

廈門大學



信息学院软件工程系

《计算机网络》实验报告

题 目 实验五 Socket API 许可认证软件

班 级 软件工程 2021 级卓越班

姓 名 黄子安

学 号 22920212204396

实验时间 2023 年 5 月 10 日

2023 年 5 月 10 日

填写说明

- 1、本文件为 Word 模板文件，建议使用 Microsoft Word 2021 打开，在可填写的区域中如实填写；
- 2、填表时勿改变字体字号，保持排版工整，打印为 PDF 文件提交；
- 3、文件总大小尽量控制在 1MB 以下，最大勿超过 5MB；
- 4、应将材料清单上传在代码托管平台上；
- 5、在实验课结束 14 天内，按原文件发送至课程 FTP 指定位置。

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。

2 实验环境

Windows11, C++, Socket API

3 实验结果

需求分析

- 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。

软件实现

1.设计概况

需要根据用户提供的用户名、口令和许可证类型返回以一个指定的许可证序列号，考虑到提供序列号和后续接收用户状态报送这两个功能实际上对服务器的需求差别较大，前者一般提供完序列号即可完成任务，后者可能会面临大量用户同时接入的并发处理，可能对设备的需求较高，因此为了满足需求将这两个程序拆分开，分为许可证提供服务器和状态报送服务器，便于管理员在不同主机上运行。

2.许可证提供服务器

使用 Socket API 获取用户输入的用户名、口令和类型，之后存入数据库并从序列号池中分配一个序列号发送给用户

```
while (true)
{
    // 接受客户端连接
    clientSocket = accept(serverSocket, (struct sockaddr*)&clientAddress, &clientAddressLength);
    if (clientSocket == INVALID_SOCKET) {
        std::cerr << "接受连接失败" << std::endl;
        continue;
    }

    char account[80], password[80], type[80];
    if (recv(clientSocket, account, 1024, 0) < 0)
    {
        cerr << "接受数据失败" << endl;
    }

    if (recv(clientSocket, password, 1024, 0) < 0)
    {
        cerr << "接受数据失败" << endl;
    }

    if (recv(clientSocket, type, 1024, 0) < 0)
    {
        cerr << "接受数据失败" << endl;
    }

    string license=get_license(account,password,type);
    cout << time_now() << " 用户: " << account << "接入, 并分配许可证" << license << endl;
    send(clientSocket, license.c_str(), license.size() + 1, 0);
}
```

```
string get_license(char account[],char password[],char type[])
{
    //todo:将信息存入数据库
    //todo:从许可证池中获取一个许可证, 这里用随机数代替
    int license = rand() + 1000000000;
    return to_string(license);
}

const int PORT = 6666;
```

3.状态报送服务器

该服务器可能会有多个用户接入，为了并发处理这些用户需求，需要在创建 Socket 之后使用多线程处理客户端接入。

```
while (true)
{
    // 接受客户端连接
    clientSocket = accept(serverSocket, (struct sockaddr*)&clientAddress, &clientAddressLength)
    if (clientSocket == INVALID_SOCKET) {
        std::cerr << "接受连接失败" << std::endl;
        continue;
    }
    // 创建线程处理客户端请求
    std::thread clientThreadObj(clientThread, clientSocket);
    clientThreadObj.detach(); // 分离线程，使其在完成任务后自动释放资源
}
```

为了区分同一序列号下的不同用户，使用 IP 地址来标志区分这些用户，在线程中获取客户端的 IP

```
// 处理客户端请求的线程函数
void clientThread(SOCKET clientSocket)
{
    // 获取客户端地址信息
    sockaddr_in clientAddr;
    int addrLen = sizeof(clientAddr);
    getpeername(clientSocket, (struct sockaddr*)&clientAddr, &addrLen);

    // 将客户端地址转换为字符串形式
    char ip[INET_ADDRSTRLEN];
    inet_ntop(AF_INET, &(clientAddr.sin_addr), ip, INET_ADDRSTRLEN);
    char buffer[1024] = { 0 };
    int bytesRead = recv(clientSocket, buffer, 1024, 0);
}
```

之后判断该许可证序列号的用户接入数量是否已满，在这里使用 C++STL 中的 map 来实现序列号和用户数量的映射关系，根据用户数量返回信息给客户端，之后客户端根据这些反馈的信息来决定用户是否有权使用当前桌面软件

