**实验1：利用Socket，设计和编写一个聊天程序**

2011763 黄天昊

1. **实验要求**
2. 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。
3. 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。
4. 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。
5. 对实现的程序进行测试。
6. **实验流程**
7. **协议设计**

网络通信协议由三个要素组成：

* 语义，解释控制信息每个部分的意义。它规定了需要发出何种控制信息，以及完成的动作与做出什么样的响应。
* 语法，用户数据与控制信息的结构与格式，以及数据出现的顺序。
* 时序，对事件发生顺序的详细说明。

*可以形象地把这三个要素描述为：语义表示要做什么，语法表示要怎么做，时序表示要做的顺序。*

下面逐一进行说明：

1. 语义

* 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不在线或不存在，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。

1. 语法

规定发送的数据报为char型数组，总长度为500，第0位为标志位，第1-20位为用户信息位，第21-499位为消息位。

1. 时序

当客户端连接到服务器端时，首先会给客户端发送用户信息（即用户名），服务器端接收到用户信息后，会将现在在线的所有用户信息发送给客户端，然后客户端开始循环询问用户要发送的消息以及何种类型的消息（群发或私发），服务器端负责接收客户端发来的报文并根据消息类型进行转发，如果为群发消息，则会转发给其他所有用户。如果为私发消息，则会转发给指定用户，若该用户不存在或未上线，则会向发送者发送私发失败的报文信息。

