中国石油大学（北京）

FTP 实验

学号：2022012101

专业：计算机科学与技术

姓名：严嘉哲

目　录

第 1 章　实验概述

1.1　实验目的

通过本次实验，掌握 FTP 文件传输协议的原理，学习在 Linux 系统中使用 Socket 接口实现 FTP 客户端与服务器程序，实现文件的上传、下载、目录操作等基本功能，并进一步实现 FTP 网页浏览器以及基于 socket 编程的网络爬虫。

1.2　实验内容

在 Linux 环境下基于 C 语言和 Socket 接口开发 FTP 客户端与服务器程序，实现客户端连接服务器并完成文件传输协议的基本操作。该 FTP 系统支持文件上传、下载、目录浏览和切换等功能，并实现了控制通道与数据通道的分离机制，确保命令控制与数据传输过程互不干扰。FTP 服务器与客户端之间的通信符合 RFC959 规则。

此外，还实现了一个基于 HTML、JavaScript 和 Python 的 Web 界面 FTP 客户端。用户可以通过浏览器以图形化方式进行 FTP 操作。该 Web 客户端支持用户登录 FTP 服务器，并提供文件上传、下载、删除、重命名以及目录浏览等功能，实现了对 FTP 服务器的可视化文件管理。

最后，实现了一个基于 C 语言的网络爬虫程序。该程序通过 Socket 通信连接指定网站，抓取网页文本内容和图像资源。获取的网页数据被保存至本地，以加深对 Socket 编程在 HTTP 协议应用中的理解。

第 2 章　FTP 实验实现

2.1　基本功能介绍

本实验实现了基于 Socket 编程的 FTP 客户端与服务器程序。其严格遵循 RFC959 协议规范，通过控制通道接收命令、通过数据通道进行数据传输。

2.1.1　FTP 客户端基本功能

建立连接（open）：客户端通过 TCP Socket 连接至服务器，实现命令通道和数据通道的双通道通信结构。命令通道用于发送控制命令（如登录验证、目录操作指令），数据通道用于传输实际文件内容或目录列表。

用户登录验证（user）：用户需要通过用户名和密码认证后，才能使用 FTP 服务进行进一步操作，确保系统安全性。

文件下载（get）：客户端通过数据通道从服务器上下载指定的文件。

文件上传（put）：客户端能够向服务器上传文件，通过数据通道将本地文件传输至服务器端。

目录查看（dir）：客户端能够列出服务器当前目录下的所有文件和子目录。

当前路径显示（pwd）：客户端可以显示服务器当前所在的工作目录路径。

目录切换（cd）：客户端能够在服务器上切换当前的工作目录。

命令帮助（?）：客户端提供命令帮助功能，显示所有可用的 FTP 命令列表及其简要描述。

安全退出（quit）：客户端能够安全退出 FTP 服务，并关闭所有打开的连接通道。

2.1.2　FTP 服务器基本功能

建立连接：接收客户端 TCP 连接请求，分别建立命令通道和数据通道，实现双通道通信。

用户登录验证（USER，PASS）：接收并验证客户端发送的用户名和密码，允许认证用户访问 FTP 服务。

文件下载（RETR）：根据客户端请求，通过数据通道将指定文件内容传输给客户端，实现文件下载功能。

文件上传（STOR）：接收客户端通过数据通道上传的文件内容，并保存到服务器指定路径，实现文件上传功能。

目录查看（LIST）：响应客户端请求，返回当前目录下所有文件和子目录的列表信息。

当前路径显示（PWD）：向客户端返回服务器当前工作目录的完整路径。

目录切换（CWD）：根据客户端命令切换当前工作目录。

命令帮助（HELP）：响应客户端请求，返回服务器支持的所有 FTP 命令及简要说明。

安全退出（QUIT）：关闭与客户端的连接，释放相关资源。

2.2　套接字函数的应用

表 2.1 使用的套接字函数介绍

函数格式	作用
socket(domain, type, protocol)	创建新的套接口，返回套接口描述符
bind(sockfd, addr, addrlen)	将套接口绑定至特定的地址和端口
listen(sockfd, backlog)	监听套接口上的客户端连接请求
accept(sockfd, addr, addrlen)	接受客户端连接，返回新套接口描述符
connect(sockfd, addr, addrlen)	客户端连接到服务器
send(sockfd, buf, len, flags)	向对端发送数据
recv(sockfd, buf, len, flags)	从对端接收数据
close(sockfd)	关闭套接口，释放资源

2.3　文件操作函数的应用

表 2.2 使用的文件操作函数介绍

函数格式	作用
fopen(filename, mode)	打开文件，返回文件指针
fclose(file)	关闭已打开的文件
fread(ptr, size, nmemb, file)	从文件读取数据到缓冲区
fwrite(ptr, size, nmemb, file)	将缓冲区数据写入文件

opendir(dirpath)	打开目录，返回目录指针
readdir(dirp)	读取目录项，返回目录条目
closedir(dirp)	关闭目录指针，释放资源
chdir(path)	更改当前工作目录
getcwd(buf, size)	获取当前工作目录的路径

2.4　客户端关键函数与功能描述

表 2.3 客户端关键函数与功能描述

关键函数	功能描述
connect_to_server(server_ip, port)	连接到 FTP 服务器
login(username, password)	用户身份认证
handle_command(command)	解析并执行用户命令
send_command(cmd_str)	发送命令到服务器
receive_response()	接收服务器响应
download_file(remote_file, local_file)	下载服务器文件到本地
upload_file(local_file, remote_file)	上传本地文件到服务器
show_directory()	显示服务器目录列表
change_directory(path)	切换服务器当前目录
show_help()	显示 FTP 命令帮助
quit()	断开连接并退出客户端

2.5　服务器关键函数与功能描述

表 2.4 服务器关键函数与功能描述

关键函数	功能描述
init_server(port)	初始化服务器，绑定端口，准备监听
accept_client(server_sock)	接受客户端连接
authenticate(username, password)	用户身份验证
process_command(cmd_str, client)	解析并处理客户端命令

send_response(client, message, code)	发送响应消息到客户端
send_file(client, filename)	发送文件到客户端
recv_file(client, filename)	接收客户端上传的文件
list_directory(client)	向客户端发送目录列表
change_directory(path, client)	切换工作目录
get_working_directory(client)	返回当前工作目录
close_connection(client)	关闭与客户端的连接

2.6　共用模块关键函数与功能描述

表 2.5 公用模块关键函数与功能描述

关键函数	功能描述
send_all(sockfd, buf, len)	发送指定长度的数据，保证全部发送完毕
recv_all(sockfd, buf, len)	接收指定长度的数据，保证完整接收
send_file(data_sock, filename)	通过数据通道发送文件内容
recv_file(data_sock, filename)	通过数据通道接收文件并保存
parse_command(input, cmd_struct)	解析命令字符串，提取命令及参数
trim_newline(str)	去除字符串末尾的换行符和回车符
create_data_connection(listen_port)	创建数据连接，返回新的数据套接字
connect_data_channel(ip, port)	连接到数据通道
get_random_port()	获取系统分配的随机端口

