

实验 1：利用 Socket，设计和编写一个聊天程序

2011763 黄天昊

一、实验要求

1. 使用流式 Socket，设计一个两人聊天协议，要求聊天信息带有时间标签。请完整地说明交互消息的类型、语法、语义、时序等具体的消息处理方式。
2. 对聊天程序进行设计。给出模块划分说明、模块的功能和模块的流程图。
3. 在 Windows 系统下，利用 C/C++对设计的程序进行实现。程序界面可以采用命令行方式，但需要给出使用方法。编写程序时，只能使用基本的 Socket 函数，不允许使用对 socket 封装后的类或架构。
4. 对实现的程序进行测试。

二、实验流程

1. 协议设计

报文采用 char 类型数组，并规定报文的最大长度为 500。

规定报文的第 0 位为类型标志位。

 为 1 时是群发功能

发送报文时，如果设置标志位为 1，则该信息为一条群发信息。

服务器收到该类型信息后，会从报文的第 1-20 位获取用户名，并从第 21-499 位获取报文信息，并将这些信息转发给其他所有用户。

客户端收到该类型信息后，会从报文的第 1-20 位获取用户名，并从第 21-499 位获取报文信息，并将这些信息打印在屏幕上。

发送时与之类似。


 为 2 时是私发功能

发送报文时，如果设置标志位为 2，则该信息为一条私发信息。


服务器收到该类型信息后，会从报文的第 1-20 位获取用户名，并从第 21-499 位获取报文信息，并将这些信息转发给其他所有用户。

客户端收到该类型信息后，会从报文的第 1-20 位获取用户名，并从第 21-499 位获取报文信息，并将这些信息打印在屏幕上。

发送时与之类似。

 为 0 时是发送当前在线列表功能

当有用户加入或退出时，服务器端会向所有用户发送一条标志位为 0，1-499 位为用户信息的报文，所有客户端会将其第 1-499 位作为当前在线用户信息打印在屏幕上。

 为 9 时是私发请求失败消息回复

如果有用户 A 私发给另一用户 B 一条消息，但是用户 B 不在线或不在线，那么服务器端会给用户 A 发送标志位为 9 的报文，并在第 1-20 位附加 B 的用户名，在 21-499 位附加错误信息。当用户 A 收到后，也会以同样的格式将信息打印在屏幕上。

2. 服务器端设计流程

1) 使用 WSStartup 函数启动 WSA

WSStartup 必须是应用程序或 DLL 调用的第一个 Windows Sockets 函数。它允许应用程序或 DLL 指明 Windows Sockets API 的版本号及获得特定 Windows Sockets 实现的细节。应用程序或 DLL 只能在一次成功的 WSStartup() 调用之后才能调用进一步的 Windows Sockets API 函数。

```
1. WSADATA wsaData; // 用来存储被 WSStartup 函数调用后返回的
   Windows Sockets 数据, 包含 Winsock.dll 执行的数据。
2. int iResult = WSStartup(MAKEWORD(2, 2), &wsaData); // 指定 socket
   规范的版本
3. if (iResult != NO_ERROR) {
4.     cout << "WSStartup failed with error: " << iResult << endl;
5.     return 1;
6. }
```

2) 创建一个监听的 socket

创建一个监听的 SOCKET 如果有 connect 的请求就新创建一个线程

```
1. SOCKET listenSocket;
2. listenSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP); // 通信
   协议: IPv4 Internet 协议; 套接字通信类型: TCP 链接; 协议特定类型: 某些
   协议只有一种类型, 则为 0
3. if (listenSocket == INVALID_SOCKET) {
4.     cout << "socket failed with error: " << WSAGetLastError() <<
   endl;
5.     WSACleanup();
6.     return 1;
7. }
```

3) 获取本机 IP

```
1. char ip[20] = { 0 };
2. struct hostent* phostinfo = gethostbyname("");
3. char* p = inet_ntoa(*(struct in_addr*)(*phostinfo->h_addr_list))
   );
4. strncpy(ip, p, sizeof(ip));
5. cout << "服务器端 ip 为: " << ip << endl;
```

