::endl;
52     }
53
54     return 0;
55 }
```

其中密钥是特殊挑选的一组，会和标准轮密钥生成结果对比。另外如果验证成功，应该会输出”DES encryption and decryption are correct!“。结果如下：

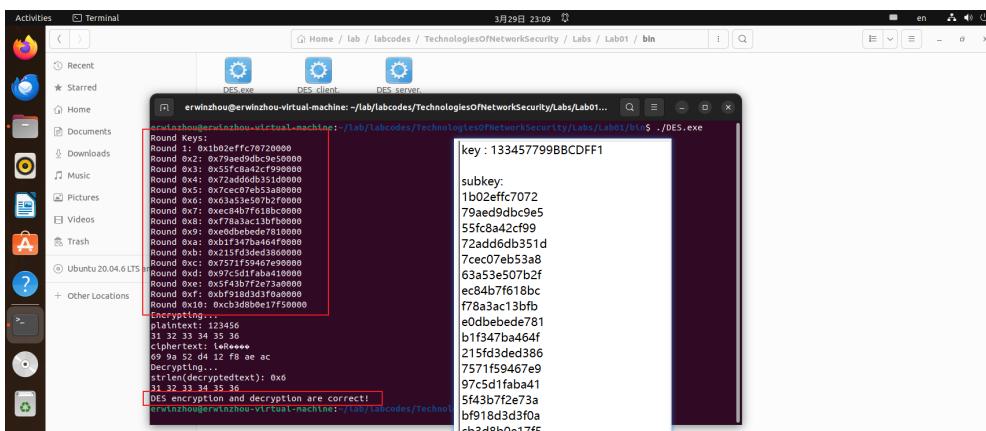


图 5.13: DES 验证结果

如图5.13所示，可以看到不仅输出了16轮的密钥与标准答案一致，并且有打印信息”DES encryption and decryption are correct!”。验证成功！

PS: 该组 16 轮密钥的结果经过验证, 十分可靠。

5.2 实验结果展示

现在对整体通信过程进行一个简单的展示。首先假设登陆的特工是 JamesBond，指挥官是 M。这里为了美观采用虚拟机终端展示。

```
erwinzhou@erwinzhou-virtual-machine:~/lab/labcodes/TechnologiesOfNetworkSecurity/Labs/Lab01/bin$ ./DES_Server.exe
-----M16 Headquarter-----
<Headquarter::System @ Fri Mar 29 23:14:28 2024 # Message:>May I have your name, sir?
Commander: M
<Headquarter::System @ Fri Mar 29 23:14:28 2024 # Message:>Welcome back , Commander M!
-----Commander M-----
<Headquarter::System @ Fri Mar 29 23:14:28 2024 # Message:>system rebooting...
-----Creating Socket-----
<Headquarter::System @ Fri Mar 29 23:14:28 2024 # Message:>Successfully creating socket!
<Headquarter::System @ Fri Mar 29 23:14:28 2024 # Message:>Successfully binding socket!
-----Listening for Agents-----
<Headquarter::System @ Fri Mar 29 23:14:28 2024 # Message:>Waiting for agents to come in...
```

图 5.14: 连接前 Server 端

如图5.14所示，可以看到服务器总部端会要求输入正确的长官名字。之后开始一系列初始化并开始 socket 监听 Agent 的连接。

此时如果 Client 端没法正确通过身份验证就会显示：如果特工正确输入了名字，则可以开始与总部正式进行通信：