1. **服务器端设计流程**
2. 使用WSAStartup函数启动WSA

WSAStartup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WSAStartup()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WSAStartup函数调用后返回的Windows Sockets数据，包含Winsock.dll执行的数据。*
2. int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData); *// 指定socket规范的版本*
3. if (iResult != NO\_ERROR) {
4. cout << "WSAStartup failed with error: " << iResult << endl;
5. return 1;
6. }
7. 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. }
8. 获取本机IP
9. char ip[20] = { 0 };
10. struct hostent\* phostinfo = gethostbyname("");
11. char\* p = inet\_ntoa(\*((struct in\_addr\*)(\*phostinfo->h\_addr\_list)));
12. strncpy(ip, p, sizeof(ip));
13. cout << "服务器端ip为：" << ip << endl;
14. 使用bind函数绑定IP地址及端口号
15. sockaddr\_in sockAddr;
16. memset(&sockAddr, 0, sizeof(sockAddr));
17. sockAddr.sin\_family = AF\_INET;
18. *// inet\_pton(AF\_INET, "10.130.106.124", &sockAddr.sin\_addr.s\_addr);*
19. 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*
20. sockAddr.sin\_port = htons(920); *// 端口号*
21. 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*
22. if (iResult == SOCKET\_ERROR) {
23. wprintf(L"bind failed with error: %ld\n", WSAGetLastError());
24. closesocket(listenSocket);
25. WSACleanup();
26. return 1;
27. }
28. 使用listen函数使socket进入监听状态
29. if (listen(listenSocket, 5) == SOCKET\_ERROR) {
30. cout << "listen failed with error: " << WSAGetLastError() << endl;
31. closesocket(listenSocket);
32. WSACleanup();
33. return 1;
34. }
35. 对于每个新的请求使用多线程处理
36. cout << "等待客户端连接..." << endl << endl;
37. while (1) {
38. sockaddr\_in clientAddr;
39. int len = sizeof(clientAddr);
40. 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*
41. if (AcceptSocket == INVALID\_SOCKET) {
42. cout << "accept failed with error: " << WSAGetLastError() << endl;
43. closesocket(listenSocket);
44. WSACleanup();
45. return 1;
46. }
47. else {
48. 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*
49. CloseHandle(hThread); *// 关闭对线程的引用*
50. }
51. }
52. 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. char curr\_username[20];
6. recv(ClientSocket, curr\_username, sizeof(curr\_username), 0); *// 接收用户名*
7. appendUser(ClientSocket, curr\_username);
8. SYSTEMTIME sysTime = { 0 };
9. GetLocalTime(&sysTime);
10. cout << endl;
11. cout << "[" << sysTime.wYear << "-" << sysTime.wMonth << "-" << sysTime.wDay << " ";
12. cout << sysTime.wHour << ":" << sysTime.wMinute << ":" << sysTime.wSecond << "] ";
13. cout << curr\_username << " 已加入三体世界     ";
14. cout << "目前在线人数：" << user\_list.size() << "人" << endl;
15. *// 向用户发送当前在线用户名单*
16. if (user\_list.size() > 0) {
17. string nameList = "";
18. for (auto it : user\_list) {
19. nameList += it.second;
20. nameList += " ";
21. }
22. char sendList[1000];
23. sendList[0] = '0';
24. for (int i = 0; i < nameList.length(); i++)sendList[i + 1] = nameList[i];
25. sendList[nameList.length()] = '\0';
26. send(ClientSocket, sendList, sizeof(sendList), 0);
27. }
28. else {
29. char sendList[1000];
30. sendList[0] = '0';
31. string msg = "当前无在线成员";
32. for (int i = 0; i < msg.length(); i++)sendList[i + 1] = msg[i];
33. send(ClientSocket, sendList, sizeof(sendList), 0);
34. }
35. *// 循环接受客户端数据*
36. int recvResult;
37. int sendResult;
38. do {
39. char recvBuf[DEFAULT\_BUFLEN] = "";
40. char sendBuf[DEFAULT\_BUFLEN] = "";
41. recvResult = recv(ClientSocket, recvBuf, DEFAULT\_BUFLEN, 0);
42. if (recvResult > 0) {
43. SYSTEMTIME logTime = { 0 };
44. GetLocalTime(&logTime);
45. if (recvBuf[0] == '0') {
46. *// 获取当前用户列表*
47. char message[DEFAULT\_BUFLEN];
48. for (int i = 1; i < DEFAULT\_BUFLEN; i++) {
49. message[i] = recvBuf[i];
50. }
51. cout << endl;
52. cout << "当前在线用户" << "] " << message << endl;
53. }
54. else if (recvBuf[0] == '1') {
55. *// 群发*
56. char message[DEFAULT\_BUFLEN];
57. for (int i = 21; i < DEFAULT\_BUFLEN; i++) {
58. message[i - 21] = recvBuf[i];
59. }
60. cout << endl;
61. cout << "[" << logTime.wYear << "-" << logTime.wMonth << "-" << logTime.wDay << " ";
62. cout << logTime.wHour << ":" << logTime.wMinute << ":" << logTime.wSecond;
63. cout << " 来自 " << curr\_username << " 的群发消息] " << message << endl;
64. *// 向其他用户分发消息*
65. for (auto it : user\_list) {
66. if (it.first != ClientSocket) {
67. sendResult = send(it.first, recvBuf, DEFAULT\_BUFLEN, 0);
