

****

信息学院软件工程系

《计算机网络》实验报告

**题　　目 实验四　CISCO IOS 路由器基本配置**

**班　　级 软件工程2019级1班**

**姓　　名 李世豪**

**学　　号 22920192204299**

**实验时间 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。

# 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。

# 实验环境

MAC bigsur 11.3 类UNIX系统

C/C++ gcc编译

# 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/expriment8/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/LicenseProject\_Client/LicenseProject\_Client/client\_log/auth\_key.txt","w");

    fprintf(file,"%s\n",key);

    fclose(file);

**return** **true**;

}

1. 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** (comfirm\_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);

    }

1. 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,"HELO",4)==0)

    {

        handle\_HELLO(request,response);

    }

**else** **if** (strncmp(request,"GBYE",4)==0)

    {

        handle\_GOODBYTE(request,response);

    }

**else**

    {

        printf("Request not vailed.");

        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\_GOODBYTE(**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");

}

1. 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** (strncmp(response,"TICK", 4)==0)

    {

        /\* code \*/

        have\_ticket=1;

        strcpy(ticket\_str,response+5);

        printf("get ticketed.\n");

**return** 0;

    }

**if** (strncmp(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** (strncmp(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;

}

1. 处理服务器崩溃问题和客户端未正常返回凭证问题。
2. 。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。
3. 。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));

            }

        }

    }

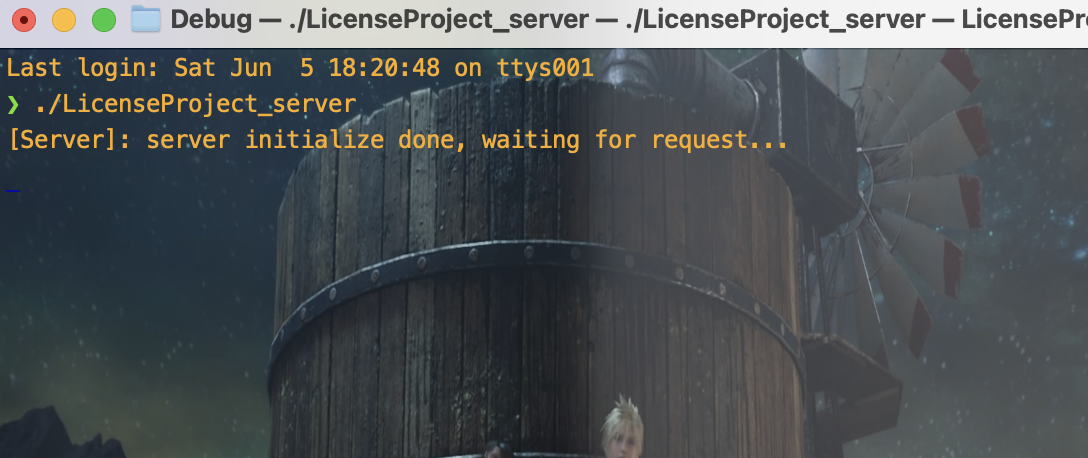
    alarm(30);

}

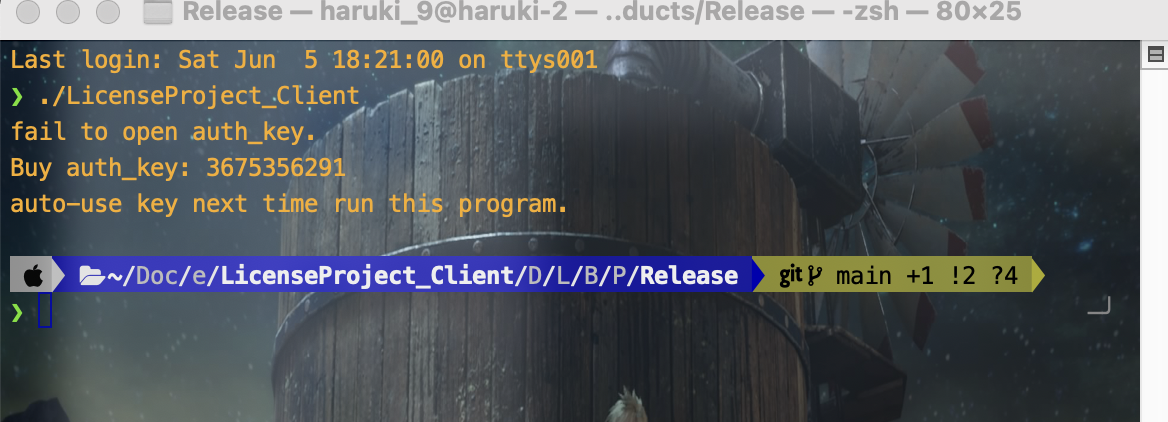
客户端响应：这里客户端响应需要使用多线程编程，但是本人并不会多线程而且没有那么多的时间，因此就不实现客户端响应这个功能先了，同时也就是说并没有完成这个处理客户端崩溃的问题。

