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	朱家顺	学号	55210425	成绩	
提交日期	3 月 1 日	座位号			

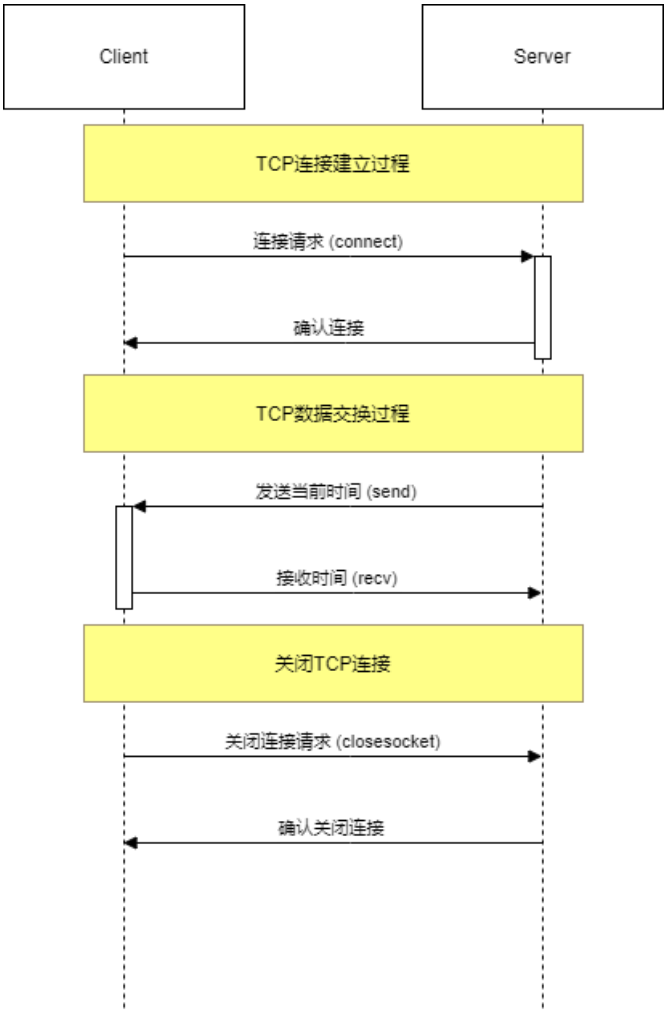
1. 实验目的

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

这个实验的内容是关于在 Windows 操作系统上开发基于 TCP 或 UDP 的客户端和服务器端程序的网络编程。

3. 实验分析



4. 问题解答

4.3 核心代码（有必要的注释）

```
    // 初始化 Winsock
    WSAStartup(MAKEWORD(2,0), &WSAData);
    server = socket(AF_INET, SOCK_STREAM, 0);

    // 设置服务器地址和端口
    addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    addr.sin_family = AF_INET;
    addr.sin_port = htons(5555);

    // 连接到服务器
    connect(server, (SOCKADDR *)&addr, sizeof(addr));
    std::cout << "Connected to server!" << std::endl;

    // 接收服务器发送的时间
    recv(server, buffer, sizeof(buffer), 0);
    std::cout << "Server time: " << buffer << std::endl;









    // 初始化 Winsock
    WSAStartup(MAKEWORD(2,0), &WSAData);
    // AF_INET: IPv4, SOCK_STREAM: TCP
    server = socket(AF_INET, SOCK_STREAM, 0);

    // 设置服务器地址和监听端口
    serverAddr.sin_addr.s_addr = INADDR_ANY;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(5555);

    // 绑定套接字
    bind(server, (SOCKADDR *)&serverAddr, sizeof(serverAddr));
    // 监听连接请求
    listen(server, 0);

    std::cout << "Listening for incoming connections..." << std::endl;

    int clientAddrSize = sizeof(clientAddr);
    // 接受客户端连接
    if ((client = accept(server, (SOCKADDR *)&clientAddr, &clientAddrSize)) !=
INVALID_SOCKET) {
        std::cout << "Client connected!" << std::endl;

        // 获取当前时间
        time_t currentTime;
        time(&currentTime);
        strcpy(buffer, ctime(&currentTime));

        // 发送时间给客户端
        send(client, buffer, sizeof(buffer), 0);
        std::cout << "Time sent: " << buffer << std::endl;
        closesocket(client);
    }

    // 关闭套接字
    closesocket(server);
    WSACleanup();
```