第 3 章 实验结果展示

3.1 客户端命令

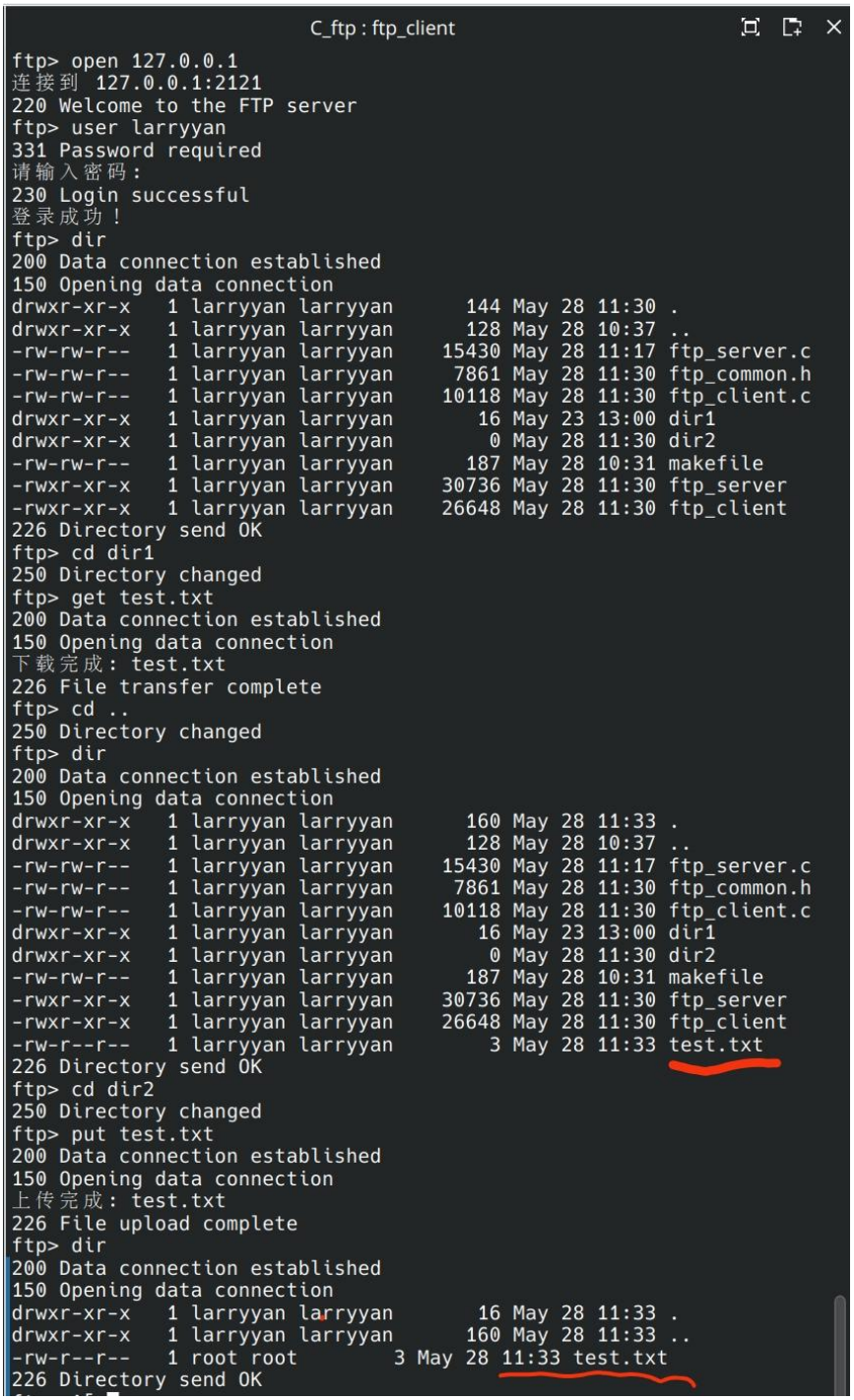

图 3.1 以本机账号为例的 FTP 客户端演示

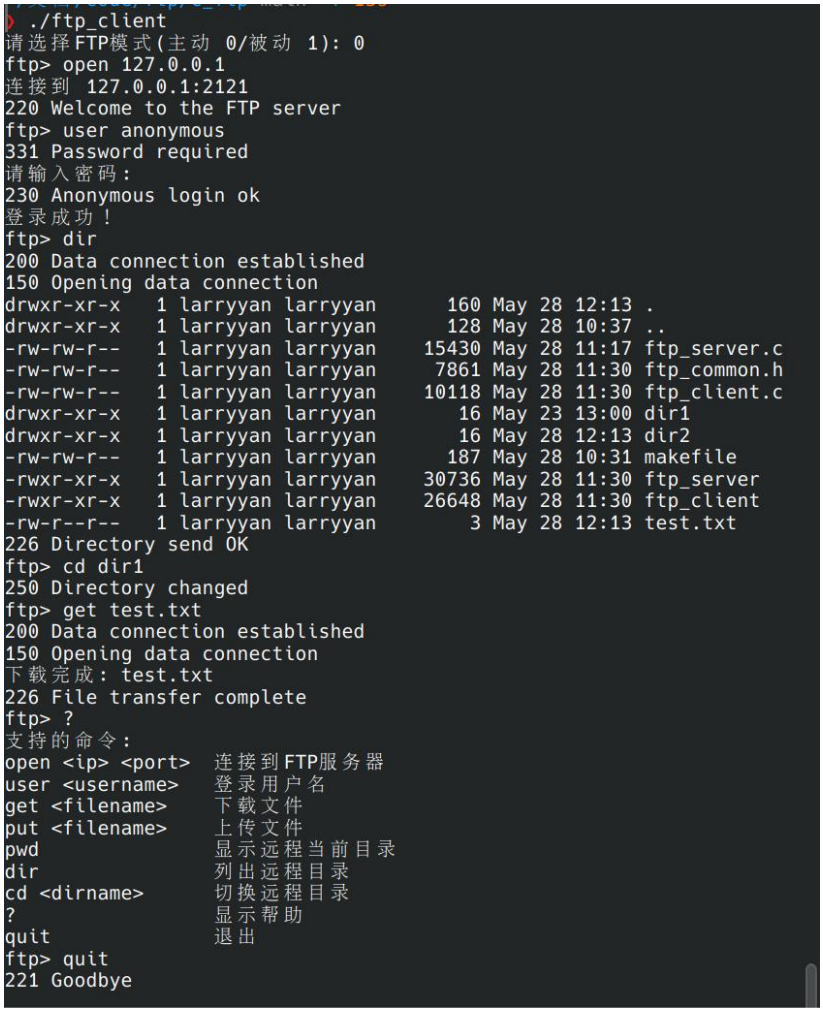

图 3.2 以匿名账号为例的 FTP 客户端演示

3.2 匿名用户命令

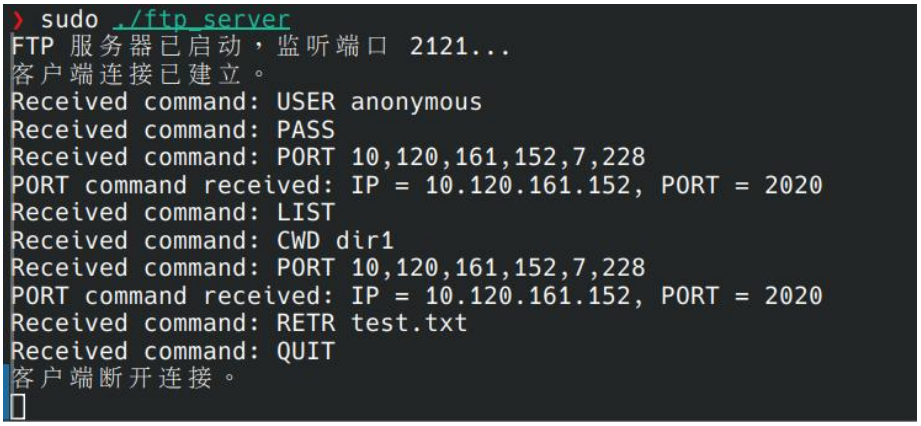

图 3.3 匿名账号的控制命令

3.3 普通用户命令

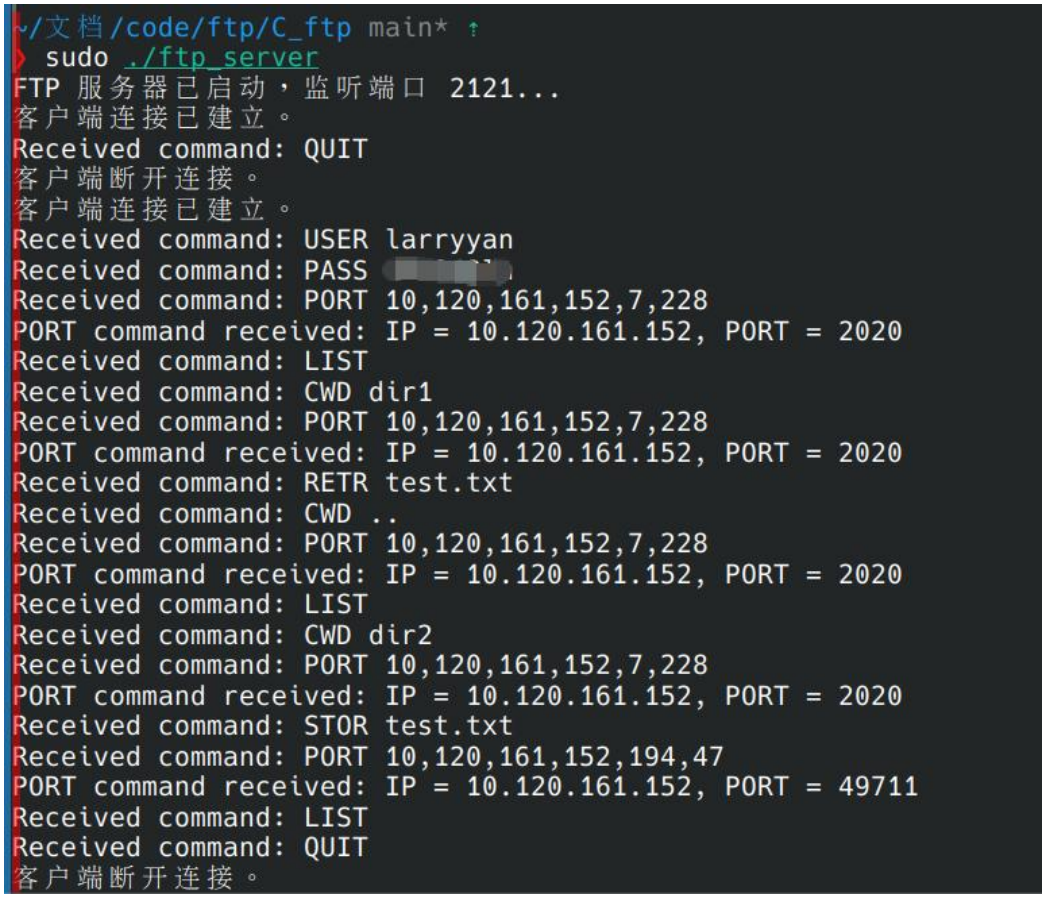

图 3.4 普通用户的控制命令

第 4 章　Web 服务器实现

本章概括性地介绍了 FTP 的 Web 文件管理系统的实现方法。在 FTP 服务器层面，对第 3 章实现的服务器命令进行了扩展，增加了特定指令以提升服务器的功能。在 Web 应用层面，后端部分基于 Python 标准库进行了模块扩展，包含了完善的接口处理流程和异常处理机制；前端则通过 HTML 和 JavaScript 构建了功能完善的页面，实现了与后端的交互。

4.1　FTP 服务器

服务器端仍旧采用第 3 章实现的服务器，对命令进行扩展。增加了“MKD”、“RMD”、“CDUP”、“TYPE”和“DELE”指令，使其支持新增文件夹、删除文件和文件夹，切换数据通道传输格式的能力。

4.2　后端实现（Python HTTP 服务）

后端基于 Python 标准库 http.server 模块，扩展 SimpleHTTPRequestHandler，实现 FTPHandler 类。该类支持多个接口，包括：/list（列目录）、/uploadFile（上传）、/download（下载）、/mkdir（新建文件夹）、/delete（删除文件/文件夹）、/rename（重命名）。

接口处理流程：接收 HTTP 请求，解析用户参数，连接 FTP 服务器，调用相应 FTP 命令（如 LIST、STOR、RETR、MKD 等），关闭 FTP 连接—返回结果。接口均支持异常处理，保证了系统的稳定性。

4.3　前端实现（index.html）

前端通过 HTML 和 JavaScript 实现，页面支持文件列表显示、文件上传、下载、新建文件夹、删除、重命名、刷新、目录切换、用户登录/退出等功能。核心逻辑通过 fetch 发送 HTTP 请求与后端交互，采用 JSON 或 FormData 格式传递参数。