```

Activities Terminal 3月29日 23:17
erwinzhou@erwinzhou-virtual-machine: ~/lab/labcodes/TechnologiesOfNetworkSecurity/Labs/Lab01/bin$ ./DES_client.exe
... Secret Hideout...
<SecretHideout::System @ Fri Mar 29 23:16:09 2024 # Message>;Stop there mate! Who are you?
Well, I am James Bond
<SecretHideout::System @ Fri Mar 29 23:16:18 2024 # Message>;Welcome back, Mr.007!
-----Agent:007-----
<SecretHideout::System @ Fri Mar 29 23:16:18 2024 # Message>;The mission is almost ready to launch !
<SecretHideout::System @ Fri Mar 29 23:16:18 2024 # Message>;Please wait for a moment, Mr.007!
-----Creating Socket-----
<SecretHideout::System @ Fri Mar 29 23:16:18 2024 # Message>;Successfully creating socket!
<SecretHideout::System @ Fri Mar 29 23:16:18 2024 # Message>;Agent, please input the IP address of the Headquarter i
n M16:
<SecretHideout::System @ Fri Mar 29 23:16:18 2024 # Message>;Use the Default IP Address
<SecretHideout::System @ Fri Mar 29 23:16:18 2024 # Message>;2:Input the IP Address Manually
-----Connecting to the Headquarter-----
<Headquarter::System @ Fri Mar 29 23:14:03 2024 # Message>;Successfully connected to the Headquarter!
<SecretHideout::System @ Fri Mar 29 23:16:27 2024 # Message>;Warning: The enemy is potentially in earshot. We are under surv
eillance, be careful!
-----Commander: M -----
<Headquarter::System @ Fri Mar 29 23:14:28 2024 # Message>;We are under surveillance, be careful!
-----Commander: M -----
<Headquarter::System @ Fri Mar 29 23:14:28 2024 # Message>;Creating Socket...
-----Binding Socket-----
<Headquarter::System @ Fri Mar 29 23:14:28 2024 # Message>;Successfully binding socket!
-----Listening for Agents-----
<Headquarter::System @ Fri Mar 29 23:14:28 2024 # Message>;Waiting for agents to come in...
<Headquarter::System @ Fri Mar 29 23:16:27 2024 # Message>;We got a agent from 0.0.0.0, port 0, socket 4
Ubuntu-20.04.6-LTS-minimal @ Fri Mar 29 23:16:27 2024 # Message>;Agent code name:007.
<Headquarter::System @ Fri Mar 29 23:16:30 2024 # Message>;Warning: The enemy is potentially in earshot. Be careful!
<Headquarter::System @ Fri Mar 29 23:16:30 2024 # Message>;Please give us further instructions!
1:Let's GO DARK! Silent Guardian, ACTIVATE!
2:Forget about their existence.
3:Terminate the connection! ASAP!
```

图 5.15: 成功连接后

如5.15所示，可以看到以下内容：

- 客户端成功通过身份验证后，进行了一系列 socket 初始化。
- 客户端可以自主选择 IP 地址或使用默认 IP 地址。在选择使用默认 IP 地址后双方成功连接。
- 服务器端显示 Agent 的 IP 地址端口，以及代号。
- 服务器端和客户端同时显示了敌人的人侵窃听！由服务器端进行下一步指示。

接下来以 SilentGuardian 为例，演示 DES 加密的通信过程：

```

<Headquarter::System @ Fri Mar 29 23:28:21 2024 # Message>;Please give us furt
her instructions!
1:Let's GO DARK! Silent Guardian, ACTIVATE!
2:Forget about their existence.
3:Terminate the connection! ASAP!
1
<Headquarter::System @ Fri Mar 29 23:28:23 2024 # Message>;Yes, sir!
<Headquarter::System @ Fri Mar 29 23:28:23 2024 # Message>;Silent Guardian is activated..
Agent! Are you prepared for your mission?
<Headquarter::System @ Fri Mar 29 23:28:36 2024 # Message>;Message Sent.

<Headquarter::007 @ Fri Mar 29 23:29:39 2024 # Message>; Yes, Sir!
Fine! You mission this time is to blow up Nankai University at Tianjin, China!
Do you copy?
<Headquarter::System @ Fri Mar 29 23:30:16 2024 # Message>;Message Sent.

<Headquarter::007 @ Fri Mar 29 23:30:50 2024 # Message>; Loud And Clear! Nanka
i is doomed! Copy that, commander!
Ok, that's all. I have to have lunch now, goodbye!
<Headquarter::System @ Fri Mar 29 23:31:16 2024 # Message>;Message Sent.

<Headquarter::007 @ Fri Mar 29 23:31:24 2024 # Message>; Goodbye, SIR!
OVER
<Headquarter::System @ Fri Mar 29 23:31:30 2024 # Message>;Communication is OV
ER.
-----Communication OVER-----
<Headquarter::System @ Fri Mar 29 23:31:30 2024 # Message>;Waiting for agents
to come in...
```

图 5.16: DES 加密通信展示

如5.16可以看到如下内容：

- 长官 Commander 下令去 blow up Nankai University(我的恶趣味 hhh。Commander 即服务器端可以正常发送消息，并且 Agent 即客户端可以正常显示)。
- 特工 Agent007 回复长官 Nankai is doomed! Agent 即客户端可以正常发送消息，并且 Commander 即服务器端可以正常显示。
- Commander 长官即服务器端通过 OVER 结束了通信，Agent 一侧也结束了通信。并且服务器一端重新开始等待新的 Agent 连入。

若等待时间超时 (5min)，则会自动结束：

```
<Headquarter::System @ Fri Mar 29 23:28:23 2024 # Message>;Yes, sir!
<Headquarter::System @ Fri Mar 29 23:28:23 2024 # Message>;Silent Guardian is activated..
Agent! Are you prepared for your mission?
<Headquarter::System @ Fri Mar 29 23:28:36 2024 # Message>;Message Sent.

