Lab5 实验报告

零、实验目的

1. 了解文件系统的基本概念和作用。
2. 了解普通磁盘的基本结构和读写方式。
3. 了解实现设备驱动的方法。
4. 掌握并实现文件系统服务的基本操作。
5. 了解微内核的基本设计思想和结构。

在之前的实验中，我们把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，而且内存中的数据存在易失性问题。因此有些数据必须保存在磁盘、光盘等外部存储设备上, 这些外部存储设备能够长期保存大量的数据，且便于将数据装载到不同进程的内存空间进行共享。为了便于管理和访问存放在外部存储设备上的数据，在操作系统中引入了文件系统。在文件系统中，文件是数据存储和访问的基本单位。对于用户而言，文件系统可以屏蔽访问外存数据的复杂性。

一、思考题

Thinking 5.1

> 如果通过 kseg0 读写设备，那么对于设备的写入会缓存到 Cache 中。这是一种错误的行为，在实际编写代码的时候这么做会引发不可预知的问题。请思考：这么做这会引发什么问题？对于不同种类的设备（如我们提到的串口设备和 IDE 磁盘）的操作会有差异吗？可以从缓存的性质和缓存更新的策略来考虑。

- 当外部设备产生中断信号或者更新数据时，此时Cache中之前旧的数据可能刚完成缓存，那么完成缓存的这一部分无法完成更新，则会发生错误的行为。
- 对于串口设备来说，读写频繁，信号多，在相同的时间内发生错误的概论远高于IDE磁盘。

Thinking 5.2

> 查找代码中的相关定义，试回答一个磁盘块中最多能存储多少个文件控制块？一个目录下最多能有多少个文件？我们的文件系统支持的单个文件最大为多大？

- 一个磁盘块中最多存储 16 个文件控制块。
- 一个目录下最多 1024 个磁盘块，即 16384 个文件。
- 支持的单个文件最大为 4MB。

Thinking 5.3

> 请思考，在满足磁盘块缓存的设计的前提下，我们实验使用的内核支持的最大磁盘大小是多少？

最大磁盘大小为 1GB。

Thinking 5.4

> 在本实验中，`fs/serv.h`、`user/include/fs.h`等文件中出现了许多宏定义，试列举你认为较为重要的宏定义，同时进行解释，并描述其主要应用之处。

- `#define DISKMAP 0x10000000`，定义了磁盘在虚拟内存中映射的起始位置
- `#define DISKMAX 0x40000000`，定义了磁盘在虚拟内存的缓冲块的大小
- `#define MAXNAMELEN 128`，文件名的长度最大值
- `#define NDIRECT 10`，最多10个直接磁盘块
- `#define FILE_STRUCT_SIZE 256`，文件控制块的大小
- `#define FILE2BLK (BLOCK_SIZE / sizeof(struct File))`，也就是每一个块有多少个文件控制块，向下取整，这里是16

Thinking 5.5

> 在 Lab4“系统调用与 fork”的实验中我们实现了极为重要的 fork 函数。那么 fork 前后的父子进程是否会共享文件描述符和定位指针呢？请在完成上述练习的基础上编写一个程序进行验证。

`fork`前后父子进程会共享文件描述符和定位指针，测试程序为：

```
int id;
if ((id = fork()) == 0) {
    struct Fd* fdd;
    fd_lookup(r, &fdd);
    debugf("child_fd's offset == %d\n", fdd->fd_offset);
} else {
    struct Fd* fdd;
    fd_lookup(r, &fdd);
    debugf("father_fd's offset == %d\n", fdd->fd_offset);
}
```

Thinking 5.6

> 请解释`File`，`Fd`，`Filefd`结构体及其各个域的作用。比如各个结构体会在哪些过程中被使用，是否对应磁盘上的物理实体还是单纯的内存数据等。说明形式自定，要求简洁明了，可大致勾勒出文件系统数据结构与物理实体的对应关系与设计框架。