4) 使用 bind 函数绑定 IP 地址及端口号

```
1. sockaddr_in sockAddr;
2. memset(&sockAddr, 0, sizeof(sockAddr));
3. sockAddr.sin_family = AF_INET;
```

```

4. // inet_pton(AF_INET, "10.130.106.124", &sockAddr.sin_addr.s_addr);
5. inet_pton(AF_INET, ip, &sockAddr.sin_addr.s_addr); // 将点分十进制的ip地址转化为用于网络传输的数值格式
6. sockAddr.sin_port = htons(920); // 端口号
7. iResult = bind(listenSocket, (SOCKADDR*)&sockAddr, sizeof(sockAddr)); // bind函数把一个本地协议地址赋予一个套接字。对于网际协议，协议地址是32位的IPv4地址或是128位的IPv6地址与16位的TCP或UDP端口号的组合
8. if (iResult == SOCKET_ERROR) {
9.     wprintf(L"bind failed with error: %ld\n", WSAGetLastError());
10. closesocket(listenSocket);
11. WSACleanup();
12. return 1;
13. }

```

5) 使用 listen 函数使 socket 进入监听状态

```

1. if (listen(listenSocket, 5) == SOCKET_ERROR) {
2.     cout << "listen failed with error: " << WSAGetLastError() << endl;
3.     closesocket(listenSocket);
4.     WSACleanup();
5.     return 1;
6. }

```

6) 对于每个新的请求使用多线程处理

```

1. cout << "等待客户端连接..." << endl << endl;
2. while (1) {
3.     sockaddr_in clientAddr;
4.     int len = sizeof(clientAddr);
5.     SOCKET AcceptSocket = accept(listenSocket, (SOCKADDR*)&clientAddr, &len); // 接受一个特定socket请求等待队列中的连接请求
6.     if (AcceptSocket == INVALID_SOCKET) {
7.         cout << "accept failed with error: " << WSAGetLastError() << endl;
8.         closesocket(listenSocket);
9.         WSACleanup();
10.        return 1;
11.    }
12.    else {
13.        HANDLE hThread = CreateThread(NULL, 0, handlerRequest, (LPVOID)AcceptSocket, 0, NULL); // 创建线程，并且传入与client通讯的套接字

```

```

14.         CloseHandle(hThread); // 关闭对线程的引用
15.     }
16. }

```

- 7) 进入线程处理函数
这里针对不同的标志位，进入不同的处理分支

```

1. DWORD WINAPI handlerRequest(LPVOID lparam)
2. {
3.     // 为每一个连接到此端口的用户创建一个线程
4.     SOCKET ClientSocket = (SOCKET)lparam;
5.
6.     char curr_username[20];
7.     recv(ClientSocket, curr_username, sizeof(curr_username), 0);
    // 接收用户名
8.     appendUser(ClientSocket, curr_username);
9.
10.    SYSTEMTIME sysTime = { 0 };
11.    GetLocalTime(&sysTime);
12.    cout << endl;
13.    cout << "[" << sysTime.wYear << "-" << sysTime.wMonth << "-"
        << sysTime.wDay << " ";
14.    cout << sysTime.wHour << ":" << sysTime.wMinute << ":" << sys
        Time.wSecond << "]" << " ";
15.    cout << curr_username << " 已加入三体世界 ";
16.    cout << "目前在线人数: " << user_list.size() << "人" << endl;
17.
18.
19.    // 向用户发送当前在线用户名单
20.    if (user_list.size() > 0) {
21.        string nameList = "";
22.        for (auto it : user_list) {
23.            nameList += it.second;
24.            nameList += " ";
25.        }
26.        char sendList[1000];
27.        sendList[0] = '\0';
28.        for (int i = 0; i < nameList.length(); i++) sendList[i + 1
            ] = nameList[i];
29.        sendList[nameList.length()] = '\0';
30.        send(ClientSocket, sendList, sizeof(sendList), 0);
31.    }
32.    else {
33.        char sendList[1000];
34.        sendList[0] = '\0';