4.4 测试方法、测试数据与测试结果

4.5 程序的使用说明

4.6 总结

TCP 客户端程序：

功能：接收服务器发送的时间。
主要过程：
初始化 Winsock。
创建 TCP 套接字。
设置服务器地址和端口（这里为本机地址 127.0.0.1 和端口 5555）。
连接到服务器。
接收从服务器发送的时间。
显示接收到的时间。
关闭套接字并清理 Winsock。
TCP 服务器程序：

功能：发送当前时间给连接的客户端。
主要过程：
初始化 Winsock。
创建 TCP 套接字。
设置服务器地址（任意地址）和监听端口（端口 5555）。
绑定套接字到服务器地址。
监听连接请求。
接受客户端连接。
获取当前时间并发送给客户端。
关闭客户端套接字。
关闭服务器套接字并清理 Winsock。

吉 林 大 学
软件学院
实 验 报 告

实验名称	并发套接字编程				
课程名称	计算机网络课程设计				
姓名	朱家顺	学号	55210425	成绩	
提交日期	3 月 15 日	座位号			

1.实验目的

实验目的：掌握 Linux 或 Windows 平台上多线程、多进程或异步 I/O 的套接字编程。

2.实验内容

编写和编译程序：

客户端程序：能够向服务器发送请求。
服务器端程序：接收来自客户端的请求，并回复当前的系统时间给客户端。
运行程序：

这两个程序可以在同一台机器上运行，也可以在不同的机器上运行。首先运行服务器端程序，然后运行客户端程序。

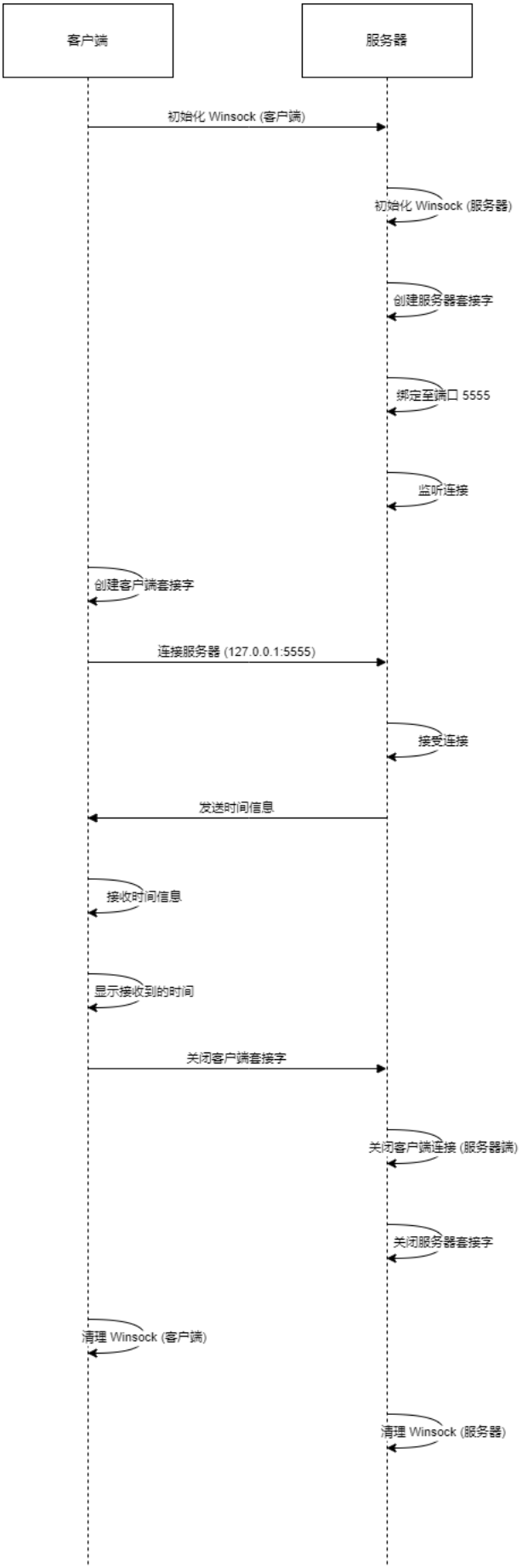
3.实验分析

解题思路
1. 编写客户机端和服务器端程序
语言选择：根据您的熟悉程度选择编程语言，例如 C++、Java 或 Python 等，它们都有支持网络编程的库和框架。
网络编程基础：理解 TCP/IP 协议，因为我们需要建立一个基于这些协议的客户端-服务器架构。
客户端程序（Client）：

功能：向服务器发送请求。
实现：创建一个 socket，连接到服务器的 IP 地址和端口上。发送一个请求消息，然后等待并接收服务器的响应。
服务器端程序（Server）：

功能：监听客户端的连接请求，并发送当前系统时间。
实现：在指定端口上创建并监听 socket。接受客户端的连接请求，获取系统当前时间，发送给客户端。
2. 运行程序
在开发完成后，需要在 Windows 环境下测试这两个程序。
先启动服务器端程序，使其监听特定端口。
然后运行客户端程序，确保它能够连接到服务器端并接收时间信息。

4.问题解答


客户端
服务器
初始化 Winsock (客户端)
初始化 Winsock (服务器)
创建服务器套接字
绑定至端口 5555
监听连接
创建客户端套接字
连接服务器 (127.0.0.1:5555)
接受连接
发送时间信息
接收时间信息
显示接收到的时间
关闭客户端套接字
关闭客户端连接 (服务器端)
关闭服务器套接字
清理 Winsock (客户端)
清理 Winsock (服务器)

4.3 核心代码（有必要的注释）

Client：

```
// 初始化 Winsock
WSAStartup(MAKEWORD(2,0), &WSAData);
server = socket(AF_INET, SOCK_STREAM, 0);

// 设置服务器地址和端口
addr.sin_addr.s_addr = inet_addr("127.0.0.1");
