单主机并发环境下实现 Kerberos

18342066 鲁沛

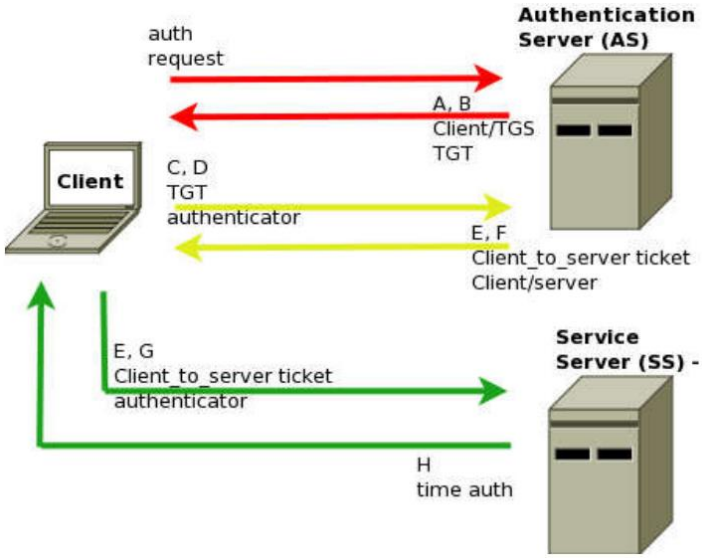
一、 原理概述

Kerberos协议主要用于计算机网络的身份鉴别(Authentication)，其特点是用户只需输入一次身份验证信息就可以凭借此验证获得的票据(ticket-granting ticket，TGT)访问多个服务，即 SSO(Single Sign On)。由于在每个 Client 和 Service 之间建立了共享密钥，使得该协议具有相当的安全性。

关键术语

- AS = Authentication Server，认证服务器；
- TGT = Ticket-Granting Ticket，票据授权票据；
- TGS = Ticket-Granting Server，票据分发中心；
- ST = Service Ticket，服务票据；
- SS = Service Server，服务服务器。

二、 Kerberos 的三个阶段



1. 身份认证

Client 向 AS 发送一个明文消息，代表用户请求服务。AS 检查数据库是否存在该用户 ID 的记录。确定的话 AS 返回以下两条消息给 Client:

消息 A: $E(K_{Client}, K_{Client-TGS})$

消息 B: $E(K_{TGS}, < client\ ID, client\ address, validity, K_{Client-TGS} >)$

消息 B 是用 TGS 密钥加密的票据授权票据 TGT，包括客户 ID、客户网络地址、票据有效期、Client-TGS 会话密钥。

Client 收到消息 A 和 B 后，应用$D(K_{client}, A)$得到$K_{Client-TGS}$用于后续与 TGS 的通信。Client 将凭借消息 B 携带的有效的 TGT 向 TGS 证明其身份。

2. 服务认证

申请服务时，Client 向 TGS 发送以下两条消息：

消息 C：service ID, B

消息 D：$E(K_{Client-TGS}, < clientID, timestamp >)$

消息 D 是用$K_{Client-TGS}$会话密钥加密的认证。

TGS 从消息 C 中获得消息 B，应用$D(K_{TGS}, B)$得到$K_{Client-TGS}$，再应用$D(K_{Client-TGS}, D)$得到认证内容，并返回给 Client 两条消息：

消息 E：service ID, ST

ST = $E(K_{SS}, < client\ ID, client\ net\ address, validity, K_{Client-SS} >)$

称为服务票据，包括客户 ID、客户网络地址、票据有效期限、 Client-SS 会话密钥。

消息 F：$E(K_{Client-TGS}, K_{Client-SS})$

3. 服务申请

Client 应用$D(K_{Client-TGS}, E)$得到$K_{Client-SS}$，然后向 SS 发出以下两条消息：

消息 E：由先前步骤得到的 service ID, ST

消息 G：$E(K_{Client-SS}, < client\ ID, timestamp >)$

消息 G 是用$K_{Client-SS}$会话密钥加密的一个新的认证。

SS 应用$D(K_{SS}, ST)$解密得到$K_{Client-SS}$，再应用$D(K_{Client-SS}, G)$解密得到认证 G，然后从中提取时间戳 TS = timestamp，返回 Client 一条消息 H 作为确认函以确认客户的身份真实，同意向该客户提供服务：

消息 H：$E(K_{Client-SS}, < client\ ID, TS + 1 >)$

Client 应用$D(K_{Client-SS}, H)$解密消息 H。如果其中的时间戳被正确更新，则 SS 可以信赖，Client 可以向 SS 发送服务请求。

认证过程至此结束，SS 向 Client 客户机提供其所请求的服务。

三、 总体结构设计

文件目录结构：3 个服务器，1 个客户端，1 个对称加密算法

```
|----AS.c
|----TGS.c
|----SS.c
|----client.c
|----DES.c
```