文件控制块是存放在是磁盘上的物理实体，包含了文件基本数据：

```
struct File {
    char f_name[MAXNAMELEN]; // filename        文件名称，最大长度为128
    uint32_t f_size; // file size in bytes  文件的大小，单位为字节
    uint32_t f_type; // file type           文件类型，有普通文件FTYPE_REG和目录
FTYPE_DIR两种
    uint32_t f_direct[NDIRECT]; // 文件的直接指针，每个文件控制块有10个直接指针，用来记
录文件的数据块在磁盘上的位置
                                // 每个磁盘块的大小为4KB，也就是这10个直接指针能够表示最
大40KB的文件
    uint32_t f_indirect; // 文件大于40KB时，需要用到间接指针。
    // ((int*)(disk[dirf->f_indirect].data))[i]
    struct File* f_dir; // the pointer to the dir where this file is in, valid only in
```

```
    memory.指向文件所属的文件目录
    char f_pad[FILE_STRUCT_SIZE - MAXNAMELEN - (3 + NDIRECT) * 4 - sizeof(void*)];
    //  是为了让整数个文件结构体占用一个磁盘块，填充结构体中剩下的字节
} __attribute__((aligned(4), packed));
```

`Fd` 是用户使用的，记录已打开文件的状态，便于用户直接使用文件描述符对文件进行操作、申请服务。是存放在内存上的数据。

```
// file descriptor
struct Fd {
    u_int fd_dev_id;    // 外设的id
    // 用户是用fd.c的用户接口是，不同的dev_id会调取不同的文件服务函数
    // fd_dev_id的取值可以是devfile.dev_id "f"  或者是 devcons.dev_id "c"
    u_int fd_offset;    // 读写的偏移量
    // 在file_read、file_write会改变这个偏移量
    // 在seek()时也会修改
    // offset会被用来找起始filebno文件块号。
    u_int fd_omode;     // 打开方式，包括只读、只写、读写
    // serve_open是会进行修改，read和write时会用到
};
```

`Filefd` 是扩充后的文件描述符，能存储更多文件信息。

```
// file descriptor + file
// 为了让Fd*类型的结构体可以存储更多信息，常常用来强转
struct Filefd {
    struct Fd f_fd;     // file descriptor 文件描述符
    u_int f_fileid;     // 文件的id
    // 会用来索引opentab[]中对应的open控制块
    struct File f_file; // 这个文件描述符对应的文件控制块
};
```

Thinking 5.7

> 图 5.9 中有多种不同形式的箭头，请解释这些不同箭头的差别，并思考我们的操作系统是如何实现对应类型的进程间通信的。

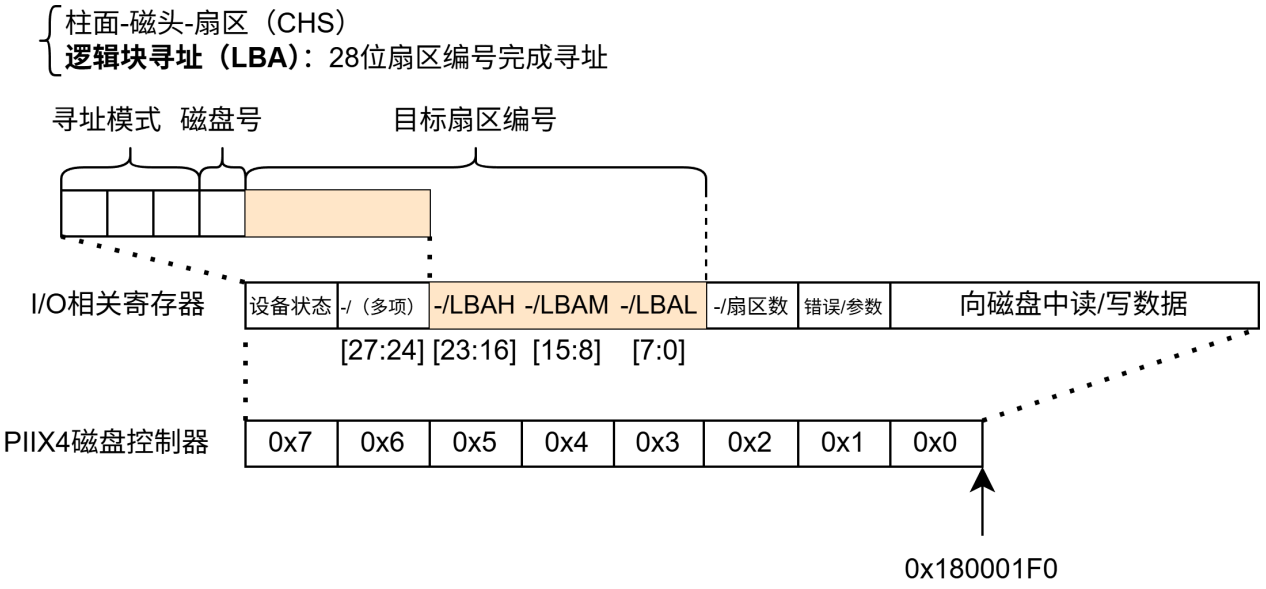
实线箭头是同步消息，指消息的发送者将消息发送出去后，暂停活动，等待消息接收者的回应消息。