4.4 效果演示

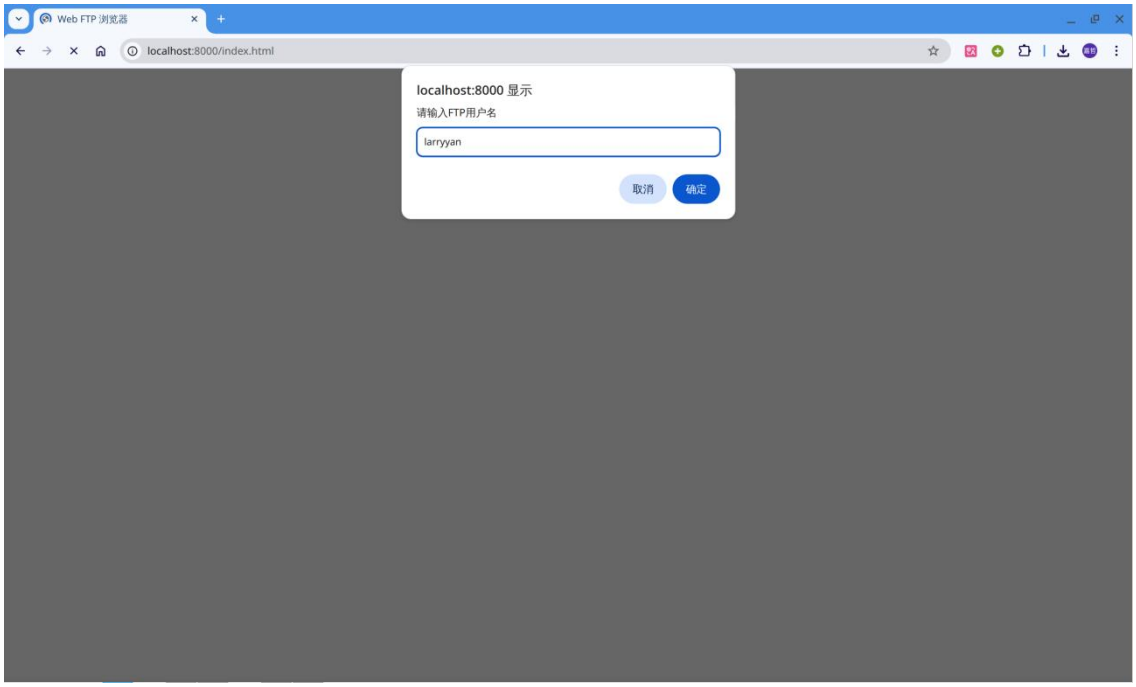

图 4.1 Web FTP 浏览器用户登录

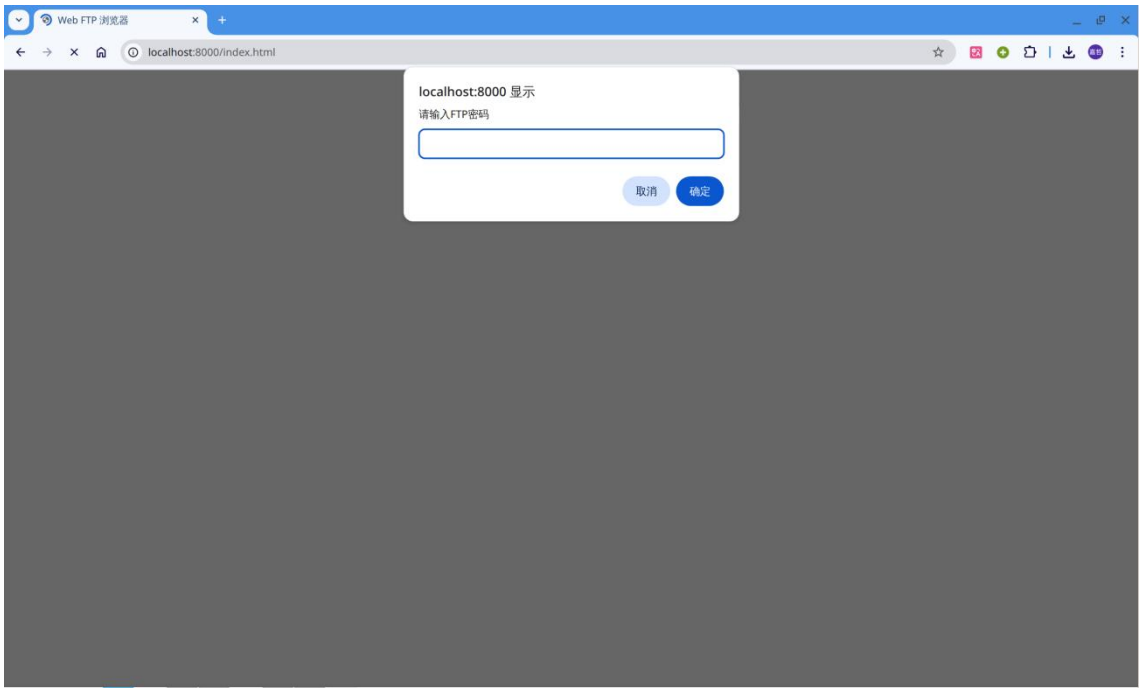

图 4.2 Web FTP 浏览器用户登录

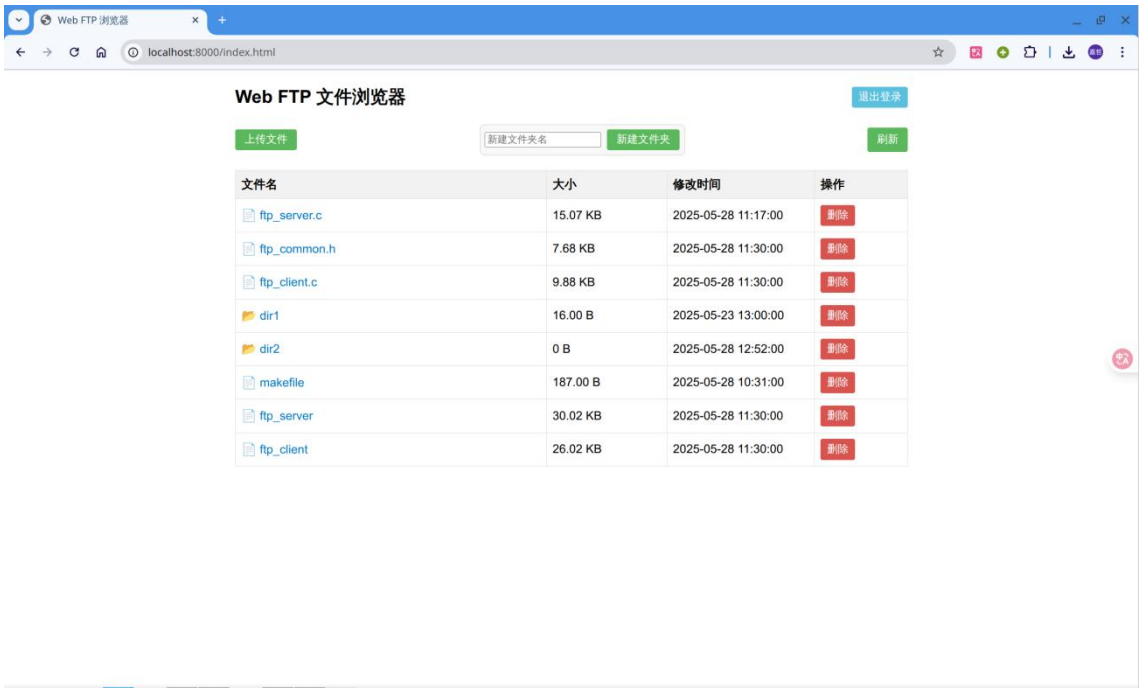

图 4.3 Web FTP 浏览器文件列表展示

图 4.4 Web FTP 浏览器进入子文件夹

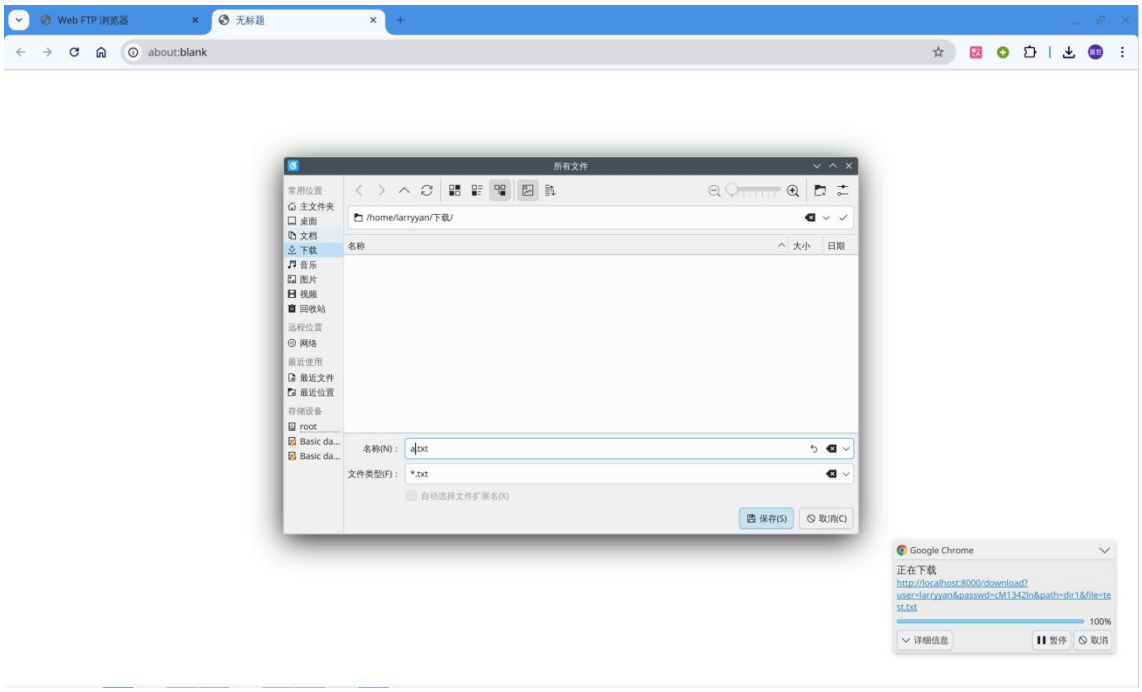

图 4.5 Web FTP 浏览器文件下载

图 4.6 Web FTP 浏览器文件上传

图 4.7 Web FTP 浏览器文件上传结果

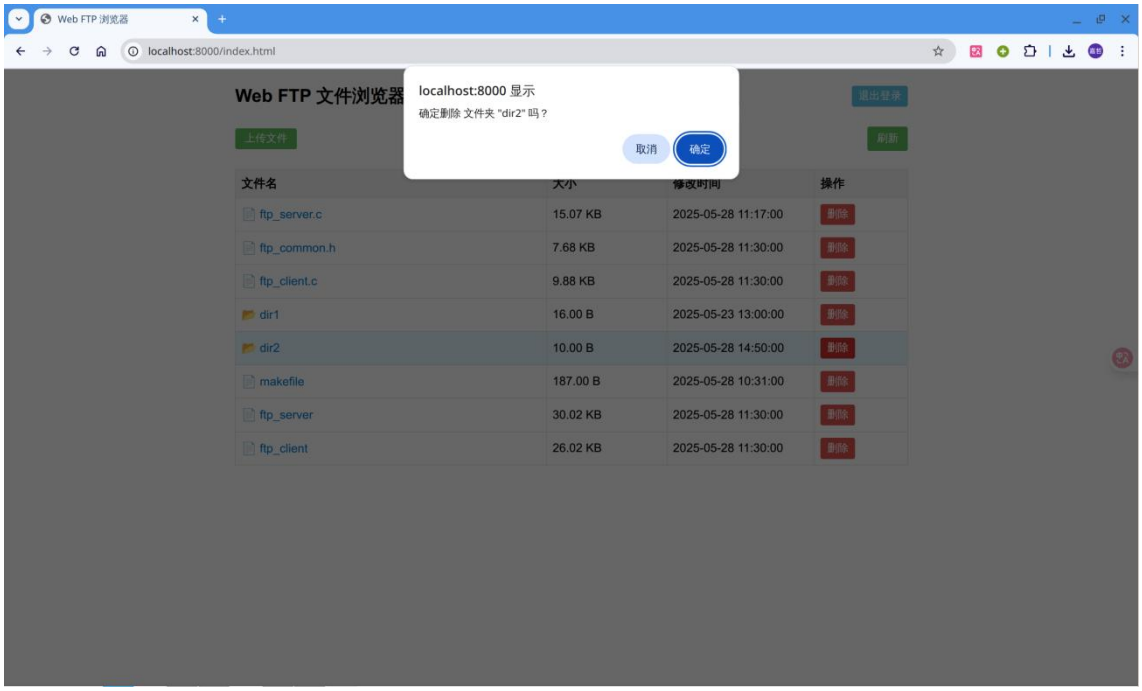

图 4.8 Web FTP 浏览器删除文件夹

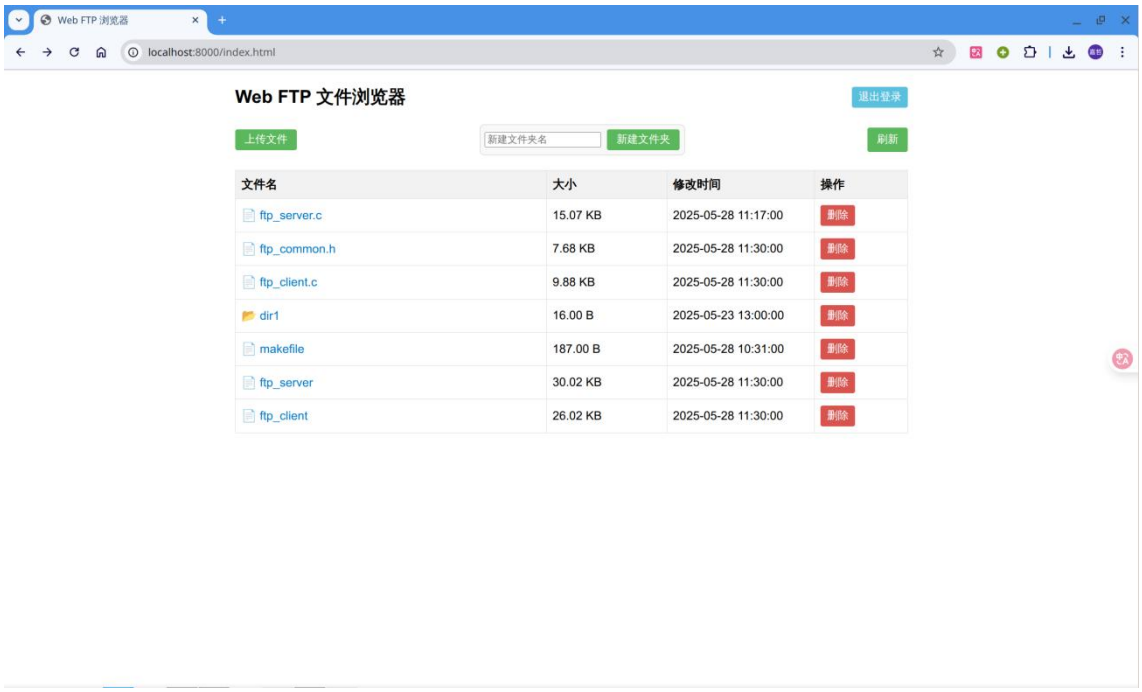

图 4.9 Web FTP 浏览器删除文件夹成功

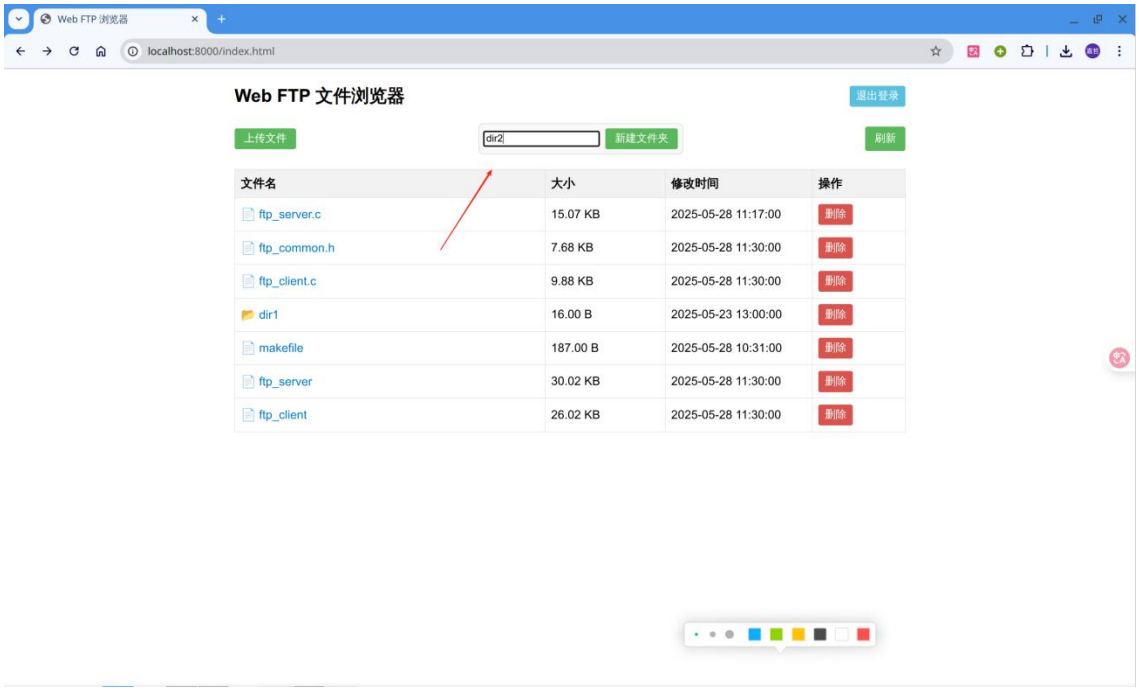

图 4.10 Web FTP 浏览器新建文件夹

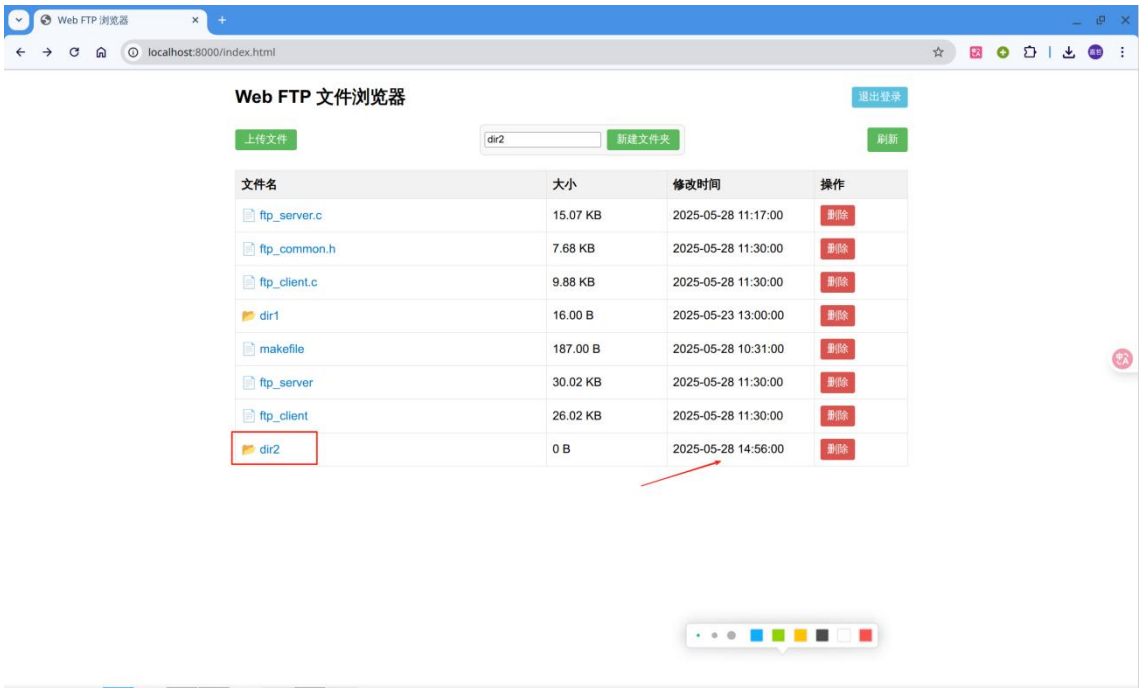

图 4.11 Web FTP 浏览器新建文件夹成功

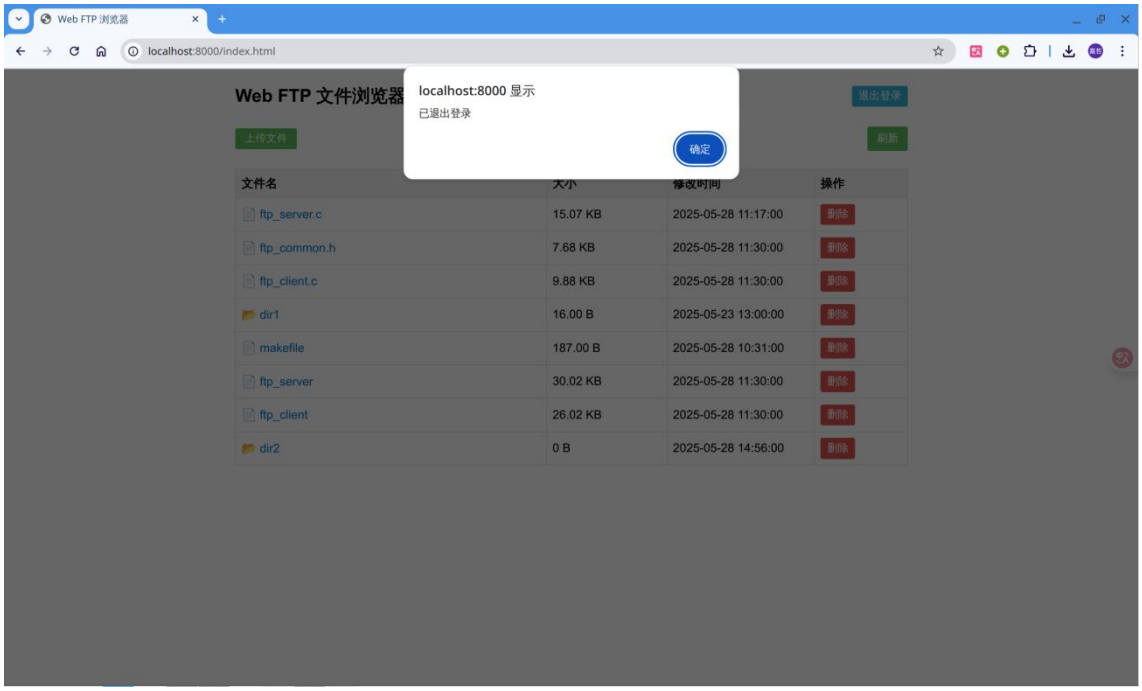

图 4.12 退出登录

```
客户端连接已建立。
Received command: USER anonymous
Received command: PASS anonymous
Received command: CWD .
Received command: TYPE A
Received command: PASV
Received command: LIST
Received command: QUIT
客户端断开连接。
客户端连接已建立。
Received command: USER anonymous
Received command: PASS anonymous
Received command: CWD .
Received command: TYPE A
Received command: PASV
Received command: LIST
Received command: QUIT
客户端断开连接。
客户端连接已建立。
Received command: USER anonymous
Received command: PASS anonymous@
Received command: CWD .
Received command: TYPE A
Received command: PASV
Received command: LIST
Received command: QUIT
客户端断开连接。
客户端连接已建立。
Received command: USER anonymous
Received command: PASS anonymous@
Received command: CWD dir1
Received command: TYPE A
Received command: PASV
Received command: LIST
Received command: QUIT
客户端断开连接。
客户端连接已建立。
Received command: USER anonymous
Received command: PASS anonymous@
Received command: CWD .
Received command: TYPE A
Received command: PASV
Received command: LIST
Received command: QUIT
客户端断开连接。
```

图 4.13 后端控制命令

访问网址后，先输入用户名和密码完成登录。登录成功后，可通过点击目录名称进入任意文件夹，页面会自动刷新显示该目录下的文件和子目录。点击文件名即可下载文件，点击“上传文件”按钮选择文件则可上传至当前目录。每个文件和文件夹旁都有“删除”按钮，确认后可以一键删除；新建文件夹也很方便，只需输入名称并点击“新建文件夹”即可。退出时，点击“退出登录”即可安全登出，整体流程简洁高效，日常文件管理非常便捷。

