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		学号		成绩	
提交日期	2024. 04. 12	座位号			

1. 实验目的

掌握 FTP 应用编程。

2. 实验内容

本实验旨在设计和实现一个基于 TCP 协议的文件传输系统，包括客户端和服务器端。客户端可以向服务器请求传输指定文件，服务器根据请求进行文件查找和传输。具体功能包括：

- 实现一个多线程的文件传输服务器，支持多个客户端同时连接和文件传输请求处理。
- 客户端能够向服务器请求指定文件，并接收服务器传输的文件内容保存到本地。

3. 实验分析

为了完成本实验，首先需要对 TCP 协议进行了解和使用，以便实现可靠的数据传输。服务器端需要能够并发处理多个客户端连接，并根据客户端请求查找并发送指定文件。客户端需要能够与服务器建立连接并发送文件请求，接收服务器传输的文件内容并保存到本地。

- 服务器端
  - 创建一个 Service 类，初始化服务器套接字并绑定到指定端口。
  - 启动服务器，监听客户端连接请求。
  - 接受客户端连接并创建新的线程处理每个连接。
  - 在处理线程中，根据客户端请求类型执行相应的文件传输操作。
- 客户端
  - 创建一个 Client 类，初始化客户端套接字并连接到指定服务器。
  - 用户输入文件名并发送给服务器。
  - 根据服务器的响应，接收文件内容并保存到本地。

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。
    - 创建一个新的线程或进程来处理客户端连接，并继续监听其他客户端连接。
    - 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)`）**
    - 在独立的线程中，接收客户端发送的请求并根据请求类型执行相应的操作。

- 如果客户端请求当前时间，调用 SendTime()函数向客户端发送本机当前时间。
- 如果客户端请求文件传输，调用 SendFile()函数向客户端发送指定文件内容。

5) 发送当前时间给客户端（`void SendTime(SOCKET clientSocket)`）

- 获取系统当前时间。
- 将当前时间格式化为字符串并发送给指定的客户端套接字。

6) 发送文件给客户端（`void SendFile(SOCKET clientSocket, const std::string& filename)`）

- 打开指定文件并读取文件内容。
- 将文件内容分段发送给客户端，直到文件全部发送完毕或发生错误。

7) 关闭服务器（`~Service()`）

- 关闭服务器套接字。等待所有子线程或进程处理完毕，释放资源。

● 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()`）

- 循环读取用户输入的命令并发送到服务器。
- 根据用户输入的命令，接收服务器的响应并进行相应的处理。
- 如果用户输入退出命令，则退出循环并关闭客户端套接字。

4) 接收服务器发送的时间信息（`void ReceiveTime()`）

- 接收服务器端发送的时间信息数据。
- 解析接收到的时间数据并在客户端控制台上显示。

5) 接收服务器发送的文件（`void ReceiveFile()`）

- 接收服务器端发送的文件数据。
- 如果服务器返回文件不存在的消息，则显示指定文件不存在。
- 提示用户输入文件保存位置，并将接收到的文件数据保存到指定位置的文件中。

6) 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;
        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,INET_ADDRSTRLEN);
        std::cout << "客户端（" << ipBuffer << "）通过端口（" <<
        ntohs(clientAddr.sin_port) << "）连接成功，套接字号：" << clientSocket <<
        std::endl;
```

```
        clientSockets.push_back(clientSocket);

        // 创建新线程处理客户端消息
        std::thread clientThread(&Service::HandleClient, this, clientSocket);
        clientThread.detach();
    }
}
void Service::HandleClient(SOCKET clientSocket) {
    int const CLIENT_MSG_SIZE = 128;
    char buffer[CLIENT_MSG_SIZE];

    while (true) {
        // 接收客户端消息
        int recvResult = recv(clientSocket, buffer, CLIENT_MSG_SIZE, 0);
        if (recvResult == SOCKET_ERROR || recvResult == 0) {
            std::cerr << "客户端（套接字：" << clientSocket << "）退出连接，对话
            中断" << std::endl;
            closesocket(clientSocket);
            // 从存储的客户端套接字列表中移除已断开连接的套接字
            auto it = std::find(clientSockets.begin(), clientSockets.end(),
                                clientSocket);
            if (it != clientSockets.end()) {
                clientSockets.erase(it);
            }
            break;
        }
        std::cout << "客户端（套接字：" << clientSocket << "）命令：" << buffer
        << std::endl;