<Headquarter::007 @ Fri Mar 29 23:29:39 2024 # Message>; Yes, Sir!
Fine! Your mission this time is to blow up Nankai University at Tianjin, China!
Do you copy?
<Headquarter::System @ Fri Mar 29 23:30:16 2024 # Message>;Message Sent.

<Headquarter::007 @ Fri Mar 29 23:30:50 2024 # Message>; Loud And Clear! Nankai is doomed! Copy that, commander!
Ok, that's all. I have to have lunch now, goodbye!
<Headquarter::System @ Fri Mar 29 23:31:16 2024 # Message>;Message Sent.

<Headquarter::007 @ Fri Mar 29 23:31:24 2024 # Message>; Goodbye, SIR!
OVER
<Headquarter::System @ Fri Mar 29 23:31:30 2024 # Message>;Communication is OVER.
-----Communication OVER-----
<Headquarter::System @ Fri Mar 29 23:31:30 2024 # Message>;Waiting for agents to come in...
<Headquarter::System @ Fri Mar 29 23:31:30 2024 # WARNING>;Too long intervals between connections.
<Headquarter::System @ Fri Mar 29 23:31:30 2024 # WARNING>;The main system will go down.
-----SYSTEM SHUTDOWN-----
```

图 5.17: 服务器端等待超时

当然即使并没有超时，**也可以通过 Ctrl+C 结束等待，详见 6.3.1**。若未超时，则可重新连入。

```
0.0.0.0, port 0, socket 4
<Headquarter::System @ Fri Mar 29 23:41:38 2024 # Message>;Agent code name:007
<Headquarter::System @ Fri Mar 29 23:41:41 2024 # WARNING>;The enemy is potentially in earshot. Be careful!
<Headquarter::System @ Fri Mar 29 23:41:41 2024 # Message>;Please give us further instructions!
1:Let's GO DARK! Silent Guardian, ACTIVATE!
2:Forget about their existence.
3:Terminate the connection! ASAP!
1
<Headquarter::System @ Fri Mar 29 23:41:42 2024 # Message>;Yes, sir!
<Headquarter::System @ Fri Mar 29 23:41:42 2024 # Message>;Silent Guardian is activated..
OVER
<Headquarter::System @ Fri Mar 29 23:41:45 2024 # Message>;Communication is OVER.
-----Communication OVER-----
<Headquarter::System @ Fri Mar 29 23:41:45 2024 # Message>;Waiting for agents to come in...
<Headquarter::System @ Fri Mar 29 23:42:12 2024 # Message>;We got a agent from 192.168.126.128, port 0, socket 5
<Headquarter::System @ Fri Mar 29 23:42:12 2024 # Message>;Agent code name:Black Widow.
<Headquarter::System @ Fri Mar 29 23:42:15 2024 # WARNING>;The enemy is potentially in earshot. Be careful!
<Headquarter::System @ Fri Mar 29 23:42:15 2024 # Message>;Please give us further instructions!
1:Let's GO DARK! Silent Guardian, ACTIVATE!
2:Forget about their existence.