第 5 章　基于 Socket 编程的简单网络爬虫

本章主要介绍基于 Socket 编程的简单网络爬虫，详细阐述了编写网络爬虫的整体思路和设计流程，对实现网络爬虫过程中使用的关键函数及其具体功能进行了描述，并展示该网络爬虫的实际运行效果。

5.1　程序思路和设计流程

本章实现一个基于 Socket 编程的简易网络爬虫，能够通过 HTTP/HTTPS 协议抓取指定静态网站的网页文本和图片资源，并将其保存到本地文件夹。

本爬虫程序采用 C 语言开发，核心流程包括：

（1）解析输入 URL，提取主机名和路径信息；

（2） 通过 Socket 建立与目标服务器的 TCP 连接（HTTP）或 SSL 连接（HTTPS），其中 SSL 连接需要调用 openssl 库；

（3）发送 HTTP GET 请求，获取网页 HTML 内容；

（4）分析并提取页面中的图片链接，依次下载保存；

（5）支持自动为每个目标网站创建对应本地下载文件夹。

5.2　关键函数与功能描述

表 5.1 爬虫程序的关键函数与功能描述

关键函数	功能描述
parse_url(url, host, path)	解析 URL，获取主机名和路径
init_openssl()	初始化 OpenSSL 环境，支持 HTTPS
cleanup_openssl()	清理 OpenSSL 环境
socket()/connect()	建立 TCP 连接
SSL_new()/SSL_connect()	建立 SSL 安全连接
send()/recv()或 SSL_write()/SSL_read()	发送和接收 HTTP 数据
parse_html_for_images(html, urls)	解析 HTML 内容，提取图片 URL
download_file(url, save_path)	下载文本或图片文件并保存本地

5.3 效果演示

运行指令：./crawler https://www.cup.edu.cn/cupai/

在命令行输入目标网址后，程序会自动连接网站并下载首页 HTML 源码，保存在本地。随后自动识别网页中的所有图片链接，将图片一并下载到本地的 download/<站点名>目录下。

图 5.1 目标网站

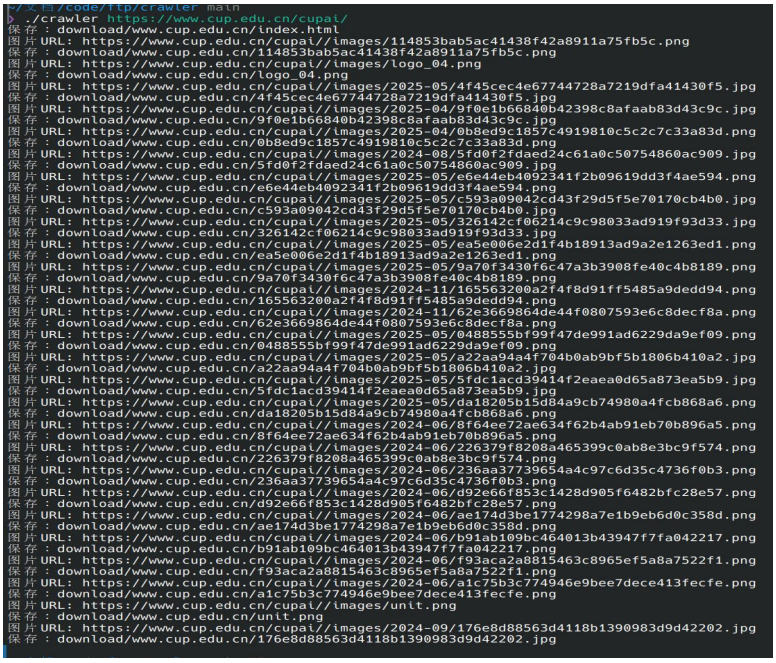

图 5.2 日志输出

图 5.3 爬虫取文件夹展示

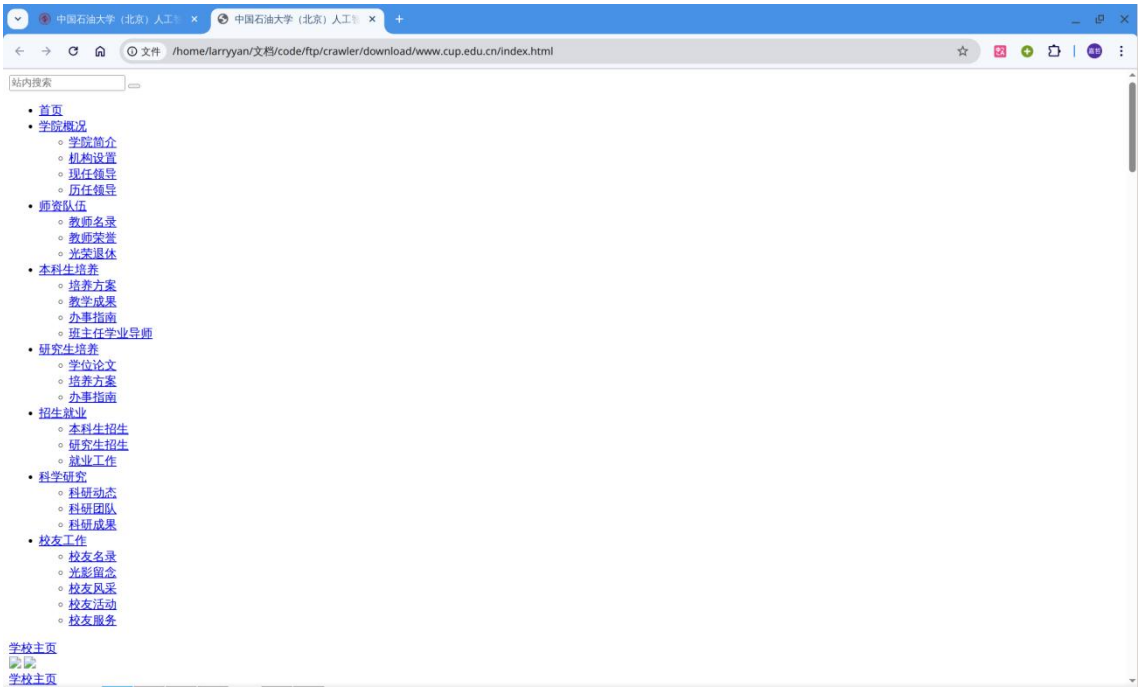

图 5.4 爬取文本信息

第 6 章　思考题解答

6.1　问题一

问题：在 FTP 中，为什么要建立两个 TCP 连接来分别传送命令和数据？

回答：

在 FTP 中建立两个 TCP 连接来分别传送命令和数据主要基于以下原因：

命令通道与数据通道分工明确。命令通道用于发送和接收控制指令，例如用户登录（USER、PASS）、切换目录（CWD）、显示目录（PWD）等；而数据通道仅在需要进行文件或目录内容传输（如 RETR、STOR、LIST 等）时临时建立和关闭。两者互不干扰，保证在进行大文件传输或长时间数据交换时，控制指令始终能被及时响应，提高了用户交互的流畅性和系统稳定性。

提升并发性和安全性。实验中，每当客户端执行 get、put、dir 等命令时，都会为数据传输单独建立一个数据连接，传输完成后及时关闭。这样既节约了系统资源，又避免了长时间占用，方便服务器管理和多客户端并发接入。

方便协议实现和扩展。从代码实现角度看，主线程始终维护命令通道，随时可以接收客户端的控制指令，而数据通道则可以由独立函数动态创建、关闭。无论主动模式（PORT）还是被动模式（PASV），都可以灵活建立数据连接，简化了代码结构，也便于定位和处理数据通道相关的异常。

6.2　问题二

问题：主动方式和被动方式的主要区别是什么？为何要设计这两种方式？

回答：

主要区别：

主动方式（PORT）：由客户端在本地随机开放一个端口并监听，客户端通过命令通道发送 PORT 命令，告知服务器其 IP 和端口，服务器主动连接该端口，建立数据通道。

被动方式（PASV）：由服务器开放一个端口监听，客户端通过命令通道发送 PASV 命令，服务器返回自己的 IP 和端口，客户端再主动连接该端口，建立数据通道。

设计这两种方式的原因：

主动模式建立连接的方式很直接，但是服务器主动连接客户端数据端口，若客户端处于 NAT 内网环境，路由器进行端口转发后，外部连接会被阻断，连接不易建立。所以为了适应 NAT 环境，市面上的客户端通常使用被动模式。

6.3 问题三

问题：当使用 FTP 下载大量小文件的时候，速度会很慢，这是什么缘故？可以怎样改进？

回答：

下载大量小文件速度慢的原因：每次下载一个小文件时都需要重新建立和关闭数据通道，频繁进行 TCP 连接的创建与释放，并发送控制命令、返回状态码，这一过程消耗的时间和系统资源远远大于小文件本身的数据传输时长。此外，每个小文件的传输都要独立完成协议交互和 IO 操作，导致控制流与数据流的往返次数剧增，操作系统和网络也因频繁打开关闭 socket、文件句柄、分配缓冲区等操作而负担加重，从而导致整体传输效率大幅下降。

改进建议：

1. 尽量将多个小文件打包为一个大文件（如 tar/zip 归档），上传或下载后再本地解压，减少数据连接建立次数。
2. 客户端可以缓存一部分小文件，集中发送，然后服务器也需支持统一接收。

第 7 章　实验结论

7.1　实验成果总结

7.1.1　FTP 协议完整实现

基于 C 语言和 Socket 接口实现了符合 RFC959 规范的 FTP 双通道通信架构，支持用户认证（匿名/普通用户）、文件上传/下载、目录管理（LIST/CWD/MKD/RMD）、多模式传输（主动 PORT/被动 PASV）等功能。

通过分离控制通道（命令交互）与数据通道（文件传输），确保了交互实时性与传输稳定性，验证了 FTP 协议的核心设计思想。

7.1.2　Web 可视化扩展

开发了基于 Python HTTP 服务与前端 HTML/JavaScript 的 Web FTP 浏览器，实现了图形化文件管理功能（上传、下载、删除、重命名、目录切换），证明了协议层与展示层解耦的可行性。

后端通过 SimpleHTTPRequestHandler 扩展实现 RESTful 接口，前端采用异步 fetch 交互，体现了现代 Web 技术的灵活性与用户体验优化。

7.1.3　网络爬虫技术实践

基于 Socket 编程实现了支持 HTTP/HTTPS 的简易爬虫，可解析 URL、建立 SSL 安全连接、提取网页文本与图片资源，并自动本地化存储，深化了对 HTTP 协议报文结构与资源抓取流程的理解。

7.2　实验收获

通过本次实验，我系统掌握了 FTP 协议的双通道架构与核心命令实现，基于 C 语言完成了符合 RFC959 规范的 FTP 服务器开发，深入理解控制通道与数据通道的协同机制。在 Web 文件管理器部分，实践了前后端分离架构，通过 Python 搭建 RESTful 接口服务，结合 HTML/JavaScript 实现可视化文件管理功能。网络爬虫模块的开发，强化了我对 HTTP/HTTPS 协议的理解，掌握了 SSL 加密通信与资源自动化抓取技术。

