

# 廈門大學



信息學院軟件工程系

## 《計算機網絡》實驗報告

題 目 实验四 CISCO IOS 路由器基本配置

班 級 軟件工程 2019 級 1 班

姓 名 李世豪

學 号 22920192204299

實驗時間 2021 年 4 月 23 日

2021 年 6 月 04 日

# 填写说明

- 1、本文件为 Word 模板文件，建议使用 Microsoft Word 2019 打开，在可填写的区域中如实填写；
- 2、填表时，勿破坏排版，勿修改字体字号，打印成 PDF 文件提交；
- 3、文件总大小尽量控制在 1MB 以下，勿超过 5MB；
- 4、应将材料清单上传在代码托管平台上；
- 5、在学期最后一节课前按要求打包发送至 [cni21@qq.com](mailto:cni21@qq.com)。

## 1 实验目的

通过完成实验，掌握应用层文件传输的原理；了解传输过程中传输层协议选用、应用层协议设计和协议开发等概念。

某远程桌面连接软件 A 需要有一个许可证程序来保证合法运行。这样的许可证（License）是带有限制的。用户提供支付费用，获得一个若干人（如：10 人或 50 人）的许可证。规则如下：

1、某组织管理员在购买许可证时，输入用户名、口令和许可证类型，许可证程序返回一个由 10 个数字组成的序列号。

2、该组织的用户第一次使用软件 A 时，输入序列号。 3、该组织用户运行软件时，向许可证服务器发送验证。

4、许可证服务器查询得到该序列号的使用人数，如果未到达上限，则返回授权指令；否则，返回拒绝指令。

5、软件 A 得到授权指令，允许用户使用软件。否则，提示用户稍后再试，退出程序。

当软件 A 或非正常退出（崩溃被其它程序中止）时，许可证服务器应在扣除使用人数时剔除它。可以使软件 A 定期（如：30 分钟）向服务器报告其状态，超过一定时间没有收到报告时，则认定崩溃。

当许可证服务器崩溃时，软件 A 应能在重新启动时恢复。许可证服务器重启后，如果新的软件 A 前来连接，服务器不可以因接受其连接而拒绝已认证用户的连接。

## 2 实验环境

MAC bigsur 11.3 类 UNIX 系统

C/C++ gcc 编译

## 3 实验结果

1. 用户访问时，如没有序列表，向服务器进行购买，发送购买请求，服务器返回一个长度为 10 的随机数列，并记录在案。用户下一次运行程序时，只需要输入序列号即可，或自动检查是否已有认证序列

服务器处理购买服务凭证：

```
//购买服务凭证
void creating_license_key(char* response){
    char sequense[10];
    for (size_t i = 0; i < 10; i++)
    {
        srand((int)time(0));
        sequense[i]=48+random()%10;
    }
    sprintf(response,"%s",sequense);
    FILE*
key_file=fopen("~/Users/haruki_9/Documents/experiment8/LicenseProject_sec/LicenseProject_server/server_log/auth_key.txt"
,"w");
    fprintf(key_file,"%s",sequense);
    fclose(key_file);
}
void handle_buy_request(char* response){
    char key[20];
    creating_license_key(key);
    sprintf(response,"RETK %s",key);
```

```

}
```

客户端发送购买请求，并记录在本地文件中：

```

bool buy_auth_key(char* key){
    char request[30];
    char response[30];
    sprintf(request,"KBUY %d",getpid());
    int
ret=sendto(client_sock_fd,request,sizeof(request),0,(struct
sockaddr*)&server_addr,server_addr_len);
    if (ret== -1)
    {
        printf("buy_auth_key request sending fail.\n");
        return false;
    }
    ret=recvfrom(client_sock_fd,response,30,0,NULL,NULL);
    if (ret== -1)
    {
        printf("recv auth_key fail.\n");
        return false;
    }
    memcpy(key,response+5,10);
    printf("%s",key);
    FILE*
file=fopen("/Users/haruki_9/Documents/expriment8/LicensePro
ject_Client/LicenseProject_Client/client_log/auth_key.txt",
"w");
    fprintf(file,"%s\n",key);
    fclose(file);
    return true;
}

