厦門大學

信息学院软件工程系

《计算机网络》实验报告

题　　目 实验五　利用 Socke API 实现许可认证软件

班　　级 软件工程 2019 级 4 班

姓　　名 陈栋

学　　号 22920192204171

实验时间 2021 年 3 月 00 日

2021 年 3 月 00 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。

1 实验目的

1、掌握应用层文件传输的原理；

2、了解传输过程中传输层协议的选定、应用层协议设计和协议开发等概念

2 实验环境

Windows10、 c++

3 实验结果

利用 c++中，网络传输的相关概念和数据函数，创建 TCP 传输协议服务器

1、服务器端：

```
public:
    server();
    void process();
    vector<string> split(char* str);
    int check(string name, string pass);



private:
    int listener;//监听套接字
    sockaddr_in  serverAddr;//IPV4的地址方式
    vector <int> socnum;//存放创建的套接字，处理多个客户端的情况

    string username;
    string password;
};
```

在 server 中声明相关端 socket 信息

```
    //成功返回一个socket（文件描述符）
    listener = socket(AF_INET, SOCK_STREAM, 0);//采用ipv4,TCP传输
    if (listener == -1)
    {
        printf("Error at socket(): %ld\n", WSAGetLastError());
        perror("创建失败");
        exit(1);
    }
    cout << "连接创建成功" << endl;
```

创建一个 socket 用于存放服务端信息

```
//
//将固定的网络地址（listener）和端口号（serverAddr）绑定在一起。
//绑定成功返回0，出错返回1
if (bind(listener, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) < 0)
{
    perror("bind error");
    exit(1);
}
```

绑定相关端口号和地址信息

```
}
//该函数仅被服务器使用
//listen声明listener处于监听状态，
//并且允许最多有6个客户端处于连接等待状态
//成功返回0失败返回-1
if (listen(listener, 6) == -1)
{
    perror("listen failed");
    exit(1);
}
```

使用 listen 监听端口

```
//服务端只设置读，不考虑写
//int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, const struct timeval
//用于确定一个或多个套接字的状态
//nfds：表示所要监视的文件描述的范围
//readfds是指向fd_set结构的指针。这个集合中加入我们所需要监视的文件可读操作的文件描述符
//writefds:指向fd_set结构的指针，这个集合中加入我们所需要监视的文件可写操作的文件描述符。
//exceptfds:指向fd_set结构的指针，这个集合中加入我们所需要监视的文件错误异常的文件描述符
//timeout:指向timeval结构体的指针，通过传入的这个timeout参数来决定select()函数的三种执行方式
//返回-1：select()函数错误，并将所有描述集合清0
//返回0：表示select()函数超时
//返回正数值表示已经准备好的描述数符
switch (select(0, &fds, &fds_writer, NULL, &timeout))
```

使用 select 来确定各个套接字的状态，并对次来进行相关操作。主要操作为返回正数是当有套接字数据帧传入时，对其进行处理

```
//如果第一个有套接字可读的消息，就建立建立连接
if (i == 0 && FD_ISSET(socnum[i], &fds))
{
    struct sockaddr_in client_address;
    socklen_t client_addrLength = sizeof(struct sockaddr_in);
    //返回一个用户的套接字
    int clientfd = accept(listener, (struct sockaddr*)&client_address, &client_addrLength);
    //添加用户，服务器上显示消息，并通知用户连接成功
    //clientfd:为新接受的从客户端发来的套接字
    socnum.push_back(clientfd);
    cout << "客户端：" << clientfd << " 成功连接本服务器" << endl;
    char ID[1024];
    sprintf(ID, "客户端id为：%d", clientfd);
    //服务器产生ID并发送给客户端让客户端知道自己的ID
    send(clientfd, ID, sizeof(ID) - 1, 0);//减去最后一个'/0'
```

当第一个数据包传入，对发送端，发送其端口序号

```
//检查集合中的指定文件描述是否完好（可读可写）
if (i != 0 && FD_ISSET(socnum[i], &fds))
{
    char buf[1024];
    memset(buf, '\0', sizeof(buf));//初始化
    int size = recv(socnum[i], buf, sizeof(buf) - 1, 0);
    //检测是否断线
    if (size == 0 || size == -1)
    {
        cout << "客户端：" << socnum[i] << "已掉线" << endl;

        closesocket(socnum[i]);//关闭这个套接字
        FD_CLR(socnum[i], &fds);//在列表列表中删除

        socnum.erase(socnum.begin() + i);//在vector数组中删除
        mount--;
    }
    //若没有掉线
    else
    {
        printf("客户端 %d 发来消息: %s \n", socnum[i], buf);
        FD_SET(socnum[i], &fds_writer);
        //提取出用户发来的序列号，用户名，密码；
        vector<string>info = split(buf);


        //对用户发来的用户名和密码进行验证
        int res=this->check(info[1], info[2]);

        //登录结果返回给客户端
        if (res == 1) {
            info[0] = "true";
        }
        else {
            info[0] = "false";
        }
        cout << info[0].c_str() << endl;
        send(socnum[i], info[0].c_str(), sizeof(info[0].c_str()), 0);
        Sleep(1000);
        FD_ZERO(&fds_writer);

    }
```