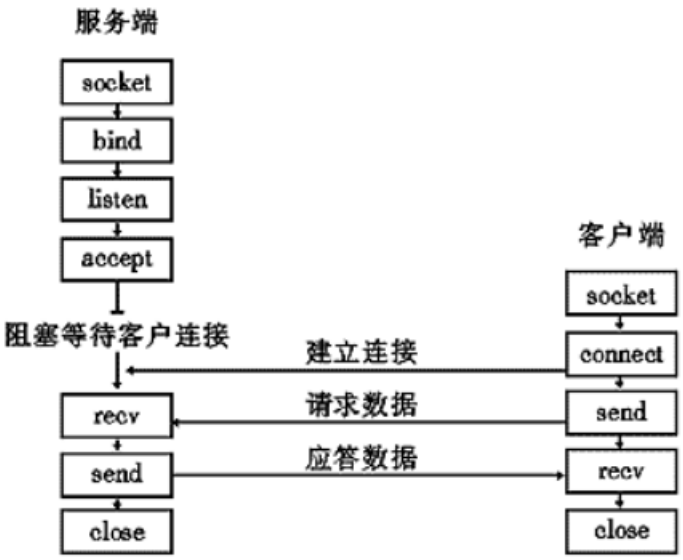
加密算法采用第一次作业实现的 DES 加密算法，加密解密函数用法如下：

```
1. char *DES_encrypt(char *input, char *key_16);
2. char *DES_decrypt(char *input, char *key_16);
```

进程间通信采用的是 **socket 套接字**，简单介绍如下：

Socket 是应用层与 TCP/IP 协议族通信的中间软件抽象层，它是一组接口。在设计模式中，Socket 其实就是一个门面模式，它把复杂的 TCP/IP 协议族隐藏在 Socket 接口后面，对用户来说，一组简单的接口就是全部，让 Socket 去组织数据，以符合指定的

协议。



服务端：

- socket 函数生成套接字;
- bind 函数绑定地址和 socket
- listen 函数开始监听
- accept 函数函数阻塞知道有客户端申请连接
- send 函数和 recv 函数用于发送和接受数据
- close 函数用于关闭套接字

客户端：

- socket 函数生成套接字;
- connect 函数讲套接字与远程服务器连接
- send 函数和 recv 函数用于发送和接受数据
- close 函数用于关闭套接字

四、 数据结构设计

三个服务器的端口号和 IP 地址:

```
1. #define AS_PORT 1500
2. #define TGS_PORT 1600
3. #define SS_PORT 1700
4. #define DEST_IP "127.0.0.1"
```

五个密钥:

K_{Client} the Master Key prenegotiated by *Client* and *AS*
K_{TGS} the symmetric key prenegotiated by *TGS* and *AS*
K_{SS} the symmetric key prenegotiated by *SS* and *TGS*
$K_{Client-TGS}$ the session key between *Client* and *TGS*
$K_{Client-SS}$ the session key between *Client* and *SS*

```
char K_client[] = "AAAAAAAAAAAAAAAA";
char K_client_TGS[] = "BBBBBBBBBBBBBBBB";
char K_TGS[] = "CCCCCCCCCCCCCCCC";
char K_client_SS[] = "DDDDDDDDDDDDDDDD";
char K_SS[] = "EEEEEEEEEEEEEEEE";
```

部分常量设计:

```
char ID[] = "Bob";              //用户 ID
char serviceID[] = "study";     //申请服务的 ID
time_t timestamp;    //客户端申请服务时的时间戳，每个阶段用到时要更新
```

五、 模块分解

1. **身份认证**
   客户端发送 ID 给 AS 服务器

```
printf("sending ID to server...\n");
send(client_socket, ID, strlen(ID), 0);
```

AS 服务器对比用户 ID 是否正确，不正确则终止服务器并通知客户端

```
char buf[MAX_DATA];        //储存接收数据
int messageLen = recv(client_socket, buf, MAX_DATA, 0);
buf[messageLen] = 0;
if(strcmp(ID, buf) != 0)
{
    printf("ID is not correct");
    send(client_socket, "ID is not correct", strlen("ID is not correct"), 0);
    return;
}
```

若正确则生成并发送消息 A 和 B 给客户端，生成消息 B 时用到了 sprintf 函数，起到连接字符串的作用，sleep 函数让消息 A 和消息 B 间隔发送以确保客户端能完整收到

```
char *A = DES_encrypt(K_client_TGS, K_client);
send(client_socket, A, strlen(A), 0);
sleep(1);   //发送 A 和 B 之间相隔 1s
char TGT[100];
sprintf(TGT, "<%s,%s,%ld,%s>", ID, inet_ntoa(client_addr->sin_addr),time(NULL) + time_limit, K_client_TGS);
char *B = DES_encrypt(TGT, K_TGS);
```

```
7. send(client_socket, B, strlen(B), 0);
```

客户端接收消息 A，如果消息 A 是报错信息则客户端终止，否则继续接收消息 B 并解密得到$K_{Client-TGS}$

```
1. messageLen = recv(client_socket, bufA, MAX_DATA, 0);
2. bufA[messageLen] = 0;
3. if(strcmp(bufA, "ID is not correct") == 0)
4. {
5.     printf("ID is not correct, please check your ID\n");
6.     return -1;
7. }
8. messageLen = recv(client_socket, bufB, MAX_DATA, 0);
9. bufB[messageLen] = 0;
10.K_client_TGS = DES_decrypt(bufA, K_client);
```

2. **服务认证**

客户端向 TGS 服务器发送消息 C 和消息 D，中间间隔 1s

```
1. sprintf(C, "%s,%s", serviceID, bufB);
2. send(client_socket, C, strlen(C), 0);
3. sleep(1);
4. sprintf(temp, "<%s,%ld>", ID, timestamp);
5. D = DES_encrypt(temp, K_client_TGS);
6. send(client_socket, D, strlen(D), 0);
```