除此之外这里还有一个状态接受功能，在线程中开启一个循环持续监听用户的报送，当客户端主动断开连接或者因为崩溃断开连接都会导致客户端 Socket 直接断开，会在服务器日志中输出用户退出的日志

```

if (bytesRead < 0)
{
    std::cerr << time_now() << "接收来自" << ip << "的消息失败" << std::endl;
    return;
}
string str = buffer;
if (license.find(str) == license.end() || license[str] < max_license)
{
    string res = "ACCEPT";
    send(clientSocket, res.c_str(), res.size() + 1, 0);
    license[str]++;
    std::cout << time_now() << " 客户接入," << " 许可证为" << str << ", IP地址为 " << ip << ", 该许可证的用户数量为" << license[str] << std::endl;
    while (true)
    {
        bytesRead = recv(clientSocket, buffer, 1024, 0);
        if (bytesRead > 0)
        {
            cout << time_now() << " 接收到许可证为" << str << ",ip地址为" << ip << "的用户状态报送" << endl;
        }
        else
        {
            cout << time_now() << " 许可证为" << str << ",ip地址为" << ip << "的用户已退出" << endl;
            license[str]--;
            return;
        }
    }
}
else
{
    string res = "REJECT";
    std::cout << time_now() << " 客户接入," << " 许可证为" << str << ", IP地址为 " << ip << ", 但该许可证的用户数量已满, 客户接入失败" << std::endl;
    send(clientSocket, res.c_str(), res.size() + 1, 0);
}

```

4.客户端

客户端会先咨询用户是否已经拥有序列号，如果未拥有序列号会和许可证提供服务器连接，许可证提供服务器和状态报送服务器可以根据管理员的需求设置 IP 地址和端口号，这里为了保证本地测试的方便两个服务器 IP 地址都设置为环回地址，通过端口号区分这两个服务器

```
int main()
{
    WSADATA wsaData;
    SOCKET clientSocket;
    struct sockaddr_in serverAddress;
    cout << "已经拥有许可证? [Y/n]:";
    char c = getchar();
    if (c == 'N' || c == 'n')
    {
        string account, password, type;
        cout << "请输入你的用户名: ";
        cin >> account;
        cout << "请输入你的密码: ";
        cin >> password;
        cout << "请输入你的用户类型: ";
        cin >> type;
        cout << "您的许可证为:" << get_license(account.c_str(), password.c_str(), type.c_str())<<"请务必牢记" << endl << endl;
    }
}
```

```
string get_license(const char account[],const char password[],const char type[])
{
    WSADATA wsaData;
    SOCKET clientSocket2;
    struct sockaddr_in serverAddress2;
    // 初始化 Winsock
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
    {
        std::cerr << "初始化 Winsock 失败" << std::endl;
        exit(-1);
    }
    // 创建套接字
    clientSocket2 = socket(AF_INET, SOCK_STREAM, 0);
    if (clientSocket2 == INVALID_SOCKET) {
        std::cerr << "创建套接字失败" << std::endl;
        exit(-1);
    }

    // 设置服务器地址和端口
    serverAddress2.sin_family = AF_INET;
    serverAddress2.sin_port = htons(6666);
    if (inet_pton(AF_INET, "127.0.0.1", &(serverAddress2.sin_addr)) <= 0) {
        std::cerr << "无效的服务器地址" << std::endl;
        exit(-1);
    }
}
```

```

// 设置服务器地址和端口
serverAddress2.sin_family = AF_INET;
serverAddress2.sin_port = htons(6666);
if (inet_pton(AF_INET, "127.0.0.1", &(serverAddress2.sin_addr)) <= 0) {
    std::cerr << "无效的服务器地址" << std::endl;
    exit(-1);
}

// 连接服务器
if (connect(clientSocket2, (struct sockaddr*)&serverAddress2, sizeof(serverAddress2)) < 0) {
    std::cerr << "连接服务器失败" << std::endl;
    exit(-1);
}

if (send(clientSocket2, account, strlen(account) + 1, 0) < 0) {
    std::cerr << "发送消息失败" << std::endl;
    exit(-1);
}
Sleep(1);
if (send(clientSocket2, password, strlen(password) + 1, 0) < 0) {
    std::cerr << "发送消息失败" << std::endl;
    exit(-1);
}
Sleep(1);
if (send(clientSocket2, type, strlen(type) + 1, 0) < 0) {
    std::cerr << "发送消息失败" << std::endl;
    exit(-1);
}
char buffer[1024] = { 0 };
if (recv(clientSocket2, buffer, 1024, 0) < 0) {
    std::cerr << "接收响应失败" << std::endl;
    exit(-1);
}
// 关闭套接字
closesocket(clientSocket2);

// 清理 Winsock
WSACleanup();
return (string)buffer;