<SecretHideout::System @ Fri Mar 29 23:41:45 2024 # Message>;Communication is OVER.
-----Communication OVER-----
erwinzhou@erwinzhou-virtual-machine:~/lab/labcodes/TechnologiesOfNetworkSecurity/Labs/Lab01/bin$ ./DES_client.exe
-----Secret Hideout-----
<SecretHideout::System @ Fri Mar 29 23:41:48 2024 # Message>;Stop there mate!
Who are you?
Well, I am Natasha Romanoff.
<SecretHideout::System @ Fri Mar 29 23:42:02 2024 # Message>;Welcome back, Ms. Black Widow!
-----Agent:Black Widow-----
<SecretHideout::System @ Fri Mar 29 23:42:02 2024 # Message>;The mission is almost ready to launch !
<SecretHideout::System @ Fri Mar 29 23:42:02 2024 # Message>;Please wait for a moment, Ms. Black Widow!
-----Creating Socket-----
<SecretHideout::System @ Fri Mar 29 23:42:02 2024 # Message>;Successfully creating socket!
<SecretHideout::System @ Fri Mar 29 23:42:02 2024 # Message>;Agent, please input the IP address of the Headquarter in M16:
<SecretHideout::System @ Fri Mar 29 23:42:02 2024 # Message>;1:Use the Default IP Address
<SecretHideout::System @ Fri Mar 29 23:42:02 2024 # Message>;2:Input the IP Address Manually
1
-----Connecting to the Headquarter-----
<SecretHideout::System @ Fri Mar 29 23:42:12 2024 # Message>;Successfully connected to the Headquarter!
<SecretHideout::System @ Fri Mar 29 23:42:15 2024 # Warning>;The enemy is pot
```

图 5.18: 服务器端重新连入一次登录多次监听

如图5.18所示，看到了 James Bond 通话结束后，Natasha Romanoff 也就是 BlackWindow 重新登陆后，依然能够正常聊天。

Phantom Hook 的过程和 SilentGuardian 类似这里就不再重复展示。到此就展示了本次实验实现的全部功能，实验非常成功！

6 实验遇到的问题及其解决方法

6.1 Visual Studio Code

6.1.1 规范化模块搭建

本次是我第一次使用 Visual Studio Code 完整地编写 C++ 程序。是因为听说一些大佬们都可以全部用 VSCode 实现。第一步我做的就是规范化我的程序。通过严格的 bin, include, src 来划分可执行文件，头文件和 cpp 文件。

这个过程最开始并不容易，但我通过解耦实现，逻辑清晰地编写了多个接口完成。其中头文件中只声明不实现，在对应的 cpp 中才实现。对我的锻炼极大。

6.1.2 编译连接问题

- 遇到的问题：**由于第一次接触 VSCode 全程编写大规模的 C++ 项目。作为编辑器的 C++ 不具备编译器和链接器等，**最开始我的编译运行等地方出了很多问题不停地报错。**
- 解决的方式：**具体而言是通过编写自带的.vscode 下面的 json 文件解决的：

1. **c_cpp_properties.json：**这个 json 配置文件负责一些设定。为了使得在 src 文件夹下的 cpp 文件中，我能够引用在其他文件夹如 src 文件中自定义的头文件，我需要通过在 **includePath** 中手动加入对应路径，

即” /lab/labcodes/TechnologiesOfNetworkSecurity/Labs/Lab01/include” 实现正确的打开和包含。

完整的 c_cpp_properties.json 文件如下：

```

1  {
2      "configurations": [
3          {
4              "name": "Linux",
5              "includePath": [
6                  "${workspaceFolder}/**",
7                  "~/lab/labcodes/TechnologiesOfNetworkSecurity/Labs/Lab01/include",
8                  "/usr/include"
9              ],
10             "defines": [],
11             "compilerPath": "/usr/bin/clang",
12             "cStandard": "c17",
13             "cppStandard": "c++14",
14             "intelliSenseMode": "linux-clang-x64"
15         }
16     ],

```

```

17     "version": 4
18 }

```

2. tasks.json: 这个 json 文件主要负责调用编译器进行正常的编译链接，它的核心是自动生成终端的 g++ 编译器指令。为了能够使得其它的.h 文件和 cpp 文件能够被正确链接，需要在 args 中添加对应的.h 和.cpp 路径，并通过-I 和-o 参数指定。除此之外，还有设定生成可执行文件路径在 bin 下。具体 json 配置如下：

