操作系统课程大作业实验报告：简化学术评审系统

1. 项目背景与目标

现代学术会议或期刊广泛使用在线评审系统，例如 OpenReview 和 EasyChair。课程项目旨在通过实现一个“简化但完整”的评审系统，帮助理解操作系统核心概念，尤其是：

- **文件系统**：superblock、inode 表、数据块、空闲位图、目录结构与路径解析
- **并发访问**：多客户端并发访问的服务端处理
- **缓存**：可配置的 **LRU 块缓存**，并输出命中、缺失及替换统计
- **网络通信**：Client-Server 架构，通过网络协议完成所有操作
- **鉴权与权限控制**：登录、会话即 Session、基于角色的权限检查

本项目采用 **C++17** 实现 Client-Server 架构，客户端为 **CLI** 即命令行界面；服务器端维护一个**自定义简化文件系统**即 VFS，将论文、评审等数据持久化到后端文件 `data.fs` 中。

---

2. 需求分析——对照课程指引

课程指引关键要求摘要如下：

- **C/S 架构**：所有操作由 client 发起，server 执行并访问服务端文件系统；支持多客户端并发
- **四种角色**：Author、Reviewer、Editor 和 Admin，不同权限与命令
- **文件系统**：superblock、inode table、data blocks、free bitmap；多级目录；文件创建、删除、读写及路径解析
- **LRU 块缓存**：容量可配置，输出 hit、miss 和 replacement 等统计
- **备份**：支持创建、列出及恢复，可选扩展为快照式备份
- **协议**：格式清晰、可扩展，支持文本或二进制格式