68. if (sendResult == SOCKET\_ERROR)cout << "send failed with error: " << WSAGetLastError() << endl;
69. }
70. }
71. }
72. else if (recvBuf[0] == '2') {
73. *// 私发*
74. char des\_user[20], message[DEFAULT\_BUFLEN];
75. for (int i = 1; i <= 20; i++)des\_user[i - 1] = recvBuf[i];
76. for (int i = 21; i < DEFAULT\_BUFLEN; i++)message[i - 21] = recvBuf[i];
77. cout << endl;
78. cout << "[" << curr\_username << " 私发给 " << des\_user << " 的消息：" << "] " << message << endl;
79. *// 向指定用户发送消息*
80. bool success = 0; *// 发送成功？*
81. for (auto it : user\_list) {
82. string sdes\_user = "";
83. for (int i = 0; des\_user[i]; i++)sdes\_user += des\_user[i];
84. if (it.second == sdes\_user) {
85. for (int i = 1; i <= 20; i++)recvBuf[i] = curr\_username[i - 1];
86. sendResult = send(it.first, recvBuf, DEFAULT\_BUFLEN, 0); *// 发送*
87. if (sendResult == SOCKET\_ERROR)cout << "send failed with error: " << WSAGetLastError() << endl;
88. else success = 1;
89. break;
90. }
91. }
92. if (!success) {
93. string smsg = "发送失败，此用户不存在或未上线";
94. char msg[100];
95. for (int i = 0; i < smsg.length(); i++)msg[i] = smsg[i];
96. msg[smsg.length()] = '\0';
97. for (int i = 1; i <= 20; i++)sendBuf[i] = des\_user[i - 1];
98. for (int i = 0; i < 100; i++)sendBuf[i + 21] = msg[i];
99. sendBuf[0] = '9';
100. sendResult = send(ClientSocket, sendBuf, DEFAULT\_BUFLEN, 0); *// 发送出错信息到原用户*
101. if (sendResult == SOCKET\_ERROR)cout << "send failed with error: " << WSAGetLastError() << endl;
102. }
103. }
104. }
105. } while (recvResult != SOCKET\_ERROR);
106. GetLocalTime(&sysTime);
107. cout << endl;
108. cout << "[" << sysTime.wYear << "-" << sysTime.wMonth << "-" << sysTime.wDay << " ";
109. cout << sysTime.wHour << ":" << sysTime.wMinute << ":" << sysTime.wSecond << "] ";
110. cout << curr\_username << " 离开了三体世界" << endl;
111. user\_list.erase(ClientSocket);
112. closesocket(ClientSocket);
113. return 0;
114. }
115. 关闭服务器端socket
116. iResult = closesocket(listenSocket);
117. if (iResult == SOCKET\_ERROR) {
118. cout << "close failed with error: " << WSAGetLastError() << endl;
119. WSACleanup();
120. return 1;
121. }
122. **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. while (1) {
5. char recvBuf[DEFAULT\_BUFLEN] = "";
6. recvResult = recv(\*recvSocket, recvBuf, DEFAULT\_BUFLEN, 0);
7. if (recvResult > 0) {
8. SYSTEMTIME systime = { 0 };
9. GetLocalTime(&systime);
10. cout << endl << endl;
11. cout << "#################################################################" << endl;
12. cout << endl;
13. cout << "[" << systime.wYear << "-" << systime.wMonth << "-" << systime.wDay << " ";
14. cout << systime.wHour << ":" << systime.wMinute << ":" << systime.wSecond;
15. if (recvBuf[0] == '0') {
16. *// 获取当前用户列表*
17. char message[DEFAULT\_BUFLEN];
18. for (int i = 1; i < DEFAULT\_BUFLEN; i++) {
19. message[i - 1] = recvBuf[i];
20. }
21. cout << " 当前在线用户" << "] " << message << endl;
22. cout << "#################################################################" << endl;
23. }
24. else if (recvBuf[0] == '1') {
25. *// 群发*
26. char user\_name[20], message[DEFAULT\_BUFLEN];
27. for (int i = 1; i <= 20; i++)user\_name[i - 1] = recvBuf[i];
28. for (int i = 21; i < DEFAULT\_BUFLEN; i++)message[i - 21] = recvBuf[i];
29. cout << " 收到来自 " << user\_name << " 群发的消息：" << "] " << message << endl;
30. cout << "#################################################################" << endl;
31. }
32. else if (recvBuf[0] == '2') {
33. *// 私发*
34. char des\_user[20], message[DEFAULT\_BUFLEN];
35. for (int i = 1; i <= 20; i++)des\_user[i - 1] = recvBuf[i];
36. for (int i = 21; i < DEFAULT\_BUFLEN; i++)message[i - 21] = recvBuf[i];
37. cout << " 收到来自 " << des\_user << " 私发的消息" << "] " << message << endl;
38. cout << "#################################################################" << endl;
39. }
40. else if (recvBuf[0] == '9') {
41. *// 收到私发错误信息*
42. char des\_user[20], message[DEFAULT\_BUFLEN];
43. for (int i = 1; i <= 20; i++)des\_user[i - 1] = recvBuf[i];
44. for (int i = 21; i < DEFAULT\_BUFLEN; i++)message[i - 21] = recvBuf[i];
45. cout << " 发送给用户 " << des\_user << " 的私发信息] " << message << endl;
46. cout << "#################################################################" << endl;
47. }
48. }
49. else {
50. closesocket(\*recvSocket);
51. return 1;
52. }
53. }
54. }