```

1 {
2   "tasks": [
3     {
4       "type": "cppbuild",
5       "label": "C/C++: g++-11 生成活动文件",
6       "command": "/usr/bin/g++-11",
7       "args": [
8         "-fdiagnostics-color=always",
9         "-g",
10        "${file}",
11        "~/lab/labcodes/TechnologiesOfNetworkSecurity/Labs/Lab01/src/utils.cpp",
12        "~/lab/labcodes/TechnologiesOfNetworkSecurity/Labs/Lab01/src/DES.cpp",
13        "~/lab/labcodes/TechnologiesOfNetworkSecurity/Labs/Lab01/src/agent.cpp",
14        "-I",
15        "~/lab/labcodes/TechnologiesOfNetworkSecurity/Labs/Lab01/include",
16        "-o",
17        "${fileDirname}../bin/${fileBasenameNoExtension}.exe",
18      ],
19      "options": {
20        "cwd": "${fileDirname}"
21      },
22      "problemMatcher": [
23        "$gcc"
24      ],
25      "group": {
26        "kind": "build",
27        "isDefault": true
28      },
29      "detail": "调试器生成的任务。"
30    }
31  ],
32  "version": "2.0.0"
33 }

```

6.2 DES

6.2.1 数据存储

- 遇到的问题：DES 算法中要求有非 32 位整数倍的存储，比如 28 位等。是否应该采用 vector 存储和操作？

- **解决的方式：**矩阵和 vector 存储和操作了许多数据不仅占用内存较多。并且 vector 的很多封装操作使得时间性能较差。**以 32bit 为界，大于 32bit 一律采用 uint64_t 进行存储。**

6.2.2 矩阵与 vector 存储时间与空间复杂度高

- **遇到的问题：**在上学期的密码学课程中，我曾经完成过 AES 加密算法的编写，不过由于当初的我通过矩阵和 vector 进行的操作。
- **解决的方式：**本次实验中我 DES 全部采用了位运算进行实现。C++ 的位运算是通过 bit 进行存储，而计算机的位运算是最快的，本质上就是寄存器之间的运算。因此本次采用了位运算后极大地加快了算法的性能。

6.2.3 填充问题

- **遇到的问题：**我在最开始编写 DES 的加密部分时候并没有编写填充，使得超过 64 位或者不足 64 位时候出现了问题。
- **解决的方式：**最终采用了 PKCS#7 填充算法，用于在加密块密码算法中处理数据长度不足块大小的情况。通过 `size_t padLength = blockSize - (data.size() % blockSize)` 计算 padding 长度。Unpadding 时只需将末尾 0-8 依次删除即可。

6.3 TCP Channel

6.3.1 进程结束不释放资源问题

- **遇到的问题：**由于上学期的计算机网路课程中我没有接触过多进程编程，之前一直使用的是多线程编程。本次实验在最开始使用 `fork` 进行编程时候遇到了很诡异的问题。

具体而言，就是在服务器端通过 `fork` 分开主进程进行循环 `accpet`，为连接而来的客户端进行单独分配进程后。此时如果想要让监听结束，父进程整体结束，使用 **VSCode 关闭终端或者直接 Terminate 进程是不行的！因为此前主进程 accept 时处于阻塞状态，突然令其终止后会导致 socket 没有正常 close，端口依然被占用！下次程序再想利用该端口会无法使用！**

- **解决的方式：**我想到了两套解决方案。具体而言：

- **signal.h：通过信号量控制。**即如果想要终止时候，可以直接通过 `Ctrl+C` 发送信号，然后来到对应的信号处理函数中令其首先 `close(sListenSocket)` 然后再 `exit`。需要注意的是使用这个方法还需要通过 `singal` 注册对应的信号处理函数。

```

1 void handle_sigint(int sig)
2 {
3     /**
4      * This is a function to handle the signal of Ctrl+C
5      * To resolve the Parent Process holding socket WHILE dying issue
6      * @param sig: the signal of Ctrl+C
7      * @return void
8      */
9     close(sListenSocket);
10    exit(0);
11 }
```

```

12  /* blablabla 注册信号处理函数 */
13  signal(SIGINT, handle_sigin);

```

- 超时结束自动退出：首先通过 setsockopt 对 socket 设置 SO_RCVTIMEO 和 SO_SNDFTIMEO，设置超时时间为 500s。之后就可以在循环中检测是否出现了 errno == EWOULDBLOCK 的情况。若出现了就代表超时，清理资源关闭 socket 后正常退出。