addr.sin_family = AF_INET;
addr.sin_port = htons(5555);

// 连接到服务器
connect(server, (SOCKADDR *)&addr, sizeof(addr));
std::cout << "Connected to server!" << std::endl;

// 接收服务器发送的时间
recv(server, buffer, sizeof(buffer), 0);
std::cout << "Server time: " << buffer << std::endl;

// 关闭套接字
closesocket(server);
WSACleanup();
```

Server：

```
// 初始化 Winsock
WSAStartup(MAKEWORD(2,0), &WSAData);
serverSocket = socket(AF_INET, SOCK_STREAM, 0);

// 设置服务器地址和端口
serverAddr.sin_addr.s_addr = INADDR_ANY;
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(5555);

// 绑定套接字
bind(serverSocket, (SOCKADDR *)&serverAddr, sizeof(serverAddr));
// 监听连接请求
listen(serverSocket, 0);

std::cout << "Listening for incoming connections..." << std::endl;

int clientAddrSize = sizeof(clientAddr);
// 循环接受客户端连接
while ((clientSocket = accept(serverSocket, (SOCKADDR *)&clientAddr,
&clientAddrSize))) {
    std::thread clientThread(handleClient, clientSocket);
    clientThread.detach(); // 分离线程，独立处理客户端;调用 detach 方法让线程
独立运行，即使主线程结束也不会影响这些线程。
}

// 关闭服务器套接字
closesocket(serverSocket);
WSACleanup();
```

4.4 测试方法、测试数据与测试结果

4.5 程序的使用说明

4.6 总结

基本功能符合预期，能够实现简单的客户端与服务器通信。

多线程处理（仅服务器）：服务器程序采用了多线程来处理客户端连接，这在并发处理多个连接时是有效的。但是，线程管理和同步机制需要谨慎处理以避免竞态条件和死锁。

吉 林 大 学
软件学院
实 验 报 告

实验名称	原始套接字编程				
课程名称	计算机网络课程设计				
姓名	朱家顺	学号	55210425	成绩	
提交日期	3 月 29 日	座位号			

1. 实验目的

实验目的：掌握原始套接字编程。

2. 实验内容

创建一个程序，使用 RAW SOCKET 捕获网络数据包：RAW SOCKET 是一种可以让程序接收和发送网络层以下数据包的套接字。这种套接字类型允许直接访问较低层的网络协议，是网络监听技术的关键组成部分。