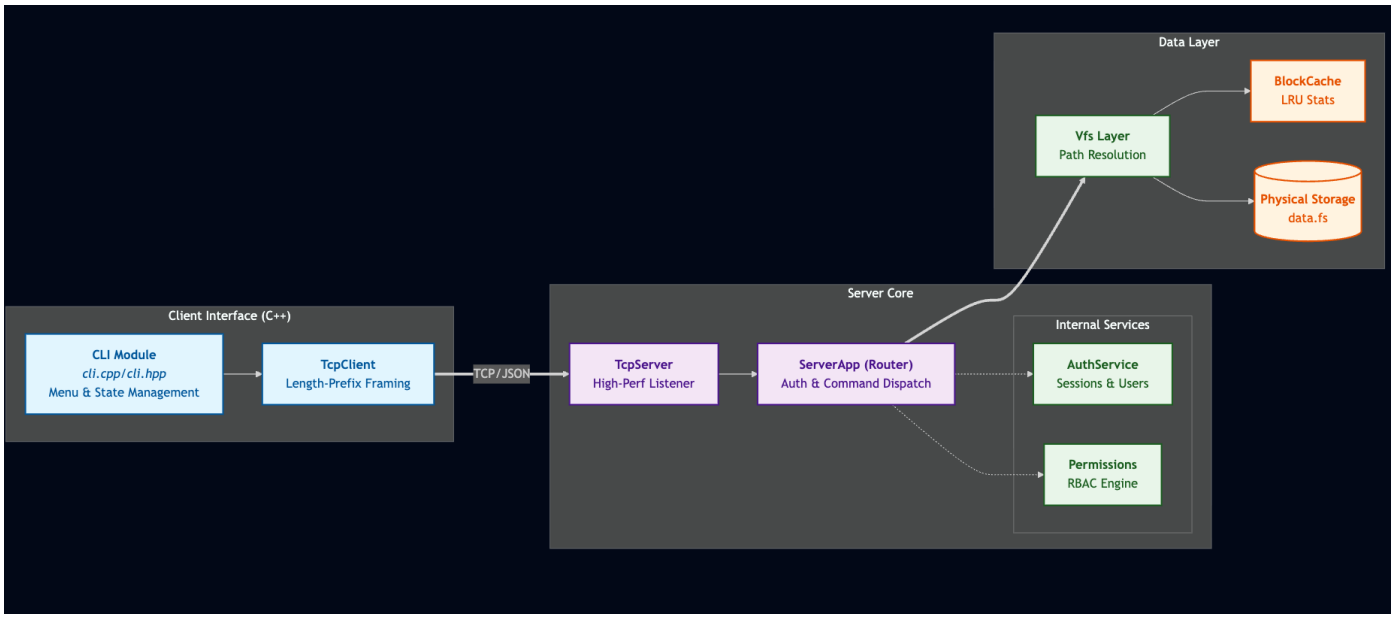
---

3. 系统总体设计

3.1 架构概览

系统分为客户端 CLI 与服务器端 ServerApp：

- **Client** 即 CLI：解析用户输入命令，封装为统一 JSON 协议，通过 TCP 发送给 server，输出响应 JSON；登录成功后自动携带 `sessionId`；本地维护“当前目录”用于 `LIST` 的默认参数。
- **Server** 即 ServerApp：接收消息、解析命令、校验会话、执行权限检查，调用自研 `Vfs` 完成数据落盘，并返回统一响应结构。



3.2 代码模块划分

项目主要模块与职责，对应 `src/` 目录：

- **`src/common/`**：公共类型与协议
  - `types.hpp`：`UserId`、`PaperId`、`Role` 和 `Credentials` 等基础类型
  - `protocol.hpp`：消息 envelope、统一命令 `Command`、序列化与反序列化、成功或错误响应构造
- **`src/domain/`**：领域模型、权限与认证
  - `auth.hpp/.cpp`：内存用户表与会话即 Session 管理，目前为教学演示用，后续可持久化到 VFS
  - `permissions.hpp/.cpp`：`Role -> Permission` 的权限矩阵
  - `paper.hpp` / `review.hpp` / `user.hpp`：领域对象定义
- **`src/server/`**：服务端实现
  - `server_app.hpp/.cpp`：命令路由、鉴权、权限检查、论文流程、FS 命令
  - `filesystem/`：自定义文件系统，包含 superblock、inode、bitmap、目录、路径解析及 LRU cache
  - `net/`：TCP Server，使用长度前缀和 JSON 数据
- **`src/client/`**：客户端实现
  - `cli.hpp/.cpp`：交互式 CLI，包含按角色的数字菜单向导
  - `net/`：TCP Client，负责一次请求与响应

---

4. 通信协议设计：可扩展的 JSON Envelope 与 Command

4.1 消息封装即 Message Envelope

通信消息由 `MessageType + payload` 即 JSON 组成，并统一序列化为 JSON 字符串：

- `type`：`"CommandRequest" | "CommandResponse" | "Error" | ...`

- `payload`：请求或响应数据即 JSON object

传输层使用 **4 字节网络序长度前缀** + **N 字节 JSON** 的 framing，避免粘包或拆包问题。

```
> LOGIN admin admin
{
  "data": {
    "role": "Admin",
    "sessionId": "sess-1-1",
    "userId": 1,
    "username": "admin"
  },
  "ok": true
}
当前角色：Admin，输入 ROLE_HELP 查看详细可用命令。
> [INFO ] Send request: {"args":["admin","admin"],"cmd":"LOGIN","rawArgs":"admin admin","sessionId":null} to 127.0.0.1:5555
[INFO ] Received response from server
[INFO ] Logged in as admin (Admin)
```

4.2 统一命令 Command

在 `payload` 内，统一使用命令结构：

- `cmd`：命令名，例如 PING、LOGIN、LIST_PAPERS
- `args`：按空格切分的参数数组
- `rawArgs`：原始参数串，用于 WRITE、REVIEW 等需要保留空格的场景
- `sessionId`：登录后携带，用于鉴权

该设计的优点：

- 命令格式明确，扩展新命令只需在 server 端路由增加分支
- 对 CLI 友好：既支持纯命令行输入，也能做“数字菜单向导”生成命令
- 对网络友好：JSON 可读、易调试、易扩展字段

