《网络程序设计》实践课

实验报告

实验名称：TCP 通信实验

实验类型：设计型

指导教师：贾浩

专业班级：

姓　　名：

学　　号：

实验评分

序号	实验分数构成	分数
1	实验结果(20 分)	
2	实验过程(40 分) （包括实验指导教师提问回答情况）	
3	实验报告书写规范（20 分）	
4	实验思考（20 分）	
实验总分		

一、实验目的

* 掌握 TCP 服务器程序和客户程序的编程流程；
* 熟悉面向连接的 C/S 程序使用的 winsock API。

二、实验设计

开发环境：Visual Studio 2022
题目一:
1、 编写一个 TCP 回显服务器，将收到的客户端信息发送给客户端，同时能在同客户端建立连接后显示客户端地址信息和当前连接数。

- Winsock 初始化：使用 WSAStartup 初始化 Winsock 库。
- 创建服务器套接字：使用 socket 函数创建一个 TCP 套接字。
- 绑定地址和端口：使用 bind 将套接字绑定到本地地址和指定端口。
- 监听连接：使用 listen 函数开始监听客户端连接。
- 接受客户端连接：使用 accept 函数接受客户端连接。
- 多线程处理客户端：为每个客户端创建一个线程，使用_beginthreadex。

2、编写一个 TCP 客户端程序能连接到你编写的服务器，接收服务器信息。主函数使用 int main(int argc,char** argv)形式传入服务器 IP 地址和端口。

- 初始化 Winsock 库。
- 创建一个 TCP 套接字。
- 通过命令行参数获取服务器的 IP 地址和端口。
- 连接到服务器。
- 在一个循环中发送消息并接收服务器的回显。
- 当用户输入"bye"时，退出循环并关闭连接。

