



南開大學  
Nankai University

计算机学院

《计算机网络》实验报告

利用 Socket 编写聊天程序

姓名：文昱韦

学号：2213125

专业：计算机科学与技术

2024 年 10 月 20 日

# 目录

<b>1 实验内容</b>	<b>1</b>
<b>2 协议设计</b>	<b>1</b>
2.1 消息类型 . . . . .	1
2.2 语法 . . . . .	1
2.3 语义 . . . . .	1
2.4 时序 . . . . .	1
<b>3 各模块功能</b>	<b>1</b>
3.1 服务器模块 . . . . .	1
3.1.1 功能概述 . . . . .	1
3.1.2 代码分析 . . . . .	2
3.2 客户端模块 . . . . .	4
3.2.1 功能概述 . . . . .	4
3.2.2 代码分析 . . . . .	5
<b>4 运行结果展示</b>	<b>6</b>
<b>5 遇到的问题及分析</b>	<b>8</b>

## 1 实验内容

利用 C 或 C++ 语言，使用流式 Socket，编写一个聊天程序。使用基本的 Socket 函数完成程序，不允许使用 CSocket 等封装后的类编写程序。程序应有基本的对话界面，但可以不是图形界面。程序应有正常的退出方式。完成的程序应能支持多人聊天，支持英文和中文聊天。

## 2 协议设计

### 2.1 消息类型

- **文本消息**：用户输入的聊天信息，发送给服务器并广播给其他客户端。
- **连接消息**：客户端成功连接服务器时，服务器发送的确认消息。
- **断开消息**：当客户端断开连接时，服务器通知所有其他客户端该客户端已断开。

### 2.2 语法

- **文本消息**：{time} Client{client\_id}: {message}"
- **连接消息**：{time} Server: Client{client\_id} is connected.
- **断开消息**：{time} Server: Client{client\_id} is disconnected.

### 2.3 语义

{time} 是以“年-月-日 时: 分: 秒”为格式的时间标识。{client\_id} 是客户端的标识（程序中使用了端口号作为标识）。{message} 是用户输入的内容。输出时，时间和其他内容分为两行显示。

### 2.4 时序

1. 客户端连接服务器。
2. 服务器确认连接并广播连接消息。
3. 客户端发送文本消息，服务器接收并广播给其他客户端。
4. 客户端输入“exit”以断开连接，服务器广播断开消息。

## 3 各模块功能

### 3.1 服务器模块

#### 3.1.1 功能概述

- 初始化 Winsock，创建和绑定套接字。

- 监听并接受客户端连接。
- 为每个客户端创建处理线程，接收并广播消息。
- 管理客户端列表，确保线程安全。

### 3.1.2 代码分析

我们使用 `vector<SOCKET>` 来存储已连接的客户端套接字。

```
1 vector<SOCKET> clients;
```

当需要向所有连接的客户端广播消息时，服务器调用该函数。函数使用 `lock_guard` 锁住互斥锁，确保对 `clients` 列表的安全访问。遍历所有客户端套接字，检查每个套接字是否为消息的发送者，如果不是，就通过 `send` 函数发送消息。

```
1 void broadcastMessage(const string &message, SOCKET sender_socket)
2 {
3     lock_guard<mutex> lock(client_mutex); // 锁住互斥锁，以确保线程安全访问客户端列表
4     for (SOCKET client_socket : clients)
5         if (client_socket != sender_socket)
6             send(client_socket, message.c_str(), message.size(), 0);
7 }
```

