《网络程序设计》理论课

实验报告

实验名称：多线程非阻塞模式网络通信实验

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

* 理解 Winsock API 的调用方法
* 了解 TCP 或 UDP 协议的工作原理
* 掌握 TCP 或 UDP 服务端程序和客户端程序的编写流程
* 熟悉非阻塞模式网络程序编写
* 熟悉多线程程序编写。

二、实验设计

开发环境：Visual Studio 2022
该程序已经完成了两个功能

1. 服务端能够接受多个客户端链接，并在服务端为每个客户端通信创建一个线程，由该线程负责和对应客户端通信。
2. 服务端把接收的客户端发送过来的信息回显给客户端，当服务端收到对方的数据为“bye”时，关闭对应套接字，结束该线程。

服务端程序通常需要完成初始化、创建套接字、绑定地址和端口、监听连接、接受连接、处理数据等步骤；客户端程序则需要完成初始化、创建套接字、连接服务器、发送和接收数据等步骤。

参考代码已经完成了阻塞模式网络通信的功能，我将修改参考代码完成非阻塞模式网络通信。

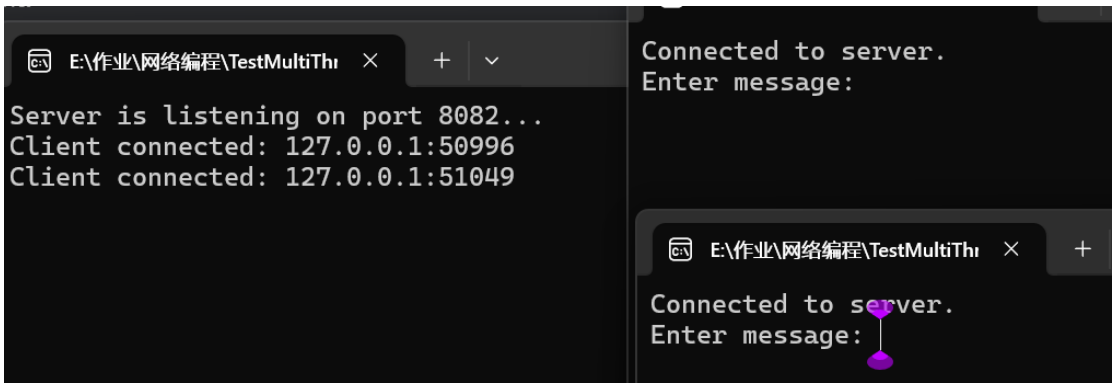
三、实验过程（包含实验结果）

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

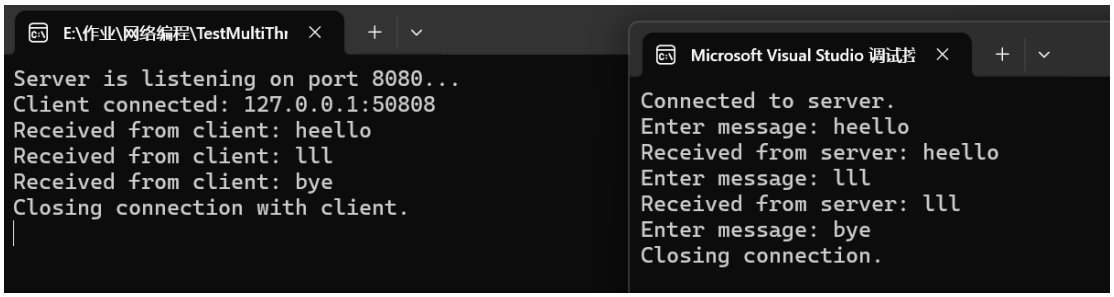
recv 接收服务端回显数据
closesocket　关闭套接字

实验任务一：
服务端能够接受多个客户端链接，并在服务端为每个客户端通信创建一个线程，由该线程负责和对应客户端通信。



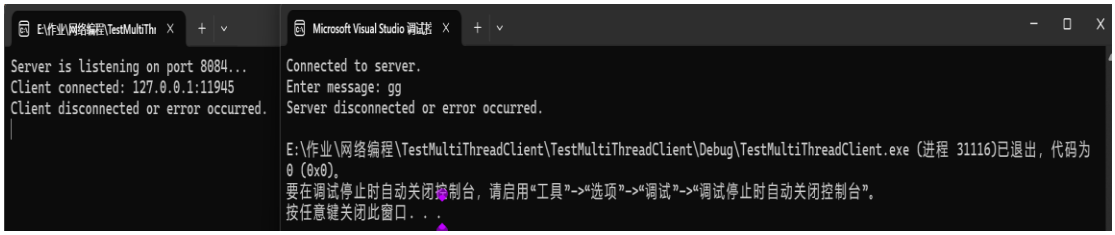
连接到了两个客户端。

实验任务二：
服务端把接收的客户端发送过来的信息回显给客户端，当服务端收到对方的数据为“bye”时，关闭对应套接字，结束该线程。



输入 bye 后线程结束。退出后不会退出会一直等待新的连接

修改代码完成非阻塞式通信：
建立连接前修改套接字工作模式 int ioctlsocket(SOCKET s, long cmd, u_long* argp);
Argp 为 1 时为非阻塞模式，即服务端和客户端 connect 前添加代码：
u_long mode = 1;
ioctlsocket(serverSocket, FIONBIO, &mode);