```

1   if (setsockopt(sListenSocket, SOL_SOCKET, SO_RCVTIMEO, (char *)&timeout,
2     sizeof(timeout)) < 0)
3   {
4     cout << "<Headquarter::System @ " + str_time1 + "# Message>:Oops! There is a
5       fatal error in setting socket!" << endl;
6     emergencyResponse();
7     // If the preceding function returns, the system will be rebooted
8     goto REBOOT;
9   }
10
11  if (setsockopt(sListenSocket, SOL_SOCKET, SO_SNDFTIMEO, (char *)&timeout,
12    sizeof(timeout)) < 0)
13  {
14    cout << "<Headquarter::System @ " + str_time1 + "# Message>:Oops! There is a
15       fatal error in setting socket!" << endl;
16    emergencyResponse();
17    // If the preceding function returns, the system will be rebooted
18    goto REBOOT;
19  }
20  /* blablabla 后面在父进程循环监听时候*/
21  if ((sAcceptSocket = accept(sListenSocket, (sockaddr *)&agentAddr, (socklen_t
22    *)&agentAddr)) < 0)
23  {
24    if (errno == EWOULDBLOCK)
25    {
26      /**
27       * TIMEOUT
28       * If the interval between connections is too long
29       * The SYSTEM will wait for the last agent to leave
30       * And then the SYSTEM will go down to save energy
31       */
32      cout << "<Headquarter::System @ " + str_time3 + "# WARNING>:Too long
33           intervals between connections." << endl;
34      cout << "<Headquarter::System @ " + str_time3 + "# WARNING>:The main
35           system will go down." << endl;
36      int status;
37      waitpid(nPid, &status, 0);
38      cout << "-----SYSTEM SHUTDOWN-----" << endl;
39      close(sListenSocket);
40      exit(0);
41    }

```

6.3.2 进程通信问题

- **遇到的问题：**多进程双全工通信时候的父进程与子进程之间，若父进程（接收进程）接收到了来自另一端的 OVER 消息，则需要杀死子进程结束；若子进程（发送进程）获取到了键盘输入的 OVER，也需要杀死父进程结束。

除此之外，另一个涉及多进程的 fork 处，即服务器端通过主进程监听，也需要等待子进程的通话结束继续开始循环。

为了避免上面提到的资源释放问题，如何解决进程间的通信？

- **解决的方法：**本次实验最终使用了 sys/wait.h, signal.h 实现进程间的通信。kill(getppid(), SIGTERM) 实现子进程杀父进程，kill(nPid, SIGTERM) 实现父进程杀子进程。同时也需要注册 SIGTERM 的信号处理函数。

```
1 void signalHandler(int sig)
2 {
3     /**
4      * Signal handler: Used for the one process to exit after receiving the signal
5      * from another
6      * @param sig: signal
7      * @return void
8      */
9     exit(0);
}
```

即服务器端通过主进程监听则是通过 wait 函数实现的等待子进程结束：

```
1 if (nPid != 0)
2 {
3     // Parent process, busy waiting for the agent to leave
4     int status;
5     waitpid(nPid, &status, 0);
6     continue;
7 }
```

7 实验结论

本次实验是本学期网络安全实验课的第一次课程作业，通过完成一个经过 DES 加密的 TCP 聊天程序，我收获颇丰，具体而言：

1. 我通过强迫自己使用 VSCode 编程，并且使用极其规范的编程模板，不仅补全了我对使用 VSCode 编写 C++ 的能力短板，更重要的让我养成了工程规范化项目的编程习惯。通过清晰的架构，让我本次代码具有极高的可读性和可维护性，我也会保持这个习惯，用于我今后的其它项目中。
2. 之前密码学课程中虽然我曾经实现过一些诸如 AES 等加密算法，不过当时我还在使用矩阵与 vector 存储，时间与空间复杂度都较高，运算也相对笨拙。本次实验通过 DES 全程采用位运算实现，不仅加深了我对 DES 算法的理解，更锻炼了我使用位运算实现各种加密算法的能力，这

能够很好地优化加密算法的性能。由于量子计算的问世，算力的快速提升，让 DES 不再靠谱。本来我计划实现 tripleDES 或者 AES 等更加稳定的算法，由于时间原因，会在未来进一步完善。

3. 此次是第一次在 Linux 下进行的网络编程，并且首次接触多进程编程，而不是传统的多线程编程。通过认真学习实验手册并查阅资料，我了解到了通过 fork 进行多进程双全工编程的方法。在编程过程中，我也遇到了诸如进程释放资源，进程通信等更加实际的多进程问题。极大地加深了我对多进程的理解，强化了我多进程编程的能力。
4. 本次实验我自主创新，通过引入特工主题，极大的发挥了主观创造性，做出了一份独一无二，也令我十分享受其中的项目！

作为本学期的第一次实验，我希望能为我接下来的网络安全学习打下良好的基础，能够将网络安全的理论知识学透，更重要的是将每次实验做好，更好地应用于实践！争取发挥更多的创造性，成为一名有趣的信安人。

参考文献