        // 处理客户端命令
        if (strcmp(buffer, "当前时间") == 0) {
            SendTime(clientSocket);
        }
        else if (strncmp(buffer, "文件传输：", 10) == 0) {
            SendFile(clientSocket, buffer + 10);
        }
        else if (strcmp(buffer, "退出") == 0) {
            std::cout << "客户端（套接字：" << clientSocket << "）成功退出连接" <<
            std::endl;
            closesocket(clientSocket);
            // 从存储的客户端套接字列表中移除已断开连接的套接字
            auto it = std::find(clientSockets.begin(), clientSockets.end(),
                                clientSocket);
            if (it != clientSockets.end()) {
                clientSockets.erase(it);
            }
            break;
        }
        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（套接字：" << clientSocket << "）
                出错，错误代码：" << WSAGetLastError() << std::endl;
            }
        }
    }

    if (clientSockets.empty()) {
        std::cout << "等待客户端连接..." << std::endl;
    }
```

```
}
void Service::SendTime(SOCKET clientSocket) {
    int const BUFFER_SIZE = 128;
    char buffer[BUFFER_SIZE];

    // 获取系统当前时间
    SYSTEMTIME systime = { 0 };
    GetLocalTime(&systime);
    sprintf_s(buffer, BUFFER_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, buffer, strlen(buffer), 0);
    if (sendResult == SOCKET_ERROR) {
        std::cerr << "发送数据给客户端（套接字：" << clientSocket << "）出错，错
        误代码：" << WSAGetLastError() << std::endl;
    }
}
void Service::SendFile(SOCKET clientSocket, const std::string& filename) {
    const int BUFFER_SIZE = 1024;
    char buffer[BUFFER_SIZE];

    std::ifstream inFile(filename, std::ios::binary);

    // 如果文件无法打开，向客户端发送文件不存在消息
    if (!inFile.is_open()) {
        const char* fileNotFoundMsg = "文件不存在";
        int msgLength = static_cast<int>(strlen(fileNotFoundMsg)) + 1;
        int sendResult = send(clientSocket, fileNotFoundMsg, msgLength, 0);

        if (sendResult == SOCKET_ERROR) {
            std::cerr << "向客户端（套接字：" << clientSocket << "）发送消息失败，
            错误代码：" << WSAGetLastError() << std::endl;
        }
        else {
            std::cerr << "客户端（套接字：" << clientSocket << "）指定文件不存在：
            " << filename << std::endl;
        }
        return;
    }

    // 循环读取文件数据并发送给客户端
    memset(buffer, 0, sizeof(buffer));
    while (!inFile.eof()) {
        inFile.read(buffer, BUFFER_SIZE);
        int bytesRead = static_cast<int>(inFile.gcount());
        if (bytesRead > 0) {
            int sendResult = send(clientSocket, buffer, bytesRead, 0);
            if (sendResult == SOCKET_ERROR) {
                std::cerr << "向客户端（套接字：" << clientSocket << "）发送文件
                数据失败，错误代码：" << WSAGetLastError() << std::endl;
                break;
            }
        }
    }

    inFile.close();

    // 检查文件是否正常发送完成
    if (inFile.eof()) {
```

```
            std::cout << "向客户端（套接字：" << clientSocket << "）传输文件完成：" <<
            filename << std::endl;
        }
        else {
            std::cerr << "向客户端（套接字：" << clientSocket << "）传输文件过程中发
            生错误：" << filename << std::endl;
            remove(filename.c_str());
        }
}
```

- 客户端

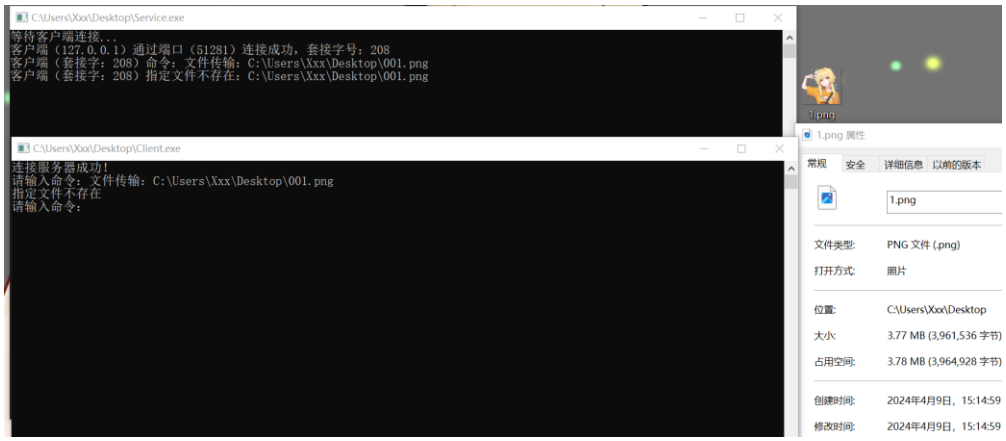
```
void Client::Run() {
    const int BUFFER_SIZE = 128;
    char buffer[BUFFER_SIZE];