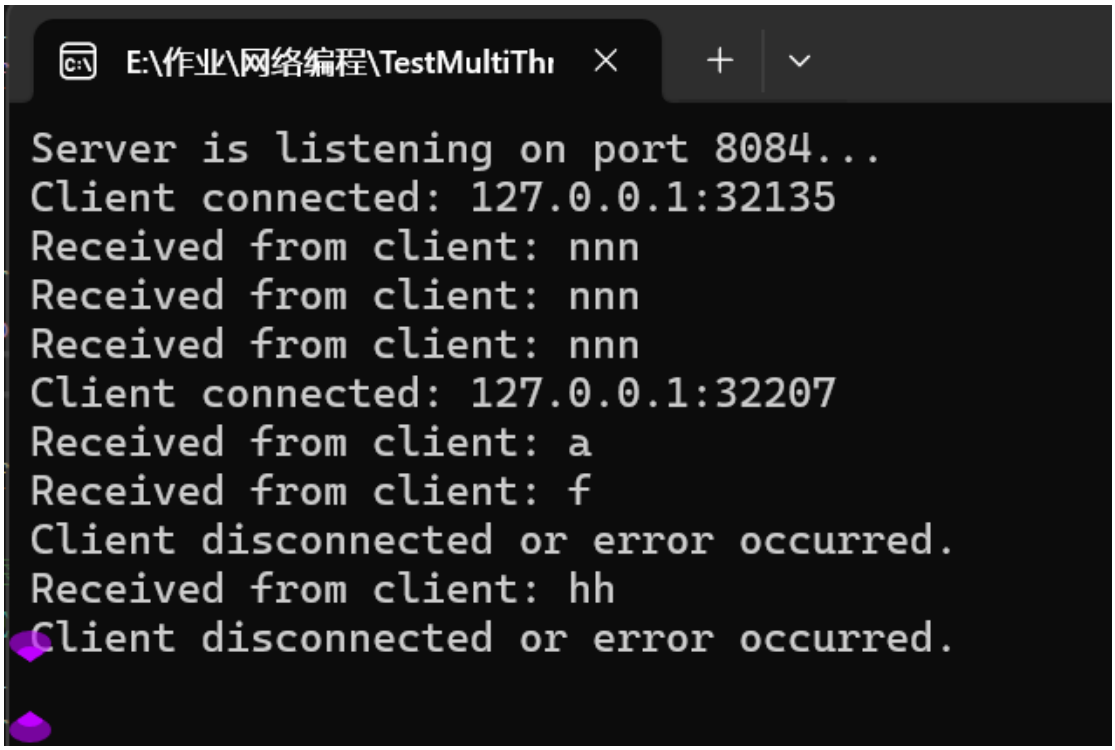


运行后结果入上图所示，原因时非阻塞模式下，如果服务器没有立即响应，recv 会立即返回 SOCKET_ERROR，并设置错误码为 WSAEWOULDBLOCK，客户端没有正确处理这种情况，导致误认为服务器断开连接。