TGS 服务器接收消息 C 和消息 D，利用 sscanf 函数分隔字符串得到 serviceID 和消息 B，然后解密消息 B 得到 TGT，再从 TGT 中分割字符串得到$K_{Client-TGS}$，然后用$K_{Client-TGS}$解密消息 D 得到认证内容，然后分割字符串得到 clientID 和客户端申请服务认证时的时间戳

```
1. int CLen = recv(client_socket, bufC, MAX_DATA, 0);
2. bufC[CLen] = 0;
3. int DLen = recv(client_socket, bufD, MAX_DATA, 0);
4. bufD[DLen] = 0;
5. sscanf(bufC, "%[^,]", serviceID);     //从 bufC 中取出“,”前的字符串
6. int start = strlen(serviceID) + 1;
7. for(int i = start; i < CLen; ++i)
8.     B[i - start] = bufC[i];
9. B[CLen - start] = 0;
10.char * TGT = DES_decrypt(B, K_TGS);
11.sscanf(TGT, "<%[^,], %[^,], %ld, %[^>]>", clientID1, client_addre
   ss, &validity, K_client_TGS);
12.char * authentication = DES_decrypt(bufD, K_client_TGS);
```

```
13.sscanf(authentication, "<%[^,], %ld>", clientID2, &timestamp)
```

TGS 服务器对比消息 B 中分隔出的 clientID 和认证内容中的 clientID，不一样则报错给客户端，然后判断客户端申请认证时的时间戳是否已经超过 TGT 中的有效期，超过则报错给客户端

```
1. if(strcmp(clientID1, clientID2) != 0)
2. {
3.     printf("authentication failed! ClientID is not correct\n");
4.     send(client_socket, "invalid ID", strlen("invalid ID"), 0);
5.     return;
6. }
7. if(timestamp > validity)
8. {
9.     printf("authentication failed! TGT expired\n");
10.    send(client_socket, "invalid TGT", strlen("invalid TGT"), 0);
11.    return;
12.}
```

TGS 服务器生成并发送消息 E 和消息 F

```
1. char temp[MAX_DATA];
2. sprintf(temp, "<%s,%s,%ld,%s>", clientID1, client_address, time(N
   ULL) + time_limit, K_client_SS);
3. char *ST = DES_encrypt(temp, K_SS);
4. sprintf(E, "<%s,%s>", serviceID, ST);
5. send(client_socket, E, strlen(E), 0);
6. sleep(1);
7. F = DES_encrypt(K_client_SS, K_client_TGS);
8. send(client_socket, F, strlen(F), 0);
```

客户端接收消息 E，如果消息 E 是报错信息则客户端终止，否则继续接收消息 F 并解密得到$K_{Client-SS}$

```
1. messageLen = recv(client_socket, bufE, MAX_DATA, 0);
2. bufE[messageLen] = 0;
3. if(strcmp(bufE, "invalid ID") == 0 || strcmp(bufE, "invalid TGT")
    == 0)
4. {
5.     printf("authentication failed, check your ID or TGT!\n");
6.     return -1;
7. }
8. messageLen = recv(client_socket, bufF, MAX_DATA, 0);
9. bufF[messageLen] = 0;
10.K_client_SS = DES_decrypt(bufF, K_client_TGS);
```

3. **服务申请**

客户端向 SS 服务器发送从 TGS 服务器收到的消息 E，然后生成并向 SS 服务器发送消息 G

```
send(client_socket, bufE, strlen(bufE), 0);
sleep(1);
sprintf(temp, "<%s,%ld>", ID, timestamp);
G = DES_encrypt(temp, K_client_SS);
send(client_socket, G, strlen(G), 0);
```

SS 服务器接收消息 E，从消息 E 中分隔出 ST，解密 ST 得到$K_{Client-SS}$。然后接收消息 G 并用$K_{Client-SS}$解密得到认证内容 G，从中分隔出 timestamp 和 clientID

```
int ELen = recv(client_socket, bufE, MAX_DATA, 0);
bufE[ELen] = 0;
sscanf(bufE, "%[^,]", serviceID);
int start = strlen(serviceID) + 1;
for(int i = start; i < ELen; ++i)
    temp[i - start] = bufE[i];
temp[ELen - start] = 0;

ST = DES_decrypt(temp, K_SS);
sscanf(ST, "<%[^,], %[^,], %ld, %[^>]>", clientID1, client_addres
s, &validity, K_client_SS);

int GLen = recv(client_socket, bufG, MAX_DATA, 0);
bufG[GLen] = 0;
authenticationG = DES_decrypt(bufG, K_client_SS);
sscanf(authenticationG, "<%[^,], %ld>", clientID2, &timestamp);
```

SS 服务器对比 ST 和认证 G 中的 clientID 来进行认证

```
if(strcmp(clientID1, clientID2) != 0)
{
    printf("authentication failed! ClientID is not correct\n");
    return;
}
```

SS 服务器将 timestamp + 1 后与 clientID 拼接生成消息 H 发送给客户端

```
sprintf(temp, "<%s,%ld>", clientID1, timestamp + 1);
H = DES_encrypt(temp, K_client_SS);
send(client_socket, H, strlen(H), 0);
```