每当连接一个客户端，就开启一个新线程，运行 `handleClient` 函数，处理服务器与单个客户端的通信。其首先创建一个缓冲区以接收数据；然后循环接收消息，格式化时间，并在控制台输出；如果接收失败，发送断开连接的消息，关闭套接字并从客户端列表中移除；发送接收到的消息给其他客户端。

```
1 void handleClient(SOCKET client_socket)
2 {
3     char buffer[BUFFER_SIZE]; // 用于存储从客户端接收的数据
4     while (true)
5     {
6         memset(buffer, 0, BUFFER_SIZE);
7         // 将缓冲区清零，以准备接收新消息
8         int bytes_received = recv(client_socket, buffer, BUFFER_SIZE, 0);
9         // 接收来自客户端的数据
10        auto currentTime = std::chrono::system_clock::now();
```

```

9         time_t timestamp = std::chrono::system_clock::to_time_t(
currentTime);
10         tm localTime;
11         localtime_s(&localTime, &timestamp);
12         char timeStr[50];
13         if (bytes_received <= 0)
14         {
15             string message = std::string("(" + timeStr + ")") + "\n" + "
Server: Client" + std::to_string(client_socket) + " is disconnected.\n
";
16             cout << message;
17             broadcastMessage(message, client_socket);
18             closesocket(client_socket);
// 关闭与该客户端的连接
19             lock_guard<mutex> lock(client_mutex);
// 锁住互斥锁，以安全地操作客户端列表
20             clients.erase(remove(clients.begin(), clients.end(),
client_socket), clients.end()); // 从列表中移除断开的客户端
21             break;
// 结束当前客户端处理线程
22         }
23         strftime(timeStr, sizeof(timeStr), "%Y-%m-%d %H:%M:%S", &
localTime); // 格式化时间输出
24         string message = std::string("(" + timeStr + ")") + "\n" + "
Client" + std::to_string(client_socket) + ": " + buffer + "\n";
25         cout << message;
26         broadcastMessage(message, client_socket); // 广播消息给其他客户端
27     }
28 }

```

主函数内初始化 Winsock 库、创建监听套接字并绑定到指定端口后，便进入主循环。主循环不断运行，服务器持续接受客户端连接。accept 函数用于阻塞地等待一个新的客户端连接。当有客户端请求连接时，它返回一个新的套接字 client\_socket，用于与该客户端进行通信。如果 client\_socket 为 INVALID\_SOCKET，则表示连接失败，打印错误信息并继续下次循环，不执行后续代码。然后，我们将新连接的客户端套接字添加到 clients 向量中，之后创建一条消息，包含连接的时间和已连接客户端的套接字标识（端口号），并将这条消息输出到控制台、调用 broadcastMessage 函数广播给其他

已连接的客户端。最后，使用 `thread` 创建一个新线程来处理新连接的客户端，通过 `handleClient` 函数进行消息接收和处理，使用 `detach` 方法使线程与主线程分离，这样新线程将独立于主线程执行。

```
1 while (1)
2 {
3     SOCKET client_socket = accept(server_socket, nullptr, nullptr); // 接
    受一个新的客户端连接
4     if (client_socket == INVALID_SOCKET)
5     {
6         cerr << "Server accept failed." << endl;
7         continue;
8     }
9     lock_guard<mutex> lock(client_mutex); // 锁住互斥锁，以线程安全地操作
    客户端列表      clients.push_back(client_socket);      // 将新连接的
    客户端添加到客户端列表
10    auto currentTime = std::chrono::system_clock::now();
11    time_t timestamp = std::chrono::system_clock::to_time_t(currentTime);
12    tm localTime;
13    localtime_s(&localTime, &timestamp);
14    char timeStr[50];
15    strftime(timeStr, sizeof(timeStr), "%Y-%m-%d %H:%M:%S", &localTime);
    // 格式化时间输出
16    string message = std::string("(" + timeStr + ")") + "\n" + "Server:
    Client" + std::to_string(client_socket) + " is connected.\n";
17    cout << message;
18    broadcastMessage(message, client_socket);
19    thread(handleClient, client_socket).detach(); // 为新客户端启动一个线
    程，并与主线程分离
20 }
```