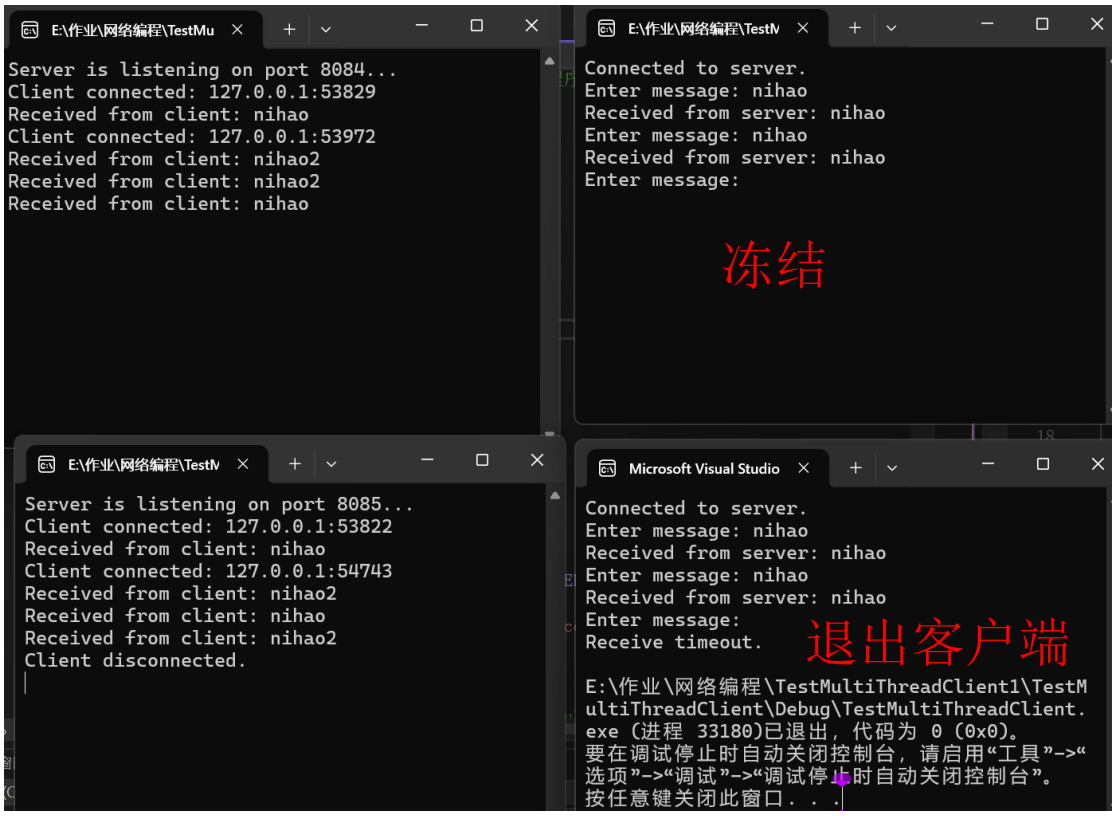
我们可以进一步修改代码：完成通信。

在服务端和客户端的 recv 和 send 调用中，增加对 WSAEWOULDBLOCK 的检查，使用 Sleep 或 select 等机制来等待数据准备好，避免立即返回错误。

结果如下图：



我们还可以与阻塞模式对比，发现如果输入为空时，阻塞模式会“窗口冻结“，这可能是因为一直在等待的原因（上图），而非阻塞模式会返回错误信息并退出客户端。



四、讨论与分析（实验思考中问题理解）

问题 1：服务端运行时报错“Bind failed”

解决方案：查看程序监听端口 8080，查看本机 8080 端口占用情况

被占用，更改端口或者解除 8080 端口占用即可，我这里更改为了 8082 端口进行实验。

实验思考 1：能否在接收数据之间不进行 bind（）调用？如果能，请说明可能的情况。

能。在 TCP 客户端、使用 UDP 协议或原始套接字的场景下，可以不调用 bind()，但服务器端或需 TCP 协议连接的场景仍需调用 bind()。

TCP 客户端连接服务器时，操作系统会自动为套接字分配一个临时端口，无需调用 bind()。

UDP 调用 connect() 后，虽不建立实际连接，但会固定对端地址并触发隐式绑定。之后可直接用 recv() 接收数据。

原始套接字通常直接监听网络层数据包，无需绑定传输层端口。

实验思考 2：阻塞模式和非阻塞模式网络程序区别。

特性	阻塞模式	非阻塞模式
线程模型	必须为每个连接创建独立线程	单线程可通过 I/O 多路复用管理多个连接
函数行为	函数调用（如 recv()）未完成时线程挂起	函数立即返回，需处理 WSAEWOULDBLOCK 错误
资源消耗	高（线程数=连接数）	低（单线程处理所有连接）
响应延迟	实时响应	需轮询检测数据就绪状态
编程复杂度	简单直观	复杂，需处理异步事件和缓冲区管理
典型 API	accept(), recv(), send()	select(), WSAEventSelect(), WSAPoll()

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）

在本次实验中，我能清晰梳理服务端和客户端的功能需求，按照 TCP 服务端和客户端程序的编写流程，规划了各阶段使用的 Winsock API 函数，对多线程处理客户端连接的设计也较为合理。

实验过程中，遇到“Bind failed”错误时，能迅速排查端口占用问题并成功解决。

对于实验知识点，我深入理解了 Winsock API 的调用方法，明白了 TCP 协议在服务端和客户端的工作流程。对多线程编程用于并发处理客户端连接有了实践体会。同时，也清晰掌握了阻塞和非阻塞模式网络程序的区别。但我也意识到在非阻塞模式编程方面，还需要进一步学习和实践，以提升自己对复杂异步事件和缓冲区管理的编程能力，未来将加强这方面的学习与练习，不断提升自己的网络编程水平。

六、 附录：关键代码（给出适当注释，可读性高）

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

// 设置客户端套接字为非阻塞模式
u_long mode = 1;
ioctlsocket(clientSocket, FIONBIO, &mode);

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

// 创建服务器套接字
serverSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (serverSocket == INVALID_SOCKET) {
    printf("Socket creation failed.\n");
    WSACleanup();
    return 1;
}
```

```
// 设置服务器套接字为非阻塞模式
u_long mode = 1;
ioctlsocket(serverSocket, FIONBIO, &mode);

// 绑定地址和端口
serverAddr.sin_family = AF_INET;
serverAddr.sin_addr.s_addr = INADDR_ANY;
serverAddr.sin_port = htons(PORT);
```