# Lab5补充指导书


Lab5补充指导书
- 实验背景
- 重要概念
- 设备驱动
- 文件系统进程
- 用户接口
- 代码梳理
- 说明


为了帮助同学们更好地熟悉文件系统部分的代码，完成实验内容，我们编写了补充指导书。在补充指导书中我们先对一些重要的概念进行了回顾，并说明了文件系统进程的基础功能。之后结合常见的文件操作介绍了文件系统的用户接口，希望可以加深同学们对文件系统工作流程的理解。最后我们分层梳理了文件系统的代码，并对一些重要的结构和函数进行了介绍，建议在此基础上对文件系统的代码进行仔细阅读，体会其中蕴含的操作系统知识和设计思想。

## 实验背景

在之前的实验中，所有的程序和数据都存放在内存中。然而内存空间的大小是有限的，且内存中的数据具有易失性。因此有些数据必须保存在磁盘、光盘等外部存储设备上。这些存储设备能够长期地保存大量的数据，且可以方便地将数据装载到不同进程的内存空间进行共享。

为了便于管理存放在外部存储设备上的数据，我们在操作系统中引入了文件系统。文件系统将文件作为数据存储和访问的单位，可看作是对用户数据的逻辑抽象。对于用户而言，文件系统可以屏蔽访问外存上数据的复杂性。

## 重要概念

以下概念对于我们深入理解文件系统，完成实验内容十分关键。

- **设备即文件**：操作系统使用文件系统来统一管理所有的外部设备。普遍意义上的“文件”指的是txt、pdf等格式的文件。而从广义上，一切带标识的、在逻辑上有完整意义的字节序列都可以称为“文件”。在我们实现的精简的文件系统中，需要对三种设备进行统一地管理，即**文件设备**（file，即狭义的“文件”）、控制台（console）和管道（pipe）。
- 在文件系统的实现中，我们的操作系统设计体现了微内核的设计思想，将文件系统的操作使用单独的**文件系统进程**进行完成。文件系统进程通过系统调用完成文件操作并通过IPC机制向普通进程提供文件服务。在完成实验时需要区分**普通进程操作、文件系统进程操作、系统调用**三者的调用关系。
- 文件系统的用户接口：为方便用户使用，文件系统向用户提供了命令行接口和程序接口。**程序接口**使得用户程序取得文件系统的服务，例如C语言中的 `fopen, fclose, fscanf, fprintf` 等，就可以对（狭义的）文件进行操作（注：这些函数并没有直接调用了操作系统的底层接口，C语言的标准库还进行了封装）。

  在文件操作中，最为重要的创建文件、打开关闭文件、读写文件、移除文件的功能我们都在Lab5的代码中进行了实现。
- 文件系统进程在较为底层的角度实现了对文件的管理，涉及到磁盘块的管理以及内存与磁盘块之间的交互。在理解文件系统进程时需要特别关注文件控制块（File结构体）的作用。

而文件系统的用户接口将底层的工作分配给了文件系统进程，因而可以使用更加简便的方式来进行管理，例如使用文件描述符（Fd结构体）来进行文件操作。

## 设备驱动

在实现文件系统高层操作之前，必须要首先完成对外部设备的驱动程序编写。由于我们是在模拟器上运行操作系统，外部设备的物理地址是完全固定的，因此我们可以通过读写固定的内核虚拟地址来实现外部设备驱动的功能。在学习MIPS的内存空间布局时我们就了解到kseg1区域具有不经过MMU映射、不经过高速缓存的特性，一般由外部设备进行使用。

在对设备驱动进行编写之前，我们需要先封装对于外部设备访问的系统调用 `syscall_write_dev` 和 `syscall_read_dev` 。该部分对应了 **Exercise 6.1**。和Lab2中访问kseg0时一样，你需要将物理地址和虚拟地址进行转换。

在外部设备上进行随意读写是无法成功与设备进行数据交换的，因此我们需要针对外部设备的特点编写驱动程序，使得我们能够与设备进行有效地交互。在指导书的 **表6.2** 中对磁盘指定偏移的作用进行了定义。