附录 A　程序源代码 ftp_common.h

用于 ftp 服务器和客户端的共用代码段

```
#ifndef FTP_COMMON_H
#define FTP_COMMON_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

// ---------------------
// 通用常量定义
// ---------------------
#define COMMAND_PORT 2121          // 命令端口
#define DATA_PORT 2020             // 默认数据端口
#define BUFFER_SIZE 1024          // 缓冲区大小
#define MAX_PATH_LEN 256          // 文件路径长度

// ---------------------
// FTP 命令定义
// ---------------------
#define CMD_USER "USER"
#define CMD_PASS "PASS"
#define CMD_RETR "RETR"
#define CMD_STOR "STOR"
#define CMD_LIST "LIST"
#define CMD_CDUP "CDUP"
#define CMD_MKD "MKD"
#define CMD_RMD "RMD"
#define CMD_CWD "CWD"
#define CMD_PWD "PWD"
```

```
    #define CMD_QUIT "QUIT"
    #define CMD_PORT "PORT"
    #define CMD_PASV "PASV"
    #define CMD_TYPE "TYPE"
    #define CMD_DELE "DELE"

    // ---------------------
    // 命令结构体（可选）
    // ---------------------
    typedef struct {
        char cmd[255];
        char arg[MAX_PATH_LEN];
    } FTPCommand;

    // 去掉字符串末尾的换行符
    void trim_newline(char *str) {
        size_t len = strlen(str);
        while (len > 0 && (str[len - 1] == '\n' || str[len - 1] == '\r'))
{
            str[-- len] = '\0';  // 去掉换行符
        }
    }

    // 解析命令字符串，提取命令和参数
    int parse_command(const char *input, FTPCommand *cmd) {
        char *space_pos = strchr(input, ' ');
        if (space_pos) {
            // 提取命令和参数
            size_t cmd_len = space_pos - input;
            strncpy(cmd->cmd, input, cmd_len);
            cmd->cmd[cmd_len] = '\0';

            // 提取参数
            strncpy(cmd->arg, space_pos + 1, MAX_PATH_LEN - 1);
            cmd->arg[MAX_PATH_LEN - 1] = '\0';  // 防止溢出
        } else {
            // 只包含命令，没有参数
            strncpy(cmd->cmd, input, MAX_PATH_LEN - 1);
            cmd->cmd[MAX_PATH_LEN - 1] = '\0';
```

```
            cmd->arg[0] = '\0';  // 参数为空
        }

        // 检查命令是否有效
        if (strcmp(cmd->cmd, CMD_USER) == 0 || strcmp(cmd->cmd, CMD_PASS)
== 0 ||
            strcmp(cmd->cmd, CMD_RETR) == 0 || strcmp(cmd->cmd, CMD_STOR)
== 0 ||
            strcmp(cmd->cmd, CMD_LIST) == 0 || strcmp(cmd->cmd, CMD_PWD) ==
0 ||
            strcmp(cmd->cmd, CMD_MKD) == 0 || strcmp(cmd->cmd, CMD_RMD) ==
0 ||
            strcmp(cmd->cmd, CMD_CWD) == 0 || strcmp(cmd->cmd, CMD_CDUP) ==
0 ||
            strcmp(cmd->cmd, CMD_PORT) == 0 || strcmp(cmd->cmd, CMD_PASV)
== 0 ||
            strcmp(cmd->cmd, CMD_DELE) == 0 || strcmp(cmd->cmd, CMD_TYPE)
== 0 ||
            strcmp(cmd->cmd, CMD_QUIT) == 0) {
            return 0;  // 成功解析
        }

        return -1;  // 无效命令
    }

    // 发送指定长度的数据
    ssize_t send_all(int sockfd, const void *buf, size_t len) {
        size_t total_sent = 0;
        ssize_t bytes_sent;

        while (total_sent < len) {
            bytes_sent = send(sockfd, (const char *)buf + total_sent, len
- total_sent, 0);
            if (bytes_sent == -1) {
                return -1;  // 错误发生，返回 -1
            }
            total_sent += bytes_sent;
        }
        return total_sent;
```

```
    }

    // 接收指定长度的数据
    ssize_t recv_all(int sockfd, void *buf, size_t len) {
        size_t total_received = 0;
        ssize_t bytes_received;

        while (total_received < len) {
            bytes_received = recv(sockfd, (char *)buf + total_received, len
- total_received, 0);
            if (bytes_received == -1) {
                return -1;  // 错误发生，返回 -1
            }
            total_received += bytes_received;

            // 客户端关闭连接
            if (bytes_received == 0) {
                break;
            }
        }
        return total_received;
    }

    // 创建数据连接（服务器端）
    int create_data_connection(int listen_port) {
        int data_sock;
        struct sockaddr_in data_addr;

        data_sock = socket(AF_INET, SOCK_STREAM, 0);
        if (data_sock < 0) {
            perror("data socket error");
            return -1;
        }

        memset(&data_addr, 0, sizeof(data_addr));
        data_addr.sin_family = AF_INET;
        data_addr.sin_addr.s_addr = INADDR_ANY;
        data_addr.sin_port = htons(listen_port);
```

```
            if (bind(data_sock, (struct sockaddr *)&data_addr,
sizeof(data_addr)) < 0) {
                perror("bind data port error");
                return -1;
            }

            if (listen(data_sock, 1) < 0) {
                perror("listen data port error");
                return -1;
            }

            int client_data_sock = accept(data_sock, NULL, NULL);
            if (client_data_sock < 0) {
                perror("accept data socket error");
                return -1;
            }

            close(data_sock);
            return client_data_sock;
        }

        // 发送文件
        int send_file(int data_sock, const char *filename) {
            FILE *file = fopen(filename, "rb");
            if (!file) {
                perror("fopen error");
                return -1;
            }

            char buffer[BUFFER_SIZE];
            size_t bytes_read;
            while ((bytes_read = fread(buffer, 1, sizeof(buffer), file)) > 0)
{
                send_all(data_sock, buffer, bytes_read);
            }

            fclose(file);
            return 0;
        }
```

```
    // 接收文件
    int recv_file(int data_sock, const char *filename) {
        FILE *file = fopen(filename, "wb");
        if (!file) {
            perror("fopen error");
            return -1;
        }

        char buffer[BUFFER_SIZE];
        ssize_t bytes_received;
             while ((bytes_received = recv_all(data_sock, buffer,
sizeof(buffer))) > 0) {
            fwrite(buffer, 1, bytes_received, file);
        }

        fclose(file);
        return 0;
    }

    // 连接到数据通道
    int connect_data_channel(const char *ip, int port) {
        int data_sock;
        struct sockaddr_in data_addr;

        data_sock = socket(AF_INET, SOCK_STREAM, 0);
        if (data_sock < 0) {
            perror("socket error");
            return -1;
        }

        memset(&data_addr, 0, sizeof(data_addr));
        data_addr.sin_family = AF_INET;
        data_addr.sin_port = htons(port);
        if (inet_pton(AF_INET, ip, &data_addr.sin_addr) <= 0) {
            perror("invalid address or address not supported");
            return -1;
        }
```

```
            if (connect(data_sock, (struct sockaddr *)&data_addr,
sizeof(data_addr)) < 0) {
            perror("connection failed");
            return -1;
        }

        return data_sock;
    }


    // 获取随机端口
    int get_random_port() {
        int sockfd;
        struct sockaddr_in addr;
        socklen_t addrlen = sizeof(addr);

        // 先尝试默认端口
        sockfd = socket(AF_INET, SOCK_STREAM, 0);
        if (sockfd < 0) return -1;

        int opt = 1;
        setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));
// 端口可复用

        memset(&addr, 0, sizeof(addr));
        addr.sin_family = AF_INET;
        addr.sin_addr.s_addr = htonl(INADDR_ANY);
        addr.sin_port = htons(DATA_PORT);

        if (bind(sockfd, (struct sockaddr*)&addr, sizeof(addr)) == 0) {
            // 默认端口可用
            close(sockfd);
            return DATA_PORT;
        }
        close(sockfd);

        // 默认端口被占用，随机分配
        sockfd = socket(AF_INET, SOCK_STREAM, 0);
        if (sockfd < 0) return -1;
```

```
    memset(&addr, 0, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = htonl(INADDR_ANY);
    addr.sin_port = 0; // 让系统分配端口

    if (bind(sockfd, (struct sockaddr*)&addr, sizeof(addr)) < 0) {
        close(sockfd);
        return -1;
    }

    if (getsockname(sockfd, (struct sockaddr*)&addr, &addrlen) < 0) {
        close(sockfd);
        return -1;
    }

    int port = ntohs(addr.sin_port);
    close(sockfd);
    return port;
}
#endif // FTP_COMMON_H
```

附录 B　程序源代码 ftp_server.c

ftp 服务器代码

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <dirent.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <pthread.h>
#include <ifaddrs.h>
#include <net/if.h>
#include <pwd.h>
#include <grp.h>
#include <shadow.h>
#include <crypt.h>

#include "ftp_common.h"

// 存储当前客户端的用户名和认证状态
typedef struct {
    int client_sock;
    int data_sock;
    int logged_in;
    int data_port;
    int passive_mode;
    char username[MAX_PATH_LEN];
    char current_dir[MAX_PATH_LEN];
} FTPClient;


// 向客户端发送响应
void send_response(int client_sock, const char *message, int code) {
    char response[BUFFER_SIZE];
    snprintf(response, sizeof(response), "%d %s\r\n", code, message);
```

```
        send(client_sock, response, strlen(response), 0);
    }

    // 登录验证（使用 Linux 系统用户/支持 anonymous 匿名用户）
    void authenticate(FTPClient *client, const char *username, const char
*password) {
        if (strcmp(username, "anonymous") == 0) {
            client->logged_in = 1;
            send_response(client->client_sock, "Anonymous login ok", 230);
            return;
        }

        struct passwd *pw = getpwnam(username);
        if (!pw) {
                send_response(client->client_sock, "Invalid username or
password", 530);
            return;
        }

        struct spwd *sp = getspnam(username);
        if (!sp) {
             send_response(client->client_sock, "Unable to access shadow
file", 550);
            return;
        }

        // 用 shadow 文件中的加密盐加密输入密码
        char *encrypted = crypt(password, sp->sp_pwdp);
        if (encrypted && strcmp(encrypted, sp->sp_pwdp) == 0) {
            client->logged_in = 1;
            send_response(client->client_sock, "Login successful", 230);
        } else {
                send_response(client->client_sock, "Invalid username or
password", 530);
        }
    }

    // 处理客户端命令
    void handle_client(FTPClient *client) {
```

```
        char buffer[BUFFER_SIZE];
        FTPCommand cmd;

        // 发送欢迎消息
        send_response(client->client_sock, "Welcome to the FTP server",
220);

        while (1) {
            memset(buffer, 0, sizeof(buffer));
            int len = recv(client->client_sock, buffer, sizeof(buffer) - 1,
0);
            if (len <= 0) {
                break;
            }

            trim_newline(buffer);

            printf("Received command: %s\n", buffer);

            if (parse_command(buffer, &cmd) < 0) {
                   send_response(client->client_sock, "Invalid command",
500);
                continue;
            }

            // 处理命令
            if (strcmp(cmd.cmd, "USER") == 0) {
                strcpy(client->username, cmd.arg);
                 send_response(client->client_sock, "Password required",
331);
                continue;

            } else if (strcmp(cmd.cmd, "PASS") == 0) {
                authenticate(client, client->username, cmd.arg);    // 验
证用户名和密码
                continue;

            } else if (strcmp(cmd.cmd, "PWD") == 0) {
                if (!client->logged_in) {
```

```
                    send_response(client->client_sock, "Please login
first", 530);
                } else {
                    char cwd[MAX_PATH_LEN];
                    getcwd(cwd, sizeof(cwd));
                    strcat(cwd, "\n");
                    send_response(client->client_sock, cwd, 257);
                }

            } else if (strcmp(cmd.cmd, "TYPE") == 0) {
                if (!client->logged_in) {
                        send_response(client->client_sock, "Please login
first", 530);
                    continue;
                }

                if (strcmp(cmd.arg, "I") == 0) {
                        send_response(client->client_sock, "Switching to
binary mode", 200);
                } else if (strcmp(cmd.arg, "A") == 0) {
                    send_response(client->client_sock, "Switching to ASCII
mode", 200);
                } else {
                      send_response(client->client_sock, "Invalid type",
501);
                }

            } else if (strcmp(cmd.cmd, "LIST") == 0) {
                if (!client->logged_in) {
                        send_response(client->client_sock, "Please login
first", 530);
                    continue;
                }

                    send_response(client->client_sock, "Opening data
connection", 150);

                DIR *dir;
                struct dirent *entry;
```

```
                char path[MAX_PATH_LEN];
                getcwd(path, sizeof(path));
                dir = opendir(path);
                if (!dir) {
                      send_response(client->client_sock, "Unable to open
directory", 550);
                    continue;
                }

                char line[BUFFER_SIZE];
                struct stat st;
                char fullpath[MAX_PATH_LEN];
                while ((entry = readdir(dir)) != NULL) {
                     snprintf(fullpath, sizeof(fullpath), "%s/%s", path,
entry->d_name);
                    if (stat(fullpath, &st) == 0) {
                        // 权限字符串
                        char perms[11] = "----------";
                        if (S_ISDIR(st.st_mode)) perms[0] = 'd';
                        if (st.st_mode & S_IRUSR) perms[1] = 'r';
                        if (st.st_mode & S_IWUSR) perms[2] = 'w';
                        if (st.st_mode & S_IXUSR) perms[3] = 'x';
                        if (st.st_mode & S_IRGRP) perms[4] = 'r';
                        if (st.st_mode & S_IWGRP) perms[5] = 'w';
                        if (st.st_mode & S_IXGRP) perms[6] = 'x';
                        if (st.st_mode & S_IROTH) perms[7] = 'r';
                        if (st.st_mode & S_IWOTH) perms[8] = 'w';
                        if (st.st_mode & S_IXOTH) perms[9] = 'x';

                        // 获取用户名和组名
                        struct passwd *pw = getpwuid(st.st_uid);
                        struct group  *gr = getgrgid(st.st_gid);
                        char *owner = pw ? pw->pw_name : "owner";
                        char *group = gr ? gr->gr_name : "group";

                        // 获取文件的修改时间
                        char timebuf[32];
                        struct tm *tm_info = localtime(&st.st_mtime);
                        strftime(timebuf, sizeof(timebuf), "%b %d %H:%M",
```

```
tm_info);

                    // 构造 ls -l 风格输出
                                        snprintf(line,  sizeof(line),
"%s %3ld %-10s %-10s %8ld %s %s\r\n",
                            perms,
                            (long)st.st_nlink,
                            owner,
                            group,
                            (long)st.st_size,
                            timebuf,
                            entry->d_name
                        );
                        send_all(client->data_sock, line, strlen(line));
                    }
                }
                closedir(dir);

                close(client->data_sock);
                client->data_sock = -1;
                 send_response(client->client_sock, "Directory send OK",
226);

            } else if (strcmp(cmd.cmd, "CWD") == 0) {
                if (!client->logged_in) {
                        send_response(client->client_sock, "Please login
first", 530);
                    continue;
                }

                if (chdir(cmd.arg) == 0) {
                            send_response(client->client_sock,  "Directory
changed", 250);
                } else {
                    send_response(client->client_sock, "Directory change
failed", 550);
                }

            }else if (strcmp(cmd.cmd, "MKD") == 0) {
```

```
                if (!client->logged_in) {
                        send_response(client->client_sock, "Please login
first", 530);
                    continue;
                }

                if (mkdir(cmd.arg, 0755) == 0) {
                    char response_msg[BUFFER_SIZE];
                    char abs_path[MAX_PATH_LEN];
                    realpath(cmd.arg, abs_path);
                     snprintf(response_msg, sizeof(response_msg), "\"%s\"
directory created", abs_path);
                        send_response(client->client_sock, response_msg,
257);
                } else {
                     send_response(client->client_sock, "Create directory
failed", 550);
                }

            } else if (strcmp(cmd.cmd, "RMD") == 0) {
                if (!client->logged_in) {
                        send_response(client->client_sock, "Please login
first", 530);
                    continue;
                }

                if (rmdir(cmd.arg) == 0) {
                          send_response(client->client_sock, "Directory
removed", 250);
                } else {
                     send_response(client->client_sock, "Remove directory
failed", 550);
                }
            } else if (strcmp(cmd.cmd, "CDUP") == 0) {
                if (!client->logged_in) {
                        send_response(client->client_sock, "Please login
first", 530);
                    continue;
                }
```

```
                if (chdir("..") == 0) {
                    send_response(client->client_sock, "Changed to parent
directory", 250);
                } else {
                    send_response(client->client_sock, "Failed to change
to parent directory", 550);
                }

            } else if (strcmp(cmd.cmd, "PORT") == 0) {
                if (!client->logged_in) {
                        send_response(client->client_sock, "Please login
first", 530);
                    continue;
                }

                // 解析 PORT 命令参数，获取客户端的 IP 地址和端口号
                char *token = strtok(cmd.arg, ",");
                char ip[16] = "";
                int port1 = 0, port2 = 0;

                // 提取 IP 地址
                for (int i = 0; token != NULL && i < 4; i++) {
                    strcat(ip, token);
                    if (i < 3) {
                        strcat(ip, ".");
                    }
                    token = strtok(NULL, ",");
                }

                // 提取端口号
                if (token != NULL) {
                    port1 = atoi(token);   // 端口的高位
                    token = strtok(NULL, ",");
                }
                if (token != NULL) {
                    port2 = atoi(token);   // 端口的低位
                }
```

```
                // 计算最终端口号
                client->data_port = (port1 * 256) + port2;

                // 打印 IP 和端口号
                printf("PORT command received: IP = %s, PORT = %d\n", ip,
client->data_port);

                            client->data_sock  =  connect_data_channel(ip,
client->data_port);
                client->passive_mode = 0; // 设置为主动模式

                    send_response(client->client_sock, "Data connection
established", 200);
            } else if (strcmp(cmd.cmd, "PASV") == 0) {
                if (!client->logged_in) {
                        send_response(client->client_sock, "Please login
first", 530);
                    continue;
                }

                // 随机分配一个临时端口
                client->data_port = get_random_port();

                // 获取本机 IP
                struct sockaddr_in addr;
                socklen_t addrlen = sizeof(addr);
                getsockname(client->client_sock, (struct sockaddr*)&addr,
&addrlen);
                unsigned int ip = ntohl(addr.sin_addr.s_addr);