发送线程调用函数：

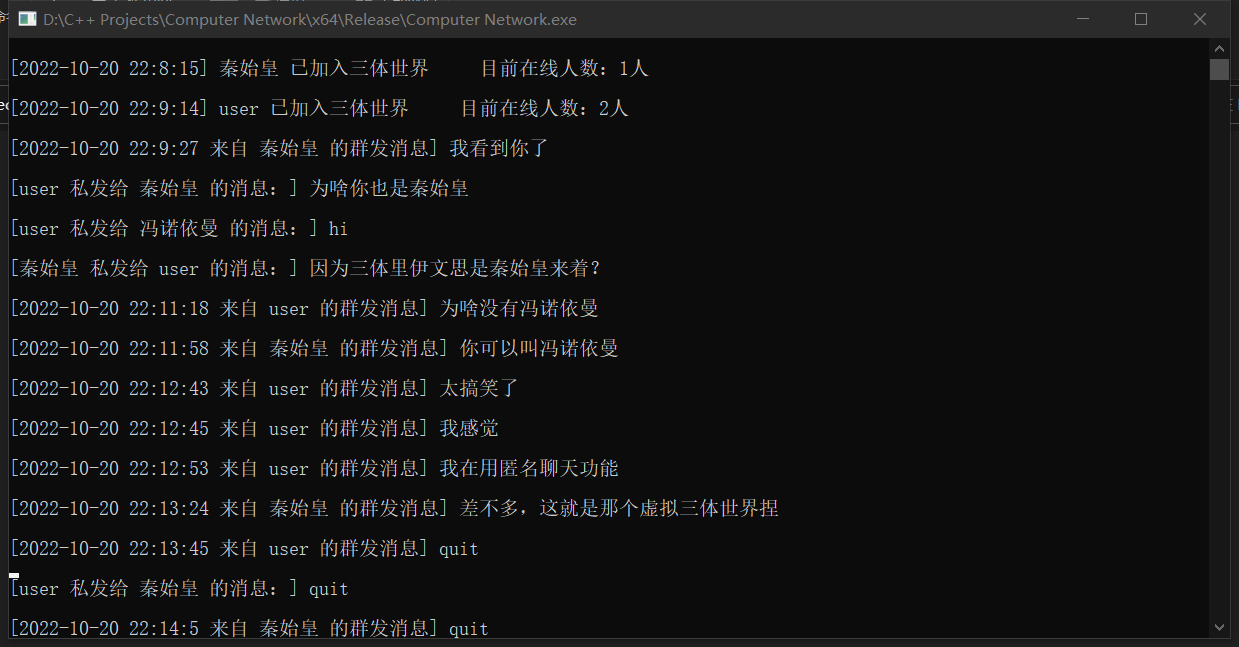
1. DWORD WINAPI send(LPVOID lparam\_socket) {
2. *// 接受消息直到quit退出聊天*
3. int sendResult;
4. SOCKET\* sendSocket = (SOCKET\*)lparam\_socket;
5. while (1)
6. {
7. *// 发送消息*
8. char sendBuf[DEFAULT\_BUFLEN] = "";
9. char temp[DEFAULT\_BUFLEN] = "";
10. cout << endl << endl;
11. cout << "-----------------------------------------------------------------" << endl;
12. cout << "请输入你的消息：";
13. cin >> temp;
14. *// 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*
15. if (temp == quit\_string) {
16. closesocket(\*sendSocket);
17. cout << endl << "您已退出" << endl;
18. return 1;
19. }
20. while (1) {
21. cout << "如群发，输入1；如私发，输入2：";
22. string flag;
23. cin >> flag;
24. if (flag == "1") {
25. sendBuf[0] = '1';
26. for (int i = 1; i <= 20; i++)sendBuf[i] = user\_name[i - 1];
27. break;
28. }
29. else if (flag == "2") {
30. sendBuf[0] = '2';
31. char des\_user[20];
32. cout << "请输入要私发的用户名：";
33. cin >> des\_user;
34. for (int i = 1; i <= 20; i++)sendBuf[i] = des\_user[i - 1];
35. break;
36. }
37. else {
38. cout << "目前不支持此功能，请重新输入" << endl;
39. }
40. }
41. for (int i = 21; i < DEFAULT\_BUFLEN; i++) {
42. sendBuf[i] = temp[i-21];
43. }
44. sendResult = send(\*sendSocket, sendBuf, DEFAULT\_BUFLEN, 0);
45. if (sendResult == SOCKET\_ERROR) {
46. cout << "send failed with error: " << WSAGetLastError() << endl;
47. closesocket(\*sendSocket);
48. WSACleanup();
49. return 1;
50. }
51. else {
52. SYSTEMTIME systime = { 0 };
53. GetLocalTime(&systime);
54. cout << endl << systime.wYear << "-" << systime.wMonth << "-" << systime.wDay << " ";
55. cout << systime.wHour << ":" << systime.wMinute << ":" << systime.wSecond;
56. cout << " 消息已成功发送" << endl;
57. cout << "-----------------------------------------------------------------" << endl;
58. }
59. Sleep(1000); *// 停顿1秒再接收输入*
60. }
61. }
62. **整体流程图**

整体流程图如下：



1. **实验结果分析**

服务器端打印消息：

****

客户端打印消息：





实验结果基本符合预期

1. **不足与反思**
2. 输入输出在打印时有时候会重叠，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。
3. 可以考虑加入消息队列机制。
4. 加入可视化UI界面增加用户体验。