北航操作系统实验报告 Lab 5

学号：71066001

姓名：陈伟杰

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。 ■

答：会引发数据的可靠性和性能问题如读取缓存中的过期内容会导致错误的结果还有如串口设备，将写入缓存中会导致数据延迟被写回设备，从而影响实时性能，不同种类的设备的操作影响也会不一样，有写回和写直达两种策略，写回缓存会在缓存满或者被替换之前一直留在缓存中，而写直达缓存则会立即写入设备。

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？ ■

答： char f_pad[BY2FILE - MAXNAMELEN - (3 + NDIRECT) * 4 - sizeof(void *)]; }

__attribute__((aligned(4), packed))

// NDIRECT 是一个常量，表示一个文件直接指向的块数

BY2FILE 代表着一个文件控制块的大小为 256B，在我们这个的文件系统中，目录块大小为 1024 字节，那么一个磁盘块 4KB 最多存 $4KB/256B$=16 4KB/256B=16 个文件控制块，一个目录最大为 4112KB，则最多能 $4112KB/256B$=16448 4112KB/256B=16448 个。1024 个磁盘块共 1024 * 4KB = 4MB。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？ ■

答：DISKMAX = 0x40000000 即 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。 ■

答：我觉得在 serv.h 文件里

1. BY2SECT：每个磁盘扇区的字节数，用于计算扇区偏移量。
2. DISKMAP：磁盘块在文件系统服务器地址空间中的映射地址，用于操作磁盘块。
3. PTE_DIRTY：页表项标志，用于标记文件系统块缓存是否被修改过。
4. DISKNO：文件系统使用的 IDE 磁盘号，指定在哪个磁盘上查找文件系统。
5. file_open、file_get_block、file_set_size、file_close、file_remove、file_flush：文件系统相关操作的函数，包括打开文件、获取文件块、设置文件大小、关闭文件、删除文件

在 fs.h 文件里

1. BY2BLK：文件系统块的大小，等于页大小，用于计算每个块包含的字节数。
2. MAXNAMELEN：文件名的最大长度，包括结尾的空字符。
3. NDIRECT：每个文件直接指向的块数。
4. BY2FILE：每个文件控制块的大小。
5. struct File：文件控制块的定义，包括文件名、大小、类型、直接块和间接块等信息。
6. FTYPE_REG 和 FTYPE_DIR：文件类型（普通文件和目录文件）

我们通过这些宏定义可以更好的理解和使用文件系统的各种元素和功能比如包括增删改查，文件大小，文件类型，定义了文件系统的对外接口

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。 ■

答：理论上父子进程会共享文件描述符和定位指针，但由于共享文件描述符所以父进程和子进程写入操作也会相互影响, 具体程序以下

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

int main() {
    int fd;
    off_t offset;
    char buffer[100];

    fd = open("example.txt", O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);
    if (fd == -1) {
        perror("open");
        exit(EXIT_FAILURE);
    }

    write(fd, "Hello, World!\n", 14);

    pid_t pid = fork();

    if (pid == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    }

    if (pid == 0) {  // 子进程
        offset = lseek(fd, 0, SEEK_SET);
        if (offset == -1) {
            perror("lseek");
            exit(EXIT_FAILURE);
        }

        read(fd, buffer, 14);
        printf("Child process read: %s", buffer);
```

```
        close(fd);
        exit(EXIT_SUCCESS);

    } else {  // 父进程

        offset = lseek(fd, 0, SEEK_END);
        if (offset == -1) {
            perror("lseek");
            exit(EXIT_FAILURE);
        }

        write(fd, "Goodbye, World!\n", 16);

        close(fd);
        exit(EXIT_SUCCESS);
    }

}

  //父进程和子进程都打开了同一个文件描述符,父进程首先把文件偏移量移到了文件末尾，并写入了一些数据。然后，父进程创建了一个子进程。
 //子进程把文件偏移量移到了文件开头，并读取了一些数据。最后，父子进程都关闭了文件描述符
```

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。 ■

答：struct File 用于描述文件的基本信息，包括文件名、大小、类型、直接块和间接块等信息，被用来管理文件系统中的文件；struct Fd 用于描述文件描述符的基本信息，包括设备 ID、偏移量和打开模式等信息，被用来管理文件系统中的文件描述符；而 struct Filefd 结合了结构体 Fd 和 struct File 的元素，用于在内存中表示文件描述符和关联的文件，其中文件描述符的基本信息存放在 Fd 字段中，而关联的文件则存放在 struct File 字段中。
这些结构体的操作通常发生在文件系统的读写、打开和关闭等操作中，其对应的数据通常是在内存中的，而非文件系统中的物理实体。

