基于 Socket 的 FTP 服务器与客户端实现

Zeng Guanyang

1 简介

本项目实现了完整的 FTP 系统，包含多线程 C 语言服务器和 Python 客户端。服务器支持并发连接、异步传输、主被动模式、断点续传、文件锁等功能，可在 Linux/Windows 平台运行。客户端提供 CLI 和 GUI 界面，支持同步和异步传输。实现了所有 Optional 功能。

代码量：**服务器**：约 **13000** 行；**客户端**：约 **4000** 行。

2 核心功能与支持的 FTP 命令

2.1 服务器端实现（C 语言）

模块设计：

- **server.c：**服务器核心，负责监听套接字管理、连接上限控制、线程池和会话生命周期管理
- **session.c：**会话状态管理，维护认证状态、当前目录、数据连接模式、续传偏移、重命名状态、统计信息等
- **command.c + handler.c：**命令注册与调度，由 handler 实现所有 FTP 命令的具体逻辑和权限校验
- **auth.c：**用户认证模块，加载 users.db（可选），支持密码哈希校验（简单实现）和权限管理
- **filesys.c：**跨平台文件系统抽象层，提供路径操作、目录遍历、文件元信息查询等功能
- **transfer.c：**数据传输引擎，管理数据通道、支持 ASCII/Binary 模式、断点续传、传输中断等
- **filelock.c：**文件锁管理器，实现基于路径的读写锁，保证并发访问安全性
- **network.c：**网络抽象层，统一 POSIX 和 Winsock API，提供简单的超时控制、非阻塞模式等
- **logger.c：**日志系统，支持分级（彩色）输出、时间戳、文件名/行号/函数名记录

支持的 FTP 命令：

- **连接管理：**USER, PASS, QUIT, REIN
- **传输模式：**PORT, PASV, TYPE, MODE, STRU
- **文件操作：**RETR, STOR, APPE, DELE, REST
- **目录操作：**LIST, NLST, CWD, CDUP, PWD, MKD, RMD
- **重命名：**RNFR, RNTO
- **传输控制：**ABOR
- **信息查询：**SYST, FEAT, SIZE, MDTM, NOOP

关键特性：

- **用户认证系统：**支持多用户管理（users.db），密码哈希校验，权限控制（读/写/删除/重命名/目录管理/管理员），匿名用户为兼容作业具有管理员权限
- **安全隔离：**所有路径相对配置根目录进行归一化验证，确保用户无法访问根目录外的文件
- **并发文件锁：**基于路径的读写锁机制，以支持多客户端同时下载，写操作互斥，防止竞态条件
- **断点续传：**REST 命令配合 RETR/STOR 实现从指定偏移量继续传输
- **中断控制：**ABOR 命令支持通过 OOB 即时中断正在进行的传输
- **会话统计：**记录上传/下载字节数、文件数量、命令总数和在线时长

2.2 客户端实现（Python）

架构：FTPClient 主控，ControlConnection/DataConnection 管理连接，CommandRegistry 实现命令注册模式，TransferManager 管理异步传输。实现 CLI 和 Tkinter GUI 界面。

命令：USER, PASS, PASV, PORT, RETR, STOR, APPE, REST, ABOR, LIST, NLST, CWD, CDUP, PWD, MKD, RMD, DELE, RNFR, RNTO, TYPE, QUIT 等

特性：同步/异步传输、GUI 断点续传、进度显示、CLI 便捷命令（upload, download, ls 等）

3 关键实现细节

3.1 服务器端核心实现

1. 多线程会话管理

服务器在 `server.c` 的 `client_thread()` 函数中处理每个客户端连接。主要流程：

- 通过 `pthread_mutex` 保护的全局计数器 `g_current_connections` 实现并发连接数限制
- 调用 `net_set_oob_inline()` 配置控制套接字接收带外数据（用于 ABOR 命令）
- 发送 220 欢迎消息后，进入命令循环：使用 `net_receive_line()` 接收命令，`net_has_urgent_data()` 检测紧急数据（但实际和正常信息流统一处理）
- 命令解析后通过 `command_dispatch()` 分发到对应的 handler 函数执行
- 会话结束时调用 `session_destroy()`，等待传输线程退出并递减连接计数

2. 异步数据传输线程

handler 函数调用 `session_start_transfer_thread()` 启动传输线程：

- 分配 `transfer_params_t` 结构体，包含传输类型（上下传等）、文件路径、偏移量等参数，转移文件锁
- 调用 `pthread_create` 创建线程
- handler 在传输启动前前发送 150 响应码，传输线程在后台执行实际 I/O
- 线程函数 `transfer_thread_func()` 调用 `transfer_send_file()` 等对应版本执行传输
- 传输完成后线程设置 `transfer_result` 并发送响应，最后清理数据连接和文件锁
- 客户端线程利用 `pthread_join()` 等待线程结束，ABOR 命令可以通过设置标志位、关闭数据传输来中止传输线程