客户端解密消息 H，若其中的 timestamp 被正确更新，则说明 SS 服务器可信

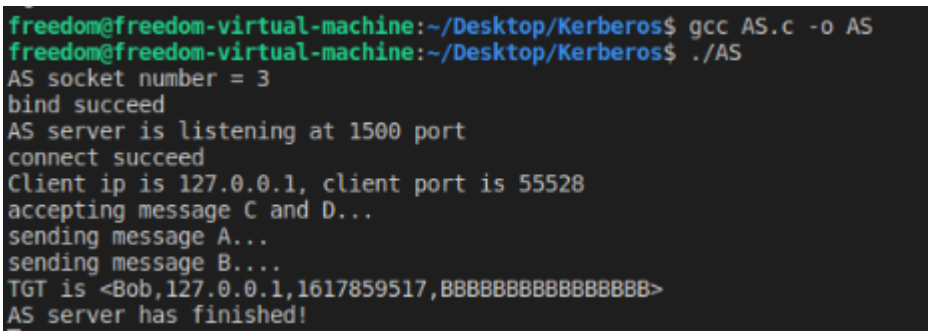
```
messageLen = recv(client_socket, bufH, MAX_DATA, 0);
bufH[messageLen] = 0;
char *H = DES_decrypt(bufH, K_client_SS);
char H_client_ID[MAX_DATA];
time_t new_timestamp;
sscanf(H, "<%[^,], %ld>", H_client_ID, &new_timestamp);

if(timestamp + 1 == new_timestamp)
{
    printf("SS is reliable!\n");
}
else
{
    printf("SS is not reliable!\n");
}
```

然后客户端可以开始使用服务。

六、 编译运行与使用

使用方法：将 5 个文件全部编译后，在 3 个独立的命令行窗口中中先运行 3 个服务端，然后在另一个命令行窗口中运行客户端即可。

AS.c 运行结果：



TGS.c 运行结果：

```
freedom@freedom-virtual-machine:~/Desktop/Kerberos$ gcc TGS.c -o TGS
freedom@freedom-virtual-machine:~/Desktop/Kerberos$ ./TGS
TGS socket number = 3
bind succeed
TGS server is listening at 1600 port
connect succeed
Client ip is 127.0.0.1, client port is 53820
accepting message C...
accepting message D...
service ID is study
decrypting message B...
K_client_TGS is BBBBBBBBBBBBBBB
decrypting message D...
message D: <Bob, 1607859517>
authentication secceed!
sending message E...
ST is <Bob,127.0.0.1,1617859518,DDDDDDDDDDDDDDD>
sending message F...
TGS server has finished!
```

SS.c 运行结果：

```
freedom@freedom-virtual-machine:~/Desktop/Kerberos$ ./SS
SS socket number = 3
bind succeed
SS server is listening at 1700 port
connect succeed
Client ip is 127.0.0.1, client port is 38572
accepting message E...
service ID is <study
ST is <Bob,127.0.0.1,1617859518,DDDDDDDDDDDDDDD>
K_client_SS is DDDDDDDDDDDDDDD
accepting message G...
authentication secceed!
sending message H...
SS server has finished!
```

client.c 运行结果：

```
freedom@freedom-virtual-machine:~/Desktop/Kerberos$ gcc client.c -o client
freedom@freedom-virtual-machine:~/Desktop/Kerberos$ ./client
+--------------------------------------------------+
connect AS server succeed, first stage begin!
sending ID to server...
accepting message A...
accepting message B...
K_client_TGS is BBBBBBBBBBBBBBB
first stage has finished!
+--------------------------------------------------+
connect TGS server succeed, second stage begin!
sending message C...
sending message D...
accepting message E...
accepting message F...
K_client_SS is DDDDDDDDDDDDDDD
second stage has finished!
+--------------------------------------------------+
connect SS server succeed, third stage begin!
sending message E...
sending message G...
accepting message H...
message H is <Bob,1607859520>
SS is reliable!
third stage has finished!
+--------------------------------------------------+
could start service!
```

七、 源代码

AS.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <time.h>
#include <unistd.h>
#include "DES.c"

#define AS_PORT 1500          //端口号
#define BACKLOG 100           //最大监听数
#define MAX_DATA 200          //缓冲区大小
#define time_limit 10000000   //有效期

char ID[] = "Bob";
char K_client[] = "AAAAAAAAAAAAAAAA";
char K_client_TGS[] = "BBBBBBBBBBBBBBBB";
char K_TGS[] = "CCCCCCCCCCCCCCCC";

void serve(struct sockaddr_in * client_addr, int client_socket)
{
    printf("Client ip is %s, client port is %d\n", inet_ntoa(client_addr->sin_addr), htons(client_addr->sin_port));
    printf("accepting message C and D...\n");
    char buf[MAX_DATA];       //储存接收数据
    int messageLen = recv(client_socket, buf, MAX_DATA, 0);
    buf[messageLen] = 0;
    if(strcmp(ID, buf) != 0)
    {
        printf("ID is not correct");
        send(client_socket, "ID is not correct", strlen("ID is not correct"), 0);
        return;
    }
    else
    {
        printf("sending message A...\n");
        char *A = DES_encrypt(K_client_TGS, K_client);
        send(client_socket, A, strlen(A), 0);