```

## 2. 用户第一次使用时，输入序列号

用户第一次使用时，在命令行中第一个参数输入 10 个序列号，则客户端会自动发送验证和请求凭证请求；若未输入参数，客户端会自动检查本地文件中是否已有序列号，若有则发送请求凭证请求，若无则返回错误信息（未拥有许可证序列），并结束程序。

```

if (args>=1&&strcmp(argv[1], "/")!=0)
{
    char key[30];
    strcpy(key, argv[1]);
    if (confirm_auth_key(key))
    {
        have_auth_key=true;
    }
    else{
        have_auth_key=false;
    }
}
else{
    search_local_auth_key();
}

if (!have_auth_key)
{
    char key[30];
    if(buy_auth_key(key)){
        printf("get auth_key: %s\n",key);
        printf("auto-use key next time run this
program.\n");
    }
    exit(0);
}

```

3. 用户运行，向服务器发送 key 凭证请求，若人数未达上限，则授权给用户；否则拒绝用户进行程序访问

服务器端处理请求主函数：

```

void deal_with_request(char* request,sockaddr_in
&client_addr){
    log_record("begin.", SERVER);
    char response[30];
    memset(response, 0, 30);
    log_record(request, CLIENT);
    printf("%s\n",request);
    if (strncmp("KBUY", request, 4)==0)
    {
        handle_buy_request(response);
    }
    else if(strncmp(request, "COMF", 4)==0){

```

```

        handle_COMF(request, response);
    }
    else if (strncmp(request,"HELLO",4)==0)
    {
        handle_HELLO(request, response);
    }
    else if (strncmp(request,"GBYE",4)==0)
    {
        handle_GOODBYE(request, response);
    }
    else
    {
        printf("Request not valid.");
        sprintf(response,"FAIL invalid request");
    }
    socklen_t len=sizeof(client_addr);
    printf("%s\n",response);
    int
ret=sendto(server_sock_fd,response,sizeof(response),0,(struct sockaddr*)&client_addr,len);
    if (ret== -1)
    {
        printf("server send response fail.");
        exit(-1);
    }
    log_record(response, SERVER);
    return;
}

```

处理 HELLO 请求：

```

void handle_HELLO(char* request,char* response){
    int pid=atoi(request+5);
    if (avail_ticket_num==0)
    {
        sprintf(response,"FAIL ticket not available");
        return;
    }

    for (int i = 0; i < MAX_ONLINES; i++)
    {
        if (ticket_list[i]==0)
        {
            ticket_list[i]=pid;

```

```

        memset(response, 0, 30);
        sprintf(response,"TICK %d.%d",pid,i+1);
        avail_ticket_num--;
        break;
    }
}

}

```

处理 GOODBYE 请求:

```

void handle_GOODBYE(char* request,char* response){
    char* token1=strtok(request+5,".");
    char* token2=strtok(NULL, ".");
    int pid=atoi(token1);
    int number=atoi(token2);
    if (pid==ticket_list[number-1])
    {
        memset(response, 0, 30);
        sprintf(response,"THANX return ticket success");
        ticket_list[number-1]=0;
        avail_ticket_num++;
        return ;
    }
    memset(response, 0, 30);
    sprintf(response,"THANX return ticket fails");
}

```

4. 客户端发送请求函数如下，发送 HELLO 和 GOODBYE 请求:

```

int get_ticket(){
    if (have_ticket==1)
    {
        return 0;
    }
    char response[30];
    char requestbuff[30];
    bzero(requestbuff,sizeof(requestbuff));
    sprintf(requestbuff,"HELO %d",pid);
    transfer_info(requestbuff,response);
    if (strlen(response)==0)
    {
        printf("get response from server fail.\n");
    }
}

```

```

        return -1;
    }
    //接下来判断得到的响应信息
    if (strcmp(response,"TICK", 4)==0)
    {
        /* code */
        have_ticket=1;
        strcpy(ticket_str,response+5);
        printf("get ticketed.\n");
        return 0;
    }
    if (strcmp(response,"FAIL",4))
    {
        printf("get ticketed fail.\n");
        return -1;
    }
    printf("response message form not support.\n");
    return -1;
}

//将空闲的许可证返还
int release_ticket(){
    if (have_ticket==0)
    {
        return 0;
    }

    char request[30];
    char response[30];
    memset(request, 0, 30);
    sprintf(request,"GBYE %s",ticket_str);
    transfer_info(request,response);
    if (strlen(response)==0)
    {
        /* code */
        printf("get response from server fail.\n");
        return -1;
    }
    if (strcmp(response,"THANX",5)==0)
    {
        /* code */
        bzero(ticket_str,sizeof(ticket_str));
        have_ticket=0;
        printf("return ticket success.\n");
        return 0;
    }
}