```

```

35.         string msg = "当前无在线成员";
36.         for (int i = 0; i < msg.length(); i++) sendList[i + 1] = m
           sg[i];
37.         send(ClientSocket, sendList, sizeof(sendList), 0);
38.     }
39.
40.
41.     // 循环接受客户端数据
42.     int recvResult;
43.     int sendResult;
44.     do {
45.         char recvBuf[DEFAULT_BUFLen] = "";
46.         char sendBuf[DEFAULT_BUFLen] = "";
47.         recvResult = recv(ClientSocket, recvBuf, DEFAULT_BUFLen,
           0);
48.         if (recvResult > 0) {
49.             SYSTEMTIME logTime = { 0 };
50.             GetLocalTime(&logTime);
51.             if (recvBuf[0] == '0') {
52.                 // 获取当前用户列表
53.                 char message[DEFAULT_BUFLen];
54.                 for (int i = 1; i < DEFAULT_BUFLen; i++) {
55.                     message[i] = recvBuf[i];
56.                 }
57.                 cout << endl;
58.                 cout << "当前在线用户" << "]" << message << endl;
59.             }
60.             else if (recvBuf[0] == '1') {
61.                 // 群发
62.                 char message[DEFAULT_BUFLen];
63.                 for (int i = 21; i < DEFAULT_BUFLen; i++) {
64.                     message[i - 21] = recvBuf[i];
65.                 }
66.
67.                 cout << endl;
68.                 cout << "[" << logTime.wYear << "-"
                   << logTime.wMonth << "-" << logTime.wDay << " ";
69.                 cout << logTime.wHour << ":" << logTime.wMinute <
                   < ":" << logTime.wSecond;
70.                 cout << " 来自 " << curr_username << " 的群发消
                   息]" << message << endl;
71.
72.                 // 向其他用户分发消息
73.                 for (auto it : user_list) {

```

```

74.             if (it.first != ClientSocket) {
75.                 sendResult = send(it.first, recvBuf, DEFA
ULT_BUFLen, 0);
76.                 if (sendResult == SOCKET_ERROR)cout << "s
end failed with error: " << WSAGetLastError() << endl;
77.             }
78.         }
79.     }
80.     else if (recvBuf[0] == '2') {
81.         // 私发
82.         char des_user[20], message[DEFAULT_BUFLen];
83.         for (int i = 1; i <= 20; i++)des_user[i - 1] = re
cvBuf[i];
84.         for (int i = 21; i < DEFAULT_BUFLen; i++)message[
i - 21] = recvBuf[i];
85.         cout << endl;
86.         cout << "[" << curr_username << " 私发
给 " << des_user << " 的消息: " << "]" << message << endl;
87.         // 向指定用户发送消息
88.         bool success = 0; // 发送成功?
89.         for (auto it : user_list) {
90.             string sdes_user = "";
91.             for (int i = 0; des_user[i]; i++)sdes_user +=
des_user[i];
92.             if (it.second == sdes_user) {
93.                 for (int i = 1; i <= 20; i++)recvBuf[i] =
curr_username[i - 1];
94.                 sendResult = send(it.first, recvBuf, DEFA
ULT_BUFLen, 0); // 发送
95.                 if (sendResult == SOCKET_ERROR)cout << "s
end failed with error: " << WSAGetLastError() << endl;
96.                 else success = 1;
97.                 break;
98.             }
99.         }
100.        if (!success) {
101.            string smsg = "发送失败, 此用户不存在或未上线";
102.            char msg[100];
103.            for (int i = 0; i < smsg.length(); i++)msg[i
] = smsg[i];
104.            msg[smsg.length()] = '\0';
105.            for (int i = 1; i <= 20; i++)sendBuf[i] = de
s_user[i - 1];