                // 组装 PASV 响应
                int h1 = (ip >> 24) & 0xFF;
                int h2 = (ip >> 16) & 0xFF;
                int h3 = (ip >> 8) & 0xFF;
                int h4 = ip & 0xFF;
                int p1 = (client->data_port >> 8) & 0xFF;
                int p2 = client->data_port & 0xFF;

                char pasv_msg[128];
```

```
                snprintf(pasv_msg, sizeof(pasv_msg),
                    "Entering Passive Mode (%d,%d,%d,%d,%d,%d)", h1, h2,
h3, h4, p1, p2);

                send_response(client->client_sock, pasv_msg, 227);

                // 创建数据通道监听
                                                client->data_sock    =
create_data_connection(client->data_port);
                if (client->data_sock < 0) {
                    send_response(client->client_sock, "Failed to create
data connection", 425);
                    continue;
                }
                client->passive_mode = 1; // 设置为被动模式

            } else if (strcmp(cmd.cmd, "RETR") == 0) {
                if (!client->logged_in) {
                    send_response(client->client_sock, "Please login
first", 530);
                    continue;
                }

                    send_response(client->client_sock, "Opening data
connection", 150);

                if (send_file(client->data_sock, cmd.arg) < 0) {
                    send_response(client->client_sock, "File transfer
failed", 550);
                }

                close(client->data_sock); // 关闭数据连接
                client->data_sock = -1;   // 重置数据连接
                // 关闭数据连接
                    send_response(client->client_sock, "File transfer
complete", 226);
                } else if (strcmp(cmd.cmd, "STOR") == 0) {
                if (!client->logged_in) {
                    send_response(client->client_sock, "Please login
```

```
first", 530);
                    continue;
                }

                    send_response(client->client_sock, "Opening data
connection", 150);

                if (recv_file(client->data_sock, cmd.arg) < 0) {
                     send_response(client->client_sock, "File upload
failed", 550);
                    close(client->data_sock);
                    client->data_sock = -1;
                    continue; // 直接跳过后续
                }

                close(client->data_sock);
                client->data_sock = -1;
                     send_response(client->client_sock, "File upload
complete", 226);
            } else if (strcmp(cmd.cmd, "DELE") == 0) {
                if (!client->logged_in) {
                     send_response(client->client_sock, "Please login
first", 530);
                    continue;
                }

                if (remove(cmd.arg) == 0) {
                     send_response(client->client_sock, "File deleted
successfully", 250);
                } else {
                     send_response(client->client_sock, "Failed to delete
file", 550);
                }
            } else if (strcmp(cmd.cmd, "QUIT") == 0) {
                send_response(client->client_sock, "Goodbye", 221);
                break;
            }
        }
    }
```

```
    void handle_client_process(int client_sock) {
        FTPClient client = { client_sock, -1, 0, 0, 0, "", "/" };
        printf("客户端连接已建立。\n");
        handle_client(&client);  // 处理客户端请求
        close(client_sock);
        printf("客户端断开连接。\n");
    }

    int main() {
        int server_sock, client_sock;
        struct sockaddr_in server_addr, client_addr;
        socklen_t client_len = sizeof(client_addr);

        // 创建命令通道 socket
        server_sock = socket(AF_INET, SOCK_STREAM, 0);
        if (server_sock < 0) {
            perror("socket error");
            exit(1);
        }

        int opt = 1;
             setsockopt(server_sock,  SOL_SOCKET,  SO_REUSEADDR,  &opt,
sizeof(opt)); // 端口可复用

        // 配置服务器地址结构
        memset(&server_addr, 0, sizeof(server_addr));
        server_addr.sin_family = AF_INET;
        server_addr.sin_addr.s_addr = INADDR_ANY;
        server_addr.sin_port = htons(COMMAND_PORT);

        // 绑定
              if  (bind(server_sock,  (struct  sockaddr*)&server_addr,
sizeof(server_addr)) < 0) {
            perror("bind error");
            exit(1);
        }

        // 监听
```

```
        if (listen(server_sock, 5) < 0) {
            perror("listen error");
            exit(1);
        }

        printf("FTP 服务器已启动，监听端口 %d...\n", COMMAND_PORT);

        while (1) {
                            client_sock   =   accept(server_sock,   (struct
sockaddr*)&client_addr, &client_len);
            if (client_sock < 0) {
                perror("accept error");
                continue;
            }

            pid_t pid = fork();
            if (pid < 0) {
                perror("fork error");
                close(client_sock);
                continue;
            } else if (pid == 0) {
                // 子进程
                close(server_sock); // 子进程不需要监听 socket
                handle_client_process(client_sock);
                exit(0);
            } else {
                // 父进程
                close(client_sock); // 父进程不处理该连接
            }
        }

        close(server_sock);
        return 0;
    }
```

附录 C　程序源代码 ftp_client.c

ftp 客户端代码

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <ifaddrs.h>
#include <net/if.h>
#include <termios.h>

#include "ftp_common.h"

char server_ip[16];
int cmd_port;
int data_port = DATA_PORT; // 数据端口
int data_sock = -1; // 数据连接套接字

int passive_mode = 0; // 被动模式

void print_help() {
    printf("支持的命令:\n");
    printf("open <ip> <port>  连接到 FTP 服务器\n");
    printf("user <username>   登录用户名\n");
    printf("get <filename>    下载文件\n");
    printf("put <filename>    上传文件\n");
    printf("pwd               显示远程当前目录\n");
    printf("dir               列出远程目录\n");
    printf("cd <dirname>      切换远程目录\n");
    printf("?                 显示帮助\n");
    printf("quit              退出\n");
}

void recv_response(int sockfd) {
```

```
    char buffer[BUFFER_SIZE] = {0};
    int n = recv(sockfd, buffer, sizeof(buffer) - 1, 0);
    if (n > 0) {
        printf("%s", buffer);
    }
}

void get_local_ip(char *ip_buf, size_t buf_size) {
    struct ifaddrs *ifaddr, *ifa;

    if (ip_buf == NULL || buf_size < INET_ADDRSTRLEN) return;

    if (getifaddrs(&ifaddr) == -1) return;
    for (ifa = ifaddr; ifa != NULL; ifa = ifa->ifa_next) {
        // 跳过无效接口和非 IPv4 地址
        if (!ifa->ifa_addr || ifa->ifa_addr->sa_family != AF_INET) {
            continue;
        }
        // 跳过回环接口 (lo)
        if (strcmp(ifa->ifa_name, "lo") == 0) {
            continue;
        }
        // 转换为字符串并复制到缓冲区
        struct sockaddr_in *addr = (struct sockaddr_in *)ifa->ifa_addr;
        inet_ntop(AF_INET, &addr->sin_addr, ip_buf, buf_size);
        break;
    }
    freeifaddrs(ifaddr);
}

int data_connect(int sockfd) {
    if (passive_mode) {
        // 被动模式：发送 PASV 命令，解析服务器返回的 IP 和端口，连接
        send_all(sockfd, "PASV\r\n", 6);
        char resp[BUFFER_SIZE] = {0};
        recv(sockfd, resp, sizeof(resp) - 1, 0);
        printf("%s", resp);

        // 解析 227 Entering Passive Mode (h1,h2,h3,h4,p1,p2)
```

```
            int h1, h2, h3, h4, p1, p2;
            char *p = strchr(resp, '(');
            if (!p) return -1;
            sscanf(p+1, "%d,%d,%d,%d,%d,%d", &h1, &h2, &h3, &h4, &p1, &p2);
            char ip[32];
            snprintf(ip, sizeof(ip), "%d.%d.%d.%d", h1, h2, h3, h4);
            int port = p1 * 256 + p2;

            // 连接数据通道
            return connect_data_channel(ip, port);
        } else {
            // 主动模式：本地监听端口，发送 PORT 命令给服务器
            // 获取本地 IP
            char local_ip[16];
            get_local_ip(local_ip, sizeof(local_ip));
            // 计算端口号
            data_port = get_random_port();
            int p1 = (data_port >> 8) & 0xFF;
            int p2 = data_port & 0xFF;
            int ip1, ip2, ip3, ip4;
            sscanf(local_ip, "%d.%d.%d.%d", &ip1, &ip2, &ip3, &ip4);

            char port_cmd[64];
                                    snprintf(port_cmd,    sizeof(port_cmd),
"PORT %d,%d,%d,%d,%d,%d\r\n", ip1, ip2, ip3, ip4, p1, p2);
            send_all(sockfd, port_cmd, strlen(port_cmd));

            int data_sock = create_data_connection(data_port);
            if (data_sock < 0) return -1;

            char resp[BUFFER_SIZE] = {0};
            recv(sockfd, resp, sizeof(resp) - 1, 0);
            printf("%s", resp);

            // 主动模式下，客户端作为服务器监听数据端口，等待服务器连接
            return data_sock;
        }
    }
```

```
void download_file(int sockfd, const char *filename) {
    data_sock = data_connect(sockfd);
    if (data_sock < 0) {
        printf("数据连接失败\n");
        return;
    }

    char cmd[BUFFER_SIZE];
    snprintf(cmd, sizeof(cmd), "RETR %s\r\n", filename);
    send_all(sockfd, cmd, strlen(cmd));
    recv_response(sockfd); // 读取服务器 150 响应

    recv_file(data_sock, filename); // 接收文件

    printf("下载完成: %s\n", filename);
    close(data_sock); // 关闭数据连接
    recv_response(sockfd); // 读取服务器 226 响应
}

void upload_file(int sockfd, const char *filename) {
    data_sock = data_connect(sockfd);
    if (data_sock < 0) {
        printf("数据连接失败\n");
        return;
    }

    char cmd[BUFFER_SIZE];
    snprintf(cmd, sizeof(cmd), "STOR %s\r\n", filename);
    send_all(sockfd, cmd, strlen(cmd));
    recv_response(sockfd); // 读取服务器 150 响应

    send_file(data_sock, filename); // 发送文件

    printf("上传完成: %s\n", filename);
    close(data_sock); // 关闭数据连接
    recv_response(sockfd); // 读取服务器 226 响应
}

void get_file_list(int sockfd) {
```

```
    data_sock = data_connect(sockfd);
    if (data_sock < 0) {
        printf("数据连接失败\n");
        return;
    }

    send_all(sockfd, "LIST\r\n", 6);
    recv_response(sockfd); // 读取服务器 150 响应

    char buffer[BUFFER_SIZE];
    int n;
    while ((n = recv(data_sock, buffer, sizeof(buffer), 0)) > 0) {
        fwrite(buffer, 1, n, stdout);
    }

    close(data_sock); // 关闭数据连接
    recv_response(sockfd); // 读取服务器 226 响应
}

void login(int sockfd, const char *username) {
    char cmd[BUFFER_SIZE];
    char resp[BUFFER_SIZE] = {0};

    // 发送 USER 命令
    snprintf(cmd, sizeof(cmd), "USER %s\r\n", username);
    send_all(sockfd, cmd, strlen(cmd));

    // 接收服务器响应
    int n = recv(sockfd, resp, sizeof(resp) - 1, 0);
    if (n > 0) {
        printf("%s", resp);
        // 331 需要输入密码
        if (strncmp(resp, "331", 3) == 0) {
            char password[BUFFER_SIZE/2];
            printf("请输入密码: ");
            fflush(stdout);
            // 关闭回显
            struct termios oldt, newt;
            tcgetattr(STDIN_FILENO, &oldt);
```

```
                newt = oldt;
                newt.c_lflag &= ~ECHO;
                tcsetattr(STDIN_FILENO, TCSANOW, &newt);

                if (!fgets(password, sizeof(password), stdin)) {
                    // 恢复回显
                    tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
                    return;
                }

                // 恢复回显
                tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
                printf("\n");

                trim_newline(password);

                // 发送 PASS 命令
                snprintf(cmd, sizeof(cmd), "PASS %s\r\n", password);
                send_all(sockfd, cmd, strlen(cmd));

                memset(resp, 0, sizeof(resp));
                n = recv(sockfd, resp, sizeof(resp) - 1, 0);
                if (n > 0) {
                    printf("%s", resp);
                    if (strncmp(resp, "230", 3) == 0) {
                        printf("登录成功! \n");
                    } else if (strncmp(resp, "530", 3) == 0) {
                        printf("登录失败，请检查用户名或密码。\n");
                    }
                }
            } else if (strncmp(resp, "230", 3) == 0) {
                printf("登录成功! \n");
                printf("Remote system type is UNIX.\nUsing binary mode to
transfer files.\n");
            } else if (strncmp(resp, "530", 3) == 0) {
                printf("登录失败，请检查用户名。\n");
            }
        }
    }
```

```c
int main() {
    int sockfd;
    struct sockaddr_in server_addr;
    char input[BUFFER_SIZE];

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        perror("socket error");
        exit(1);
    }

    printf("请选择 FTP 模式(主动 0/被动 1): ");
    scanf("%d", &passive_mode);
    getchar(); // 清除换行符

    while (1) {
        printf("ftp> ");
        fflush(stdout);
        if (!fgets(input, sizeof(input), stdin)) break;
        trim_newline(input);

        if(strncmp(input, "open ", 5) == 0) {
            // 支持 open <ip> 或 open <ip> <port>
            char ip[16] = {0}, port[6] = {0};
            int cnt = sscanf(input + 5, "%15s %5s", ip, port);
            if (cnt == 1) {
                strcpy(server_ip, ip);
                cmd_port = COMMAND_PORT;      // 默认端口
            } else if (cnt == 2) {
                strcpy(server_ip, ip);
                cmd_port = atoi(port);
            } else {
                printf("用法: open <ip> [port]\n");
                continue;
            }
            printf("连接到 %s:%d\n", server_ip, cmd_port);
            close(sockfd);
            sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

```
                if (sockfd < 0) {
                    perror("socket error");
                    exit(1);
                }
                memset(&server_addr, 0, sizeof(server_addr));
                server_addr.sin_family = AF_INET;
                server_addr.sin_port = htons(cmd_port);
                server_addr.sin_addr.s_addr = inet_addr(server_ip);
                    if (connect(sockfd, (struct sockaddr*)&server_addr,
sizeof(server_addr)) < 0) {
                    perror("connect error");
                    exit(1);
                }
                recv_response(sockfd);
            } else if (strncmp(input, "user ", 5) == 0) {
                char username[BUFFER_SIZE];
                sscanf(input + 5, "%s", username);
                login(sockfd, username);
            } else if (strncmp(input, "quit", 4) == 0) {
                send_all(sockfd, "QUIT\r\n", 6);
                recv_response(sockfd);
                break;
            } else if (strncmp(input, "?", 1) == 0) {
                print_help();
            } else if (strncmp(input, "pwd", 3) == 0) {
                send_all(sockfd, "PWD\r\n", 5);
                recv_response(sockfd);
            } else if (strncmp(input, "dir", 3) == 0) {
                get_file_list(sockfd);
            } else if (strncmp(input, "cd ", 3) == 0) {
                char cmd[BUFFER_SIZE];
                snprintf(cmd, sizeof(cmd), "CWD %s\r\n", input + 3);
                send_all(sockfd, cmd, strlen(cmd));
                recv_response(sockfd);
            } else if (strncmp(input, "get ", 4) == 0) {
                download_file(sockfd, input + 4);
            } else if (strncmp(input, "put ", 4) == 0) {
                upload_file(sockfd, input + 4);
            } else {
```