```
struct File {
    char f_name[MAXNAMELEN]; // 文件名
    uint32_t f_size;      // 文件大小（字节）
    uint32_t f_type;      // 文件类型
    uint32_t f_direct[NDIRECT];//直接块的指针数组
    uint32_t f_indirect;//间接块的指针


struct Fd {
    u_int fd_dev_id;//文件描述符所属设备的ID
    u_int fd_offset;//文件描述符的偏移量
    u_int fd_omode;//文件描述符的打开模式
};

struct Filefd {
    struct Fd f_fd;//文件描述符
    u_int f_fileid;//文件的ID
    struct File f_file;//文件信息
};
```

Thinking 5.7 图5.7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。 ■

答：
黑三角箭头搭配黑实线表示的是同步消息，是消息的发送者把进程控制传递给消息的接收者，然后暂停活动等待回应；

空心箭头搭配黑色虚线表示的是返回消息，与同步消息结合使用的；

我们的进程通过和 file_server 这个进程的通信，来操作文件。

在使用者进程中，我们可以调用 user/file.c 内的函数，如 open，还有 fd.c 的 fd_alloc 然后通过 fsipc.c 中的这些函数比如 fsipc_alloc, fsipc 与文件系统进行数据传输。当文件系统进程完成 serv 和 fs 的初始化后，它将运行 serv 函数，以开始与使用者进程的通信。在这个函数中，我们可以调用 ipc_recv 函数，从而完成通信的过程。

实验困难：

本次实验对于本人来说还是有点困难，相比之前的 lab 这次 lab 进入了文件系统也更难了，代码文件也多了，其复杂度也高了，整体消耗的时间反而不是在写代码而是在一直读懂整个文件系统的调用关系并理解，一直在反复看指导书的文件系统总览图，比如了解文件系统的基本功能和用户接口，磁盘的编写，通过下面指导书的图也让我也让我在阅读代码的时候有帮助。

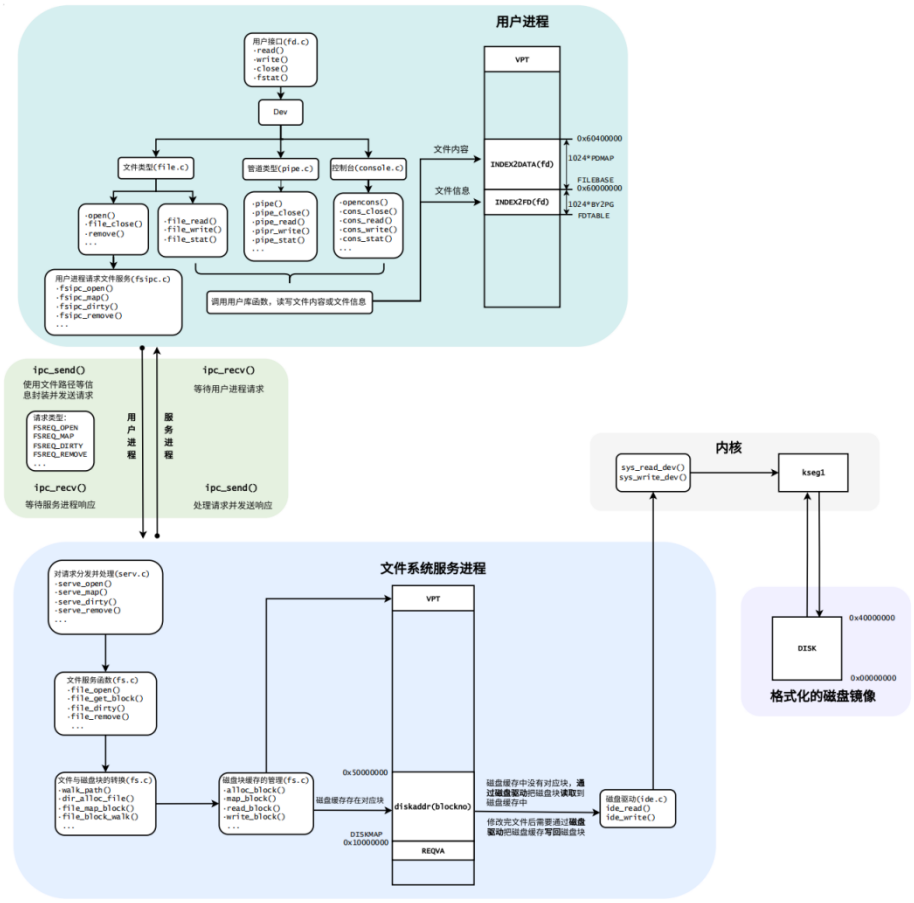

图 5.1: 文件系统总览图

心得体会：

随着 lab5 的结束，操作系统也快要接近尾声了，自己的上机一直不顺利，一直在想自己有没有学会，但相比刚开始的接触，现在自己在调试，在找 bug 的路上也越来越成熟了，其整个操作系统的原理也一次又一次的加深了，通过这次实验，自己对文件系统的了解也不在向之前的接触如此的少，也让我知道操作系统的奥秘。