        sleep(1);
```

```
41.         printf("sending message B....\n");
42.         char TGT[100];
43.         sprintf(TGT, "<%s,%s,%ld,%s>", ID, inet_ntoa(client_addr-
    >sin_addr),time(NULL) + time_limit, K_client_TGS);
44.         printf("TGT is %s\n", TGT);
45.         char *B = DES_encrypt(TGT, K_TGS);
46.         send(client_socket, B, strlen(B), 0);
47.
48.         printf("AS server has finished!\n");
49.     }
50.}
51.
52.int main()
53.{
54.     int AS_socket, client_socket;      //AS 用于监听的套接字和建立连
    接后的套接字
55.     struct sockaddr_in AS_addr;        //AS 的地址信息结构体，下面有
    具体的属性赋值
56.     struct sockaddr_in client_addr;    //客户端地址信息
57.     int addr_len = sizeof(struct sockaddr_in);
58.
59.     AS_socket = socket(AF_INET, SOCK_STREAM, 0);      //建立 socket
60.     if (AS_socket == -1)
61.     {
62.         perror("AS socket");
63.         return -1;
64.     }
65.     else
66.     {
67.         printf("AS socket number = %d\n", AS_socket);
68.     }
69.     AS_addr.sin_family = AF_INET;                    //用于网络通信
70.     AS_addr.sin_port = htons(AS_PORT);               //端口号
71.     AS_addr.sin_addr.s_addr = htonl(INADDR_ANY);     //IP，括号内容
    表示本机 IP
72.     bzero(&(AS_addr.sin_zero), 8);           //将 sin_zero 字段置 0
73.     if (bind(AS_socket, (struct sockaddr *)&AS_addr, sizeof(struc
    t sockaddr)) < 0) //绑定地址结构体和 socket
74.     {
75.         perror("bind error");
76.         return -1;
77.     }
78.     else
79.     {
```

```
        printf("bind succeed\n");
    }
    listen(AS_socket, BACKLOG);     //开启监听 ， 第二个参数是最大监听数
    printf("AS server is listening at %d port\n", AS_PORT);

    while (1)
    {
        //阻塞直到有客户端连接
        client_socket = accept(AS_socket, (struct sockaddr *)&client_addr, (socklen_t *)&addr_len);
        if (client_socket == -1)
        {
            printf("connect client failed");
            return -1;
        }

