南開大學
Nankai University

计算机学院

计算机网络实验报告

实验 1：利用 Socket 编写一个聊天程序

姓名：张铭徐

学号：2113615

专业：计算机科学与技术

2023 年 10 月 13 日

目录

1 实验内容及要求

1.1 实验内容

在本次实验中，我们拟实现一个利用 Socket 编写的聊天程序，使得其能够在本地进行进程间的通信，具体而言，要求如下：

- （1）给出你聊天协议的完整说明。
- （2）利用 C 或 C++ 语言，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。
- （3）使用流式套接字、采用多线程（或多进程）方式完成程序。
- （4）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。
- （5）完成的程序应能支持多人聊天，支持英文和中文聊天。
- （6）编写的程序应该结构清晰，具有较好的可读性。
- （7）在实验中观察是否有数据的丢失，提交源码和实验报告。

在本次实验中，我们使用了 C++ 语言，利用了 Winsock2 库进行对应的训练，并拟实验一些拓展功能，在后面的章节中我们会给出详细的通信协议，设计思路以及代码描述。

1.2 实验背景知识描述

我们本次实验实际上是在进行数据通信，我们拟采用 Socket 编程；Socket 是一个非常核心的概念，我们在实验之前，先考虑一些基础的内容和知识点。

- 套接字，也称之为 Socket 是一个抽象表示，用于在网络上的两个节点之间进行双向通信。它有一个唯一的组合：IP 地址和端口号。在大多数编程语言和操作系统中，Socket 通常表示为一个提供网络服务的接口或对象。其分为流套接字（Stream Sockets）：使用 TCP（传输控制协议）进行通信，它是可靠的、面向连接的；以及数据报套接字（Datagram Sockets）：使用 UDP（用户数据报协议）进行通信，它是不可靠的、无连接的。
- IP 地址是网络上的每台计算机或设备的唯一标识符；端口号是在计算机内部用来标识特定进程的数字。端口号和 IP 地址结合在一起为我们提供了一个完整的地址，使得数据可以准确地发送到网络中的正确位置。我们可以形象化的将 IP 地址理解为一个大的货物交易港口，而端口 (Port) 则可以理解为在港口中的不同口岸，每一个口岸都可以进行商品的往来运输，那么同样的，在计算机网络中，我们确定 IP 地址，实际上是在确定对应的港口，而确定端口号，则是确定我们到底需要将商品从哪一个口岸运输过去。
- TCP/IP 是一个协议族，定义了数据如何在网络上进行传输。其中，TCP 负责数据的传输，而 IP 负责数据的路由。TCP 可以保证数据包的顺序和完整性；在数据传输之前，需要建立连接，并且允许在通信双方之间进行双向数据传输。而 IP 负责将数据包路由到目标地址；无需保证数据包的顺序或完整性。

接下来我们考虑为什么计算机之间可以建立通信？计算机网络的基础是通信协议。协议是一组规则和约定，计算机遵循这些规则来传输数据。当两台计算机遵循相同的协议时，它们就可以互相通信。

例如，当我们在浏览器中打开一个网页时，我们的计算机（客户端）使用 HTTP 或 HTTPS 协议向服务器发送请求。服务器根据这些协议的规则响应请求，返回网页的内容。对于基本的通信而言，其可以抽象为以下几个步骤：

- 连接建立: 例如，在 TCP 中，使用三次握手来建立连接。
- 数据传输: 数据被分割成多个小包进行传输。
- 连接终止: 例如，在 TCP 中，使用四次挥手来终止连接。

那么在本次实验中，我们同样遵循上述三个步骤，具体的内容将在后续的技术路线部分详细说明。

1.3 实验平台介绍

本次实验拟采用 Windows-11 系统，使用 Windows Sockets 2 (Winsock2) 进行网络通信。开发环境采用 VScode 编译器，使用 GCC 进行后续的编译过程。