    while (true) {
        std::cout << "请输入命令：";
        std::cin.getline(buffer, BUFFER_SIZE);

        // 发送命令到服务器
        int sendResult = send(clientSocket, buffer, strlen(buffer) + 1, 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;
            break;
        }

        // 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
        if (strcmp(buffer, "当前时间") == 0) {
            ReceiveTime();
        }
        else if (strncmp(buffer, "文件传输：", 10) == 0) {
            ReceiveFile();
        }
        else {
            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;
        }
    }
}
void Client::ReceiveTime() {
    const int BUFFER_SIZE = 128;
    char buffer[BUFFER_SIZE];
    memset(buffer, 0, sizeof(buffer));

    // 接收服务器端发送的时间信息
    int recvResult = recv(clientSocket, buffer, BUFFER_SIZE, 0);
    if (recvResult == SOCKET_ERROR) {
        std::cerr << "接收服务器时间信息出错，错误代码：" << WSAGetLastError() <<
         std::endl;
```

```
        return;
    }

    if (recvResult == 0) {
        std::cerr << "服务器断开连接，无法获取时间信息" << std::endl;
        return;
    }

    // 显示服务器返回的时间
    std::cout << "服务器当前时间：" << buffer << std::endl;
}
void Client::ReceiveFile() {
    const int BUFFER_SIZE = 1024;
    char buffer[BUFFER_SIZE];

    // 检查文件是否存在
    int recvResult = recv(clientSocket, buffer, BUFFER_SIZE, 0);
    if (recvResult == SOCKET_ERROR || recvResult == 0) {
        std::cerr << "接收文件数据出错或连接关闭，错误代码：" << WSAGetLastError()
         << std::endl;
        return;
    }
    if (strcmp(buffer, "文件不存在") == 0) {
        std::cerr << "指定文件不存在" << std::endl;
        return;
    }

    std::string filename;
    std::cout << "请输入文件保存位置：";
    std::getline(std::cin, filename);

    // 打开本地文件用于接收数据
    std::ofstream outFile(filename, std::ios::binary);
    if (!outFile) {
        std::cerr << "无法打开文件以保存接收的数据：" << filename << std::endl;
        return;
    }

    // 循环接收数据直到接收完整文件
    while (true) {
        // 写入当前接收到的数据块
        outFile.write(buffer, recvResult);

        // 判断是否接收完整文件，根据实际情况进行处理
        if (recvResult < BUFFER_SIZE) {
            break;
        }

        // 接收数据块
        recvResult = recv(clientSocket, buffer, BUFFER_SIZE, 0);
        if (recvResult == SOCKET_ERROR || recvResult == 0) {
            std::cerr<<"接收文件数据出错或连接关闭，错误代码："<<WSAGetLastError()
            << std::endl;
            outFile.close();
            remove(filename.c_str());
            return;
        }
    }