        printf("connect succeed\n");
        pid_t pid = fork();
        if(pid == 0)
            serve(&client_addr, client_socket);
    }
     close(AS_socket);
     return 0;
}
```

TGS.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <time.h>
#include <unistd.h>
#include "DES.c"

#define TGS_PORT 1600          //端口号
#define BACKLOG 100            //最大监听数
#define MAX_DATA 200           //缓冲区大小
#define time_limit 10000000    //有效期

char K_TGS[] = "CCCCCCCCCCCCCCCC";
```

```
17.char K_client_SS[] = "DDDDDDDDDDDDDDDD";
18.char K_SS[] = "EEEEEEEEEEEEEEEE";
19.
20.void serve(struct sockaddr_in * client_addr, int client_socket)
21.{
22.    char bufC[MAX_DATA];              //接收 client 发送的消息 C 的
   缓冲区
23.    char bufD[MAX_DATA];              //接收 client 发送的消息 D 的
   缓冲区
24.    char serviceID[MAX_DATA];         //客户端申请的服务 ID
25.    char B[MAX_DATA];                 //消息 C 中包含的消息 B
26.    char clientID1[MAX_DATA];         //消息 C 中包含的消息 B 中得到
   的 clientID
27.    char clientID2[MAX_DATA];         //消息 D 中解压得到的
   clientID
28.    char client_address[MAX_DATA];    //消息 C 中包含的消息 B 中得到
   的客户端地址
29.    time_t validity;                  //消息 C 中包含的消息 B 中得到
   的 TGT 票据有效期
30.    time_t timestamp;                 //申请认证时客户端的时间戳
31.    char K_client_TGS[MAX_DATA];      //Client-TGS 会话密钥
32.    char E[MAX_DATA];                 //发送给 client 的消息 E
33.    char *F;                          //发送给 client 的消息 F
34.
35.    printf("Client ip is %s, client port is %d\n", inet_ntoa(clie
   nt_addr->sin_addr), htons(client_addr->sin_port));
36.    printf("accepting message C...\n");
37.    int CLen = recv(client_socket, bufC, MAX_DATA, 0);
38.    bufC[CLen] = 0;
39.    printf("accepting message D...\n");
40.    int DLen = recv(client_socket, bufD, MAX_DATA, 0);
41.    bufD[DLen] = 0;
42.
43.    //得到 serviceID 和消息 B
44.    sscanf(bufC, "%[^,]", serviceID);  //从 bufC 中取出“,”前的字符串
45.    printf("service ID is %s\n", serviceID);
46.    int start = strlen(serviceID) + 1;
47.    for(int i = start; i < CLen; ++i)
48.        B[i - start] = bufC[i];
49.    B[CLen - start] = 0;
50.
51.    printf("decrypting message B...\n");
52.    char * TGT = DES_decrypt(B, K_TGS);
```

```
53.     sscanf(TGT, "<%[^,], %[^,], %ld, %[^>]>", clientID1, client_a
    ddress, &validity, K_client_TGS);
54.     printf("K_client_TGS is %s\n", K_client_TGS);
55.
56.     printf("decrypting message D...\n");
57.     char * authentication = DES_decrypt(bufD, K_client_TGS);
58.     sscanf(authentication, "<%[^,], %ld>", clientID2, &timestamp)
    ;
59.     printf("message D: <%s, %ld>\n", clientID2, timestamp);
60.
61.     //对比消息B和消息D中的clientID来进行认证
62.     if(strcmp(clientID1, clientID2) != 0)
63.     {
64.         printf("authentication failed! ClientID is not correct\n"
    );
65.         send(client_socket, "invalid ID", strlen("invalid ID"), 0
    );
66.         return;
67.     }
68.     if(timestamp > validity)
69.     {
70.         printf("authentication failed! TGT expired\n");
71.         send(client_socket, "invalid TGT", strlen("invalid TGT"),
     0);
72.         return;
73.     }
74.     printf("authentication secceed!\n");
75.
76.     //生成并发送消息E
77.     printf("sending message E...\n");
78.     char temp[MAX_DATA];
79.     sprintf(temp, "<%s,%s,%ld,%s>", clientID1, client_address, ti
    me(NULL) + time_limit, K_client_SS);
80.     printf("ST is %s\n", temp);
81.     char *ST = DES_encrypt(temp, K_SS);       //服务票据，包括
    clientID、客户网络地址、票据有效期限、Client-SS会话密钥
82.     sprintf(E, "<%s,%s>", serviceID, ST);
83.     send(client_socket, E, strlen(E), 0);
84.
85.     sleep(1);
86.     //生成并发送消息F
87.     printf("sending message F...\n");
88.     F = DES_encrypt(K_client_SS, K_client_TGS);
89.     send(client_socket, F, strlen(F), 0);
```

```
90.
91.     printf("TGS server has finished!\n");
92.}
93.
94.int main()
95.{
96.     int TGS_socket, client_socket;      //TGS 用于监听的套接字和建立连接后的套接字
97.     struct sockaddr_in TGS_addr;        //TGS 的地址信息结构体，下面有具体的属性赋值
98.     struct sockaddr_in client_addr;     //客户端地址信息
99.     int addr_len = sizeof(struct sockaddr_in);
100.
101.
102.     TGS_socket = socket(AF_INET, SOCK_STREAM, 0);      //建立 socket
103.     if (TGS_socket == -1)
104.     {
105.         perror("TGS socket");
106.         return -1;
107.     }
108.     else
109.     {
110.         printf("TGS socket number = %d\n", TGS_socket);
111.     }
112.     TGS_addr.sin_family = AF_INET;                  //用于网络通信
113.     TGS_addr.sin_port = htons(TGS_PORT);            //端口号
114.     TGS_addr.sin_addr.s_addr = htonl(INADDR_ANY);  //IP，括号内容表示本机 IP
115.     bzero(&(TGS_addr.sin_zero), 8);         //将 sin_zero 字段置 0
116.     if (bind(TGS_socket, (struct sockaddr *)&TGS_addr, sizeof(struct sockaddr)) < 0) //绑定地址结构体和 socket
117.     {
118.         perror("bind error");
119.         return -1;
120.     }
121.     else
122.     {
123.         printf("bind succeed\n");
124.     }
125.     listen(TGS_socket, BACKLOG);      //开启监听 ，第二个参数是最大监听数
126.     printf("TGS server is listening at %d port\n", TGS_PORT);
127.
```

```
    while (1)
    {
        //阻塞直到有客户端连接
        client_socket = accept(TGS_socket, (struct sockaddr *)&client_addr, (socklen_t *)&addr_len);
        if (client_socket == -1)
        {
            printf("connect client failed");
            return -1;
        }

        printf("connect succeed\n");
        pid_t pid = fork();
        if(pid == 0)
            serve(&client_addr, client_socket);
    }
    close(TGS_socket);
    return 0;
}
```

SS.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <time.h>
#include <unistd.h>
#include "DES.c"

#define SS_PORT 1700          //端口号
#define BACKLOG 100           //最大监听数
#define MAX_DATA 200          //缓冲区大小
#define time_limit 10000000   //有效期


char K_SS[] = "EEEEEEEEEEEEEEE";

void serve(struct sockaddr_in * client_addr, int client_socket)
{
    char bufE[MAX_DATA];                //接收消息E的缓冲区
    char bufG[MAX_DATA];                //接收消息G的缓冲区
```