3、测试你编写的程序，将测试数据、测试结果和结果分析写入实验报告。

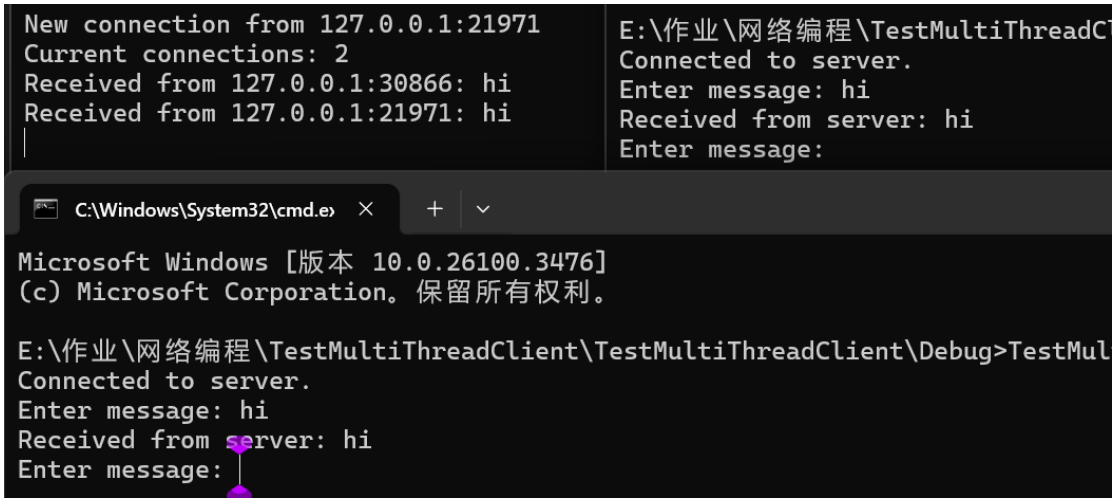
三、实验过程（包含实验结果）

先编写服务器代码，再编写客户端代码，打开客户端传入参数进行测试

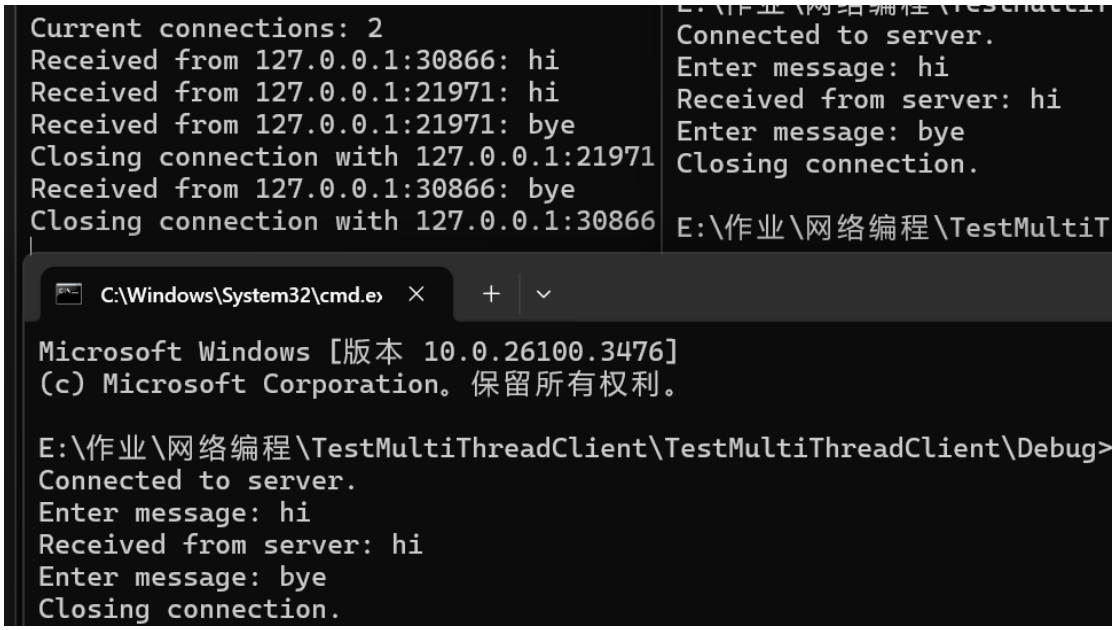
服务端监听 8084 端口，客户端连接本地 8084 端口，连接成功，返回客户端地址 127.0.0.1:30866 和当前连接数 1

再建立一个客户端连接，返回客户端地址 127.0.0.1:21971 和当前连接数 2

发送消息，会有消息来源地址和消息内容



关闭连接，输入 bye 即可关闭



四、讨论与分析（实验思考中问题理解）

问题 1

解决方案： 一开始从 vs 处运行客户端代码始终退出进程，是参数传入问题，后面了解到参

数输入需要使用可执行文件传入参数，就再 cmd 处输入命令“TestMultiThreadClient.exe 127.0.0.1 8084”即可传参成功，

实验思考 1：accept()函数，connect（ ）函数会阻塞吗？如果阻塞，说明在什么情况下阻塞。请给出在 VC 环境下的验证方法。

如果连接队列中没有已完成的连接，accept()函数会阻塞，直到有新的连接请求到来。如果服务器未响应或者网络状况不佳，connect()函数会阻塞，直到连接成功建立或者超时。

```
while (1) {
    // 接受客户端连接
    clientSocket = accept(serverSocket, (struct sockaddr*)&clientAddr, &clientAddrSize);
    if (clientSocket == INVALID_SOCKET) {
        printf("Accept failed.\n");
        continue;
    }

    printf("Client connected: %s:%d\n", inet_ntoa(clientAddr.sin_addr), ntohs(clientAddr.sin_port));
```

```
E:\作业\网络编程\TestMultiThr
Server is listening on port 8084...
```

当服务端没有接收到客户端的连接请求时，服务端会阻塞，一直处于监听状态，等待连接请求。此时若客户端发起请求则进行 accept 连接

```
E:\作业\网络编程\TestMultiThr
Connected to server.
Enter message:
```

```
E:\作业\网络编程\TestMultiThr
Server is listening on port 8084...
Client connected: 127.0.0.1:10858
```

而如果客户端先启动，服务端未启动时，connect 会阻塞

```
E:\作业\网络编程\TestMultiThr
Trying to connect to the server...
```

实验思考 2：connect()函数调用触发什么过程？

检查套接字状态：检查传入的套接字描述符是否有效，以及该套接字是否已经处于连接状态或者其他不适合发起连接的状态。

分配本地端口：为该套接字在系统可用的端口范围内随机选择分配一个本地端口号，用于标识客户端在网络中的位置。

建立连接状态：将套接字的状态设置为 CONNECTING，表示正在尝试与服务器建立连接。

调用网络协议栈：将连接请求传递给网络协议栈，由网络协议栈负责 TCP 通信。

TCP 连接：在使用 TCP 协议进行通信时，connect() 函数会触发三次握手，当三次握手完成后，TCP 连接就建立成功，客户端和服务器可以开始进行数据传输。

实验思考 3：你在服务端和客户端分别使用了哪些 Winsock API 函数，起什么作用？

服务端使用的 Winsock API 函数：

WSAStartup　　初始化 Winsock 库，指定版本（2.2）

socket 创建 TCP 套接字（AF_INET+SOCK_STREAM）
bind 绑定套接字到本地地址（INADDR_ANY:8082）
listen 启动监听模式，设置最大等待连接数（SOMAXCONN）
accept 接受客户端连接，返回新套接字
recv 接收客户端数据
send 回显数据给客户端
closesocket 关闭套接字
WSACleanup 清理 Winsock 资源

