Lab5 实验报告

Author: 刘佳隆

Student ID: 518010910009

Email: liujl01@sjtu.edu.cn

练习 1

阅读 user/chcore-libc/libchcore/porting/overrides/src/chcore-port/file.c 的 chcore_openat 函数，分析 ChCore 是如何处理 openat 系统调用的，关注 IPC 的调用过程以及 IPC 请求的内容。

在ChCore系统中，`chcore_openat`函数以目录描述符和文件路径为参考点打开文件。其处理`openat`调用及IPC交互的详细分析如下：

1. **分配文件描述符（FD）**：首先，通过调用`alloc_fd`为即将打开的文件分配一个文件描述符。文件描述符将用于标识和追踪打开的文件。
2. **生成完整路径**: 接着，通过`generate_full_path`函数基于给定的`dirfd`和`pathname`生成一个完整的文件路径。
3. **解析完整路径以确定挂载ID和服务器路径**: 使用`parse_full_path`解析刚生成的全路径，以确定目标文件所在的文件系统的挂载ID以及相对于该文件系统根的路径（`server_path`）。
4. **准备IPC消息和结构**:
   - 根据挂载ID获取对应的IPC结构体`mounted_fs_ipc_struct`。
   - 初始化`fd_record_extension`结构体，用于存储与该FD关联的额外信息，包括挂载ID和文件路径。
   - 创建IPC消息结构体`ipc_msg`，并填充`fs_request`结构体作为其数据部分。`fs_request`结构体包含了文件系统请求的具体内容，如请求类型（这里是`FS_REQ_OPEN`）、新分配的文件描述符、文件路径名、访问标志（`flags`）和权限模式（`mode`）。
5. **发起IPC调用至文件系统服务器**: 使用`ipc_call`函数发送之前构造的IPC结构体和消息到文件系统服务器。此调用会阻塞等待服务器响应，服务器根据请求执行实际的`open`操作。
6. **处理IPC响应**:
   - 如果IPC调用成功（返回值非负），则更新本地的文件描述符表（`fd_dic`），设置文件描述符类型为文件（`FD_TYPE_FILE`），并关联相应的文件操作集（`fd_op`）。最后，函数返回分配的文件描述符作为成功打开文件的标识。
   - 若调用失败，则通过`free_fd`释放之前分配的文件描述符资源。
7. **资源清理**: 无论成功还是失败，函数都会负责释放之前分配的全路径字符串内存，并销毁IPC消息结构体，确保资源的正确管理。

练习7

思考 ChCore 当前实现 VFS 的方式有何利弊？如果让你在微内核操作系统上实现 VFS 抽象，你会如何实现？

ChCore 中将页缓存、icache、dcache 等数据管理放在文件系统中而不是FS管理器中，其优势是防止在文件系统中调用FS管理器的函数，造成大量的 IPC 开销，其缺点是需要在每个文件系统中实现数据管理功能。

我们可以借助 libOS 的思想，将 VFS 中需要重复实现的功能作为库供文件系统使用，这样可以减少开发量。