```

```

    printf("return ticket fail.\n");
    return -1;
}

```

## 5. 处理服务器崩溃问题和客户端未正常返回凭证问题。

(1)。处理服务器崩溃：这里若是一一发送确认请求到客户端，未必能保证一定得到正确的反馈，因此默认客户端发送请求时，若当前凭证没有人使用，则认为该凭证在服务器未崩溃前属于该客户，又或者使用本地文件实时记录 ticket 凭证的使用情况，但是只适合与小型的服务器，一旦大型服务器可能出现增加崩溃现象。这里不再展示代码。

(2)。处理客户端未正常返回：设置定时器，使用 signal 函数，每隔一段时间，服务器对在运行客户端逐个发送确认运行请求，如果客户端返回信息，则认为其仍在运行，否则认为该客户端未正常退出。

服务器处理崩溃部分代码如下：

```

signal(SIGALRM, handle_client_lost);
alarm(30);
void handle_client_lost(int arg){
    char* request="ALIV";
    char response[4];
    for (int i=0; i<MAX_ONLINES; i++) {
        if (ticket_list[i]!=0) {
            int len=sizeof(sockaddr);
            int ret=sendto(server_sock_fd, request,
sizeof(response), 0, &tickets_list[i].addr, len);
            if (ret== -1) {
                printf("[Server]: fail to send alive
check.\n");
                return;
            }
            ret=recvfrom(server_sock_fd, &response, 4, 0,
&tickets_list[i].addr, (socklen_t*)&len);
            if (ret== -1) {

```

```

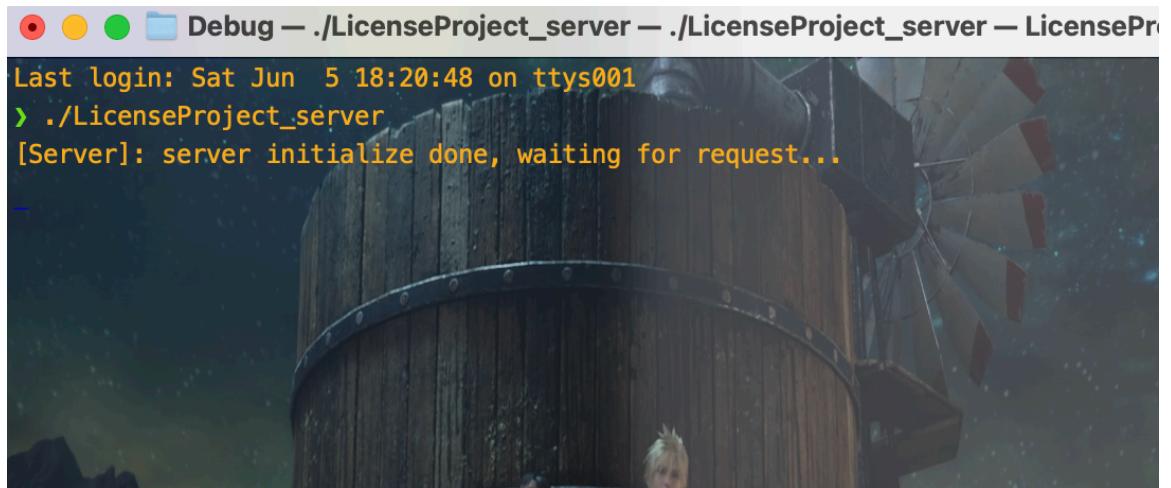
        printf("[Server]: fail to recv live
response.\n");
        return;
    }
    if (memcmp(&response, "LIVE", 4)!=0) {
        printf("[Server]: reclaim ticket
success.\n");
        ticket_list[i]=0;
        tickets_list[i].pid=0;
        bzero(&tickets_list[i].addr,sizeof(sockaddr
));
    }
}
alarm(30);
}

