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：当外部设备更新数据时，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。

串口设备读写频繁，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？

答： 一个文件控制块的大小为256B。 一个磁盘块4KB最多存储4KB/256B=16个文件控制块；

一个目录最多使用1024个磁盘块，最多能有1024*16=16384个文件。

单个文件最多使用1024个磁盘块，最大为1024*4KB = 4MB.

```
#define BY2FILE 256
struct File {
    char f_name[MAXNAMELEN]; // filename
    uint32_t f_size;     // file size in bytes
    uint32_t f_type;     // file type
    uint32_t f_direct[NDIRECT];   //直接指针
    uint32_t f_indirect;     //指向间接磁盘块

    struct File *f_dir; // the pointer to the dir where this file is in, valid only in memory.
    char f_pad[BY2FILE - MAXNAMELEN - (3 + NDIRECT) * 4 - sizeof(void *)];  //填充剩下字节
} __attribute__((aligned(4), packed));
```

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：用kseg1区域映射磁盘，因此最多处理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。

```c
/*** user/include/fs.h    ***/
struct File {
    char f_name[MAXNAMELEN]; // filename
    uint32_t f_size;     // file size in bytes
    uint32_t f_type;     // file type
    uint32_t f_direct[NDIRECT];
    uint32_t f_indirect;
    struct File *f_dir; // the pointer to the dir where this file is in, valid only in memory.
    char f_pad[BY2FILE - MAXNAMELEN - (3 + NDIRECT) * 4 - sizeof(void *)];
} __attribute__((aligned(4), packed));
```

文件控制块File结构体中记录文件数据块在磁盘上的位置。文件控制块有`10`个直接指针`f_direct`来表示最大40KB的文件，另外还有`1014`个间接指针（前10个保留不用）`f_indirect`去指向一个间接磁盘块; 结构体中对数据定位有关的十分重要。

```c
#define BY2SECT 512         /* Bytes per disk sector */
#define DISKMAP 0x10000000
#define DISKMAX 0x40000000
```

```c
#define BY2BLK BY2PG
// Maximum size of a filename (a single path component), including null
#define MAXNAMELEN 128
```

`BY2SECT`表示 1 个扇区的大小是 512 字节，`DISKMAP 和 DISKMAX`表示缓冲区地址范围为 0x10000000--0x3fffffff;

一个磁盘块大小`BY2BLK`等于一个页面大小（4KB），文件名最长为`MAXNAMELEN` =128Bytes;

这些宏定义了基本的数据的大小，限定了范围。

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：会共享。因为一个进程所有的文件描述符都存储在`[FDTABLE, FILEBASE)`这一地址空间中；在执行fork后，会把父进程页表中映射相应地址的页表项拷贝复制到子进程页表中。

```c
//文件内容：
"This is a different massage"
//
    int r, n, fdnum;
    char buf[512];

    fdnum = open("/newfile", O_RDWR);   //open for reading and writing
    if ((r = fork()) == 0) {   //子进程
        n = read(fdnum, buf, 4);
        debugf("[child] buffer is \'%s\'\n", buf);  //debugf输出
    } else {
```

```
        n = read(fdnum, buf, 4);
        debugf("[father] buffer is \'%s\'\n", buf);
    }
}
// 结果：
[father] buffer is 'This'
[child] buffer is ' is '
```

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。

File：

```
struct File {
    char f_name[MAXNAMELEN]; // filename
    uint32_t f_size;     // file size in bytes
    uint32_t f_type;     // file type
    uint32_t f_direct[NDIRECT];   //直接指针
    uint32_t f_indirect;     //指向间接磁盘块

    struct File *f_dir; // the pointer to the dir where this file is in, valid only in memory.
    char f_pad[BY2FILE - MAXNAMELEN - (3 + NDIRECT) * 4 - sizeof(void *)];  //填充剩下字节
} __attribute__((aligned(4), packed));
```

Fd：记录已打开文件的状态。文件描述符起到描述用户对于文件操作的作用。对应的是磁盘映射到内存中的数据（被用户使用）。

```
struct Fd {
    u_int fd_dev_id;  //文件对应设备
    u_int fd_offset;  //文件读写偏移量
    u_int fd_omode;  //文件读写模式
};
```

Filefd：Filefd 结构体的第一个成员就是Fd，可将`Fd*`强制转换为`Filefd*`来获取File；因为Fd存储信息可能有限。

```
struct Filefd {
    struct Fd f_fd;     //文件描述符
    u_int f_fileid;     //文件Id
    struct File f_file;  //文件控制块
};
```