虚线箭头是异步消息或返回消息，直接将消息发送出去，不进行等待，不需要知道返回值。

二、难点分析

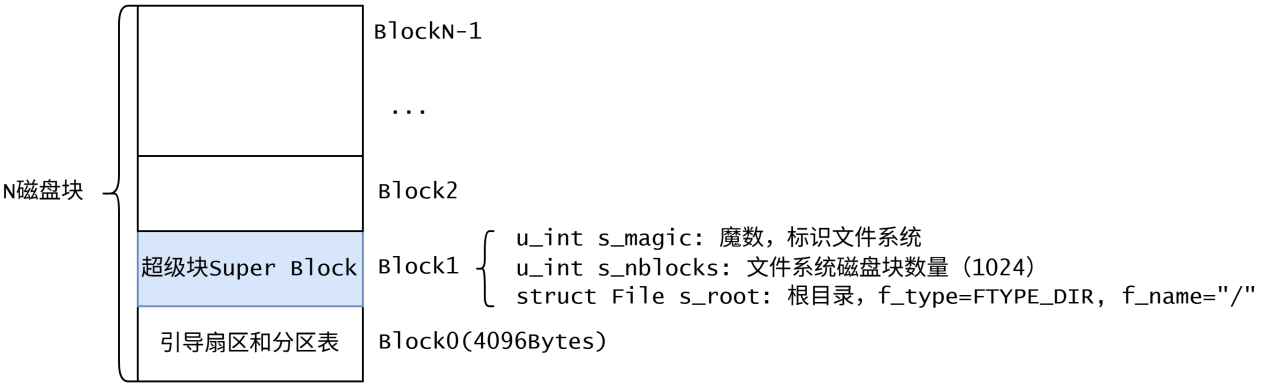
IDE 磁盘操作

实验中使用 LBA 寻址模式，由扇区编号即可完成寻址。



读取、写入流程：等待 IDE 设备就绪->设置操作扇区数目->设置操作扇区号的 [7:0] 位->设置操作扇区号的 [15:8] 位->设置操作扇区号的 [23:16] 位->设置操作扇区号的 [27:24] 位、设置扇区寻址模式、设置磁盘编号->设置 IDE 设备为读/写状态->等待 IDE 设备就绪->读取/写入数据->检查 IDE 设备状态->完成扇区读取/写入