```

```

106.             for (int i = 0; i < 100; i++)sendBuf[i + 21]
                = msg[i];
107.             sendBuf[0] = '9';
108.             sendResult = send(ClientSocket, sendBuf, DEF
                AULT_BUFLen, 0); // 发送出错信息到原用户
109.             if (sendResult == SOCKET_ERROR)cout << "send
                failed with error: " << WSAGetLastError() << endl;
110.             }
111.         }
112.     }
113. } while (recvResult != SOCKET_ERROR);
114.
115.     GetLocalTime(&sysTime);
116.     cout << endl;
117.     cout << "[" << sysTime.wYear << "-" << sysTime.wMonth << "-"
        << sysTime.wDay << " ";
118.     cout << sysTime.wHour << ":" << sysTime.wMinute << ":" << sy
        sTime.wSecond << "]" << endl;
119.     cout << curr_username << " 离开了三体世界" << endl;
120.
121.     user_list.erase(ClientSocket);
122.     closesocket(ClientSocket);
123.     return 0;
124. }

```

8) 关闭服务器端 socket

```

1. iResult = closesocket(listenSocket);
2. if (iResult == SOCKET_ERROR) {
3.     cout << "close failed with error: " << WSAGetLastError()
        << endl;
4.     WSACleanup();
5.     return 1;
6. }

```

3. 客户端设计流程

客户端前几步与服务器端类似，这里不予赘述。

需要注意的是：

开启两个线程，一个负责接收，一个负责发送

```

1. HANDLE hThread[2];
2. hThread[0] = CreateThread(NULL, 0, recv, (LPVOID)&connectSocket,
    0, NULL);

```

```
3. hThread[1] = CreateThread(NULL, 0, send, (LPVOID)&connectSocket,
    0, NULL);
```

接收线程调用函数:

```
1. DWORD WINAPI recv(LPVOID lparam_socket) {
2.     int recvResult;
3.     SOCKET* recvSocket = (SOCKET*)lparam_socket; // 一定要使用指针, 因为要指向 connect socket 的位置
4.
5.     while (1) {
6.         char recvBuf[DEFAULT_BUFLen] = "";
7.         recvResult = recv(*recvSocket, recvBuf, DEFAULT_BUFLen, 0
            );
8.         if (recvResult > 0) {
9.             SYSTEMTIME systime = { 0 };
10.            GetLocalTime(&systime);
11.            cout << endl << endl;
12.            cout << "#####
                #####" << endl;
13.            cout << endl;
14.            cout << "[" << systime.wYear << "-"
                " << systime.wMonth << "-" << systime.wDay << " ";
15.            cout << systime.wHour << ":" << systime.wMinute << ":"
                " << systime.wSecond;
16.            if (recvBuf[0] == '0') {
17.                // 获取当前用户列表
18.                char message[DEFAULT_BUFLen];
19.                for (int i = 1; i < DEFAULT_BUFLen; i++) {
20.                    message[i - 1] = recvBuf[i];
21.                }
22.                cout << " 当前在线用户
                    " << "]" << message << endl;
23.                cout << "#####
                    #####" << endl;
24.            }
25.            else if (recvBuf[0] == '1') {
26.                // 群发
27.                char user_name[20], message[DEFAULT_BUFLen];
28.                for (int i = 1; i <= 20; i++)user_name[i - 1] = r
                    ecvBuf[i];
29.                for (int i = 21; i < DEFAULT_BUFLen; i++)message[
                    i - 21] = recvBuf[i];
30.                cout << " 收到来自 " << user_name << " 群发的消息:
                    " << "]" << message << endl;
```



```

31.             cout << "#####
#####" << endl;
32.         }
33.         else if (recvBuf[0] == '2') {
34.             // 私发
35.             char des_user[20], message[DEFAULT_BUFLen];
36.             for (int i = 1; i <= 20; i++)des_user[i - 1] = re
cvBuf[i];
37.             for (int i = 21; i < DEFAULT_BUFLen; i++)message[
i - 21] = recvBuf[i];
38.             cout << " 收到来自 " << des_user << " 私发的消息
" << "]" << message << endl;
39.             cout << "#####
#####" << endl;
40.         }
41.         else if (recvBuf[0] == '9') {
42.             // 收到私发错误信息
43.             char des_user[20], message[DEFAULT_BUFLen];
44.             for (int i = 1; i <= 20; i++)des_user[i - 1] = re
cvBuf[i];
45.             for (int i = 21; i < DEFAULT_BUFLen; i++)message[
i - 21] = recvBuf[i];
46.             cout << " 发送给用户 " << des_user << " 的私发信
息]" << message << endl;
47.             cout << "#####
#####" << endl;
48.         }
49.     }
50.     else {
51.         closesocket(*recvSocket);
52.         return 1;
53.     }
54. }
55. }