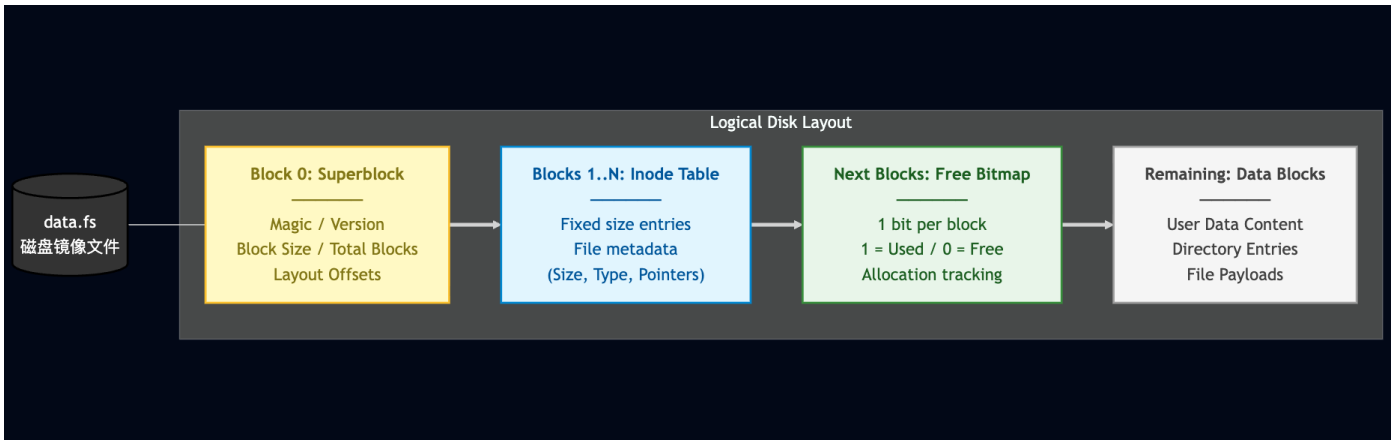
---

5. 自定义文件系统即 VFS 设计

5.1 磁盘布局即 data.fs

服务器在启动时挂载 `data.fs`，若不存在或 magic 不匹配则自动格式化。当前实现采用固定布局，以满足指引中的 superblock、inode、bitmap 和 data blocks 要求：

- **Block Size**：4096 bytes
- **Total Blocks**：1024 blocks
- **Superblock**：block 0
- **Inode Table**：固定若干 blocks，当前为 8 blocks
- **Free Bitmap**：固定若干 blocks，当前为 1 block，用于位图管理 data blocks
- **Data Blocks**：其余 blocks



5.2 Inode 与数据块管理

- 每个 inode 记录：`id`、是否目录、文件大小、若干 **direct blocks**，当前仅支持直接块数组
- 数据块分配：使用 **free bitmap**，扫描第一个空闲 bit 并置位，释放时清零
- 读写块：通过 `readBlock/writeBlock` 访问 `data.fs` 指定 block offset

当前限制，已在后续计划中安排增强：

- 文件大小受限于 `MaxDirectBlocks * blockSize`，目前无间接块或多级索引
- 目录采用“单块目录”，仅使用 `directBlocks[0]` 存储目录项

5.3 目录结构与路径解析

- 支持 **多级路径解析**：将 `/a/b/c` 切分组件逐级在目录项中查找 inode
- 目录项（DirEntry）为定长结构：`inodeId + name[60]`
- 支持：`MKDIR <path>`、`LIST /`、创建或删除文件与目录，删除目录要求为空目录

5.4 LRU 块缓存：可配置与统计

`BlockCache` 为内存态 LRU 缓存：

- 容量 `capacity` 可通过服务器启动参数或环境变量配置，见第 7 节
- 统计项：`hits/misses/replacements/entries/capacity`
- server 命令 `VIEW_SYSTEM_STATUS` 会回传当前缓存统计，便于演示与评估

```
---------------
> VIEW_SYSTEM_STATUS
{
  "data": {
    "blockCache": {
      "capacity": 64,
      "entries": 2,
      "hits": 0,
      "misses": 2,
      "replacements": 0
    },
    "papers": 0,
    "reviews": 0,
    "sessions": 1,
    "users": 5
  },
  "ok": true
}
```

