吉 林 大 学
软件学院
实 验 报 告

实验名称	Windows 平台简单套接字编程				
课程名称	计算机网络课程设计				
姓名		学号		成绩	
提交日期	2024.04.12	座位号			

1. 实验目的

掌握 Windows 平台上简单的客户机端和服务器端的套接字编程。

2. 实验内容

本实验旨在通过设计和实现基于 TCP 协议的简单客户端和服务器端通信程序，以加深对套接字编程原理和实践的理解，并掌握在 Windows 平台上进行网络编程的基本技能。

具体要求包括：①服务器端能够监听特定端口，接受客户端连接，并根据客户端请求返回系统当前时间。②客户端能够连接到服务器端，向服务器发送特定命令（如请求当前时间），并接收服务器端的响应信息并在控制台上显示。

3. 实验分析

- 服务器端
  - 初始化 Winsock 库，创建 TCP 套接字，并将其绑定到指定端口。
  - 开始监听客户端连接请求，一旦有连接请求到达，就接受连接。
  - 处理客户端的请求，如果收到请求为获取当前时间，则获取系统当前时间并返回给客户端。
- 客户端
  - 初始化 Winsock 库，创建 TCP 套接字，并连接到服务器指定的地址和端口。
  - 向服务器发送特定命令（如请求当前时间）。
  - 接收服务器端的响应信息（如当前时间），并在控制台上显示。

4. 问题解答

- 服务器端程序（`class Service`）
  - **1) 初始化服务器（`Service(int port)`）**
    - 创建服务器套接字 serverSocket。
    - 设置服务器地址结构体 serverAddr 中的端口号为指定的 port。
    - 调用 bind()函数将服务器套接字绑定到指定端口。
  - **2) 启动服务器（`void Start()`）**
    - 调用 listen()函数开始监听客户端连接。
    - 在一个循环中调用 AcceptClient()函数接受客户端连接。
  - **3) 接受客户端连接（`void AcceptClient()`）**
    - 调用 AcceptClient()函数接受客户端连接，获取客户端套接字和地址信息。
    - 在接受到客户端连接后，调用 HandleClient()函数处理客户端消息。
  - **4) 处理客户端消息（`void HandleClient(SOCKET clientSocket)`）**
    - 接收客户端发送的请求。
    - 根据请求的类型，执行相应的操作。在这个问题中，需要向客户端发送本机当前的时间。

5) 关闭服务器（~Service()）
- 关闭服务器套接字。

● 客户端程序（class Client）

1) 初始化客户端（Client(const char* serverIP, int serverPort)）
- 创建客户端套接字 clientSocket。
- 设置服务器地址结构体 serverAddr 中的 IP 地址和端口号为指定的 serverIP 和 serverPort。

2) 连接服务器（bool Connect()）
- 调用 Connect()函数连接到服务器。

3) 运行客户端（void Run()）
- 发送请求给服务器端。
- 接收服务器返回的消息，即本机当前的时间。

4) 关闭客户端（~Client()）
- 关闭客户端套接字。

4.3 核心代码（有必要的注释）

● 服务器端

```
void Service::AcceptClient() {
    SOCKADDR_IN clientAddr;
    int addrSize = sizeof(clientAddr);

    // 接受客户端连接，返回客户端套接字
    SOCKET clientSocket = accept(serverSocket, (struct sockaddr*)&clientAddr,
                                 &addrSize);

    if (clientSocket == INVALID_SOCKET) {
        std::cerr << "接受客户端连接失败，错误代码：" << WSAGetLastError() <<
        std::endl;
        closesocket(clientSocket);
        return;
    }
    else {
        // 获取客户端 IP 和端口信息
        char ipBuffer[INET_ADDRSTRLEN];
        inet_ntop(AF_INET, &(clientAddr.sin_addr), ipBuffer, NET_ADDRSTRLEN);
        std::cout << "客户端（" << ipBuffer << "）通过端口（" <<
        ntohs(clientAddr.sin_port) << "）连接成功" << std::endl;

        // 处理客户端消息
        HandleClient(clientSocket);
    }
}
void Service::HandleClient(SOCKET clientSocket) {
    int const CLIENT_MSG_SIZE = 128;
    char inMSG[CLIENT_MSG_SIZE];
    char outMSG[CLIENT_MSG_SIZE];

    while (true) {
        // 接收客户端消息
        int recvResult = recv(clientSocket, inMSG, CLIENT_MSG_SIZE, 0);
        if (recvResult == SOCKET_ERROR) {
            std::cerr << "对话中断，错误代码：" << WSAGetLastError() << std::endl;
            closesocket(clientSocket);
            return;
        }
        std::cout << "客户端命令：" << inMSG << std::endl;
```

```
        // 处理客户端命令
        if (strcmp(inMSG, "当前时间") == 0) {
            // 获取系统当前时间
            SYSTEMTIME systime = { 0 };
            GetLocalTime(&systime);
            sprintf_s(outMSG, CLIENT_MSG_SIZE, "%d-%02d-%02d %02d:%02d:%02d",
                systime.wYear, systime.wMonth, systime.wDay,
                systime.wHour, systime.wMinute, systime.wSecond);

            // 发送时间信息给客户端
            int sendResult = send(clientSocket, outMSG, strlen(outMSG), 0);
            if (sendResult == SOCKET_ERROR) {
                std::cerr<<"发送数据给客户端出错，错误代码："<<WSAGetLastError()
                << std::endl;
            }
        }
        else if (strcmp(inMSG, "退出") == 0) {
            std::cout << "客户端成功退出连接" << std::endl;
            closesocket(clientSocket);
            return;
        }
        else {
            const char* message = "无效命令";
            int sendResult = send(clientSocket, message, strlen(message), 0);
            if (sendResult == SOCKET_ERROR) {
                std::cerr<<"发送数据给客户端出错，错误代码："<<WSAGetLastError()
                << std::endl;
            }
        }
    }
}
```