```

发送线程调用函数:

```

1. DWORD WINAPI send(LPVOID lparam_socket) {
2.
3.     // 接受消息直到quit 退出聊天
4.     int sendResult;
5.     SOCKET* sendSocket = (SOCKET*)lparam_socket;
6.
7.     while (1)
8.     {

```

```

9.         // 发送消息
10.        char sendBuf[DEFAULT_BUFLen] = "";
11.        char temp[DEFAULT_BUFLen] = "";
12.        cout << endl << endl;
13.        cout << "-----" << endl;
14.        cout << "请输入你的消息: ";
15.        cin >> temp;
16.        // cin.getline(temp, DEFAULT_BUFLen); // 保证可以输入空格,
        getline 函数设置好了以换行符为结束
17.        if (temp == quit_string) {
18.            closesocket(*sendSocket);
19.            cout << endl << "您已退出" << endl;
20.            return 1;
21.        }
22.        while (1) {
23.            cout << "如群发, 输入 1; 如私发, 输入 2: ";
24.            string flag;
25.            cin >> flag;
26.            if (flag == "1") {
27.                sendBuf[0] = '1';
28.                for (int i = 1; i <= 20; i++) sendBuf[i] = user_name[i - 1];
29.                break;
30.            }
31.            else if (flag == "2") {
32.                sendBuf[0] = '2';
33.                char des_user[20];
34.                cout << "请输入要私发的用户名: ";
35.                cin >> des_user;
36.                for (int i = 1; i <= 20; i++) sendBuf[i] = des_user[i - 1];
37.                break;
38.            }
39.            else {
40.                cout << "目前不支持此功能, 请重新输入" << endl;
41.            }
42.        }
43.        for (int i = 21; i < DEFAULT_BUFLen; i++) {
44.            sendBuf[i] = temp[i-21];
45.        }
46.        sendResult = send(*sendSocket, sendBuf, DEFAULT_BUFLen, 0);
47.        if (sendResult == SOCKET_ERROR) {

```

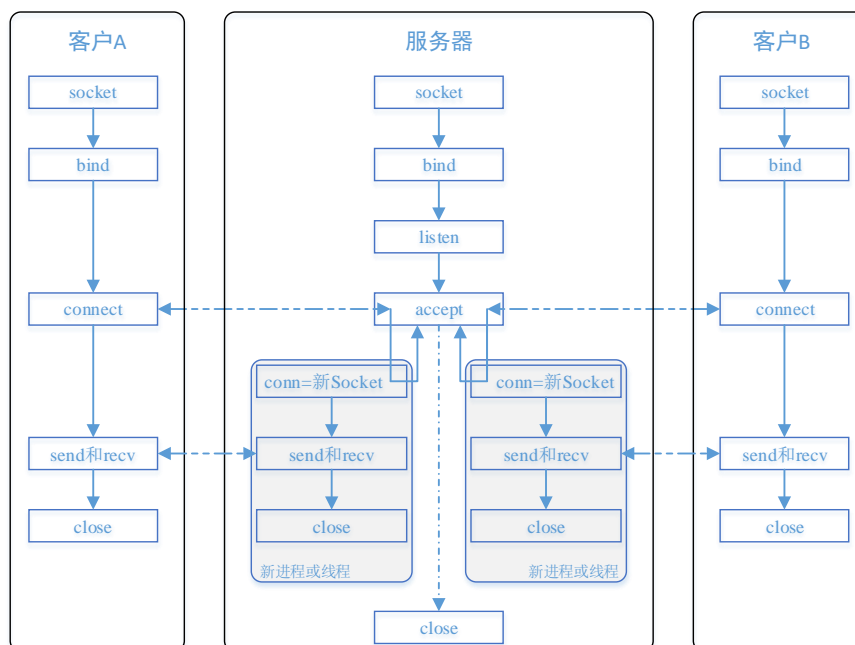
```

48.         cout << "send failed with error: " << WSAGetLastError
    () << endl;
49.         closesocket(*sendSocket);
50.         WSACleanup();
51.         return 1;
52.     }
53.     else {
54.         SYSTEMTIME systime = { 0 };
55.         GetLocalTime(&systime);
56.         cout << endl << systime.wYear << "-"
    " << systime.wMonth << "-" << systime.wDay << " ";
57.         cout << systime.wHour << ":" << systime.wMinute << ":"
    " << systime.wSecond;
58.         cout << " 消息已成功发送" << endl;
59.         cout << "-----"
    "-----" << endl;
60.     }
61.     Sleep(1000); // 停顿1秒再接收输入
62. }
63.}