2 技术路线

2.1 库函数使用说明

本次实验我们采用 Winsock2 库进行 Socket 编程实验，我们使用的库函数功能如下所示：

- **WSAStartup()**: 初始化 Windows Sockets 库。
- **socket()**: 创建一个新的套接字。
- **bind()**: 将套接字与特定的 IP 地址和端口号关联起来。
- **listen()**: 开始在绑定的地址和端口上监听传入的连接。
- **accept()**: 接受一个传入的连接请求。
- **connect()**: 请求连接到服务器。
- **send()**: 发送数据到套接字。
- **recv()**: 从套接字接收数据。
- **closesocket()**: 关闭套接字，释放资源。
- **WSACleanup()**: 终止对套接字库的使用，清理资源。
- **inet_addr()**: 将以点分隔的 IPv4 地址转换为长整数格式。

2.2 拟实现功能说明

我们本次实验实现一个 Socket 通信程序,实现两个端:服务器端 (*server.cpp*) 和用户端 (*client.cpp*),其中服务器端起到的主要作用是新建一个服务器，指定一个端口持续性进行监听，当一个客户端试图连接时，服务器接受这个连接并创建一个新的线程来处理这个客户端的消息，服务器从连接的客户端接收消息，并将其广播到应该广播的客户端位置；而客户端的主要作用是连接到服务器的指定地址和端口，并发送数据，接受服务器端返回的信息。

除了要求的内容外，我们仍实现了昵称，用户编号等内容，并可选择将消息私发给某一个用户，也即实现了私聊功能！具体的功能实现原理详见后文拓展功能说明。

2.3 技术路线

正如前文所说，我们在本次实验中遵循连接建立，数据传输，连接终止这三部分，我们分别对这三部分予以说明:

2.3.1 连接建立

在服务器端 在服务器端，我们使用以下代码来进行初始化的连接建立:

server.cpp

```
WSADATA wsaData;
WSAStartup(MAKEWORD(2, 2), &wsaData);
SOCKET serverSocket = socket(AF_INET, SOCK_STREAM, 0);
sockaddr_in serverAddr;
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(PORT);
serverAddr.sin_addr.S_un.S_addr = INADDR_ANY;
bind(serverSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
listen(serverSocket, 5);
```

具体而言，我们使用 WSAStartup() 初始化 Winsock 库，然后使用 socket() 创建一个新的套接字来监听客户端的连接请求；使用 bind() 将新创建的套接字与特定的 IP 地址和端口号绑定；使用 listen() 开始在该地址和端口上监听传入的连接请求，最后当有客户端请求连接时，使用 accept() 来接受这个连接请求。

在客户端 类似的，我们使用 WSAStartup() 初始化 Winsock 库，并使用 socket() 创建一个新的套接字，然后我们使用 connect() 请求连接到服务器的指定地址和端口，一旦连接成功，客户端可以开始与服务器交换信息。

client.cpp

```
WSADATA wsaData;
WSAStartup(MAKEWORD(2, 2), &wsaData);
SOCKET clientSocket = socket(AF_INET, SOCK_STREAM, 0);
sockaddr_in serverAddr;
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(PORT);
serverAddr.sin_addr.S_un.S_addr = inet_addr(SERVER_IP);
```

```
8     connect(clientSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
```

2.3.2 数据传输

在服务器端 对于每个已连接的客户端，服务器都会创建一个新线程来处理它的消息，然后使用 recv() 从连接的客户端接收消息。并根据消息的内容，服务器会决定是否广播这个消息到所有客户端或仅仅发送给特定的客户端，最后使用 send() 将消息发送到指定的客户端。

server.cpp

```
1     while (true) {
2         SOCKET clientSocket = accept(serverSocket, NULL, NULL);
3         int clientID = globalID++;
4         clients.push_back({ clientSocket, clientID });
5         char buffer[1024];
6         recv(clientSocket, buffer, sizeof(buffer), 0);  // 接收昵称
7         std::cout<<buffer<<std::endl;
8         send(clientSocket, (char*)&clientID, sizeof(clientID), 0);
9         clientNicknames[clientID] = buffer;
10        std::thread(handleClient, clientSocket, std::ref(clientID)).detach();
11    }
```