```
23.     char *authenticationG;              //从消息 G 中解密得到的认证 G
24.     char serviceID[MAX_DATA];           //消息 E 解密得到
25.     char *ST;                           //服务票据
26.     char K_client_SS[MAX_DATA];
27.     char clientID1[MAX_DATA];           //ST 中得到用户 ID
28.     char clientID2[MAX_DATA];           //消息 G 中解密得到用户
    ID
29.     char client_address[MAX_DATA];      //ST 中得到的客户端地址
30.     time_t validity;                    //ST 中得到的票据有效期
31.     time_t timestamp;                   //申请服务时客户端的时间戳
32.     char *H;                            //发回给客户端的消息 H
33.     char temp[MAX_DATA];                //可重用的临时缓冲区
34.
35.     printf("Client ip is %s, client port is %d\n", inet_ntoa(clie
    nt_addr->sin_addr), htons(client_addr->sin_port));
36.     //接收消息 E
37.     printf("accepting message E...\n");
38.     int ELen = recv(client_socket, bufE, MAX_DATA, 0);
39.     bufE[ELen] = 0;
40.     sscanf(bufE, "%[^,]", serviceID);
41.     int start = strlen(serviceID) + 1;
42.     for(int i = start; i < ELen; ++i)
43.         temp[i - start] = bufE[i];
44.     temp[ELen - start] = 0;
45.     printf("service ID is %s\n", serviceID);
46.
47.     //解密得到 K_client_SS
48.     ST = DES_decrypt(temp, K_SS);
49.     printf("ST is %s\n", ST);
50.     sscanf(ST, "<%[^,], %[^,], %ld, %[^>]>", clientID1, client_ad
    dress, &validity, K_client_SS);
51.     printf("K_client_SS is %s\n", K_client_SS);
52.
53.     //接收消息 G 并解密得到认证内容 G
54.     printf("accepting message G...\n");
55.     int GLen = recv(client_socket, bufG, MAX_DATA, 0);
56.     bufG[GLen] = 0;
57.     authenticationG = DES_decrypt(bufG, K_client_SS);
58.     sscanf(authenticationG, "<%[^,], %ld>", clientID2, &timestamp
    );
59.
60.     //对比 ST 和认证 G 中的 clientID 来进行认证
61.     if(strcmp(clientID1, clientID2) != 0)
62.     {
```

```
        printf("authentication failed! ClientID is not correct\n");
        return;
    }
    printf("authentication secceed!\n");

    //生成并发送消息H
    sprintf(temp, "<%s,%ld>", clientID1, timestamp + 1);
    H = DES_encrypt(temp, K_client_SS);
    printf("sending message H...\n");
    send(client_socket, H, strlen(H), 0);

    printf("SS server has finished!\n");
}

int main()
{
    int SS_socket, client_socket;     //SS 用于监听的套接字和建立连接后的套接字
    struct sockaddr_in SS_addr;       //SS 的地址信息结构体，下面有具体的属性赋值
    struct sockaddr_in client_addr;   //客户端地址信息
    int addr_len = sizeof(struct sockaddr_in);


    SS_socket = socket(AF_INET, SOCK_STREAM, 0);     //建立 socket
    if (SS_socket == -1)
    {
        perror("SS socket");
        return -1;
    }
    else
    {
        printf("SS socket number = %d\n", SS_socket);
    }
    SS_addr.sin_family = AF_INET;                    //用于网络通信
    SS_addr.sin_port = htons(SS_PORT);               //端口号
    SS_addr.sin_addr.s_addr = htonl(INADDR_ANY);     //IP，括号内容表示本机 IP
    bzero(&(SS_addr.sin_zero), 8);           //将 sin_zero 字段置 0
    if (bind(SS_socket, (struct sockaddr *)&SS_addr, sizeof(struct sockaddr)) < 0) //绑定地址结构体和 socket
    {
        perror("bind error");
```

```
        return -1;
    }
    else
    {
        printf("bind succeed\n");
    }
    listen(SS_socket, BACKLOG);     //开启监听 ， 第二个参数是最大监听数
    printf("SS server is listening at %d port\n", SS_PORT);

    while (1)
    {
        //阻塞直到有客户端连接
        client_socket = accept(SS_socket, (struct sockaddr *)&client_addr, (socklen_t *)&addr_len);
        if (client_socket == -1)
        {
            printf("connect client failed");
            return -1;
        }