程序运行效果如下：

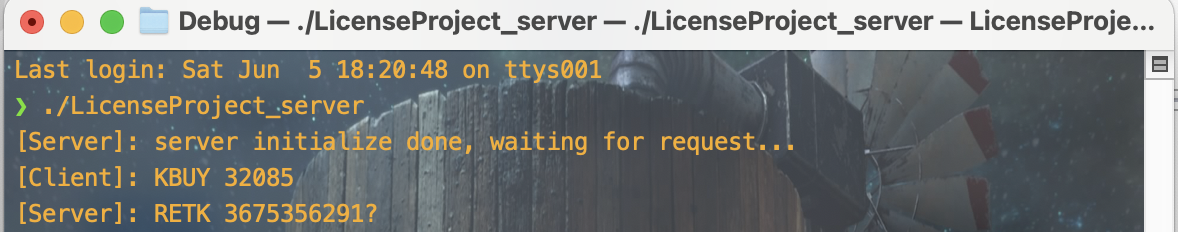
1. 开始运行Server：



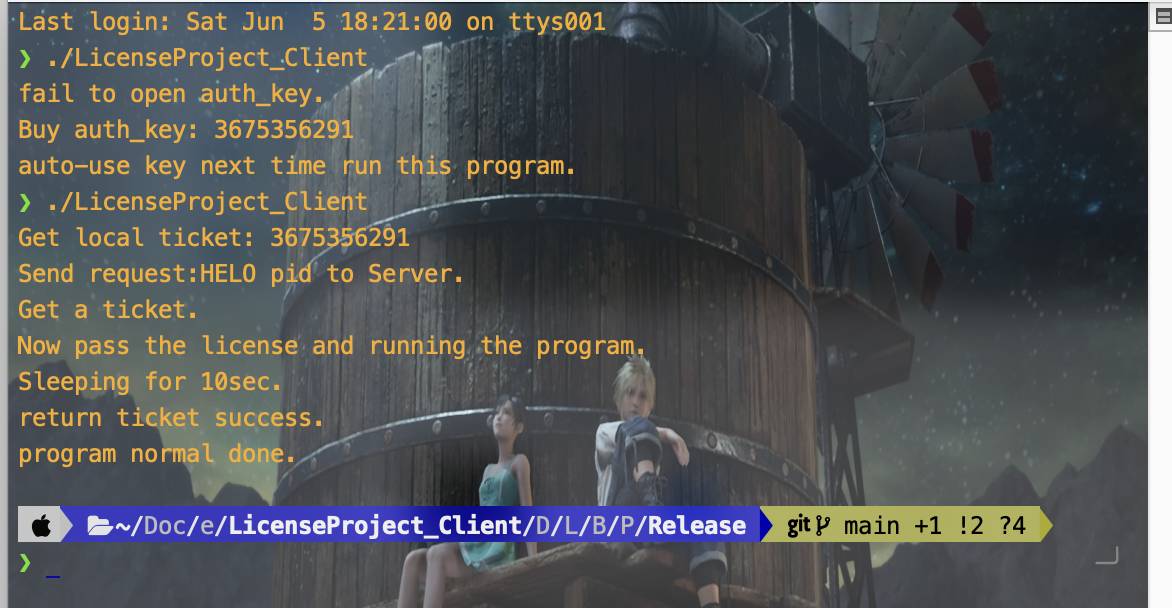
2. 用户第一次运行程序并发送购买认证序列号请求：



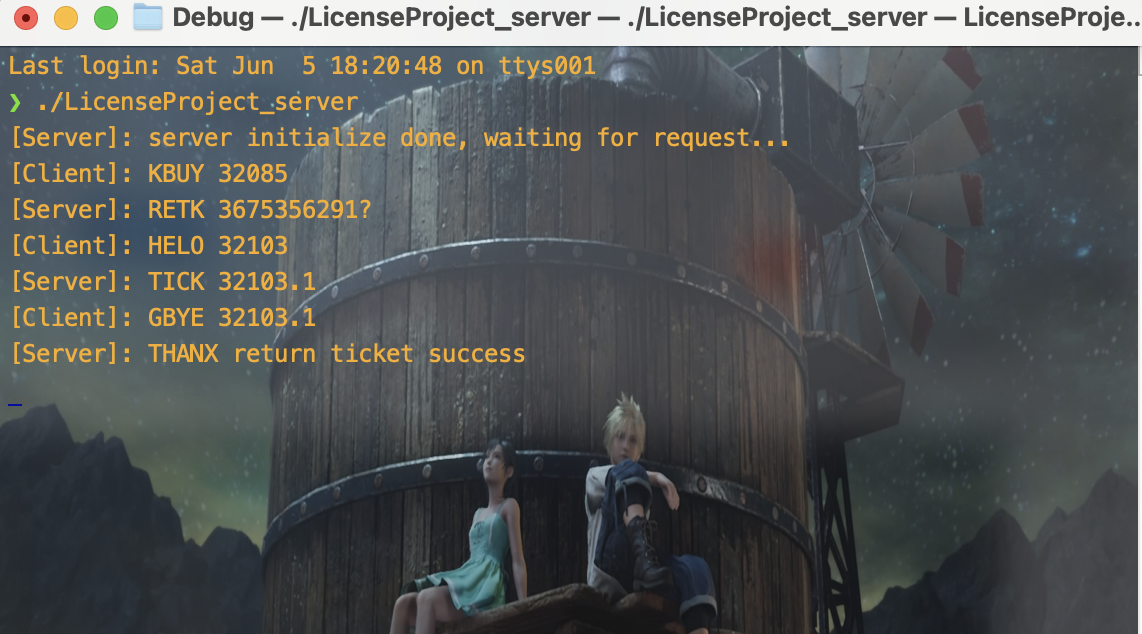
1. 服务器处理用户购买请求：



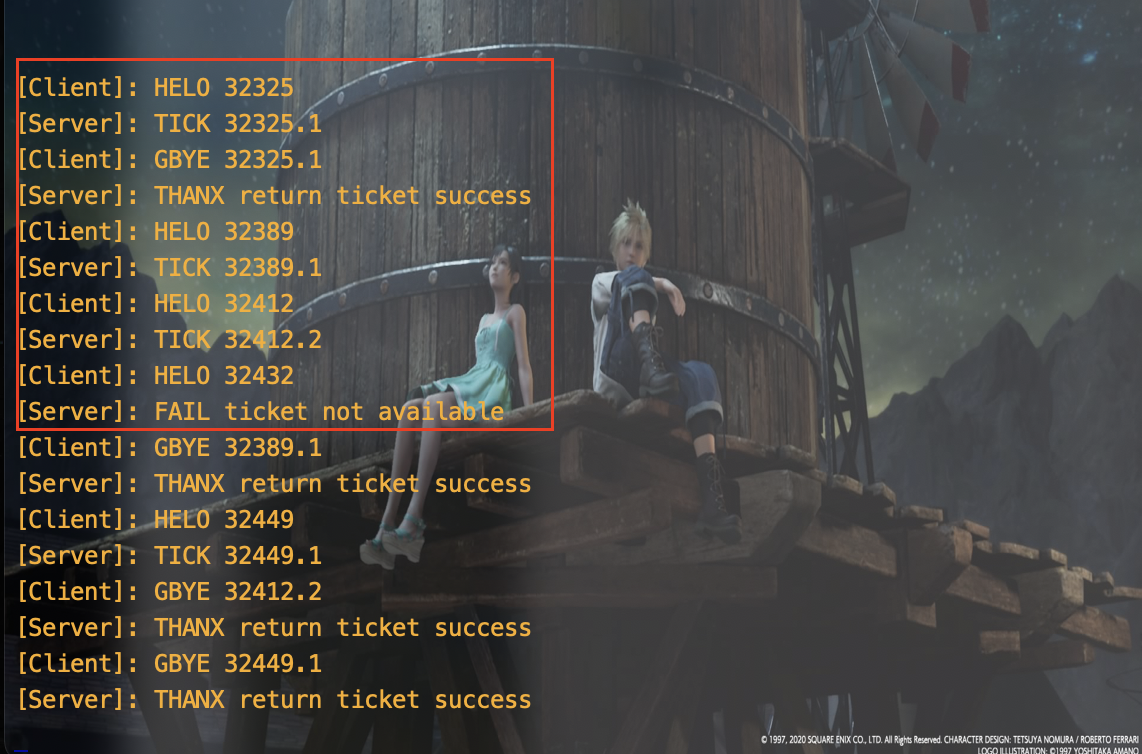
1. 用户获得认证序列号后第一次运行程序：



1. 服务器处理用户认证请求：



1. 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）：



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。

# 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>

Gitee：<https://gitee.com/haruki9/computer-network_-labs>

# 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。