在客户端 在客户端，我们对消息进行必要的处理，加入一些必要信息后，将整合后的消息通过 send() 发送消息到服务器。并且使用 recv() 来接收来自服务器的消息，也就是他人对于该用户的通讯信息。当然，在这部分我们可以看到，我们加入了一些特定的功能，例如退出程序的标识符，私聊的标识符等等。

client.cpp

```
1     char buffer[1024];
2     Message msg;
3     msg.senderID = clientID;
4     strcpy(msg.nickname, nickname);
5     while (true) {
6         const char* endline = "Quit";
7         const char* func = "private";
8         std::cin.getline(msg.content, sizeof(msg.content));
9         std::time_t now = std::time(nullptr);
10        std::strftime(msg.time, sizeof(msg.time), "%Y-%m-%d %H:%M:%S",
              std::localtime(&now));
11        if (strcmp(msg.content, endline) == 0) {
12            std::cout << "Thank you for using! The program will shutdown!" <<
                  std::endl;
13            break;
14        }
15        if(strcmp(msg.content,func) == 0){
16            std::cout << "You will send a private message! Please enter the clientID:
                  ";
```

```
            std::cin >> msg.targetID;
            std::cout << "Please enter the message: ";
            std::cin.getline(msg.content, sizeof(msg.content));
        } else {
            msg.targetID = -1;  // Broadcast
        }
        send(clientSocket, (char*)&msg, sizeof(msg), 0);
    }
```

2.3.3 连接终止

```
    closesocket(serverSocket);
    WSACleanup();
```

对于客户端和服务器端而言，我们都使用 closesocket() 来关闭连接。使用 WSACleanup() 终止对 Winsock 库的使用并释放资源。

3 通信协议说明

3.1 协议概述

本聊天协议基于 TCP，确保数据的可靠性和有序性。协议定义了客户端和服务器之间的消息格式和交互方式，使得客户端可以发送消息给其他客户端，私聊或群聊。

3.2 基础技术

TCP: 传输控制协议，为网络上的计算机之间在无连接的通信网络上进行可靠的数据通信。它处理数据如何打包、寻址、传输、接收等问题。

Socket: 一个网络通信的端点，可以生成网络上的进程之间的通信链路。

3.3 消息格式

消息是使用以下结构发送和接收的：

消息格式.cpp

```
struct Message {
    int senderID;
    int targetID;
    char nickname[50];
    char content[512];
    char time[50];
};
```

senderID: 唯一确定一个客户端的标识符。targetID: 指定消息的接收方。特殊值-1 表示消息是广播的，应发送给所有连接的客户端。nickname: 表示发送消息的客户端的昵称。content: 实际的消息内容。time: 消息发送的时间戳。

3.4 交互模式

广播: 当客户端发送的消息的 targetID 为-1 时，服务器会将此消息广播给所有在线的客户端。

私聊: 客户端可以通过指定一个具体的 targetID 来发送私有消息给另一个客户端。

3.5 命令与控制

Quit: 客户端可以发送一个特殊的”Quit” 消息来安全地断开与服务器的连接。

Private: 要发送私有消息，客户端首先发送一个”private” 命令，然后系统将提示他们输入目标用户的 ID 和消息内容。

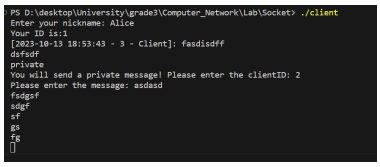
3.6 安全和错误处理

当前版本的协议不包括消息加密、身份验证或错误恢复功能。未来版本可以考虑加入这些功能来增强安全性和可靠性，此外，我们的数据传输是明文传输，包括发送者的 ID，发送的内容，目标 ID 等，这样很容易被抓包软件将数据包捕获，甚至无需破译即可得到相应的敏感信息，后续的内容可以在安全性上做以改良。