        printf("connect succeed\n");
        pid_t pid = fork();
        if(pid == 0)
            serve(&client_addr, client_socket);
    }
    close(SS_socket);
    return 0;
}
```

client.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <time.h>
#include "DES.c"

#define AS_PORT 1500
#define TGS_PORT 1600
```

```
#define SS_PORT 1700
#define DEST_IP "127.0.0.1"    //本机回环IP地址
#define MAX_DATA 200           //缓冲区大小

char ID[] = "Bob";
char K_client[] = "AAAAAAAAAAAAAAAA";

struct sockaddr_in * set_sock_addr(int port)
{
    struct sockaddr_in *server_addr =  (struct sockaddr_in *)mall
oc(sizeof(struct sockaddr_in));
    server_addr->sin_family = AF_INET;
    server_addr->sin_addr.s_addr - inet_addr(DEST_IP);
    server_addr->sin_port = htons(port);
    bzero(&(server_addr->sin_zero), 8);
    return server_addr;
}

int create_socket()
{
    int client_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (client_socket == -1)
        perror("client socket");
    return client_socket;
}

int main()
{
    int client_socket;
    int messageLen;                 //recv函数返回值
    time_t timestamp;               //客户端申请服务时的时间戳，每个
阶段用到时要更新
    char temp[MAX_DATA];            //可重用的临时缓冲区
    char bufA[MAX_DATA];            //存放消息A的缓冲区
    char bufB[MAX_DATA];            //存放消息B的缓冲区
    char C[MAX_DATA];               //消息C
    char *D;                        //消息D
    char bufE[MAX_DATA];            //存放消息E的缓冲区
    char bufF[MAX_DATA];            //存放消息F的缓冲区
    char *G;                        //消息G
    char bufH[MAX_DATA];            //存放消息H的缓冲区
    char *K_client_TGS;
    char *K_client_SS;
    char serviceID[] = "study";     //申请服务的ID
```

```
55.
56.
57.     //设置三个服务器的地址
58.     struct sockaddr_in * AS_addr = set_sock_addr(AS_PORT);
59.     struct sockaddr_in * TGS_addr = set_sock_addr(TGS_PORT);
60.     struct sockaddr_in * SS_addr = set_sock_addr(SS_PORT);
61.
62.     //第一阶段
63.     if((client_socket = create_socket()) == -1)
64.         return -1;
65.
66.     //与AS 服务器建立连接
67.     if (connect(client_socket, (struct sockaddr *)AS_addr, sizeof
    (struct sockaddr_in)) == -1)
68.     {
69.         perror("AS connect");
70.         return -1;
71.     }
72.     printf("+-----------------------------------------------------
    -+\n");
73.     printf("connect AS server succeed, first stage begin!\n");
74.
75.     printf("sending ID to server...\n");
76.     send(client_socket, ID, strlen(ID), 0);
77.
78.     printf("accepting message A...\n");
79.     messageLen = recv(client_socket, bufA, MAX_DATA, 0);
80.     bufA[messageLen] = 0;
81.     if(strcmp(bufA, "ID is not correct") == 0)
82.     {
83.         printf("ID is not correct, please check your ID\n");
84.         return -1;
85.     }
86.
87.     printf("accepting message B...\n");
88.     messageLen = recv(client_socket, bufB, MAX_DATA, 0);
89.     bufB[messageLen] = 0;
90.
91.     K_client_TGS = DES_decrypt(bufA, K_client);
92.     printf("K_client_TGS is %s\n", K_client_TGS);
93.     close(client_socket);
94.     printf("first stage has finished!\n");
95.     printf("+-----------------------------------------------------
    -+\n");
```

```
96.
97.     //第二阶段
98.     if((client_socket = create_socket()) == -1)
99.         return -1;
100.
101.     //与TGS 服务器建立连接
102.     if (connect(client_socket, (struct sockaddr *)TGS_addr, size
    of(struct sockaddr_in)) == -1)
103.     {
104.         perror("TGS connect");
105.         return -1;
106.     }
107.     printf("connect TGS server succeed, second stage begin!\n");
108.     timestamp = time(NULL);
109.
110.     //生成并发送消息C
111.     printf("sending message C...\n");
112.     sprintf(C, "%s,%s", serviceID, bufB);
113.     send(client_socket, C, strlen(C), 0);
114.
115.     sleep(1);
116.     //生成并发送消息D
117.     printf("sending message D...\n");
118.     sprintf(temp, "<%s,%ld>", ID, timestamp);
119.     D = DES_encrypt(temp, K_client_TGS);
120.     send(client_socket, D, strlen(D), 0);
121.
122.     printf("accepting message E...\n");
123.     messageLen = recv(client_socket, bufE, MAX_DATA, 0);
124.     bufE[messageLen] = 0;
125.
126.     //认证过程
127.     if(strcmp(bufE, "invalid ID") == 0 || strcmp(bufE, "invalid
    TGT") == 0)
128.     {
129.         printf("authentication failed, check your ID or TGT!\n")
    ;
130.         return -1;
131.     }
132.
133.     printf("accepting message F...\n");
134.     messageLen = recv(client_socket, bufF, MAX_DATA, 0);
135.     bufF[messageLen] = 0;
136.     K_client_SS = DES_decrypt(bufF, K_client_TGS);
```

```
137.     printf("K_client_SS is %s\n", K_client_SS);
138.     close(client_socket);
139.     printf("second stage has finished!\n");
140.     printf("+-------------------------------------------------
    --+\n");
141. 
142.     //第三阶段
143.     if((client_socket = create_socket()) == -1)
144.         return -1;
145. 
146.     //与SS 服务器建立连接
147.     if (connect(client_socket, (struct sockaddr *)SS_addr, sizeo
    f(struct sockaddr_in)) == -1)
148.     {
149.         perror("TGS connect");
150.         return -1;
151.     }
152.     printf("connect SS server succeed, third stage begin!\n");
153.     timestamp = time(NULL);
154. 
155.     //发送消息E
156.     printf("sending message E...\n");
157.     send(client_socket, bufE, strlen(bufE), 0);
158.     sleep(1);
159. 
160.     //生成并发送消息G
161.     printf("sending message G...\n");
162.     sprintf(temp, "<%s,%ld>", ID, timestamp);
163.     G = DES_encrypt(temp, K_client_SS);
164.     send(client_socket, G, strlen(G), 0);
165. 
166.     //接收消息H
167.     printf("accepting message H...\n");
168.     messageLen = recv(client_socket, bufH, MAX_DATA, 0);
169.     bufH[messageLen] = 0;
170.     char *H = DES_decrypt(bufH, K_client_SS);
171.     printf("message H is %s\n", H);
172.     char H_client_ID[MAX_DATA];
173.     time_t new_timestamp;
174.     sscanf(H, "<%[^,], %ld>", H_client_ID, &new_timestamp);
175. 
176.     if(timestamp + 1 == new_timestamp)
177.     {
178.         printf("SS is reliable!\n");
```

```
179.         printf("third stage has finished!\n");
180.         printf("+-----------------------------------------------
   ------+\n");
181.         printf("could start service!\n");
182.     }
183.     else
184.     {
185.         printf("SS is not reliable!\n");
186.     }
187.     return 0;
188. }
```