一言以蔽之，对于虚拟磁盘的读写就是在磁盘特定偏移的位置做一些写操作，之后虚拟磁盘会自动完成指定数据在磁盘和缓冲区中的迁移，我们只需要去缓存区放数据或读数据就可以，同时还可以通过查看特定位置判断读写操作是否成功。这部分对应了 **Exercise 6.2**。对于IDE磁盘的读取可以有多种实现方式，如使用C语言的系统调用读写对应位置、使用MIPS直接读写对应位置。

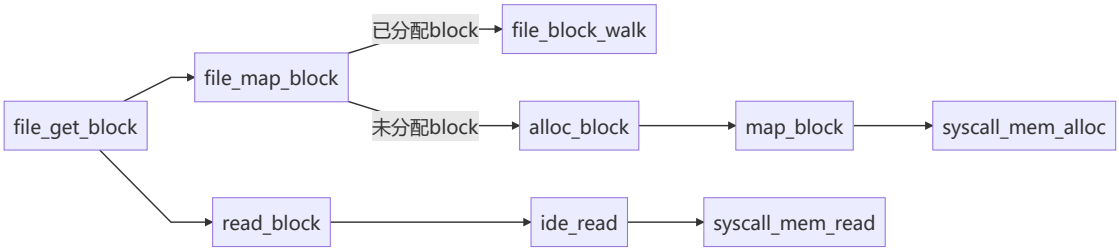
其实，读写磁盘并不是我们第一次与外部设备进行交互。在Lab3开启时钟中断时，我们使用了汇编函数set_timer。这个函数通过向Gxemul的实时钟（rtc）写入中断频率，开启了硬件的时钟中断。

## 文件系统进程

文件系统进程大致完成了三个部分的工作：

1. 将磁盘空间以块为单位划分并使用**位图法**管理，之后基于索引的方式管理文件在磁盘上的放置。这一部分主要通过操作 `fs/fs.c` 文件中的 `bitmap` 来完成，主要使用了 `block_is_free, free_block` 函数。该部分对应了 **Exercise 6.3**。
2. 管理磁盘上的文件块与内存空间的映射。利用位图法管理资源后，可以使用 `alloc_block` 来分配磁盘块并完成向内存的映射。

   观察 `file_get_block` 函数我们发现，映射过程分为两步：为读入内存的磁盘块分配对应的物理内存，接着把磁盘内容以块为单位读入内存中的相应位置。这两个步骤都需要借助系统调用来完成。




该部分对应了 **Exercise 6.5, Exercise 6.6。**

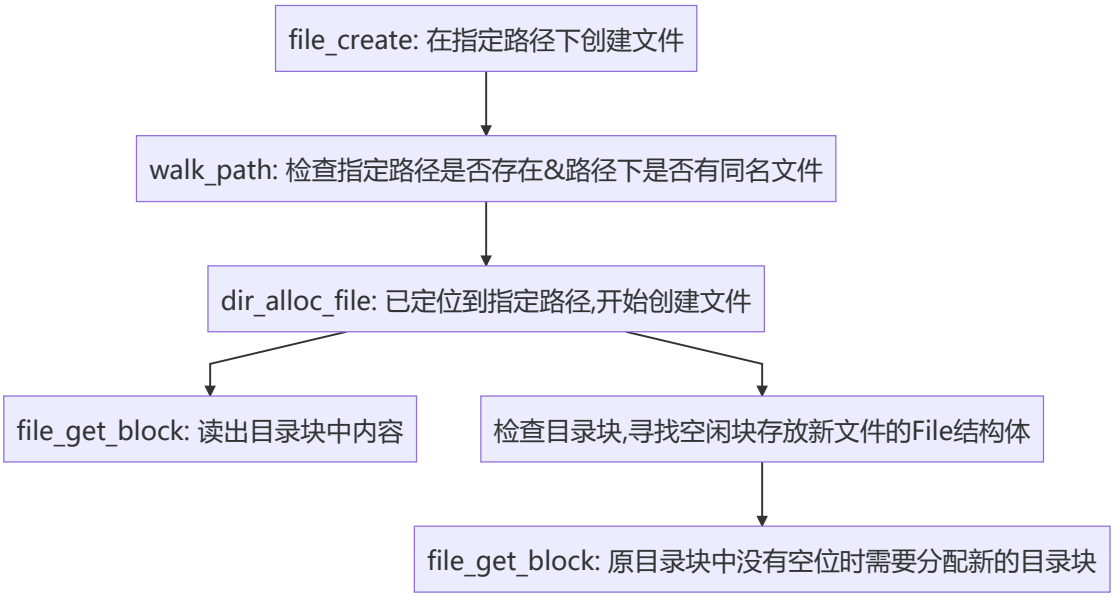
3. 在前两个功能的基础上，实现文件操作，如 `file_read, file_open, dir_lookup` 等。实现文件操作需要大量使用第二部分提供的 `file_get_block` 函数，用于**读文件**（将磁盘块与内存建立映射并将块中的内容读取到内存中）。该部分对应了 **Exercise 6.7。**