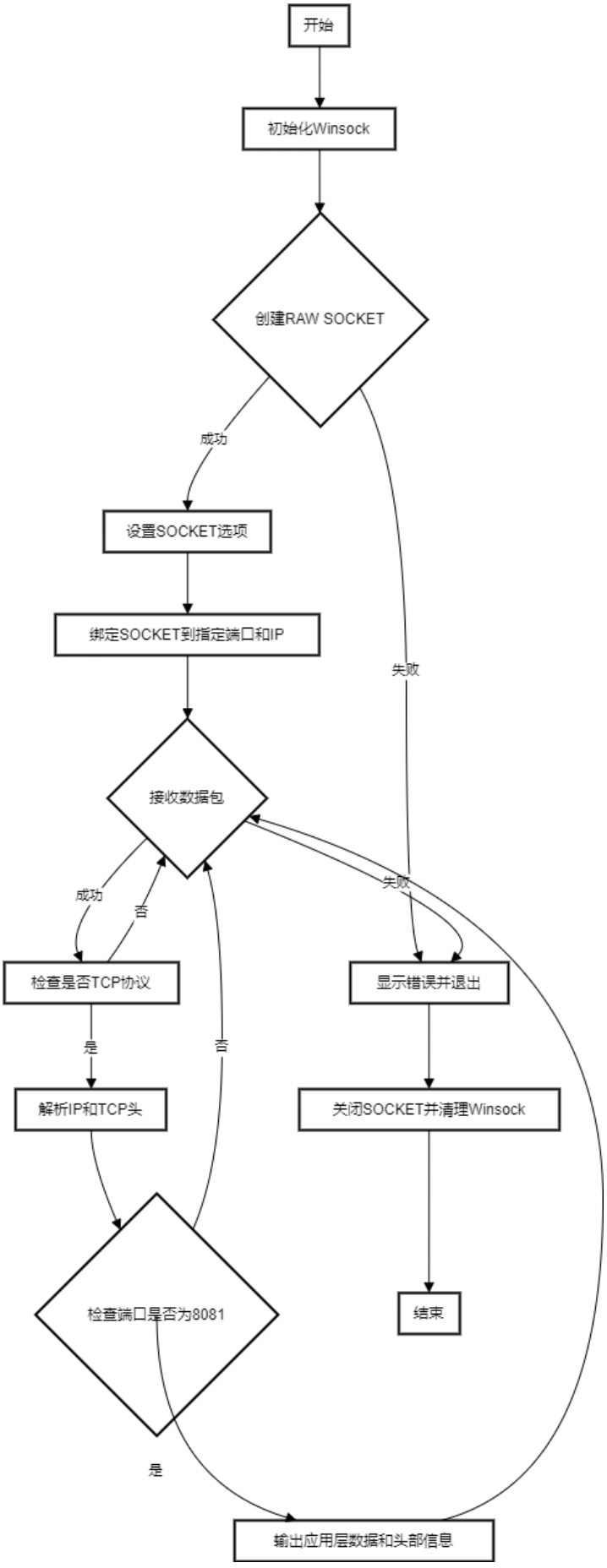
RAW SOCKET 相比于流式套接字（SOCKET_STREAM，即 TCP）和数据报套接字（SOCKET_DGRAM，即 UDP），提供了更底层的网络控制。它通常用于处理 IP 层、ICMP、IGMP 协议等。
在使用 RAW SOCKET 时，你可以直接构造或解析 IP 层及其以上的数据。例如，可以用来实现类似于 ping 的功能（使用 ICMP 协议）。
实现应用层数据的抓取：需要捕获特定应用（如第二节课中的并发服务器程序）的服务器端或客户端的应用层数据，并打印输出。