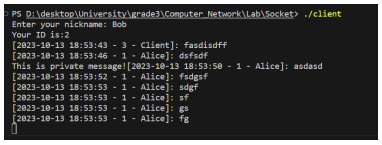
4 实验运行截图

我们将运行 *server.cpp* 以及 *client.cpp* 并对输出结果做以分析：我们首先打开三个用户端，在进入用户端时，系统会提示输入昵称信息，然后用户需要输入昵称并按回车键，然后系统会接收到对应的昵称并返回一个 ClientID 给用户，然后用户就可以自行输入信息进行实验。

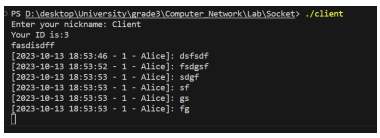
我们可以输入关键词”private” 进入私聊模式，在私聊模式下，用户需要输入即将发送的 ClientID，然后输入对应的消息，只有对应的用户才会接收到该用户发送的消息，其余用户则不会收到该信息。同时，我们系统定义了 “Quit” 关键词作为退出时的标识符，当用户输入到 “Quit” 时，系统会自动的退出，同时服务器端会接收到对应断开连接的标识。



((a)) ID1 用户截图



((b)) ID2 用户截图

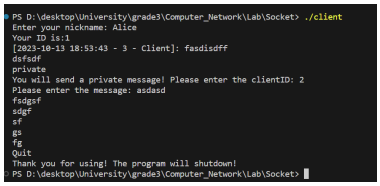


((c)) ID3 用户截图

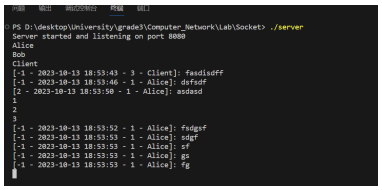
图 4.1: 三个用户的截图

上述三幅图展示了用户端的信息，我们从图中可以看到用户 ID 的分配过程，用户昵称的输入过程，以及公开信息，私聊信息的相关内容，我们可以看到，上述要求实现的较为完美，我们在下面给出退出的运行结果：如图2(a)所示：同样的，在退出系统后，终端会输出退出的标识符，然后该文件执行停止，同样的，在 server 端也会有断链提示。

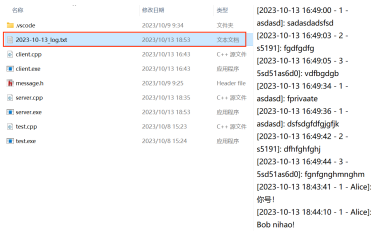
一个完整的项目通常来说会有一定的日志输出，我们在这里同样在服务器端有对应的日志输出，在运行时，系统会自动的输出接收到的消息的相关内容到一个文件中，文件路径与 *server.cpp* 同目录，文件命名为” 日期 _log.txt”。在该文件中有对应的日志输出。图2(b)和图2(c)则分别展示了服务器端的接受信息以及日志输出的相关内容。



((a)) 退出截图



((b)) 服务器端截图



((c)) 日志截图

图 4.2: 服务器与日志截图

5 实验总结

本次实验的目标是设计并实现一个基于 Socket 的通信程序。通过实验，我们成功地构建了一个简单的聊天系统，该系统由服务器端（server.cpp）和客户端（client.cpp）组成。

在本次实验中，我深入了解 Socket 编程：我们深入探讨了 Socket 编程的各个关键步骤，包括建立连接、数据传输和连接的终止，并理解了多线程在数据传输方面的应用：在服务器端，为了处理多个客户端的连接，我们使用了多线程技术。这让我们更好地理解了并发和多线程编程的重要性和挑战。我们设计了一个简单的聊天协议，该协议定义了消息的结构和如何在客户端和服务器之间传输。这个过程教会了我们如何创建并使用自定义协议。

总的来说，这次的实验为我们提供了一个宝贵的机会，使我们深入了解网络编程和 Socket 通信。通过亲自设计和实现聊天系统，我们对这些概念和技术有了更深入的理解。

我们遵循程明明老师的 DOCX 倡议，本次实验源码以及实验报告可以在计算机网络实验仓库中找到！