```

4. 整体流程图

整体流程图如下：



三、 实验结果分析

服务器端打印消息：

```
D:\C++ Projects\Computer Network\x64\Release\Computer Network.exe

[2022-10-20 22:8:15] 秦始皇 已加入三体世界      目前在线人数: 1人
[2022-10-20 22:9:14] user 已加入三体世界      目前在线人数: 2人
[2022-10-20 22:9:27 来自 秦始皇 的群发消息] 我看到你了
[user 私发给 秦始皇 的消息: ] 为啥你也是秦始皇
[user 私发给 冯诺依曼 的消息: ] hi
[秦始皇 私发给 user 的消息: ] 因为三体里伊文思是秦始皇来着?
[2022-10-20 22:11:18 来自 user 的群发消息] 为啥没有冯诺依曼
[2022-10-20 22:11:58 来自 秦始皇 的群发消息] 你可以叫冯诺依曼
[2022-10-20 22:12:43 来自 user 的群发消息] 太搞笑了
[2022-10-20 22:12:45 来自 user 的群发消息] 我感觉
[2022-10-20 22:12:53 来自 user 的群发消息] 我在用匿名聊天功能
[2022-10-20 22:13:24 来自 秦始皇 的群发消息] 差不多, 这就是那个虚拟三体世界捏
[2022-10-20 22:13:45 来自 user 的群发消息] quit
[user 私发给 秦始皇 的消息: ] quit
[2022-10-20 22:14:5 来自 秦始皇 的群发消息] quit
```

客户端打印消息:

```
F:\All assignments during college\Junior(first semester)\Computer Network\Computer-Network\lab1\Client.exe

请输入要连接的服务器ip地址: 172.17.144.1
请输入你的用户名: 秦始皇

*****
*
* 欢迎 秦始皇 进入三体世界
*
*
* 当前在线用户
*
* 秦始皇
*
*
*
* 输入quit以退出三体世界
*
*****

-----
请输入你的消息: 啦啦啦
如群发, 输入1; 如私发, 输入2: 1
2022-10-20 22:41:34 消息已成功发送

-----
请输入你的消息:
```

```
F:\All assignments during college\Junior(first semester)\Computer Network\Computer-Network\lab1\Client.exe F:\All assignments during college\Junior(first semester)\Computer Network\Computer-Network\lab1\Server.exe

请输入要连接的服务器ip地址: 172.17.144.1
请输入你的用户名: 冯·诺伊曼

*****
*
* 欢迎 冯·诺伊曼 进入三体世界
*
*
* 当前在线用户
*
* 秦始皇 冯·诺伊曼
*
*
* 输入quit以退出三体世界
*
*****

-----
请输入你的消息: hello
如群发, 输入1; 如私发, 输入2: 2
请输入要私发的用户名: 秦始皇
2022-10-20 22:45:55 消息已成功发送

-----
请输入你的消息:

-----
[2022-10-20 22:45:55 收到来自 冯·诺伊曼 私发的消息] hello
hello
如群发, 输入1; 如私发, 输入2: 1
2022-10-20 22:46:4 消息已成功发送

-----
请输入你的消息:
```

实验结果基本符合预期

四、 不足与反思

1. 输入输出在打印时有时会重叠,比如上图中,原本应该等待用户输入,但由于来了新消息,所以在屏幕打印了消息,再接收输入。
2. 可以考虑加入消息队列机制
3. 加入可视化 UI 界面增加用户体验