    // 检查文件接收过程中是否出现错误
    if (!outFile.good()) {
```

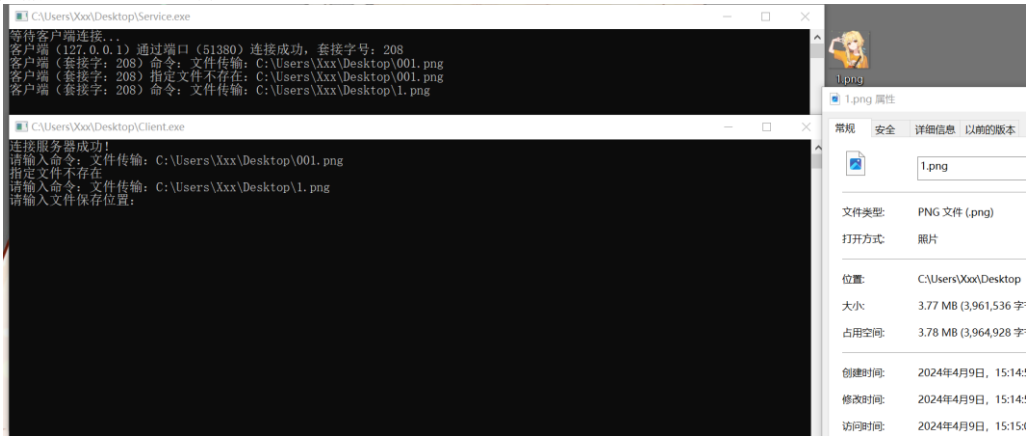
```
        std::cerr << "写入文件数据出错" << std::endl;
        outFile.close();
        remove(filename.c_str());
        return;
    }

    // 关闭本地文件
    outFile.close();
    std::cout << "文件接收完成: " << filename << std::endl;
}
```

4.4 测试方法、测试数据与测试结果

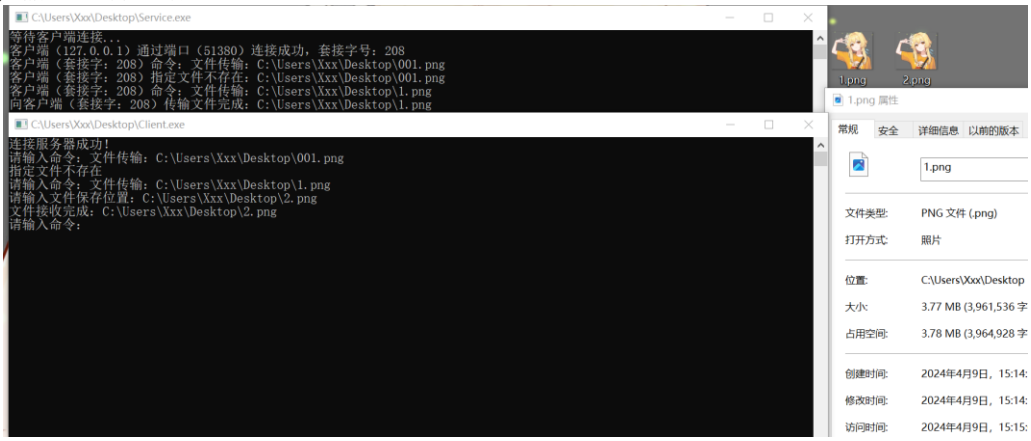
1) 输入不存在的文件位置



2) 输入正确的文件位置



3) 输入文件要存放的位置



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”“文件传输”和“退出”。若命令为“当前时间”，则服务器端会获取当前系统时间并将其返回给客户端；若命令为“文件传输”，则服务器端会获取文件内容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”“文件传输”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。
  - **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：**程序实现了基于 TCP 的文件传输功能，客户端可以向服务器端请求文件，并将服务器端指定的文件传输到客户端。文件传输过程中，程序能够稳定地将文件数据从服务器端发送到客户端，并在客户端保存为本地文件。
  - **稳定性：**程序整体功能较为完整，实现了基本的服务器-客户端通信和文件传输功能。通过 TCP 协议保证了数据传输的稳定性和可靠性。

- 遇到的问题及解决办法
  - **文件传输异常：**如果文件传输过程中出现异常，例如文件不存在或网络中断，需要程序能够正确处理异常情况，并向用户提供相关信息。
  - **性能优化：**对于大文件传输，程序可能需要优化文件读取和发送的方式，以提高传输效率和减少资源消耗。

- 程序特色说明
  - **并发处理能力：**通过多线程机制实现了服务器端的并发处理能力，能够同时处理多个客户端的连接和请求，提高了系统的性能和效率。
  - **模块化设计：**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，使得代码结构清晰，易于理解和维护。
  - **基于 TCP 协议：**使用 TCP 协议进行通信，保证了数据传输的可靠性和有序性，适用于需要稳定通信的场景。
  - **灵活的文件传输：**用户可以指定任意位置的文件进行传输，并自定义下载文件的保存位置。