数据的输出格式要求包括应用层数据（例如时间值）、传输层信息（源端口和目标端口）以及网络层信息（源 IP 和目标 IP）。

3. 实验分析

提示：对该问题进行分析，给出解题思路与解题方法

4. 问题解答


开始
初始化Winsock
创建RAW SOCKET
成功
失败
设置SOCKET选项
绑定SOCKET到指定端口和IP
接收数据包
成功
否
失败
检查是否TCP协议
显示错误并退出
是
否
解析IP和TCP头
关闭SOCKET并清理Winsock
检查端口是否为8081
结束
是
输出应用层数据和头部信息

4.3 核心代码（有必要的注释）

```
WSADATA wsadata;
// 初始化 Winsock
if (WSAStartup(MAKEWORD(2, 2), &wsadata) == SOCKET_ERROR) {
    cerr << "winsock InitializationFailed:" << WSAGetLastError() << endl;
    return -1;
}

SOCKADDR_IN saddr;
// 设置套接字地址结构
memset(&saddr, 0, sizeof(SOCKADDR_IN));
saddr.sin_family = AF_INET;
saddr.sin_port = htons(8081);
saddr.sin_addr.s_addr = inet_addr("127.0.0.1");

SOCKET rawSocket;
// 创建原始套接字
if ((rawSocket = socket(AF_INET, SOCK_RAW, IPPROTO_IP)) == INVALID_SOCKET)
{
    cerr << "FailedToCreateRawSocket:" << WSAGetLastError() << endl;
    return -1;
}

int on = 1;
// 设置套接字选项
setsockopt(rawSocket, SOL_SOCKET, SO_REUSEADDR, (char *) &on, sizeof(on));

// 绑定套接字
if (bind(rawSocket, (SOCKADDR *) &saddr, sizeof(SOCKADDR)) == SOCKET_ERROR)
{
    cerr << "BindingRawSocketFailed:" << WSAGetLastError() << endl;
    return -1;
}

DWORD dwlen[10], dwlenrtned = 0, optval = 1;
// 允许套接字接收所有经过本机的网络数据包
WSAIoctl(rawSocket, SIO_RCVALL, &optval, sizeof(optval), &dwlen,
sizeof(dwlen), &dwlenrtned, NULL, NULL);

char buffer[65535]{};
IPHeader *ipHeader = (IPHeader *) buffer;
TCPHeader *tcpHeader = (TCPHeader *) (buffer + 20);
while (true) {
    int size = recv(rawSocket, buffer, sizeof(buffer), 0);

    if (size == SOCKET_ERROR) {
        cerr << "FailedToReceiveData" << WSAGetLastError() << endl;
        return -1;
    }

    if (ipHeader->protocol == IPPROTO_TCP) {
        if (ntohs(tcpHeader->dport) == 8081 ||
            ntohs(tcpHeader->sport) == 8081) {
            cout <<
"Applications++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++" << endl;
            cout << "Data: " << buffer + 40 << endl;
```

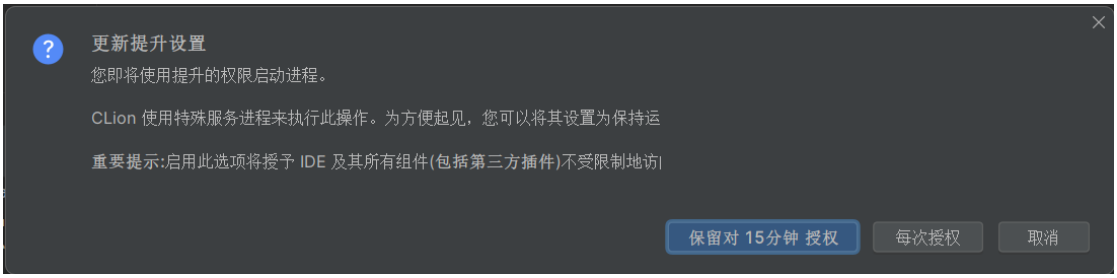
```
            cout <<
"Transportation++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++" << endl;
            cout << "Src Port: " << ntohs(tcpHeader->sport) << endl;
            cout << "Dest Port: " << ntohs(tcpHeader->dport) << endl;

            cout <<
"Network++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++" << endl;
            cout<<"Src IP: "<<inet_ntoa(*(in_addr*)&ipHeader->sourceip)
<< endl;
            cout<<"Dest IP: "<<inet_ntoa(*(in_addr*)&ipHeader->destip)
<< endl;

            cout << endl << endl << endl;
        }
    }
}
```