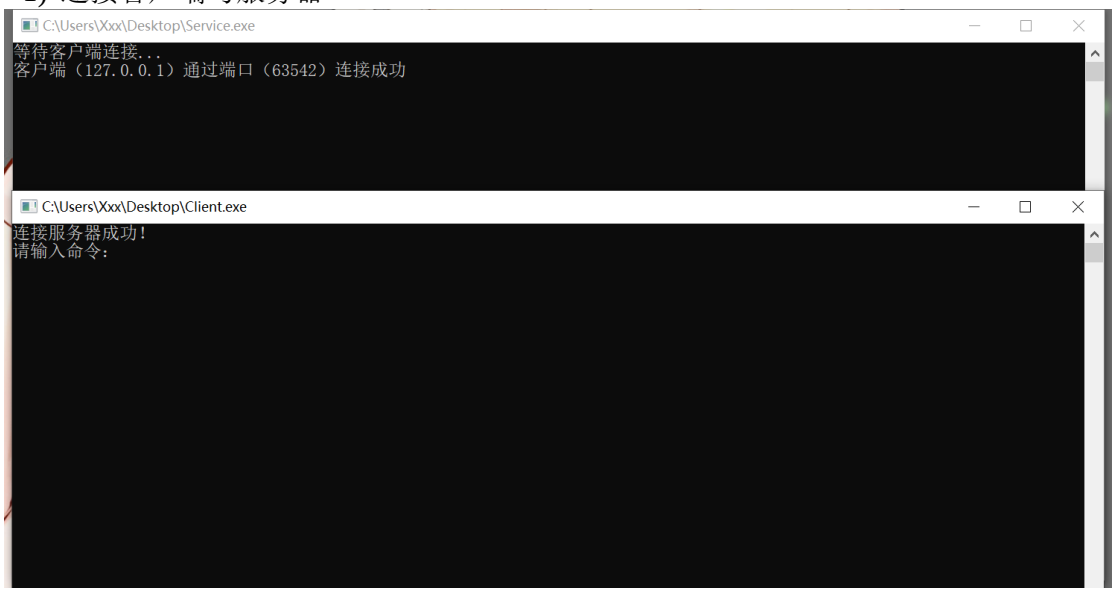
- 客户端

```
void Client::Run() {
    const int BUFFER_SIZE = 128;
    char buffer[BUFFER_SIZE];

    while (true) {
        std::cout << "请输入命令：";
        std::cin >> buffer;

        // 发送命令到服务器
        int sendResult = send(clientSocket, buffer, sizeof(buffer), 0);
        if (sendResult == SOCKET_ERROR) {
            std::cerr << "发送数据给服务器端出错，错误代码：" << WSAGetLastError()
            << std::endl;
        }

        // 如果输入命令是退出，则退出循环
        if (strcmp(buffer, "退出") == 0) {
            break;
        }

        // 接收服务器响应
        memset(buffer, 0, sizeof(buffer));
        int recvResult = recv(clientSocket, buffer, sizeof(buffer), 0);
        if (recvResult == SOCKET_ERROR) {
            std::cerr << "服务器断开连接，对话中断"<< std::endl;
            closesocket(clientSocket);
```

```
            return;
        }
        std::cout << "服务器端应答：" << buffer << std::endl;
    }
}
```