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。

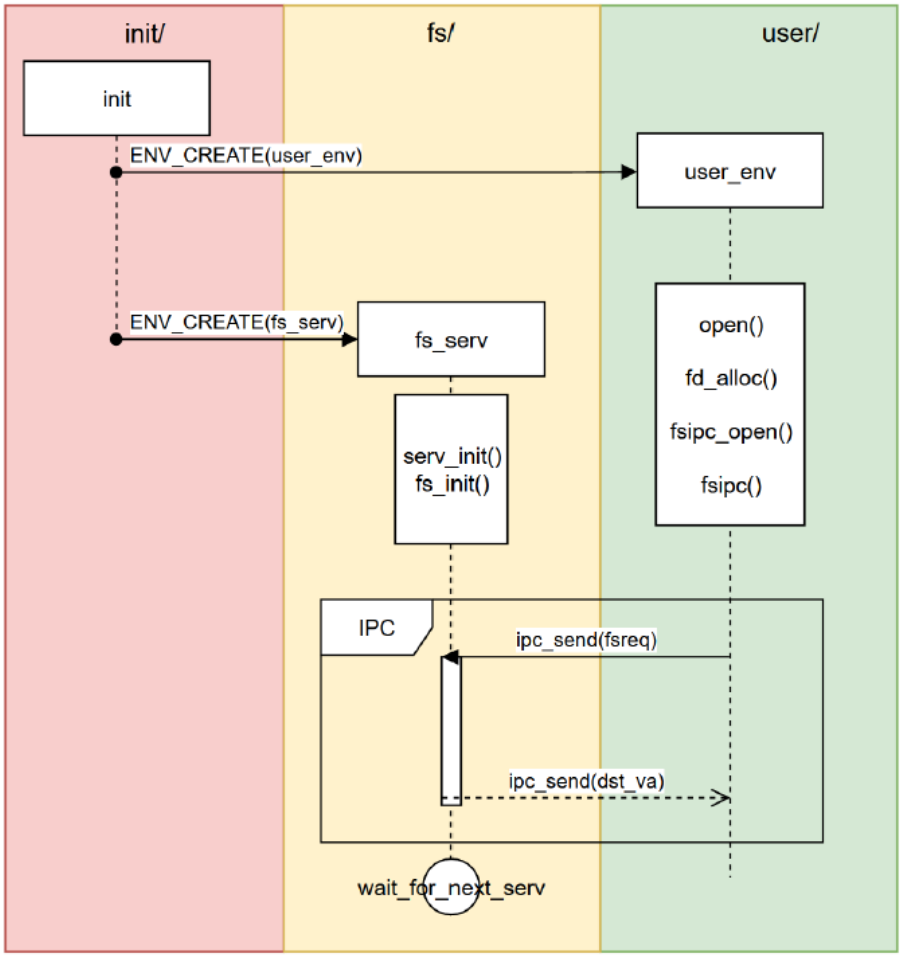
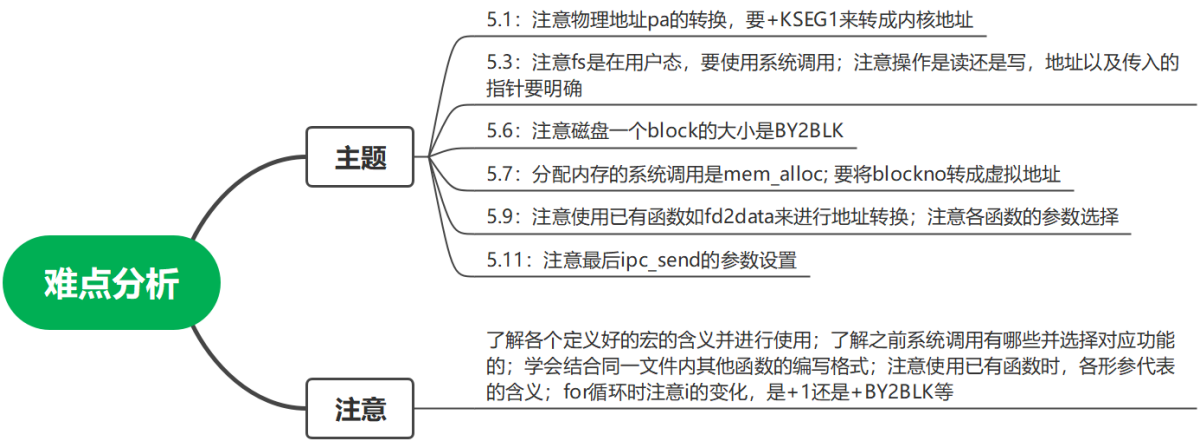


图 5.7: 文件系统服务时序图

刚开始ENV_CREATE创建一个用户进程，然后又创建一个fs文件系统进程。这两个进程同时运行；在fs初始化后以及用户处要与fs进行进程间通信时，两个进程通过IPC实现；用户进程先把fsreq请求发送给fs，fs进程运行一段时间后再把目标返回给用户进程；之后fs进程等待下一次的通信服务。

操作系统通过IPC来实现进程间通信。发送方先调用`ipc_send`函数，在函数内部通过死循环来持续向接收方发送信息。当接收方成功接收到消息时，`ipc_send`函数跳出循环结束；发送方再调`ipc_recv`函数主动放弃CPU，等待接收返回信息。

难点分析



实验体会

本次实验首先了解了设备驱动的编写，了解了磁盘的运作；然后介绍了文件系统中各个结构。主要还是要学会结合注释编写代码，并且主动去了解注释中提示的已经编写好的函数，学会正确调用。在编写过程中尤其要注意函数中各参数的含义，并及时了解各函数之间的调用关系。