**note: 文件系统进程部分的测试在 `fs/test.c` 中，测试文件系统进程的底层文件操作能否正常进行。**

# 用户接口

MOS的文件系统对创建、打开、移除、读写文件等重要操作都进行了支持，但是实现的方式有一定的差别。我们选取create、open和read操作展示了文件系统对于重要文件操作的实现逻辑。

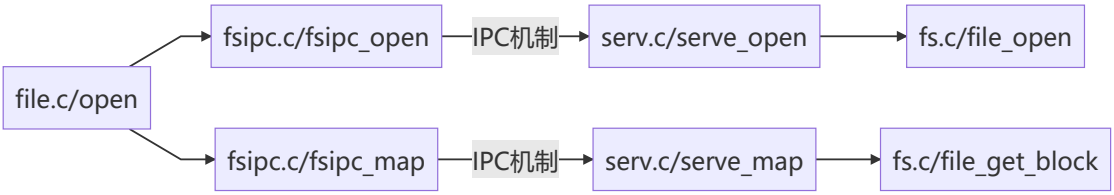
- create操作（在 `fs/fs.c/file_create` 已经实现了创建文件的全过程）




在 **Exercise 6.4** 中，我们需要参考以上的文件创建过程在 `fsformat.c` 中完成另一个创建文件的函数 `create_file` （该函数的流程与 `dir_alloc_file` 十分接近，需要在已经定位到指定目录的情况下创建相应的文件）。

观察 `fsformat.c` 的头文件和 `MakeFile` 文件可以发现， `fsformat.c` 的编译使用的是普通的 gcc 编译器并使用了一些 C 语言的标准库，和其他部分的编译流程不相同。使用 `fsformat.c` 生成镜像文件 `fs.img` 可以模拟与真实的磁盘文件设备的交互。需要注意的是，我们的文件系统和 `fsformat.c` 中生成的镜像文件符合相同的文件组织格式，因此使用我们实现的文件系统可以成功读出镜像文件 `fs.img` 中的内容。`fsformat.c` 中的主函数十分灵活，可以通过修改命令行参数来生成不同的镜像文件。

- open & remove（通过文件系统IPC实现，以open为例）




在使用文件系统IPC机制打开文件之后，open函数还需要对文件描述符进行填充并将文件内容映射到内存中。参照该流程可以帮助我们更好地完成 **Exercise 6.8** 和 **Exercise 6.10**。