---

6. 鉴权与权限控制设计

6.1 会话 Session 即当前实现

`AuthService` 维护：

- `usersByName_`：用户名映射用户信息，当前使用明文密码，仅用于教学演示
- `sessionsById_`：sessionId 映射 Session，登录后生成 `sess-<uid>-<counter>`

客户端登录后保存 `sessionId`，后续请求自动携带，服务端统一校验会话有效性。

6.2 权限模型, 基于角色的访问控制

`Permission` 抽象高层动作，包括上传论文、分配审稿、查看状态、管理用户及备份等，通过 `hasPermission(role, permission)` 决定是否允许执行。

本实现将权限检查集中在 server 命令处理阶段，做到：

- **未登录拒绝**：如论文操作、系统状态等
- **角色限制**：Author、Reviewer、Editor 和 Admin 各自只允许对应操作

---

7. 业务流程与数据落盘设计：论文评审

7.1 数据目录约定，存储在 VFS 内

论文相关数据全部写入 `Vfs`，例如：

- `/papers/`：论文根目录
- `/papers/<id>/meta.txt`：论文元信息，包含 id、authorId、status 和 title
- `/papers/<id>/content.txt`：论文正文
- `/papers/<id>/reviewers.txt`：审稿人列表，其中 userId 每行一个
- `/papers/<id>/reviews/<reviewerId>.txt`：审稿意见文件，内容为 decision 和 comments
- `/system/next_paper_id`：自增 ID 计数，采用简单实现

```
LIST_PAPERS
[INFO ] Send request: {"args":[],"cmd":"LIST_PAPERS","sessionId":"sess-2-2"} to 127.0.0.1:5555
[INFO ] Received response from server
{
  "data": {
    "papers": [
      {
        "authorId": 2,
        "id": 1,
        "status": "Submitted",
        "title": "How"
      }
    ]
  },
  "ok": true
}
```

7.2 关键命令

- **Author**
  - `SUBMIT <Title> <Content...>`：提交新论文，其中 title 当前建议不含空格
  - `LIST_PAPERS`：查看自己的论文
  - `GET_PAPER <PaperID>`：查看论文内容
  - `LIST_REVIEWS <PaperID>`：查看自己论文的评审意见或状态
- **Reviewer**
  - `LIST_PAPERS`：仅列出分配给自己的论文，通过 reviewers.txt 判断
  - `GET_PAPER <PaperID>`：仅允许查看被分配论文

    - `REVIEW <PaperID> <Decision> <Comments...>`：提交评审，Decision 可选 ACCEPT、REJECT、MINOR 或 MAJOR
- **Editor**
    - `ASSIGN <PaperID> <ReviewerUsername>`：分配审稿人
    - `DECISION <PaperID> <ACCEPT/REJECT>`：最终决定，更新 meta 状态
    - 便捷命令：`ASSIGN_REVIEWER` / `VIEW_REVIEW_STATUS` / `MAKE_FINAL_DECISION`，内部转成基础命令
- **Admin**
    - `MANAGE_USERS LIST/ADD/REMOVE/UPDATE_ROLE/RESET_PASSWORD ...`：用户管理，当前为内存态
    - `VIEW_SYSTEM_STATUS`：统计 users、sessions、papers、reviews 以及 block cache 指标
    - `BACKUP` / `RESTORE`：当前为接口与权限框架已就绪，实际备份落地见第 9 节计划

```
ROLE_HELP
当前用户: author  角色: Author
=== Author 指引 ===
命令:
  SUBMIT <Title> <Content...>            - 上传新论文
    Title: 不含空格（建议用下划线代替）
  LIST_PAPERS                            - 查看我的论文列表
  GET_PAPER <PaperID>                    - 查看论文详情（含正文）
  LIST_REVIEWS <PaperID>                 - 查看评审意见/状态（仅限自己的论文）
[Author 数字菜单]
  1）提交新论文
  2）查看我的论文列表
  3）查看论文详情
  4）查看评审意见/状态
  （直接输入数字开始操作；也可以直接输入原始命令）
----------------
>
```