4.4 测试方法、测试数据与测试结果

4.5 程序的使用说明

4.6 总结

需要申请权限（即使在管理员权限下）：



吉 林 大 学
软件学院
实 验 报 告

实验名称	FTP 综合应用编程				
课程名称	计算机网络课程设计				
姓名	朱家顺	学号	55210425	成绩	
提交日期	4 月 11 日	座位号			

1. 实验目的

实验目的：掌握 FTP 应用编程。

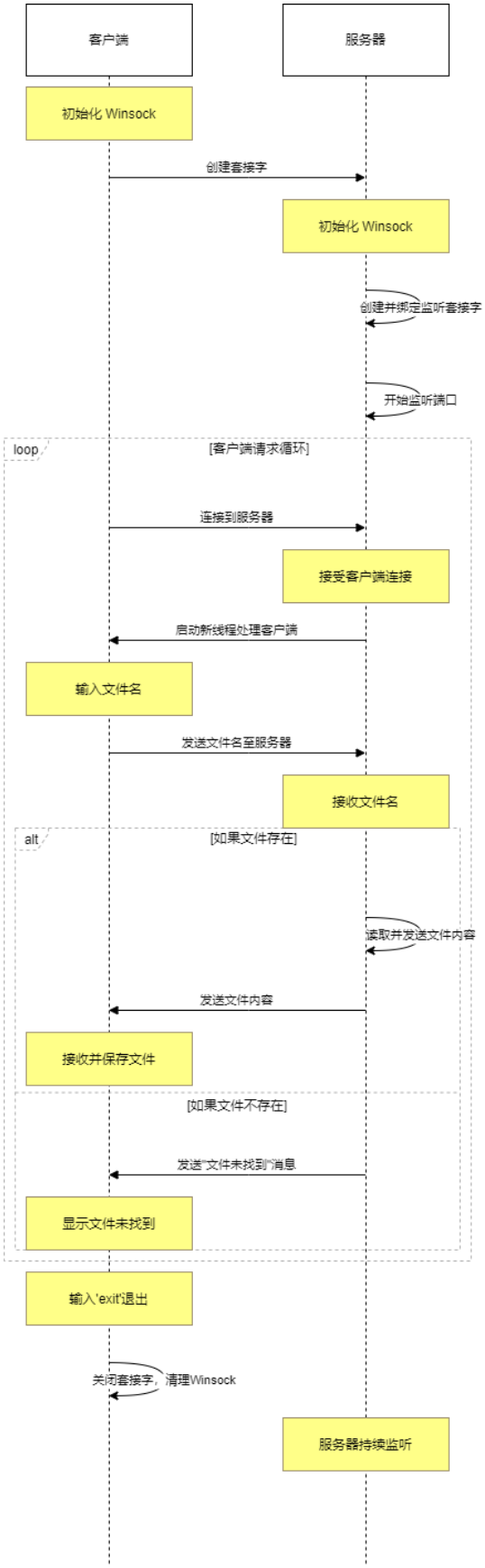
2. 实验内容

改造现有软件：基于之前实验的基础，改造成一个能够传输特定文件的点对点文件传输软件。
开发多客户端支持的文件传输服务器：设计并实现一个服务器，能同时处理多个客户端的文件传输请求。
客户端-服务器交互：客户端输入文件名以请求下载，服务器在特定目录查找并传输文件，或反馈文件不存在的信息。

3. 实验分析

思路
了解 FTP 原理：研究 FTP 协议基础。
环境准备：选择编程语言和工具。
软件改造：在现有基础上添加指定文件传输功能。
服务器开发：实现多客户端支持和文件传输逻辑。
客户端开发：编写代码以输入文件名，请求和接收文件。
测试：测试程序的功能和稳定性。
方法
研究 FTP：阅读相关文档。
搭建环境：安装编程语言和必要库。
修改现有代码：增加文件指定传输功能。
编写服务器代码：实现多连接处理和文件查找、传输。
编写客户端代码：实现文件名输入，发送请求，接收和保存文件。
程序测试：在不同条件下进行测试，确保程序正确运行。