```
                printf("未知命令，输入 ? 查看帮助。\n");
            }
        }

        close(sockfd);
        return 0;
    }
```

附录 D　程序源代码 ftp_client_web.py

ftp 文件管理 web 网页后端

```
from http.server import SimpleHTTPRequestHandler, HTTPServer
from urllib.parse import urlparse, parse_qs
from ftplib import FTP
import json
import cgi
import io
import re
from datetime import datetime

FTP_HOST = "127.0.0.1"
FTP_PORT = 2121

def parse_ftp_list_line(line):
    """
    解析 FTP LIST 命令返回的单行，类 Unix 格式:
    drwxr-xr-x  2 owner group 4096 Jan 01 12:34 dirname
    -rw-r--r--  1 owner group 1234 Jan 01 2022 filename.txt
    """
    regex=
r'^([\-ld])([rwx\-]{9})\s+\d+\s+\S+\s+\S+\s+(\d+)\s+(\w{3})\s+(\d{1,2})\
s+([\d:]{4,5}|\d{4})\s+(.+)$'
    m = re.match(regex, line)
    if not m:
        return None
    type_flag = m.group(1)
    size = int(m.group(3))
    month = m.group(4)
    day = int(m.group(5))
    time_or_year = m.group(6)
    name = m.group(7)

    ftype = "dir" if type_flag == "d" else "file"

    current_year = datetime.now().year
    try:
```

```
            if ':' in time_or_year:
                dt_str = f"{month} {day} {current_year} {time_or_year}"
                dt = datetime.strptime(dt_str, "%b %d %Y %H:%M")
            else:
                dt_str = f"{month} {day} {time_or_year} 00:00"
                dt = datetime.strptime(dt_str, "%b %d %Y %H:%M")
            mtime = dt.strftime("%Y-%m-%d %H:%M:%S")
        except Exception:
            mtime = None

        return {"name": name, "type": ftype, "size": size, "modify": mtime}


    class FTPHandler(SimpleHTTPRequestHandler):

        def do_POST(self):
            if self.path == "/list":
                length = int(self.headers.get('Content-Length'))
                data = json.loads(self.rfile.read(length).decode())
                user = data.get("user")
                passwd = data.get("passwd")
                path = data.get("path", ".")

                try:
                    ftp = FTP()
                    ftp.connect(FTP_HOST, FTP_PORT)
                    ftp.login(user, passwd)
                    ftp.cwd(path)

                    lines = []
                    ftp.dir(lines.append)

                    items = []
                    for line in lines:
                        item = parse_ftp_list_line(line)
                        if item:
                            items.append(item)
                    ftp.quit()
                    self.send_response(200)
```

```
                    self.send_header("Content-Type", "application/json")
                    self.end_headers()
                                    self.wfile.write(json.dumps({"files":
items}).encode())
                except Exception as e:
                    self.send_response(200)
                    self.send_header("Content-Type", "application/json")
                    self.end_headers()
                                    self.wfile.write(json.dumps({"error":
str(e)}).encode())

            elif self.path == "/uploadFile":
                content_type = self.headers.get('Content-Type')
                if not content_type:
                    self.send_error(400, "Missing Content-Type")
                    return
                ctype, pdict = cgi.parse_header(content_type)
                if ctype != 'multipart/form-data':
                                self.send_error(400, "Content-Type must be
multipart/form-data")
                    return

                pdict['boundary'] = bytes(pdict['boundary'], "utf-8")
                length = int(self.headers.get('Content-Length'))

                # 使用 FieldStorage 读取上传的文件和表单字段
                fs = cgi.FieldStorage(
                    fp=self.rfile,
                    headers=self.headers,
                    environ={'REQUEST_METHOD': 'POST'},
                    keep_blank_values=True
                )
                user = fs.getvalue("user")
                passwd = fs.getvalue("passwd")
                path = fs.getvalue("path") or "."

                if "file" not in fs:
                    self.send_error(400, "No file uploaded")
                    return
```

```
                file_field = fs["file"]
                filename = file_field.filename
                filecontent = file_field.file.read()

                try:
                    ftp = FTP()
                    ftp.connect(FTP_HOST, FTP_PORT)
                    ftp.login(user, passwd)
                    ftp.cwd(path)
                                        ftp.storbinary(f"STOR  {filename}",
io.BytesIO(filecontent))
                    ftp.quit()

                    self.send_response(200)
                    self.send_header("Content-Type", "application/json")
                    self.end_headers()
                                    self.wfile.write(json.dumps({"success":
True}).encode())
                except Exception as e:
                    self.send_response(200)
                    self.send_header("Content-Type", "application/json")
                    self.end_headers()
                           self.wfile.write(json.dumps({"success": False,
"error": str(e)}).encode())

            elif self.path == "/delete":
                length = int(self.headers.get('Content-Length'))
                data = json.loads(self.rfile.read(length).decode())
                user, passwd = data.get("user"), data.get("passwd")

                try:
                    ftp = FTP()
                    ftp.connect(FTP_HOST, FTP_PORT)
                    ftp.login(user, passwd)

                    # 文件和目录分别删除
                    if "dir" in data:
                        path = data.get("path", ".")
```

```
                    dirname = data.get("dir")
                    ftp.cwd(path)
                    ftp.rmd(dirname)

                else:
                    path = data.get("path", ".")
                    filename = data.get("file")
                    ftp.cwd(path)
                    ftp.delete(filename)

                # 关闭 FTP 连接
                ftp.quit()

                self.send_response(200)
                self.send_header("Content-Type", "application/json")
                self.end_headers()
                            self.wfile.write(json.dumps({"success":
True}).encode())
            except Exception as e:
                self.send_response(200)
                self.send_header("Content-Type", "application/json")
                self.end_headers()
                    self.wfile.write(json.dumps({"success": False,
"error": str(e)}).encode())

        elif self.path == "/rename":
            length = int(self.headers.get('Content-Length'))
            data = json.loads(self.rfile.read(length).decode())
            user, passwd = data.get("user"), data.get("passwd")
            path = data.get("path", ".")
            oldname = data.get("oldname")
            newname = data.get("newname")

            try:
                ftp = FTP()
                ftp.connect(FTP_HOST, FTP_PORT)
                ftp.login(user, passwd)
                ftp.cwd(path)
                ftp.rename(oldname, newname)
```

```
                    ftp.quit()

                    self.send_response(200)
                    self.send_header("Content-Type", "application/json")
                    self.end_headers()
                                self.wfile.write(json.dumps({"success":
True}).encode())
                except Exception as e:
                    self.send_response(200)
                    self.send_header("Content-Type", "application/json")
                    self.end_headers()
                           self.wfile.write(json.dumps({"success": False,
"error": str(e)}).encode())

            elif self.path == "/mkdir":
                length = int(self.headers.get('Content-Length'))
                data = json.loads(self.rfile.read(length).decode())
                user, passwd = data.get("user"), data.get("passwd")
                path = data.get("path", ".")
                dirname = data.get("name")

                try:
                    ftp = FTP()
                    ftp.connect(FTP_HOST, FTP_PORT)
                    ftp.login(user, passwd)
                    ftp.cwd(path)
                    ftp.mkd(dirname)
                    ftp.quit()

                    self.send_response(200)
                    self.send_header("Content-Type", "application/json")
                    self.end_headers()
                                self.wfile.write(json.dumps({"success":
True}).encode())
                except Exception as e:
                    self.send_response(200)
                    self.send_header("Content-Type", "application/json")
                    self.end_headers()
                           self.wfile.write(json.dumps({"success": False,
```

```
"error": str(e)}).encode())

            else:
                self.send_error(404, "Not Found")

        def do_GET(self):
            if self.path.startswith("/download"):
                query = parse_qs(urlparse(self.path).query)
                user = query.get("user", [""])[0]
                passwd = query.get("passwd", [""])[0]
                path = query.get("path", ["."])[0]
                filename = query.get("file", [""])[0]

                try:
                    ftp = FTP()
                    ftp.connect(FTP_HOST, FTP_PORT)
                    ftp.login(user, passwd)
                    ftp.cwd(path)

                    self.send_response(200)
                                        self.send_header("Content-Type",
"application/octet-stream")
                     self.send_header("Content-Disposition", f'attachment;
filename="{filename}"')
                    self.end_headers()

                    ftp.retrbinary(f"RETR {filename}", self.wfile.write)
                    ftp.quit()

                except Exception as e:
                    self.send_error(500, f"FTP 下载失败: {str(e)}")
            else:
                super().do_GET()
    if __name__ == "__main__":
        print("🌐 Web FTP 浏览器启动: http://localhost:8000/index.html")
        server = HTTPServer(("0.0.0.0", 8000), FTPHandler)
        server.serve_forever()
```

附录 E　程序源代码 index.html

ftp 文件管理 web 网页前端

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
    <meta charset="UTF-8" />
    <title>Web FTP 浏览器</title>
    <style>
        body { font-family: Arial, sans-serif; max-width: 900px; margin: 20px auto; }
        .header-container { display: flex; justify-content: space-between; align-items: center; margin-bottom: 20px; }
        h2 { margin: 0; }
        .action-bar { display: flex; justify-content: space-between; align-items: center; margin-bottom: 20px; gap: 10px; }
        #uploadSection, #newFolderSection { padding: 6px; border: 1px solid #ccc; border-radius: 6px; background: #f9f9f9; }
        #reloadBtn { padding: 6px 12px; font-size: 0.9em; cursor: pointer; border: 1px solid #5cb85c; background: #5cb85c; color: white; border-radius: 3px; }
        #reloadBtn:hover { background: #4cae4c; border-color: #449d44; }
        table { width: 100%; border-collapse: collapse; table-layout: fixed; }
        th, td { border: 1px solid #ddd; padding: 8px; text-align: left; word-wrap: break-word; }
        th { background-color: #f2f2f2; user-select: none; }
        tr:hover { background-color: #e6f7ff; }
        a { color: #007acc; text-decoration: none; }
        a:hover { text-decoration: underline; }
        button { padding: 4px 8px; font-size: 0.9em; cursor: pointer; border: 1px solid #d9534f; background: #d9534f; color: white; border-radius: 3px; }
        button:hover { background: #c9302c; border-color: #ac2925; }
        #backBtn { margin-bottom: 10px; cursor: pointer; color: #007acc; display: inline-block; font-weight: bold; }
        #backBtn:hover { text-decoration: underline; }
```

```
            #logoutBtn { padding: 4px 8px; background: #5bc0de; color:
white; border: 1px solid #46b8da; border-radius: 3px; cursor: pointer; }
            #logoutBtn:hover { background: #31b0d5; }
            .green-btn {
                padding: 4px 12px;
                font-size: 0.9em;
                cursor: pointer;
                border: 1px solid #5cb85c;
                background: #5cb85c;
                color: white;
                border-radius: 3px;
                transition: background 0.2s, border-color 0.2s;
            }
            .green-btn:hover {
                background: #4cae4c;
                border-color: #449d44;
            }
        </style>
    </head>
    <body>

    <div class="header-container">
        <h2>Web FTP 文件浏览器</h2>
        <button id="logoutBtn" style="display:none;" onclick="logout()">
退出登录</button>
    </div>

    <div class="action-bar">
        <div id="uploadSection" style="display: flex; flex-direction: row;
gap: 16px; align-items: center; padding: 0; border: none; background:
none;">
            <!-- 上传文件 -->
            <div style="display: flex; align-items: center; gap: 8px;">
                <input type="file" id="uploadFile" style="display:none;"
/>
                                <button  type="button"  class="green-btn"
id="uploadFileBtn">上传文件</button>
            </div>
        </div>
```

```
        <div id="newFolderSection">
            <form id="newFolderForm" style="display: flex; align-items:
center; gap: 8px;">
                <input type="text" id="newFolderName" placeholder="新建文
件夹名" autocomplete="off" />
                    <button type="submit" class="green-btn">新建文件夹
</button>
            </form>
        </div>
        <button id="reloadBtn" onclick="reloadDirectory()">刷新</button>
    </div>

    <span id="backBtn" style="display:none;">🔙 返回上级目录</span>

    <table>
        <thead>
            <tr>
                <th style="width:55%">文件名</th>
                <th style="width:20%">大小</th>
                <th style="width:25%">修改时间</th>
                <th style="width:15%">操作</th>
            </tr>
        </thead>
        <tbody id="filelist">
                <tr><td colspan="4" style="text-align:center;">正在加
载...</td></tr>
        </tbody>
    </table>

    <script>
        let currentPath = ".";
        let username = localStorage.getItem("ftp_user") || "";
        let password = localStorage.getItem("ftp_passwd") || "";

        document.getElementById("backBtn").onclick = () => {
            if (currentPath === "." || currentPath === "") return;
            // 直接去掉最后一个/及其后面的部分
            let idx = currentPath.lastIndexOf("/");
            let newPath = idx > 0 ? currentPath.slice(0, idx) : ".";
```

```
            navigateTo(newPath);
        };

        function reloadDirectory() {
            loadDirectory(currentPath);
        }

        function navigateTo(path) {
            currentPath = normalizePath(path);
            loadDirectory(currentPath);
            history.pushState({}, "", window.location.pathname);
        }

        function normalizePath(path) {
            // 只做简单的多余/去除，不做..和.的处理
            if (!path || path === "") return ".";
            // 去除首尾多余的/
            path = path.replace(/^\/+|\/+$/g, "");
            return path === "" ? "." : path;
        }

        async function loadDirectory(path = ".") {
            currentPath = normalizePath(path);
                      document.getElementById("backBtn").style.display  =
(currentPath !== ".") ? "inline-block" : "none";
                    document.getElementById("logoutBtn").style.display  =
"inline-block";

            if (!username) {
                username = prompt("请输入 FTP 用户名");
                if (!username) {
                    alert("必须输入用户名！");
                    return;
                }
                if (username === "anonymous") {
                    password = "";
                } else {
                    password = prompt("请输入 FTP 密码");
                    if (!password) {
```

```
                        alert("必须输入密码！");
                        return;
                    }
                }
                localStorage.setItem("ftp_user", username);
                localStorage.setItem("ftp_passwd", password);
            }

            // 请求目录
            const res = await fetch(`/list`, {
                method: "POST",
                headers: { "Content-Type": "application/json" },
                body: JSON.stringify({ user: username, passwd: password,
path: currentPath })
            });
            const data = await res.json();

            // 检查无权限，立即弹出登录
                if (data.error && (data.error.includes("无权限") ||
data.error.includes("认证") || data.error.includes("密码") ||
data.error.includes("未登录"))) {
                localStorage.removeItem("ftp_user");
                localStorage.removeItem("ftp_passwd");
                username = prompt("无权限或未登录，请输入 FTP 用户名");
                if (!username) {
                    alert("必须输入用户名！");
                    return;
                }
                if (username === "anonymous") {
                    password = "";
                } else {
                    password = prompt("请输入 FTP 密码");
                    if (!password) {
                        alert("必须输入密码！");
                        return;
                    }
                }
                localStorage.setItem("ftp_user", username);
                localStorage.setItem("ftp_passwd", password);
```

```
                // 重新请求
                return loadDirectory(currentPath);
            }

            const tbody = document.getElementById("filelist");
            tbody.innerHTML = "";

            const files = (data.files || []).filter(item => item.name !==
"." && item.name !== "..");

            if (files.length) {
                files.forEach(item => {
                    const tr = document.createElement("tr");

                    const nameTd = document.createElement("td");
                     const hrefPath = (currentPath === "." ? item.name :
`${currentPath}/${item.name}`);
                    nameTd.innerHTML = item.type === "dir"
                                                        ? `<a href="#"
onclick="navigateTo('${hrefPath}')">🗁 ${item.name}</a>`
                                                        : `<a
href="/download?user=${encodeURIComponent(username)}&passwd=${encodeURI
Component(password)}&path=${encodeURIComponent(currentPath)}&file=${enc
odeURIComponent(item.name)}" target="_blank">🗎 ${item.name}</a>`;
                    tr.appendChild(nameTd);

                    const sizeTd = document.createElement("td");
                                sizeTd.textContent = item.size >= 0 ?
formatSize(item.size) : "—";
                    tr.appendChild(sizeTd);

                    const timeTd = document.createElement("td");
                    timeTd.textContent = item.modify || "—";
                    tr.appendChild(timeTd);

                    const opTd = document.createElement("td");
                    const delBtn = document.createElement("button");
                    delBtn.textContent = "删除";
                    delBtn.onclick = async e => {
```

```
                        e.stopPropagation();
                        if (!confirm(`确定删除 ${item.type === "dir" ? "文
件夹" : "文件"} "${item.name}" 吗？`)) return;
                        const resp = await fetch(`/delete`, {
                            method: "POST",
                                                headers: { "Content-Type":
"application/json" },
                            body: JSON.stringify({
                                user: username,
                                passwd: password,
                                path: currentPath,
                                file: item.name
                            })
                        });
                        const result = await resp.json();
                        if (result.success) {
                            alert("删除成功");
                            loadDirectory(currentPath);
                        } else {
                            alert("删除失败: " + result.error);
                        }
                    };
                    opTd.appendChild(delBtn);
                    tr.appendChild(opTd);

                    tbody.appendChild(tr);
                });
            } else {
                                tbody.innerHTML = `<tr><td colspan="4"
style="text-align:center; color:#999;">空目录或无权限查看</td></tr>`;
            }
        }

        function formatSize(bytes) {
            if (bytes === 0) return '0 B';
            const k = 1024;
            const sizes = ['B', 'KB', 'MB', 'GB', 'TB'];
            const i = Math.floor(Math.log(bytes) / Math.log(k));
            return (bytes / Math.pow(k, i)).toFixed(2) + ' ' + sizes[i];
```

```
        }

        // 文件上传按钮
        document.getElementById("uploadFileBtn").onclick = function() {
            document.getElementById("uploadFile").click();
        };
        document.getElementById("uploadFile").onchange = async function()
{
            const fileInput = this;
            if (!fileInput.files.length) return;
            const file = fileInput.files[0];
            const formData = new FormData();
            formData.append("file", file);
            formData.append("user", username);
            formData.append("passwd", password);
            formData.append("path", currentPath);
            formData.append("filename", file.name);

            const res = await fetch("/uploadFile", { method: "POST", body:
formData });
            const data = await res.json();
                  if (!data.success) alert(`上传 ${file.name} 失败:
${data.error}`);
            else alert("上传完成");
            fileInput.value = ""; // 重置
            loadDirectory(currentPath);
        };

             document.getElementById("newFolderForm").onsubmit = async
function(e) {
            e.preventDefault();
                                              const     folderName     =
document.getElementById("newFolderName").value.trim();
            if (!folderName) return alert("请输入文件夹名称");
            const res = await fetch("/mkdir", {
                method: "POST",
                headers: { "Content-Type": "application/json" },
                body: JSON.stringify({ user: username, passwd: password,
path: currentPath, name: folderName })
```

```
            });
            const result = await res.json();
            if (result.success) {
                alert("文件夹创建成功");
                loadDirectory(currentPath);
            } else {
                alert("创建失败: " + result.error);
            }
        };

        function logout() {
            localStorage.removeItem("ftp_user");
            localStorage.removeItem("ftp_passwd");
            alert("已退出登录");
            window.location.reload();
        }

        loadDirectory(currentPath);
    </script>

    </body>
    </html>
```