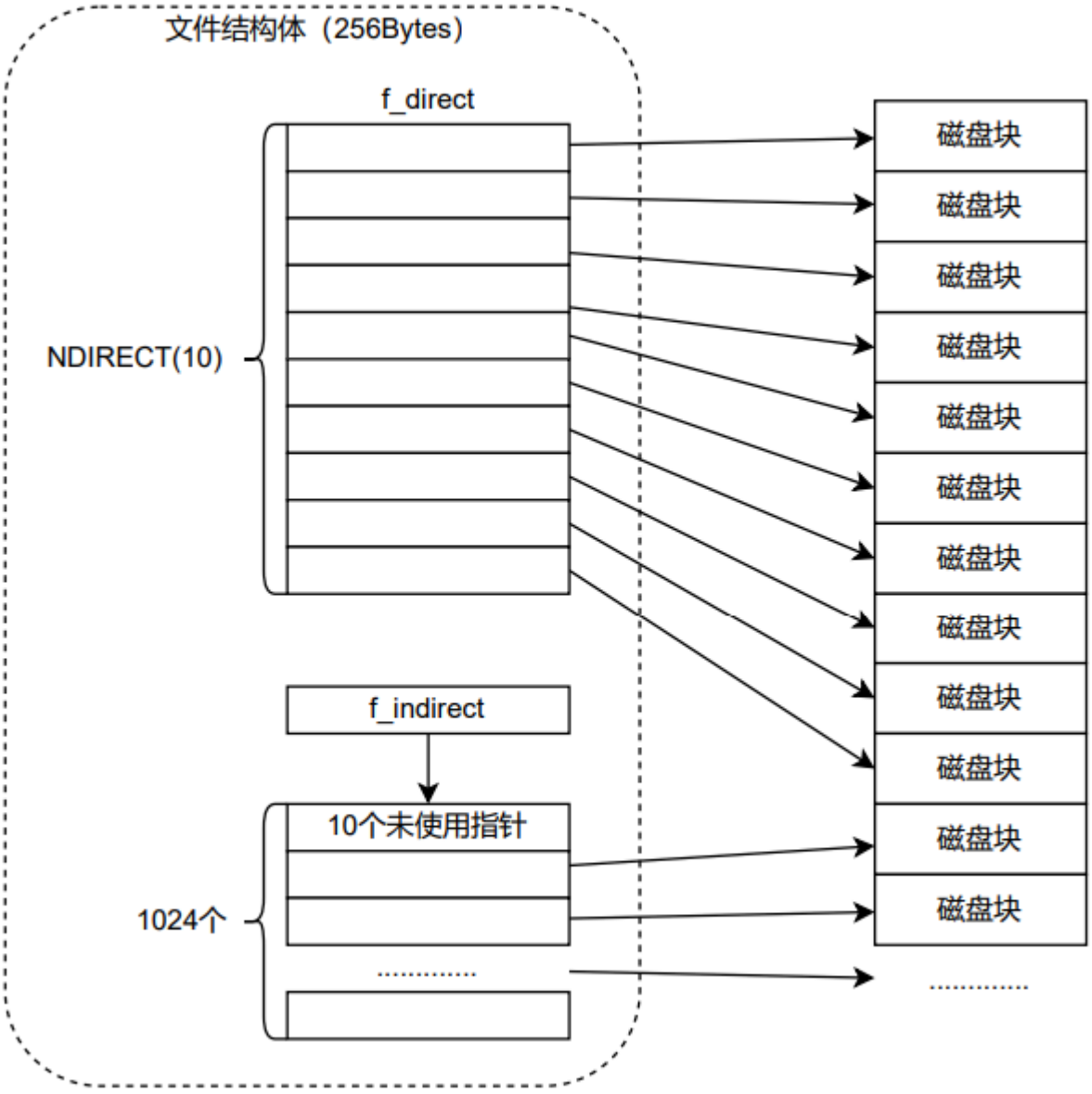
磁盘空间布局



文件控制块

```
struct File {
    char f_name[MAXNAMELEN]; // 文件名，最大为 MAXNALELEN == 128
    uint32_t f_size; // 文件大小，字节
    uint32_t f_type; // 文件类型，普通文件 FTYPE_REG，目录 FTYPE_DIR
    uint32_t f_direct[NDIRECT]; // 直接指针，每个控制块 10 个直接指针，表示最大 40KB 的
文件
    uint32_t f_indirect; // 间接指针

    struct File *f_dir; // 指向文件所属文件目录
    char f_pad[FILE_STRUCT_SIZE - MAXNAMELEN - (3 + NDIRECT) * 4 - sizeof(void *)]; //
为了让整数个文件结构体占用一个磁盘块，填充结构体剩余的字节
} __attribute__((aligned(4), packed));
```



新增主要函数作用

- `sys_write_dev`， `sys_read_dev`：（`kern/syscall_all.c`）系统调用，实现设备的读写操作。以用户虚拟地址、设备物理地址、读写字节长度，作为参数，在内核空间完成 I/O 操作。
- `ide_read`， `ide_write`：（`fs/ide.c`）实现用户态下对磁盘的读写操作，操作过程是 PIIX4 磁盘控制器 I/O 相关寄存器的设置。
- `free_block`：（`fs/fs.c`）释放磁盘块，位图中对应位的值设置为 1。
- `create_file`：（`tools/fsformat.c`）将一个文件或指定目录下的文件按照目录结构写入到 `target/fs.img` 。
- `disk_addr`：（`fs/fs.c`）由磁盘块块号计算对应的虚存起始地址。
- `map_block`：（`fs/fs.c`）检查磁盘块是否映射到内存，没有则分配一页内存。
- `unmap_block`：（`fs/fs.c`）解除磁盘块和物理内存对应映射，回收内存。
- `dir_loopup`：（`fs/fs.c`）查找某个目录下是否存在指定文件。
- `open`：（`user/lib/file.c`）打开文件，返回文件描述符的编号。
- `write`、 `read`：（`user/lib/fd.c`）读写文件。
- `serve_remove`：（`fs/serv.c`）删除指定目录文件， ipc 发送。
- `fsipc_remove`：（`user/lib/fsipc.c`）删除指定目录文件。

- `remove`：（`user/lib/file.c`）删除指定目录文件，调用 `fsipc_remove`。

三、实验体会

Lab5 整体流程感觉和之前的实验调性一致，整个系统是比较相似的。

四、原创说明

感谢：
os-lab5实验报告 | hugo