4. 问题解答



4.3 核心代码（有必要的注释）

Client：

```
WSADATA wsaData;
SOCKET serverSocket;
sockaddr_in serverAddr{};
int iResult;

// 初始化 Winsock
iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
if (iResult != 0) {
    cerr << "WSAStartup 失败: " << iResult << endl;
    return 1;
}

// 创建套接字
serverSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (serverSocket == INVALID_SOCKET) {
    cerr << "创建套接字失败: " << WSAGetLastError() << endl;
    WSACleanup();
    return 1;
}

// 设置服务器地址和端口
serverAddr.sin_family = AF_INET;
serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
serverAddr.sin_port = htons(PORT);

// 连接到服务器
iResult = connect(serverSocket, (SOCKADDR*) &serverAddr,
sizeof(serverAddr));
if (iResult == SOCKET_ERROR) {
    cerr << "连接失败: " << WSAGetLastError() << endl;
    closesocket(serverSocket);
    WSACleanup();
    return 1;
}

// 循环接收文件
while (true) {
    cout << "输入文件名（或输入'exit'退出）: ";
    char fileName[BUFFER_SIZE];
    cin.getline(fileName, BUFFER_SIZE);

    if (strcmp(fileName, "exit") == 0) break;

    // 发送文件名
    send(serverSocket, fileName, strlen(fileName), 0);

    char message[BUFFER_SIZE]{};
    // 接收文件内容
    int size = recv(serverSocket, message, BUFFER_SIZE, 0);

    // 处理文件不存在的情况
    if (strcmp(message, "File not found") == 0) {
        cout << "服务器上未找到文件。" << endl;
    } else {
        // 将接收到的文件内容写入新文件
        ofstream outFile("received_" + string(fileName), ios::binary);
        outFile.write(message, size);
        cout << "文件接收成功: " << message << endl;
```

```
        outFile.close();
    }
}

// 关闭套接字和清理 Winsock
closesocket(serverSocket);
WSACleanup();
```

Server：

```
WSADATA wsaData;
// 初始化 Winsock
if (WSAStartup(MAKEWORD(2, 2), &wsaData) == SOCKET_ERROR) {
    cerr << "WSAStartup 失败。" << endl;
    return -1;
}

// 创建监听套接字
SOCKET listenSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (listenSocket == INVALID_SOCKET) {
    cerr << "创建套接字失败：" << WSAGetLastError() << endl;
    WSACleanup();
    return -1;
}

sockaddr_in serverAddr{};
serverAddr.sin_family = AF_INET;
serverAddr.sin_addr.s_addr = htonl(INADDR_ANY);
serverAddr.sin_port = htons(PORT);
// 绑定套接字
if (bind(listenSocket, (sockaddr*) &serverAddr, sizeof(serverAddr)) ==
SOCKET_ERROR) {
    cerr << "绑定失败：" << WSAGetLastError() << endl;
    closesocket(listenSocket);
    WSACleanup();
    return -1;
}

// 开始监听
if (listen(listenSocket, SOMAXCONN) == SOCKET_ERROR) {
    cerr << "监听失败：" << WSAGetLastError() << endl;
    closesocket(listenSocket);
    WSACleanup();
    return -1;
}

cout << "服务器正在监听端口 " << PORT << endl;

while (true) {
    // 接受客户端连接
    SOCKET clientSocket = accept(listenSocket, nullptr, nullptr);
    if (clientSocket == INVALID_SOCKET) {
        cerr << "接收失败：" << WSAGetLastError() << endl;
        continue;
    }

    // 创建新线程处理客户端请求
    thread clientThread(handleClient, clientSocket);
    clientThread.detach();
```

```
}

closesocket(listenSocket);
WSACleanup();
```

4.4 测试方法、测试数据与测试结果

4.5 程序的使用说明

4.6 总结

提示：对程序进行分析、评价运行效果，总结遇到的问题及解决办法；特色说明