没有掉线时，对传入信息进行验证是否可以登录，访问数据

2、发送端

```
client::client()
{
    user = 0;
    writing = 0;
    serverAddr.sin_family = PF_INET;
    serverAddr.sin_port = SERVER_PORT;
    serverAddr.sin_addr.s_addr = inet_addr(SERVER_IP);//将字符串类型转换uint32_t
}
```

首先也是配置端口信息

```cpp
int client::init()
{
    int   Ret;
    WSADATA   wsaData;                          // 用于初始化套接字环境
                                                // 初始化WinSock环境
    //使用Socket之前必须调用WSAStartup函数，此函数在应用程序中用来初始化Windows Sockets DLL，
    //只有此函数调用成功后，应用程序才可以调用Windows SocketsDLL中的其他API函数，否则后面的任何函数都将调用失败。

    if ((Ret = WSAStartup(MAKEWORD(2, 2), &wsaData)) != 0)
    {
        cout << "WSAStartup()   failed   with   error" << Ret << endl;
        WSACleanup();
    }

    user = socket(AF_INET, SOCK_STREAM, 0);//采用ipv4,TCP传输
    if (user <= 0)
    {
        cout << "Error at socket():" << WSAGetLastError() << endl;
        exit(1);
    };
    cout << "成功建立连接" << endl;//创建成功
    //阻塞式的等待服务器连接
    if (connect(user, (const sockaddr *)&serverAddr, sizeof(serverAddr)) < 0)
    {
        cout << "Error at socket():" << WSAGetLastError() << endl;
        return 1;
    }
    cout << "连接 IP:" << SERVER_IP << " Port:" << SERVER_PORT << " 成功" << endl;//创建成功
    return 0;
}
```

初始化套接字的设备端口信息，相比于服务器端，少了 listen

```cpp
    //服务端有信息发送回来时，接收消息
    if (FD_ISSET(user, &fdread))
    {
        int size = recv(user, recvbuf, sizeof(recvbuf), 0);
        if (size > 0)
        {
            cout << "服务端回复:" << recvbuf << endl;
            memset(recvbuf, '\0', sizeof(recvbuf));
        }
        else
        {
            cout << "服务端关闭，正在等待服务端开启" << endl;
            Sleep(10000);//等待10秒后尝试重连
            cout << endl;
            cout << "开始尝试重新连接服务器" << endl << endl;
            stats = 1;//表明服务器崩溃了
            return;
        }
        writing = 1;

    }
```