---

8. 当前进展总结

8.1 已完成及可演示

- **Client-Server 基础通信**：TCP 加长度前缀及 JSON 序列化，支持请求-响应
- **统一命令协议**：`CommandRequest/Response` 加 `Command{cmd,args,rawArgs,sessionId}`
- **鉴权与会话**：`LOGIN` 加 session 自动携带与校验
- **权限控制**：RBAC 权限矩阵已实现，并在关键命令处校验
- **自定义文件系统核心结构**：superblock、inode table、free bitmap 及 data blocks 的基本布局与读写
- **路径解析与目录操作**：支持多级路径解析，`MKDIR/LIST/WRITE/READ/RM/RMDIR`

- **LRU 块缓存**：容量可配置、统计可查询，通过 `VIEW_SYSTEM_STATUS`
- **论文评审主流程之简化版**：提交论文、分配审稿人、提交评审、查看评审、给出最终决定

8.2 部分完成

- **目录能力**：目前目录为“单块目录”，目录项数量受 block size 限制；创建目录不自动递归创建父目录
- **文件能力**：仅 direct blocks；文件大小受限；写入为覆盖式写入
- **用户与会话存储**：用户与会话目前在内存中，未持久化到 VFS

8.3 未完成及需补齐

- **多客户端并发**：当前 TCP server 为阻塞式“一次 accept 一个连接、处理一条消息就关闭”，并且每轮 `serveOnce` 重新创建、绑定并监听 socket；尚未达到“并发多客户端”的要求
- **备份与恢复落地**：`BACKUP/RESTORE` 已有命令入口与权限框架，但未实现实际快照、复制及恢复逻辑

---

9. 构建运行与演示步骤

9.1 构建

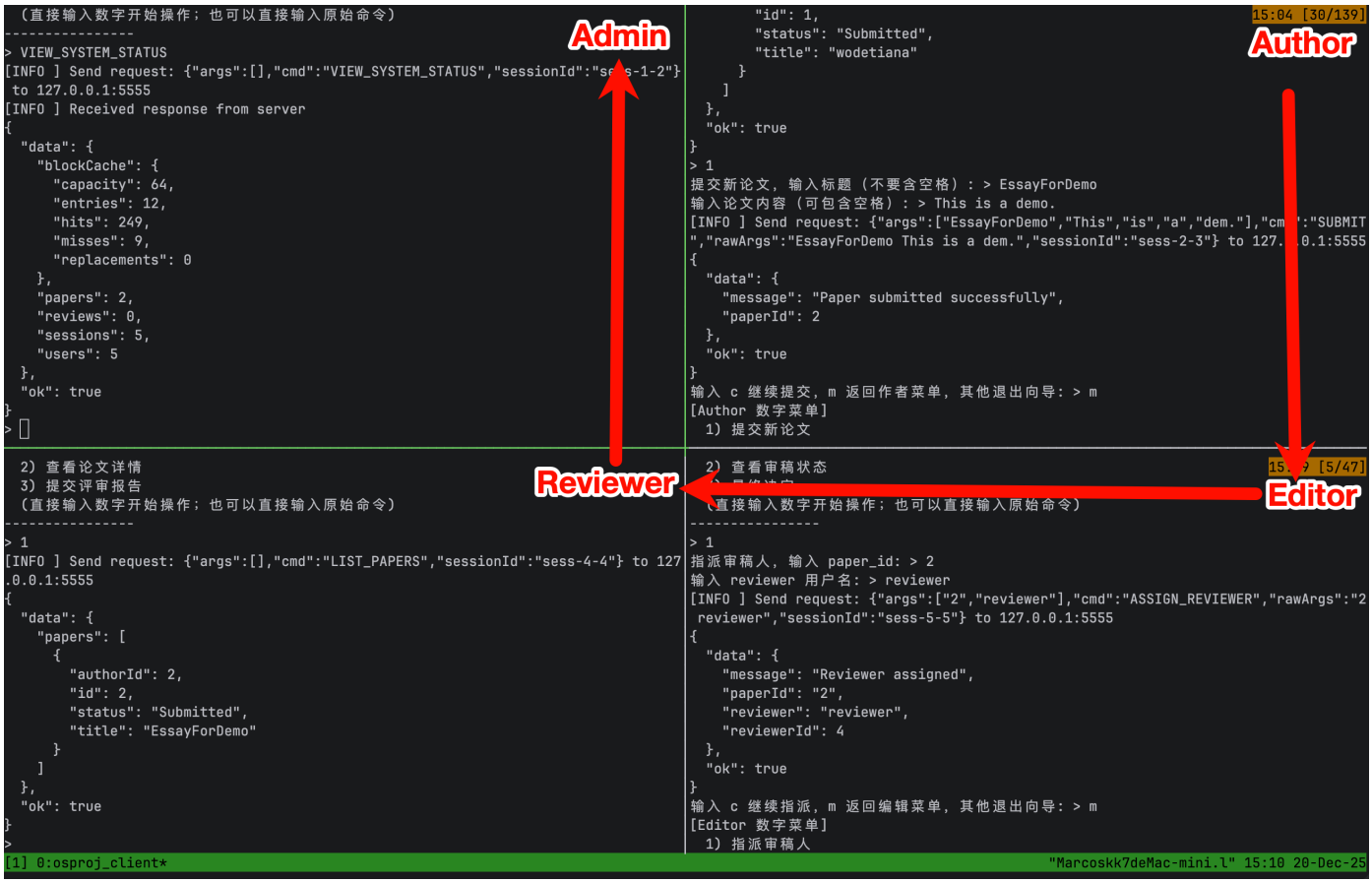
项目使用 CMake 构建，根目录为 `CMakeLists.txt`，子模块位于 `src/CMakeLists.txt`：