```

当用户拥有许可证后会根据服务器中该许可证序列号的在线用户数量来确定该用户是否可以使用软件

```

// 发送数据到服务器
string license;
cout << "请输入10位许可证: ";
while (cin >> license && license.size() != 10) cout << " 许可证不合法, 请重新输入: ";
if (send(clientSocket, license.c_str(), license.size() + 1, 0) < 0) {
    std::cerr << "发送消息失败" << std::endl;
    return -1;
}

// 接收服务器的响应
char buffer[1024] = { 0 };
if (recv(clientSocket, buffer, 1024, 0) < 0) {
    std::cerr << "接收响应失败" << std::endl;
    return -1;
}
string s = buffer;
if (s == "ACCEPT")
{
    cout << "允许使用" << endl;
    stopFlag = false;
    thread t(SendStatusReport, clientSocket);
    //使用软件A
    softwareA();
    stopFlag = true;
    t.detach();
}
else if (s == "REJECT")
{
    cout << "当前许可证用户已满, 请稍后再尝试";
    Sleep(10);
}

```

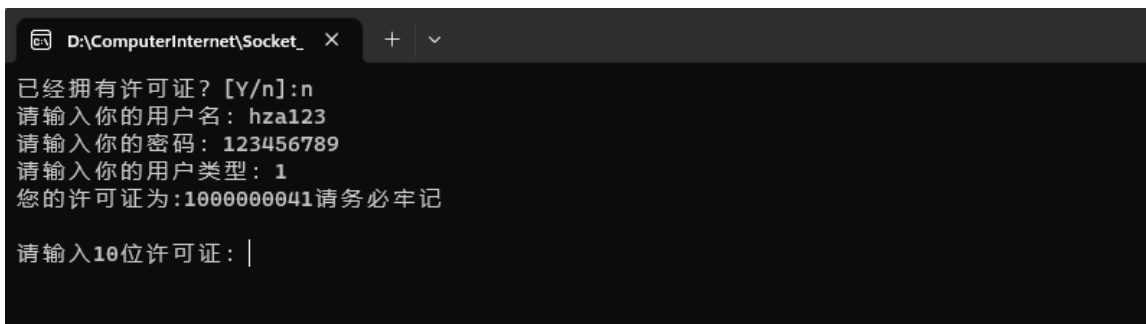

之后会开启一个线程用于客户端状态的报送，每隔一段时间向服务器发送一次状态报送，之后会将对应的线程进行阻塞直到下一次发送

```
void SendStatusReport(SOCKET clientSocket)
{
    while (!stopFlag)
    {
        // 发送状态报告到服务器
        string report = "status report";
        if (send(clientSocket, report.c_str(), report.size() + 1, 0) < 0)
        {
            cerr << "发送状态报告失败" << std::endl;
        }
        this_thread::sleep_for(chrono::seconds(1800));
    }
}
```

```
string s = buffer;
if (s == "ACCEPT")
{
    cout << "允许使用" << endl;
    stopFlag = false;
    thread t(SendStatusReport, clientSocket);
    //使用软件A
    softwareA();
    stopFlag = true;
    t.detach();
}
else if (s == "REFLECT")
```