```

客户端响应：这里客户端响应需要使用多线程编程，但是本人并不会多线程而且没有那么多的时间，因此就不实现客户端响应这个功能先了，同时也就是说并没有完成这个处理客户端崩溃的问题。

程序运行效果如下：

1. 开始运行 Server:

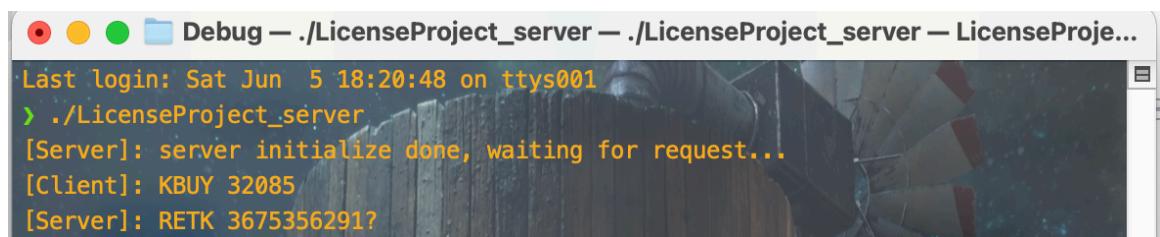


2. 用户第一次运行程序并发送购买认证序列号请求：



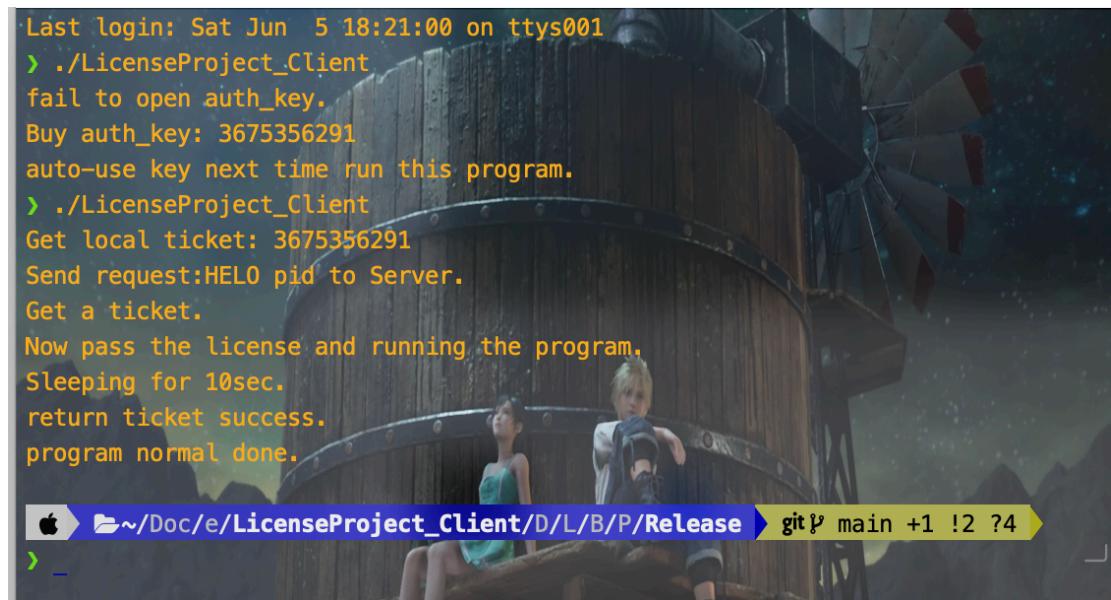
```
Last login: Sat Jun  5 18:21:00 on ttys001
> ./LicenseProject_Client
fail to open auth_key.
Buy auth_key: 3675356291
auto-use key next time run this program.
```

3. 服务器处理用户购买请求:



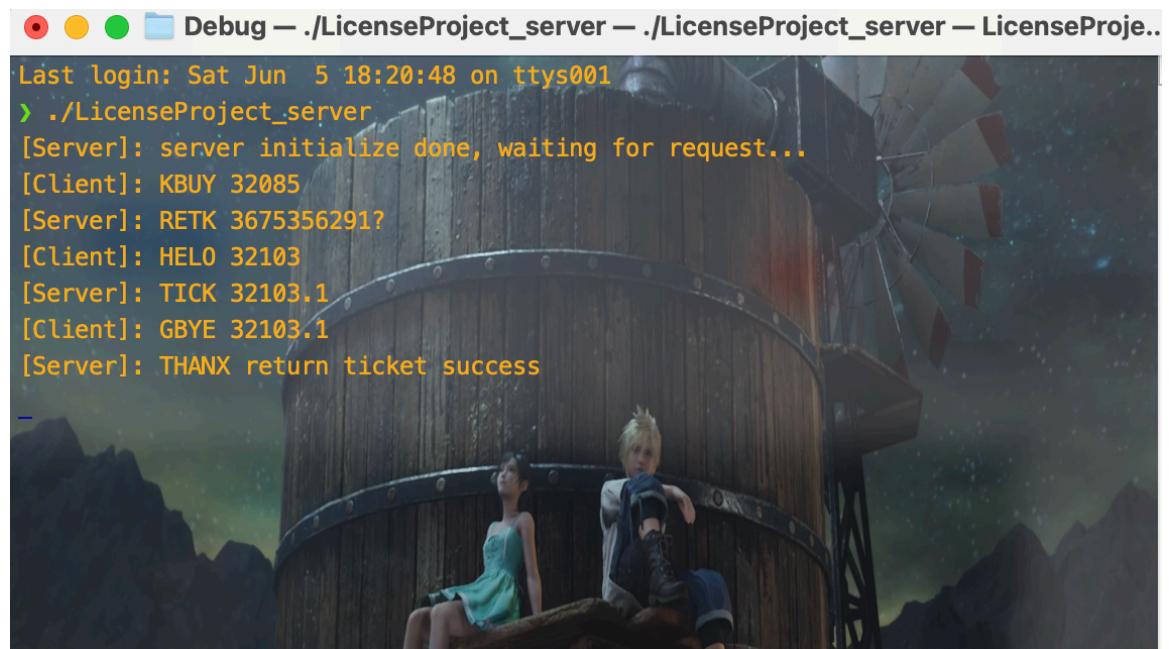
```
Last login: Sat Jun  5 18:20:48 on ttys001
> ./LicenseProject_server
[Server]: server initialize done, waiting for request...
[Client]: KBUY 32085
[Server]: RETK 3675356291?
```

3. 用户获得认证序列号后第一次运行程序:



```
Last login: Sat Jun  5 18:21:00 on ttys001
> ./LicenseProject_Client
fail to open auth_key.
Buy auth_key: 3675356291
auto-use key next time run this program.
> ./LicenseProject_Client
Get local ticket: 3675356291
Send request:HELLO pid to Server.
Get a ticket.
Now pass the license and running the program.
Sleeping for 10sec.
return ticket success.
program normal done.
```

4. 服务器处理用户认证请求:

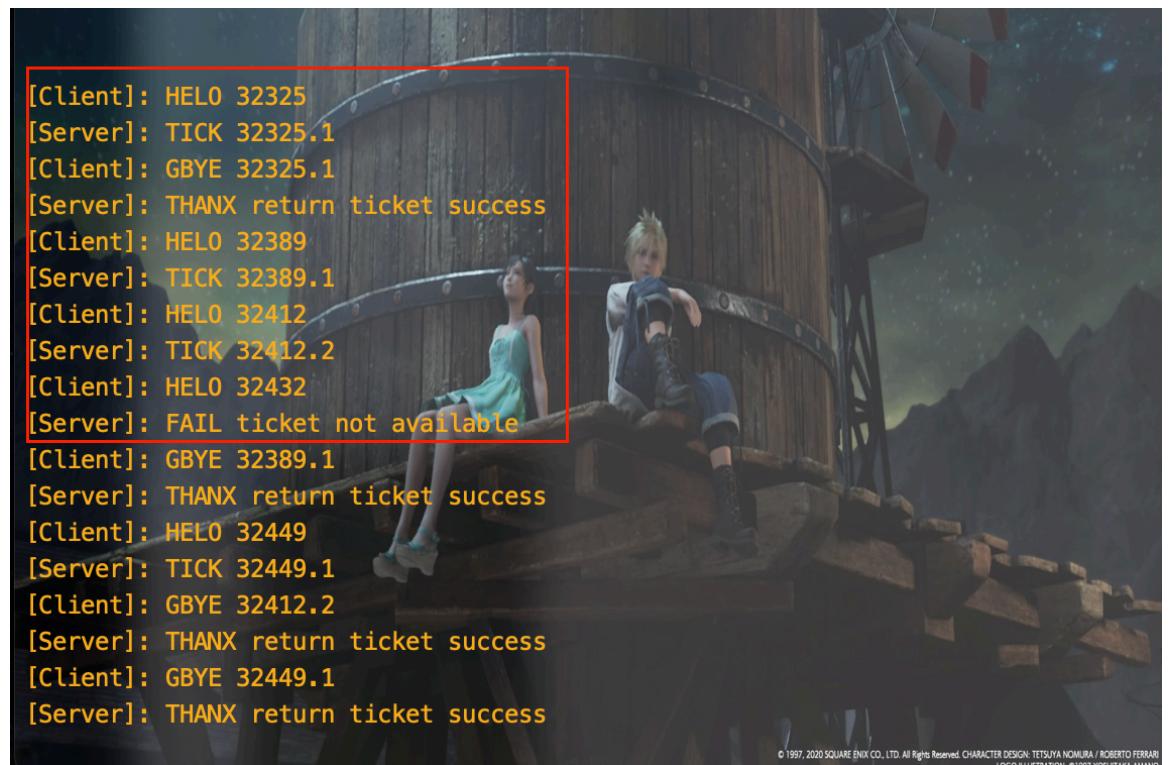


```

Debug — ./LicenseProject_server — ./LicenseProject_server — LicenseProje..
Last login: Sat Jun  5 18:20:48 on ttys001
> ./LicenseProject_server
[Server]: server initialize done, waiting for request...
[Client]: KBUY 32085
[Server]: RETK 3675356291?
[Client]: HELO 32103
[Server]: TICK 32103.1
[Client]: GBYE 32103.1
[Server]: THANX return ticket success

```

5. 服务器处理认证 tickets 队列满状态（此时为了测试设置最大同时在线数为 2 个客户端）：



```

[Client]: HELO 32325
[Server]: TICK 32325.1
[Client]: GBYE 32325.1
[Server]: THANX return ticket success
[Client]: HELO 32389
[Server]: TICK 32389.1
[Client]: HELO 32412
[Server]: TICK 32412.2
[Client]: HELO 32432
[Server]: FAIL ticket not available
[Client]: GBYE 32389.1
[Server]: THANX return ticket success
[Client]: HELO 32449
[Server]: TICK 32449.1
[Client]: GBYE 32412.2
[Server]: THANX return ticket success
[Client]: GBYE 32449.1
[Server]: THANX return ticket success

```

C 1997, 2020 SQUARE ENIX CO., LTD. All Rights Reserved. CHARACTER DESIGN: TETSUYA NOMURA / ROBERTO FERRARI  
LOGO ILLUSTRATION: ©1997 YOSHITAKA AMANO

可以看到，当有超过两个同时在使用的程序时，第 3 个程序请求认可 tickets 将会被拒绝。

最后建立所谓的网页管理后台基本是没有能力实现，所以这里就不再去花时间去做了。这里只是完成最基本的服务器与客户端最基本的自定义的通信协议，而在此之外的对于多线程处理服务器崩溃，处理客户端未正常返回票证这里不做回应，也是本人的能力比较差，不足以能够完成这些要求，但个人追求不高，只求能够基本完成任务即可，虽然实际上也没有真正完成就是了，但有一句话说得好，做到自己能做的就好，其余的就交给时间去处理。

## 4 实验代码

本次实验的代码已上传于以下代码仓库：代码放置于 Github，地址如下：

Github: [https://github.com/Haruki9/Computer-Network\\_Labs/tree/main](https://github.com/Haruki9/Computer-Network_Labs/tree/main)

Gitee: [https://gitee.com/haruki9/computer-network\\_labs](https://gitee.com/haruki9/computer-network_labs)

## 5 实验总结

通过本次实验，了解了怎么制定自定义的协议，即需要通过客户端和服务器的互相应答配置，同时通过编程加深了 SOCKET 的使用和深度了解。