- 生成产物：
  - `osproj_server`
  - `osproj_client`

```
✘ marcoskk7@Marcoskk7deMac-mini /Volumes/External/Cpp/OsFinalProject/build  master ±+ cmake ..
-- The CXX compiler identification is AppleClang 17.0.0.17000603
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (0.6s)
-- Generating done (0.0s)
-- Build files have been written to: /Volumes/External/Cpp/OsFinalProject/build
marcoskk7@Marcoskk7deMac-mini /Volumes/External/Cpp/OsFinalProject/build  master ±+ cmake --build
[  6%] Building CXX object src/CMakeFiles/osproj_domain.dir/domain/user.cpp.o
[ 13%] Building CXX object src/CMakeFiles/osproj_domain.dir/domain/permissions.cpp.o
[ 20%] Building CXX object src/CMakeFiles/osproj_domain.dir/domain/auth.cpp.o
[ 26%] Linking CXX static library libosproj_domain.a
[ 26%] Built target osproj_domain
[ 33%] Building CXX object src/CMakeFiles/osproj_fs.dir/server/filesystem/vfs.cpp.o
[ 40%] Linking CXX static library libosproj_fs.a
[ 40%] Built target osproj_fs
[ 46%] Building CXX object src/CMakeFiles/osproj_server_core.dir/server/server_app.cpp.o
[ 53%] Building CXX object src/CMakeFiles/osproj_server_core.dir/server/net/tcp_server.cpp.o
[ 60%] Linking CXX static library libosproj_server_core.a
[ 60%] Built target osproj_server_core
[ 66%] Building CXX object src/CMakeFiles/osproj_server.dir/server/main.cpp.o
[ 73%] Linking CXX executable osproj_server
[ 73%] Built target osproj_server
[ 80%] Building CXX object src/CMakeFiles/osproj_client.dir/client/main.cpp.o
[ 86%] Building CXX object src/CMakeFiles/osproj_client.dir/client/cli.cpp.o
[ 93%] Building CXX object src/CMakeFiles/osproj_client.dir/client/net/tcp_client.cpp.o
[100%] Linking CXX executable osproj_client
[100%] Built target osproj_client
```