## 3.2 客户端模块

### 3.2.1 功能概述

- 初始化 Winsock，创建套接字并连接到服务器。
- 启动接收线程，实时接收并显示服务器消息。
- 主线程负责读取用户输入并发送消息给服务器。

- 处理用户输入 “exit” 关闭连接。

### 3.2.2 代码分析

当一个新的客户端创建时，我们启动一个新线程运行 `receiveMessages` 函数，用于从服务器接收消息。其重要通过 `recv` 函数用于接收数据。如果成功接收到数据，则将其输出到控制台。

```
1 void receiveMessages(SOCKET client_socket)
2 {
3     char buffer[BUFFER_SIZE]; // 用于存储接收的数据
4     while (true)
5     {
6         memset(buffer, 0, BUFFER_SIZE); // 清零缓冲区
7         int bytes_received = recv(client_socket, buffer, BUFFER_SIZE, 0);
8         // 接收数据
9         if (bytes_received > 0)
10             cout << buffer;
11     }
```

主函数内，先通过 `WSAStartup` 用于初始化 Winsock 库，使用 `socket` 函数创建一个 TCP 套接字，并创建了一个 `sockaddr_in` 结构体来存储服务器的地址信息。

然后，我们通过 `connect` 函数连接到指定的服务器。连接成功后，使用 `thread` 启动一个新线程，调用 `receiveMessages` 函数接收来自服务器的消息，并使用 `detach` 分离该线程。进入主循环，从标准输入读取用户输入的消息，如果输入为 “exit”，则退出循环；其他信息使用 `send` 函数将用户输入的消息发送给服务器。

```
1 // 连接到服务器
2 if (connect(client_socket, (sockaddr *)&server_addr, sizeof(server_addr))
3     == SOCKET_ERROR)
4 {
5     cout << "Connection to server failed." << endl;
6     closesocket(client_socket); // 关闭套接字
7     WSACleanup();               // 清理Winsock资源
8     return -1;
9 }
10 cout << "(Connected to server. You can start chatting! Type \"exit\" to
    end the conversation.)" << endl;
```

```
11 thread(receiveMessages, client_socket).detach(); // 启动一个新线程接收服
    务器的消息，并与主线程分离
12
13 char message[BUFFER_SIZE]; // 用于存储用户输入的消息
14 while (true)
15 {
16     cin.getline(message, BUFFER_SIZE); // 从标准输入读取用户输入的消息
17     if (strcmp(message, "exit") == 0)
18     {
19         cout << "Exiting chat..." << endl;
20         break;
21     }
22     send(client_socket, message, strlen(message), 0); // 将用户输入的消息
    发送给服务器
23 }
```