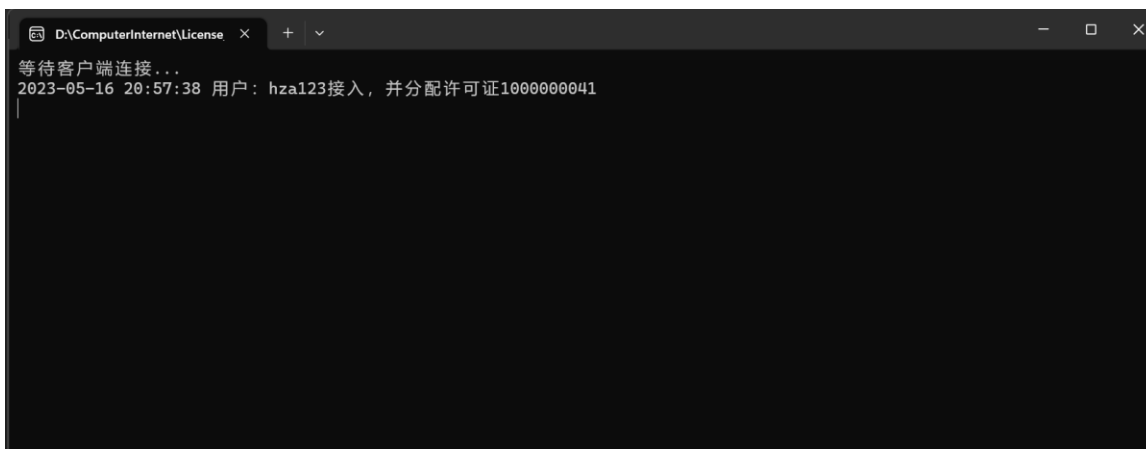
5.运行效果

先开启两个服务器，等待客户端接入，当获取到用户没有许可证时会连接对应的服务器，根据所提供的用户名、密码和用户类型分配一个序列号，同时服务器将对应的信息存储到数据库中



```
D:\ComputerInternet\Socket_ X + v
已经拥有许可证? [Y/n]:n
请输入你的用户名: hza123
请输入你的密码: 123456789
请输入你的用户类型: 1
您的许可证为:1000000041请务必牢记

请输入10位许可证: |
```



```
D:\ComputerInternet\License X + v - □ X
等待客户端连接...
2023-05-16 20:57:38 用户: hza123接入, 并分配许可证1000000041
|
```

之后用户输入许可证进行登录，此时用户获得软件的使用许可权，同时用户会每隔一段时间向服务器报送状态，服务器接收到状态并作为日志打印输出



```
D:\ComputerInternet\Socket_ X + v
已经拥有许可证? [Y/n]:n
请输入你的用户名: hza123
请输入你的密码: 123456789
请输入你的用户类型: 1
您的许可证为:1000000041请务必牢记

请输入10位许可证: 1000000041
允许使用
使用软件A中, 按任意键结束软件A使用
|
```

```

D:\ComputerInternet\Socket_ x + v
等待客户端连接...
2023-05-16 20:59:25 客户接入, 许可证为1000000041, IP地址为 127.0.0.1, 该许可证的用户数量为1
2023-05-16 20:59:25 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 20:59:30 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 20:59:35 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 20:59:40 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 20:59:45 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 20:59:50 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 20:59:55 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:00:00 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:00:05 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:00:10 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:00:15 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:00:20 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:00:25 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:00:30 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送

```

此时假设接入用户 2，服务器会对这两个用户进行并发服务，如图所示服务器的日志中输出用户接入信息，并且都会打印两个用户的状态报送日志，同理更多个用户接入时也会利用多线程实现并发处理

```

2023-05-16 21:03:40 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:03:45 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:03:50 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:03:55 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:04:00 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:04:05 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:04:08 客户接入, 许可证为1234567890, IP地址为 127.0.0.1, 该许可证的用户数量为1
2023-05-16 21:04:08 接收到许可证为1234567890,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:04:10 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:04:13 接收到许可证为1234567890,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:04:16 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送

```

如果用户 2 对应的许可证序列号已经达到人数上限，那么将会被拒绝使用软件，可以在服务器的日志中找到对应的拒绝信息

```

您的许可证为:1000018407请务必牢记

请输入10位许可证: 1000000041
许可证不合法, 请重新输入: 1000000041
当前许可证用户已满, 请稍后再尝试
D:\ComputerInternet\Socket_Client\x64\Debug\Socket_Client.exe (进程 4552)已退出, 代码为 0。
按任意键关闭此窗口。 . . |

2023-05-16 21:01:20 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:01:25 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:01:30 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:01:35 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:01:40 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:01:45 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:01:50 客户接入, 许可证为1000000041, IP地址为 127.0.0.1, 但该许可证的用户数量已满, 客户接入失败
2023-05-16 21:01:50 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:01:55 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:02:00 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:02:05 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:02:10 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:02:15 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送

```

当用户主动退出或者崩溃时，会打印出对应的退出日志，同时将对应许可证序列号的在线人数扣除

```
2023-05-16 21:05:01 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:05:03 接收到许可证为1234567890,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:05:06 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:05:08 接收到许可证为1234567890,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:05:11 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:05:13 接收到许可证为1234567890,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:05:16 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:05:18 接收到许可证为1234567890,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:05:21 接收到许可证为1000000041,ip地址为127.0.0.1的用户状态报送
2023-05-16 21:05:21 许可证为1000000041,ip地址为127.0.0.1的用户已退出
```

4 实验代码

本次实验的代码已上传于以下代码仓库：<https://gitee.com/aaaz718/Lab5.git>

5 实验总结

1.对 Socket 编程有了初步的了解，在应用层的角度对网络层的 IP 和传输层的 TCP 等协议有了进一步的了解

2.对多线程编程有了初步的了解，简单了解了线程、阻塞、进程等概念。可以使用它们来实现一些需要的程序效果

3.这个实验难点之一在于需求分析，因此在软件编写的时候进行充分的需求分析十分必要，不然容易导致后续的工作无法正确进行