4.4 测试方法、测试数据与测试结果

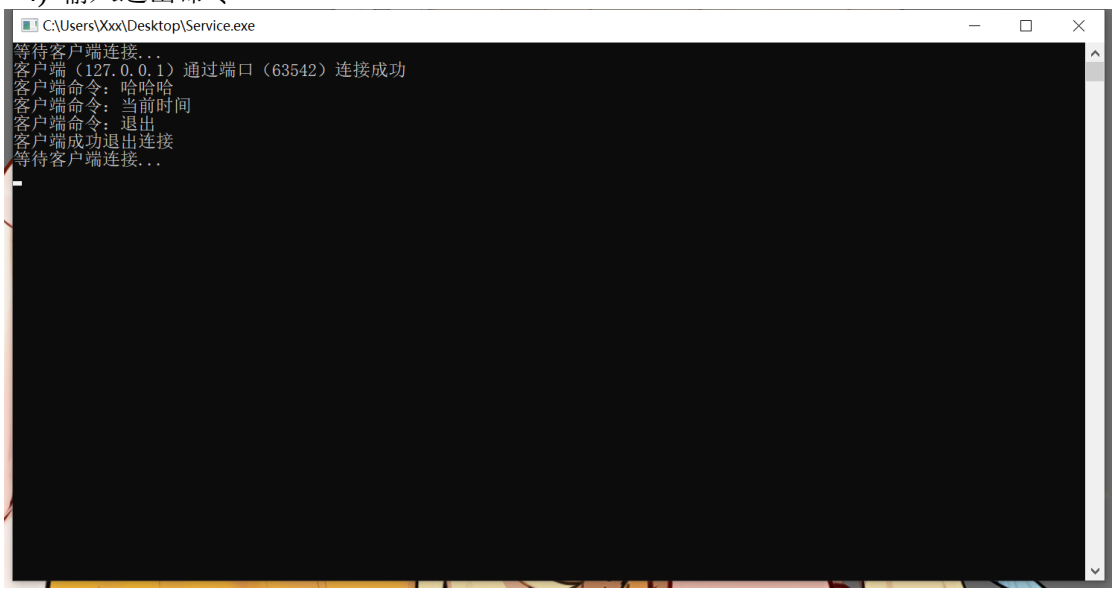
1) 连接客户端与服务器



2) 输入无效命令

3) 输入获取当前时间命令

4) 输入退出命令



4.5 程序的使用说明

- **服务器端**
  - **启动服务器端程序：**请首先编译并运行服务器端程序。该程序将在指定的端口上开始监听客户端的连接请求。
  - **等待客户端连接：**一旦服务器端程序启动，它将持续监听客户端的连接请求。当有客户端连接时，服务器端会显示客户端的 IP 地址和端口号，并等待接收客户端发送的命令。
  - **接收和处理客户端请求：**当客户端向服务器端发送命令时，服务器端会接收该命令并根据命令内容执行相应的操作。当前支持的命令包括“当前时间”和“退出”，服务器端会获取当前系统时间并将其返回给客户端。
  - **关闭服务器端：**直接关闭服务器端程序。

- **客户端**
  - **启动客户端程序：**编译并运行客户端程序，程序已设置好服务器的 IP 地址和端口号，客户端将尝试连接到指定的服务器。

- **发送命令给服务器：**在客户端程序中，输入要发送给服务器的命令。目前支持的命令包括 “当前时间”和“退出”。输入完命令后，按 Enter 键将命令发送给服务器。
- **接收和显示服务器响应：**客户端将等待服务器返回响应。一旦收到服务器的响应，客户端将在屏幕上显示响应内容，例如当前系统时间。
- **发送其他命令：**您可以反复发送不同的命令给服务器，请求其他服务或功能。服务器将根据收到的命令执行相应操作并返回结果。
- **退出客户端：**要退出客户端程序，可以输入特定的退出命令（例如“退出”）或通过其他退出方式。客户端程序将关闭与服务器的连接并退出运行。

4.6 总结

- **程序运行效果评价**
  - **功能实现：**服务器端能够成功监听指定端口，并接受客户端的连接请求。客户端能够连接到服务器端，并发送指定命令，接收服务器端的响应，例如获取当前时间。
  - **稳定性：**程序在一般情况下稳定运行，能够正常处理客户端连接和消息交互。

- **遇到的问题及解决办法**
  - **编译和环境配置：**在 Windows 平台上使用 VC++进行编译需要确保正确配置开发环境和链接网络库。（解决办法：在编译前确保已安装并配置好 Visual Studio，并正确链接 Winsock 库。）
  - **错误处理和异常情况：**程序可能会遇到网络连接异常、命令错误等情况，需要增加错误处理和异常处理机制。（解决办法：优化程序代码，增加错误检查和异常处理，提高程序的健壮性和容错性。）

- **程序特色说明**
  - **模块化设计：**程序采用面向对象的模块化设计，将服务器端和客户端功能封装为类，提高了代码的可维护性和复用性。
  - **面向对象编程：**通过定义 Service 类和 Client 类，实现了面向对象的套接字编程，使程序结构清晰，易于理解和扩展。
  - **基于 TCP 协议：**使用 TCP 协议进行通信，保证数据传输的可靠性和有序性，适用于需要可靠数据传输的场景。
  - **交互式功能：**客户端和服务器端可以实现简单的交互，发送命令和接收响应，体现了套接字通信的基本应用。