3. 被动模式竞态问题处理 (Thanks to Claude-Sonnet-4.5)

被动模式下，服务器需要在监听套接字上 `accept()` 客户端连接。实现中遇到并解决了多个竞态问题：

问题 1：高并发下（500 个）`select()` 显示可读但 `accept()` 返回 `EWOULDBLOCK/EAGAIN`

原因：`select()` 和 `accept()` 之间存在时间窗口，期间连接可能被撤销或被其他进程接受

解决方案：

- 将监听套接字设为非阻塞模式（`net_set_nonblocking()`）
- 实现重试机制：`accept()` 失败时检测是否为 `would_block` 错误，若是则短暂休眠（10ms）后重试，最多重试 3 次
- 每次重试前检查会话状态（`should_quit`）和套接字是否仍然有效，避免在已销毁的会话上操作

问题 2：会话销毁时主线程阻塞在 `select()` 中

原因：`select()` 在等待连接时持有会话锁会导致死锁，但释放锁后无法安全检测套接字是否被关闭

解决方案：

- 在调用 `select()` 前释放 `pthread_mutex` 锁，允许其他线程（如会话清理）继续执行

- 保存监听套接字的文件描述符 `listen_sock`，`select()` 返回后重新获取锁
- 验证 `session->data_listen_socket` 是否仍等于保存的 `listen_sock`，若不同说明套接字已被关闭/重建，立即中止
- 当套接字被关闭时，`select()` 会返回错误，正常处理错误并返回即可

4. 文件锁

`filelock.c` 使用链表维护全局锁表 `g_file_lock_entries`，每个节点 `file_lock_entry_t` 包含：

- `path`：文件路径；`readers/writers`：当前持有读/写锁的数量；`waiting_writers`：等待写锁的数量
- `pthread_cond_t cond`：条件变量实现锁等待队列
- `file_lock_acquire_shared()`：获得共享锁，等待无排它锁定，有非阻塞版本
- `file_lock_acquire_exclusive()`：获得排它锁，等待无其他锁定，同样由有非阻塞版本
- 释放锁时利用 `pthread_cond_broadcast()` 唤醒所有等待线程；若没有线程持有锁则移除节点

5. ABOR 中断机制

- 控制连接启用 OOB inline 模式，ABOR 命令可作为紧急数据发送
- 主线程检测到 ABOR 后设置 `session->transfer_should_abort = 1`
- 传输线程在每次 I/O 循环前调用 `session_should_abort_transfer()` 检查标志位
- 若需中断，线程设置 `transfer_result = TRANSFER_STATUS_ABORTED` 并跳出循环
- 主线程可调用 `session_close_data_connection()` 关闭数据套接字，使阻塞 I/O 立即返回
- 传输结束时根据 `transfer_result` 发送对应响应码，如果传输正在执行，handler 提前发送 426 响应码

3.2 客户端核心实现

1. 连接管理

`ControlConnection` 继承 `BaseConnection`，使用 `threading.RLock` 保护接收操作：

- `recv_line()`：逐字节接收直到遇到 CRLF（`\r\n`）
- `recv_multiline()`：检测首行是否为多行响应（码-消息），循环接收直到遇到”码 < 空格 >” 结束标记
- `DataConnection` 支持主被动模式：`setup_passive()` 连接到服务器指定的地址；`setup_active()` 创建监听套接字并通过 `accept()` 等待服务器连接

2. 命令注册与 Handler 模式

`CommandRegistry` 维护 `commands` 字典（命令名 →Handler 类）：

- 注册：`register(command_name, handler_class)` 将 Handler 类关联到命令名
- 执行：`execute(command, *args, **kwargs)` 实例化对应 Handler 并调用 `execute()` 方法
- 这样新增命令只需定义继承 `CommandHandler` 的子类，实现 `execute()`，无需修改核心逻辑

3. 传输管理器

`TransferManager` 使用 `threading` 管理异步传输：

- 维护 `transfers` 字典（transfer_id→Transfer 对象），每个 `Transfer` 包含状态、进度、线程等信息
- `Transfer` 对象使用 `pause_event` 和 `cancel_event` 实现暂停/取消功能
- 传输线程在循环中检查 `pause_event`（暂停则阻塞）和 `cancel_event`（取消则退出）
- 支持进度回调：`progress_callback(bytes_transferred, total_size)` 定期报告进度
- 因（目前）FTP 单数据连接限制，`max_concurrent` 实际限制为 1，避免数据连接冲突

4. GUI 实现与测试兼容

通过在 `main.py` 中加入参数控制实现兼容，比如 `-g` 进入图形化界面，默认进入 CLI 模式。CLI 模式下，不使用 `-v` 则进入测试兼容版本，避免交互提示。GUI 实现借助了 Claude-Sonnet-4.5 的帮助