附录 F　程序源代码 crawler.c

基于 socket 的简单 C 语言静态网页爬虫

```
    // 基于 Socket+OpenSSL 的简易 HTTPS 网络爬虫（抓取文本和图片，保存到 download 文件夹）
    // 编译：gcc -o crawler_https crawler_https.c -lssl -lcrypto
    // 运行：./crawler https://example.com

    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    #include <unistd.h>
    #include <sys/socket.h>
    #include <arpa/inet.h>
    #include <netdb.h>
    #include <sys/stat.h>
    #include <fcntl.h>
    #include <openssl/ssl.h>
    #include <openssl/err.h>

    #define BUFFER_SIZE 8192

    void parse_url(const char *url, char *host, char *path, int *port, int *is_https) {
        *port = 80;
        *is_https = 0;
        if (strncmp(url, "https://", 8) == 0) {
            *is_https = 1;
            *port = 443;
            url += 8;
        } else if (strncmp(url, "http://", 7) == 0) {
            url += 7;
        }
        const char *slash = strchr(url, '/');
        if (slash) {
            strncpy(host, url, slash - url);
            host[slash - url] = '\0';
            strcpy(path, slash);
```

```
        } else {
            strcpy(host, url);
            strcpy(path, "/");
        }
        char *colon = strchr(host, ':');
        if (colon) {
            *port = atoi(colon + 1);
            *colon = '\0';
        }
    }

    int connect_host(const char *host, int port) {
        struct hostent *he = gethostbyname(host);
        if (!he) return -1;
        int sock = socket(AF_INET, SOCK_STREAM, 0);
        struct sockaddr_in addr = {0};
        addr.sin_family = AF_INET;
        addr.sin_port = htons(port);
        memcpy(&addr.sin_addr, he->h_addr_list[0], he->h_length);
        if (connect(sock, (struct sockaddr*)&addr, sizeof(addr)) < 0) {
            close(sock);
            return -1;
        }
        return sock;
    }

    void save_file(const char *host, const char *fname, const char *data,
size_t len) {
        char dirpath[256];
        sprintf(dirpath, "download/%s", host);
        mkdir(dirpath, 0755); // 创建主机子目录

        char fullpath[512];
        sprintf(fullpath, "%s/%s", dirpath, fname);
        FILE *f = fopen(fullpath, "wb");
        if (f) {
            fwrite(data, 1, len, f);
            fclose(f);
            printf("保存: %s\n", fullpath);
```

```
        }
    }

    int ssl_send_recv(SSL *ssl, const char *req, char **out_buf, size_t
*out_size) {
        char buf[BUFFER_SIZE];
        int header = 1, n;
        size_t file_size = 0, cap = BUFFER_SIZE * 4;
        char *file_data = malloc(cap);
        SSL_write(ssl, req, strlen(req));
        while ((n = SSL_read(ssl, buf, sizeof(buf))) > 0) {
            if (header) {
                char *p = strstr(buf, "\r\n\r\n");
                if (p) {
                    size_t hlen = p + 4 - buf;
                    memcpy(file_data, buf + hlen, n - hlen);
                    file_size = n - hlen;
                    header = 0;
                }
            } else {
                    if (file_size + n > cap) { cap *= 2; file_data =
realloc(file_data, cap); }
                memcpy(file_data + file_size, buf, n);
                file_size += n;
            }
        }
        *out_buf = file_data;
        *out_size = file_size;
        return 0;
    }

    void fetch_url(const char *url, int is_img) {
        char host[128], path[256];
        int port, is_https;
        parse_url(url, host, path, &port, &is_https);
        int sock = connect_host(host, port);
        if (sock < 0) { puts("连接失败"); return; }

        char req[512];
```

```
        snprintf(req, sizeof(req),
            "GET %s HTTP/1.0\r\nHost: %s\r\nConnection: close\r\n\r\n",
            path, host);

        char *file_data = NULL;
        size_t file_size = 0;
        if (!is_https) {
            // HTTP 方式
            char buf[BUFFER_SIZE];
            int header = 1, n;
            size_t cap = BUFFER_SIZE * 4;
            file_data = malloc(cap);
            send(sock, req, strlen(req), 0);
            while ((n = recv(sock, buf, sizeof(buf), 0)) > 0) {
                if (header) {
                    char *p = strstr(buf, "\r\n\r\n");
                    if (p) {
                        size_t hlen = p + 4 - buf;
                        memcpy(file_data, buf + hlen, n - hlen);
                        file_size = n - hlen;
                        header = 0;
                    }
                } else {
                       if (file_size + n > cap) { cap *= 2; file_data =
realloc(file_data, cap); }
                    memcpy(file_data + file_size, buf, n);
                    file_size += n;
                }
            }
        } else {
            // HTTPS 方式
            SSL_library_init();
            OpenSSL_add_all_algorithms();
            SSL_load_error_strings();
            const SSL_METHOD *method = TLS_client_method();
            SSL_CTX *ctx = SSL_CTX_new(method);
            SSL *ssl = SSL_new(ctx);
            SSL_set_fd(ssl, sock);
            if (SSL_connect(ssl) <= 0) {
```

```
                SSL_free(ssl); SSL_CTX_free(ctx); close(sock); puts("SSL
连接失败"); return;
            }
            ssl_send_recv(ssl, req, &file_data, &file_size);
            SSL_shutdown(ssl); SSL_free(ssl); SSL_CTX_free(ctx);
        }
        close(sock);
        // 生成文件名
        const char *fname = strrchr(path, '/');
        if (!fname || strlen(fname) <= 1) fname = is_img ? "index.jpg" :
"index.html"; else fname++;
        save_file(host, fname, file_data, file_size);
        free(file_data);
    }

    void mkdir_download() { mkdir("download", 0755); }

    void find_and_fetch_imgs(const char *html, size_t len, const char
*base_url) {
        const char *p = html;
        while ((p = strstr(p, "<img"))) {
            const char *src = strstr(p, "src=");
            if (!src) break;
            src += 4;
            while (*src == ' ' || *src == '\'' || *src == '"') src++;
            char img_url[512] = {0};
            int j = 0;
            while (*src && *src != '"' && *src != '\'' && *src != ' ' && *src !=
'>') {
                img_url[j++] = *src++;
            }
            img_url[j] = '\0';
            char full_url[512];
             if (strncmp(img_url, "http://", 7) == 0 || strncmp(img_url,
"https://", 8) == 0) {
                strcpy(full_url, img_url);
            } else if (img_url[0] == '/') {
                char host[128], path[256]; int port, is_https;
                parse_url(base_url, host, path, &port, &is_https);
```

```
                        sprintf(full_url, is_https ? "https://%s%s" :
"http://%s%s", host, img_url);
            } else {
                sprintf(full_url, "%s/%s", base_url, img_url);
            }
              if (strstr(img_url, ".jpg") || strstr(img_url, ".png") ||
strstr(img_url, ".jpeg") || strstr(img_url, ".gif")) {
                printf("图片 URL: %s\n", full_url);
                fetch_url(full_url, 1);
            }
            p = src;
        }
    }

    int main(int argc, char *argv[]) {
        if (argc < 2) {
            printf("用法: %s <网址>\n", argv[0]);
            return 1;
        }
        mkdir_download();
        char host[128], path[256]; int port, is_https;
        parse_url(argv[1], host, path, &port, &is_https);
        int sock = connect_host(host, port);
        if (sock < 0) { puts("连接失败"); return 1; }
        char req[512];
        snprintf(req, sizeof(req),
            "GET %s HTTP/1.0\r\nHost: %s\r\nConnection: close\r\n\r\n",
            path, host);
        char *html = NULL;
        size_t html_size = 0;
        if (!is_https) {
            // HTTP
            char buf[BUFFER_SIZE];
            int n, header = 1;
            size_t cap = BUFFER_SIZE * 10;
            html = malloc(cap);
            send(sock, req, strlen(req), 0);
            while ((n = recv(sock, buf, sizeof(buf), 0)) > 0) {
                if (header) {
```

```
                    char *p = strstr(buf, "\r\n\r\n");
                    if (p) {
                        size_t hlen = p + 4 - buf;
                        memcpy(html, buf + hlen, n - hlen);
                        html_size = n - hlen;
                        header = 0;
                    }
                } else {
                    if (html_size + n > cap) { cap *= 2; html = realloc(html,
cap); }
                    memcpy(html + html_size, buf, n);
                    html_size += n;
                }
            }
        } else {
            // HTTPS
            SSL_library_init();
            OpenSSL_add_all_algorithms();
            SSL_load_error_strings();
            const SSL_METHOD *method = TLS_client_method();
            SSL_CTX *ctx = SSL_CTX_new(method);
            SSL *ssl = SSL_new(ctx);
            SSL_set_fd(ssl, sock);
            if (SSL_connect(ssl) <= 0) {
                SSL_free(ssl); SSL_CTX_free(ctx); close(sock); puts("SSL
连接失败"); return 1;
            }
            char buf[BUFFER_SIZE];
            int n, header = 1;
            size_t cap = BUFFER_SIZE * 10;
            html = malloc(cap);
            SSL_write(ssl, req, strlen(req));
            while ((n = SSL_read(ssl, buf, sizeof(buf))) > 0) {
                if (header) {
                    char *p = strstr(buf, "\r\n\r\n");
                    if (p) {
                        size_t hlen = p + 4 - buf;
                        memcpy(html, buf + hlen, n - hlen);
                        html_size = n - hlen;
```

```
                        header = 0;
                    }
                } else {
                    if (html_size + n > cap) { cap *= 2; html = realloc(html,
cap); }
                    memcpy(html + html_size, buf, n);
                    html_size += n;
                }
            }
            SSL_shutdown(ssl); SSL_free(ssl); SSL_CTX_free(ctx);
        }
        close(sock);
        save_file(host, "index.html", html, html_size);
        find_and_fetch_imgs(html, html_size, argv[1]);
        free(html);
        return 0;
    }
```