9.2 运行

- 启动服务器：
  - `./build/src/osproj_server [port] [cacheCapacity]`
  - 环境变量：`OSP_CACHE_CAPACITY`，可覆盖默认缓存容量
- 启动客户端：
  - `./build/src/osproj_client`，默认连接 127.0.0.1:5555



10. 后续计划

10.1 网络并发改造

将 `TcpServer` 从“单连接单请求加每轮重建 listen socket”改为“长期监听加并发处理”：

- **多线程 per-connection**
  - 主线程 `listen + accept`，每个连接交给工作线程处理，可使用线程池
  - 注意：`Vfs`、`AuthService` 共享数据需要加锁，例如 `std::mutex`，避免并发写导致数据损坏

10.2 备份与恢复功能落地

目标：支持 `BACKUP <path>`、`RESTORE <path>` 并可演示：

- 最简单可行实现：对 `data.fs` 做整文件拷贝，即快照式
  - `BACKUP`：将当前 `data.fs` 复制到指定备份文件或备份目录
  - `RESTORE`：停止对外服务或加全局锁，替换 `data.fs` 并重新 mount
- 进阶：多版本备份列表、校验、增量或写时复制快照

10.3 文件系统能力增强

- 支持多块目录，目录文件可跨多个 data blocks
- 支持间接块或 extent，提高单文件容量
- 支持更完整的错误码与一致性，即崩溃恢复与日志

- 持久化用户与会话，例如将用户表落盘到 `/system/users`

10.4 业务增强：可选扩展

- 自动分配审稿人，包括领域匹配与冲突检测
- 版本历史，即论文修订版管理
- 更严格的状态机，如 Submitted → UnderReview → Accepted/Rejected

---

11. 测试与验证策略

- **协议正确性**：构造非法 JSON、缺字段或错误 type，验证 server 返回 Error，例如 PARSE_ERROR 或 INVALID_TYPE
- **权限边界**：Author 访问他人论文应拒绝；Reviewer 未分配论文应拒绝
- **文件系统基本操作**：`MKDIR/WRITE/READ/RM/RMDIR/LIST` 覆盖常见路径
- **缓存统计**：重复读同一文件块应提高 `hits`，并可通过 `VIEW_SYSTEM_STATUS` 观察变化

---

12. 风险分析与对策

- **并发导致的数据一致性问题**：后续并发化必须引入锁或更细粒度的同步；对 VFS 写操作做互斥保护
- **VFS 简化设计的上限**：目录或文件大小受限；需在答辩时明确“简化点与扩展路线”
- **备份恢复时的在线一致性**：恢复期间需要暂停服务或全局锁，避免一边恢复一边写入

---

13. 团队分工：

- **成员**：
  - `王梓衡`：总体架构、协议设计与服务器命令路由，自定义文件系统，包含 superblock、inode、bitmap、目录及缓存。
  - `宋曦`：前端系统架构与可视化交互实现，响应式 UI 设计、文件树动态呈现及前后端通信封装。
  - `张叶涵&李雨桐`：客户端 CLI、四个角色业务流程、权限矩阵与备份恢复。

---

14. 附录：常用命令速查即 CLI

- **基础**：`PING`、`LOGIN <u> <p>`、`ROLE_HELP`
- **文件系统**：`LIST [path]`、`MKDIR <path>`、`WRITE <path> <content...>`、`READ <path>`、`RM <path>`、`RMDIR <path>`、`CD <path>`
- **论文流程**：`SUBMIT`、`LIST_PAPERS`、`GET_PAPER`、`ASSIGN/ASSIGN_REVIEWER`、`REVIEW`、`LIST_REVIEWS/VIEW_REVIEW_STATUS`、`DECISION/MAKE_FINAL_DECISION`
- **管理**：`MANAGE_USERS ...`、`BACKUP`、`RESTORE`、`VIEW_SYSTEM_STATUS`