## 4 运行结果展示

首先启动 `server.exe`，如图4.1所示。

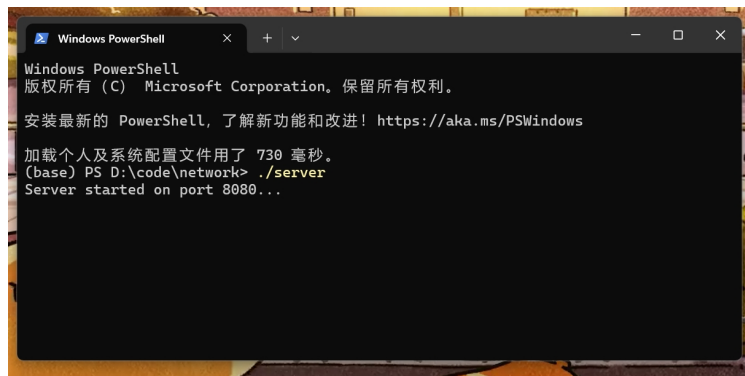


图 4.1: 启动 server

然后，我们依次启动两个客户端，连接时间与连接消息显示在了服务器端和已经连接的客户端，如图4.2所示。



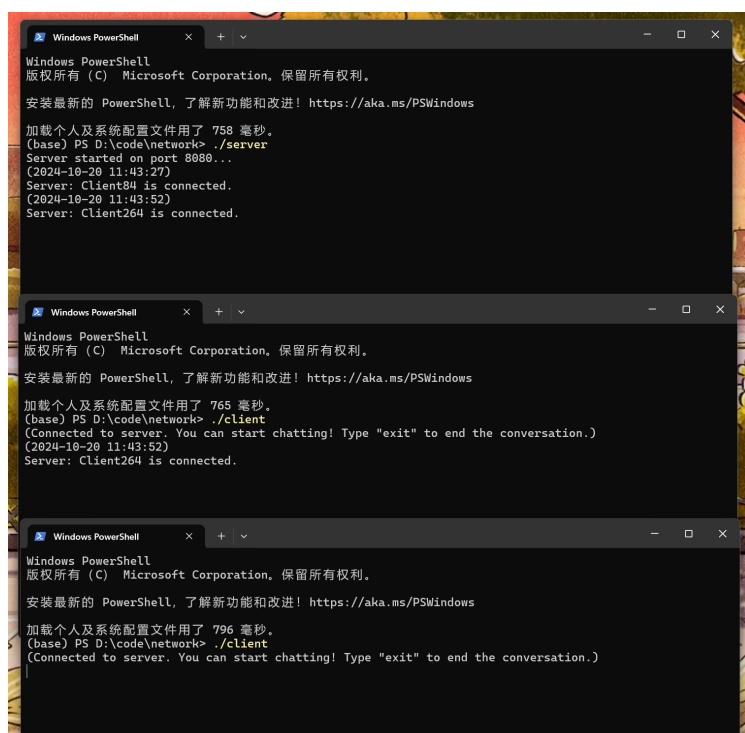


图 4.2: 启动 client

之后，我们从客户端 84 输入消息“你好”，发送消息的时间以及消息内容输出到了服务器端和其他客户端上（图4.3）。然后，我们从客户端 264 输入消息“hi”，发送消息的时间以及消息内容也输出到了服务器端和其他客户端上（图4.4）。

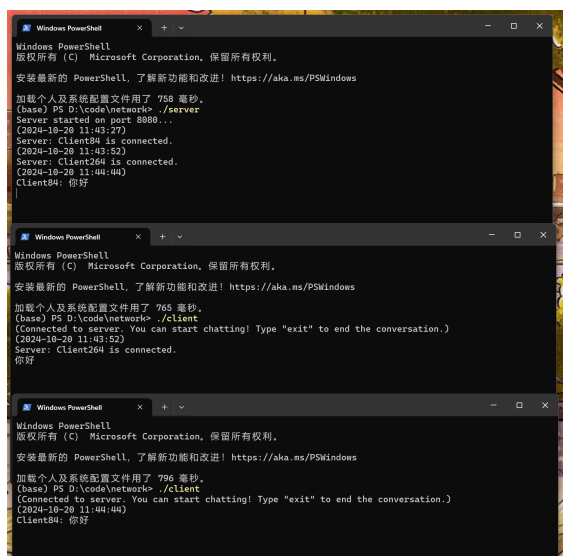


图 4.3: 客户端 84 输入消息“你好”

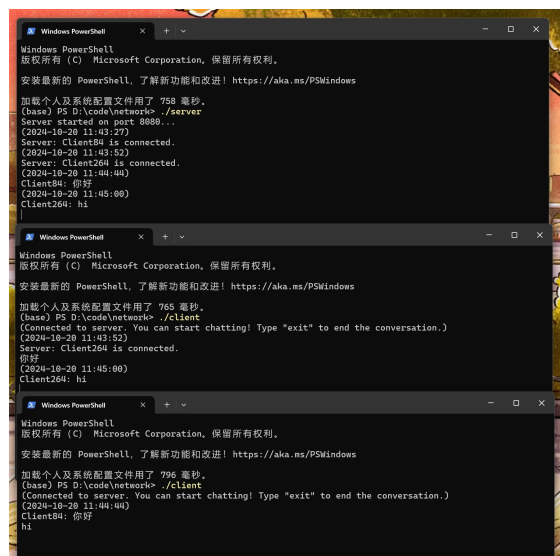
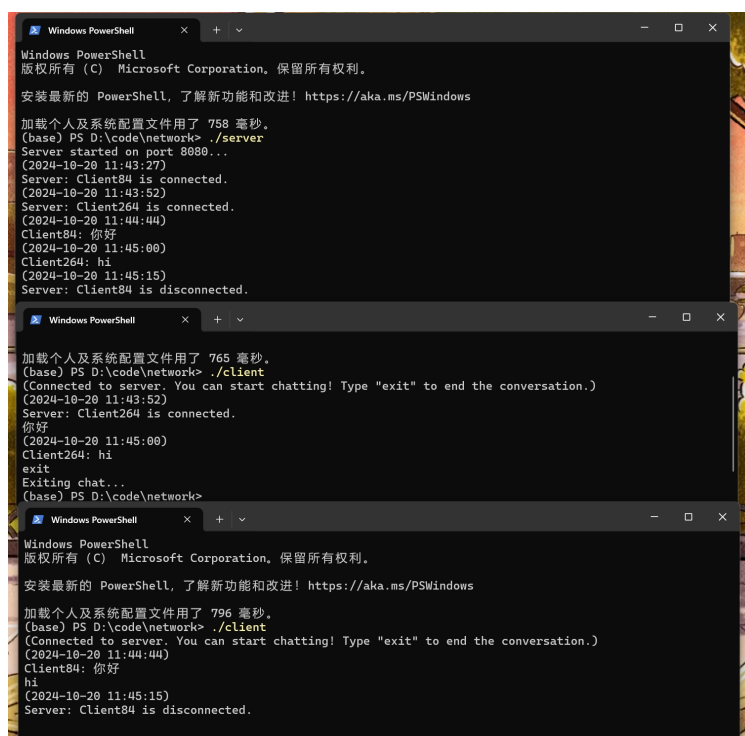