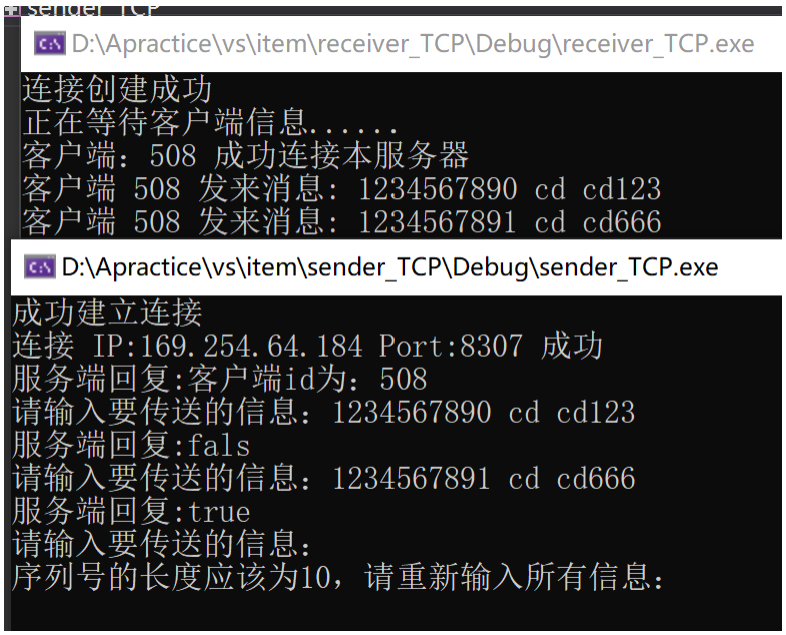
接受服务器端传来的消息

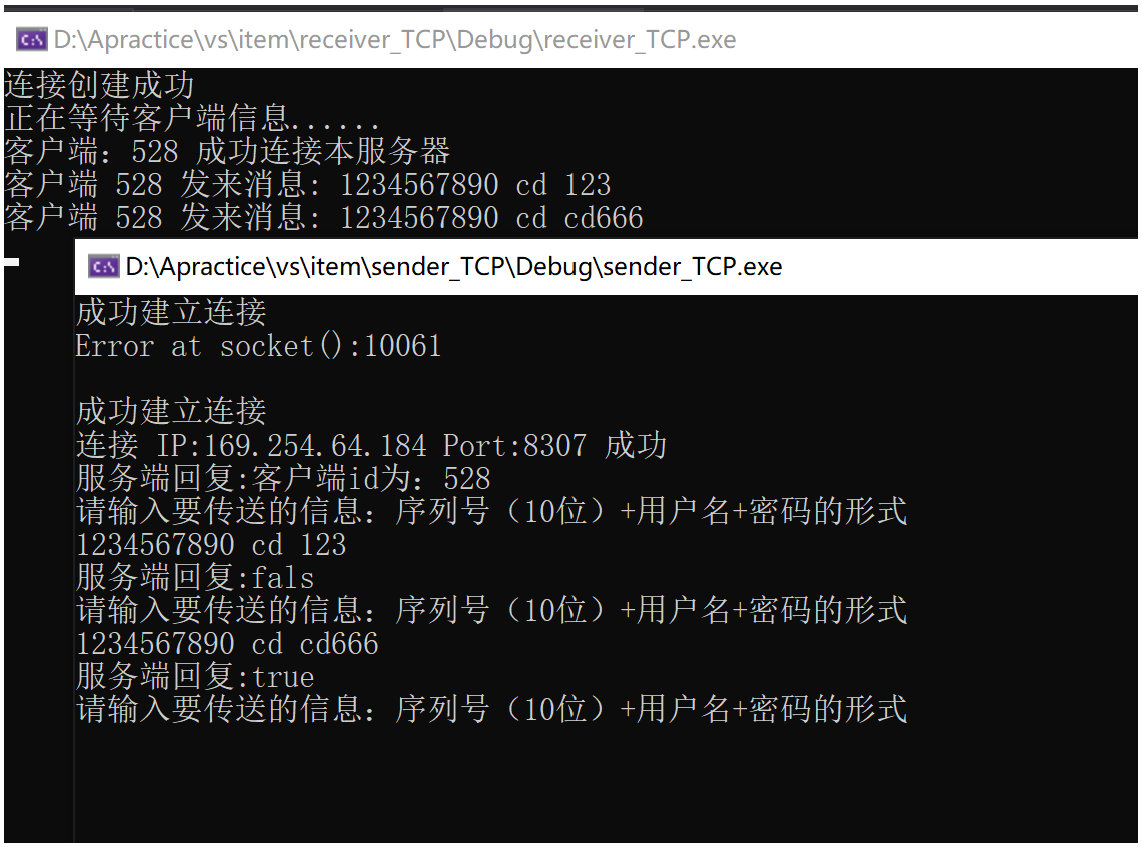
```
    }
    if (FD_ISSET(user, &fedwrite))
    {
        FD_ZERO(&fedwrite);//将fedwrite清零
        writing = 1;//设置为写状态


        sendata();
    }
    break;
}
```

```
//将客户端输入的序列号发送给服务端以求验证
string str;
if(info.size()==3)str = info[0] + " " + info[1] + " " + info[2];
else str = info[0] + " " + info[1];

send(user, str.c_str(), sizeof(str), 0);//发送消息

writing = 0;//发送完之后，写状态结束
```

传输数据信息到服务器端，进行信息验证

3、实验结果




D:\Apractice\vs\item\receiver_TCP\Debug\receiver_TCP.exe
连接创建成功
正在等待客户端信息.....
客户端：528 成功连接本服务器
客户端 528 发来消息：1234567890 cd 123
客户端 528 发来消息：1234567890 cd cd666
D:\Apractice\vs\item\sender_TCP\Debug\sender_TCP.exe
成功建立连接
Error at socket():10061
成功建立连接
连接 IP:169.254.64.184 Port:8307 成功
服务端回复:客户端id为：528
请输入要传送的信息：序列号（10位）+用户名+密码的形式
1234567890 cd 123
服务端回复:fals
请输入要传送的信息：序列号（10位）+用户名+密码的形式
1234567890 cd cd666
服务端回复:true
请输入要传送的信息：序列号（10位）+用户名+密码的形式


D:\Apractice\vs\item\receiver_TCP\Debug\receiver_TCP.exe
连接创建成功
正在等待客户端信息.....
客户端：396 成功连接本服务器
D:\Apractice\vs\item\sender_TCP\Debug\sender_TCP.exe
成功建立连接
连接服务器失败：No error
Error at socket():10061
正在尝试重连
成功建立连接
连接服务器失败：No error
Error at socket():10061
正在尝试重连
成功建立连接
连接 IP:169.254.64.184 Port:8307 成功
服务端回复:客户端id为：396
请输入要传送的信息：

4 实验代码

本次实验的代码已上传于以下代码仓库：cd888888/network: report (github.com)

5 实验总结

了解了网络编程时，使用网络传输协议的使用。

当建立想要使用网络层传输时服务器端和客户端建立的一般步骤

服务器端：初始化 winsocket 环境->创建套接字->绑定端口信息和地址信息->创建对端口的监听->通过 select 获取监听到的端口状态->建立连接->身份验证->信息传输->.

客户端开始的创建过程与之类似，少了对端口的监听。

注：同时附上学习同学的 TCP 和 UDP 服务器的代码