客户端使用的 Winsock API 函数：
WSAStartup 初始化 Winsock 库，指定版本（2.2）
socket 创建 TCP 套接字
connect 主动连接服务端（127.0.0.1:8082）
send 发送用户输入数据
recv 接收服务端回显数据
closesocket 关闭套接字

五、实验者自评（从实验设计、实验过程、对实验知识点的理解上给出客观公正的自我评价）

通过本次实验，我对 TCP 服务器和客户端程序的编程流程有了更深入的理解，熟悉了面向连接的 C/S 程序中 Winsock API 的使用。对于实验思考中的问题，我能够准确分析和解释，如 accept() 和 connect() 函数的阻塞情况、connect() 函数调用触发的过程等，表明我对实验知识点有较好的掌握。但在实际应用中，对于一些 API 函数的细节和使用场景还需要进一步加深理解，以提高编程的效率和质量。

六、 附录：关键代码（给出适当注释，可读性高）

```
#define PORT 8084
#define BUFFER_SIZE 1024

// 当前连接数
int connectionCount = 0;

// 客户端通信线程函数
unsigned int __stdcall ClientHandler(void* lpParam) {
    SOCKET clientSocket = *(SOCKET*)lpParam;
    char buffer[BUFFER_SIZE];
    int recvSize;

    // 获取客户端地址信息
    struct sockaddr_in clientAddr;
```

```
        int clientAddrSize = sizeof(clientAddr);
        getpeername(clientSocket, (struct sockaddr*)&clientAddr, &clientAddrSize);
        char clientIp[INET_ADDRSTRLEN];
        inet_ntop(AF_INET, &clientAddr.sin_addr, clientIp, INET_ADDRSTRLEN);
        int clientPort = ntohs(clientAddr.sin_port);

        // 显示客户端连接信息
        printf("New connection from %s:%d\n", clientIp, clientPort);
        connectionCount++;
        // 显示当前连接数
        printf("Current connections: %d\n", connectionCount);
        while (1) {
            // 接收客户端数据
            recvSize = recv(clientSocket, buffer, BUFFER_SIZE, 0);
            if (recvSize <= 0) {
                printf("Client %s:%d disconnected or error occurred.\n", clientIp, clientPort);
                break;
            }

            buffer[recvSize] = '\0'; // 确保字符串以 null 结尾
            printf("Received from %s:%d: %s\n", clientIp, clientPort, buffer);

            // 回显数据给客户端
            send(clientSocket, buffer, recvSize, 0);

            // 如果收到 "bye"，关闭连接
            if (strcmp(buffer, "bye") == 0) {
                printf("Closing connection with %s:%d\n", clientIp, clientPort);
                break;
            }
        }

        // 关闭客户端套接字
        closesocket(clientSocket);
        connectionCount--;

        return 0;
    }

    #define BUFFER_SIZE 1024
    int main(int argc, char** argv) {
        WSADATA wsaData;
        SOCKET clientSocket;
        struct sockaddr_in serverAddr;
```

```
    char buffer[BUFFER_SIZE];
    int recvSize;

    // 检查命令行参数
    if (argc != 3) {
        printf("Usage: %s <server_ip> <port>\n", argv[0]);
        return 1;
    }

    const char* serverIp = argv[1];
    const int port = atoi(argv[2]);

    // 初始化 Winsock
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        printf("WSAStartup failed.\n");
        return 1;
    }

    // 创建客户端套接字
    clientSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (clientSocket == INVALID_SOCKET) {
        printf("Socket creation failed.\n");
        WSACleanup();
        return 1;
    }

    // 设置服务器地址
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = inet_addr(serverIp);
    serverAddr.sin_port = htons(port);

    // 连接到服务器
    if (connect(clientSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) ==
SOCKET_ERROR) {
        printf("Connection failed.\n");
        closesocket(clientSocket);
        WSACleanup();
        return 1;
    }

    printf("Connected to server.\n");

    while (1) {
        // 输入消息
```

```
        printf("Enter message: ");
        fgets(buffer, BUFFER_SIZE, stdin);
        buffer[strcspn(buffer, "\n")] = '\0'; //  去掉换行符

        //  发送消息
        send(clientSocket, buffer, strlen(buffer), 0);

        //  如果输入  "bye"，退出循环
        if (strcmp(buffer, "bye") == 0) {
            printf("Closing connection.\n");
            break;
        }

        //  接收服务器回显
        recvSize = recv(clientSocket, buffer, BUFFER_SIZE, 0);
        if (recvSize <= 0) {
            printf("Server disconnected or error occurred.\n");
            break;
        }

        buffer[recvSize] = '\0'; //  确保字符串以  null  结尾
        printf("Received from server: %s\n", buffer);
    }

    //  关闭客户端套接字
    closesocket(clientSocket);
    WSACleanup();

    return 0;
}
```