- read & write（不通过文件系统IPC，以read为例）


参照该流程可以帮助我们更好地完成 **Exercise 6.9**。

**note: 用户接口的测试在 `user/fstest.c` 中，测试普通用户进程能否正确执行常见的文件操作。**

## 代码梳理

我们使用文件系统的大致结构如下表所示。

| 层次 | 功能 | 主要文件 |
| --- | --- | --- |
| 文件系统用户接口 | 为用户提供文件系统的访问接口 | user/file.c, user/fd.c |
| 文件系统抽象层 | 将用户访问请求转交给文件系统，利用文件系统进程提供服务 | user/fsipc.c, fs/serv.c |
| 文件系统具体实现 | 一个基于索引的文件系统实例，并提供与内存进行映射的机制 | fs/fsformat.c, fs/fs.c |
| 文件系统设备接口 | 实现各种设备的驱动接口 | fs/ide.c, lib/syscall_all.c |

文件系统部分主要的结构体及功能

| 文件 | 主要结构体定义 | 功能 |
| --- | --- | --- |
| include/fs.h | File, Super | 文件结构体、超级块结构体 |
|  | Fsreq_open, Fsreq_remove | 进行文件系统IPC请求的结构体 |
| user/fd.h | Dev | 适用于三种设备的结构 |
|  | Fd, Filefd | 文件描述符相关的结构 |
| fs/serv.c | Open | 记录打开文件的结构 |

文件系统部分各个文件的作用及主要函数，各个函数的详细功能请参考注释。

| 文件 | 功能 | 主要函数 |
| --- | --- | --- |
| user/fstest.c | 用户进程：测试文件系统的各项功能 | umain |
| fs/test.c | 文件系统进程：测试文件系统的各项功能 | fs_test |
| user/file.c | 1. 为用户进程提供访问文件的接口，例如打开文件、移除文件、读写文件 | open, remove |
| | 2. 文件系统进程对文件设备的读写操作 | file_read, file_write |
| user/console.c | 为用户进程提供访问控制台（console）的接口 | |
| user/pipe.c | 为用户进程提供访问管道（pipe）的接口 | |
| user/fd.c | 1. 使用文件描述符来作为用户程序管理、操作文件资源的方式 | fd_alloc, fd_lookup |
| | 2. 为用户进程提供对设备的访问接口（读写等操作） | read, write |
| user/fsipc.c | 文件系统IPC机制的用户进程部分，通过IPC机制向文件系统进程转发请求，完成打开关闭文件等操作 | fsipc, fsipc_open, fsipc_remove |
| fs/serv.c | 1. 文件系统IPC机制的文件系统进程部分，文件系统进程完成服务并向用户进程发送信息 | serve, serve_open, serve_remove |
| | 2. 创建文件系统进程：初始化文件系统并开启服务 | umain, serve_init |
| fs/fsformat.c | 创建磁盘镜像文件gxemul/fs.img，用于测试操作系统的文件管理功能 | create_file, make_link_block, write_file, write_directory |
| fs/fs.c | 1. 初始化文件系统 | fs_init, read_super |
| | 2. 使用位图法管理磁盘 | read_bitmap, block_is_free, free_block |
| | 3. 建立磁盘地址和物理内存之间的缓存映射 | diskaddr, map_block, unmap_block, read_block, write_block, alloc_block |
| | 4. 文件系统进程的文件操作 | dir_lookup, file_get_block, walk_path, dir_alloc_file, file_truncate |
| fs/ide.c | 磁盘驱动程序，对 Simulated IDE disk 进行读写操作 | ide_read, ide_write |
| lib/syscall_all.c | 使用系统调用对设备进行读写 | sys_write_dev, sys_read_dev |

# 说明

本次文件系统实验需要完成的代码填空部分较少，但是整个文件系统的代码量较为庞大，接近整个系统代码总量的一半。希望在补充指导书的帮助下，同学们能够完成对文件系统代码的通读，对MOS文件系统的设计有一个清晰的理解，体会其中的精妙之处和不足之处，欢迎大家在讨论区中对相关问题发表见解。**温馨提示：在课上测试中，测试的内容将不仅局限于需要填空的函数。**

若发现指导书中出现错误请与我们联系（微信：wyx2063959711），十分感谢。