图 4.4: 客户端 264 输入消息“hi”

最后，我们在客户端 264 输入“exit”，该客户端成功退出，其断开连接的时间及消息都输出到了服务器和其他已连接的客户端上，如图4.5所示。



```
Windows PowerShell
版权所有 (C) Microsoft Corporation. 保留所有权利。

安装最新的 PowerShell, 了解新功能和改进! https://aka.ms/PSWindows

加载个人及系统配置文件用了 758 毫秒。
(base) PS D:\code\network> ./server
Server started on port 8880...
(2024-10-20 11:43:27)
Server: Client84 is connected.
(2024-10-20 11:43:52)
Server: Client264 is connected.
(2024-10-20 11:44:44)
Client84: 你好
(2024-10-20 11:45:00)
Client264: hi
(2024-10-20 11:45:15)
Server: Client84 is disconnected.

Windows PowerShell
版权所有 (C) Microsoft Corporation. 保留所有权利。

安装最新的 PowerShell, 了解新功能和改进! https://aka.ms/PSWindows

加载个人及系统配置文件用了 765 毫秒。
(base) PS D:\code\network> ./client
(Connected to server. You can start chatting! Type "exit" to end the conversation.)
(2024-10-20 11:43:52)
Server: Client264 is connected.
你好
(2024-10-20 11:45:00)
Client264: hi
exit
Exiting chat...
(base) PS D:\code\network>

Windows PowerShell
版权所有 (C) Microsoft Corporation. 保留所有权利。

安装最新的 PowerShell, 了解新功能和改进! https://aka.ms/PSWindows

加载个人及系统配置文件用了 796 毫秒。
(base) PS D:\code\network> ./client
(Connected to server. You can start chatting! Type "exit" to end the conversation.)
(2024-10-20 11:44:44)
Client84: 你好
hi
(2024-10-20 11:45:15)
Server: Client84 is disconnected.
```

图 4.5: client 断开连接

以上结果显示, 程序能够正常运行。

## 5 遇到的问题及分析

在处理输出时间的相关内容时, 发现了每次打印消息的时间总是上一次发送消息的时间, 例如某个客户端在第 1 分和第 5 分分别发送了两条消息, 第 5 分发送的消息显示的时间却是第 1 分。分析后发现, 在 `handleClient` 函数里, 函数总是“卡”在 `recv` 函数那里, 直到接收到一条新消息。而我记录时间的相关代码在这个函数之前。也就是说, 当一次消息接收到后所用到的时间, 是在接收到这条消息之前就已经初始化好的 (具体而言, 是在上一次接收消息时, 就立刻执行了获取时间的相关代码)。因此, 应该将获取时间的代码放到 `recv` 函数之后, 这样才能正确输出每次接收到消息时的时间。