Lab 4

思考题

Thinking 5.1

1. 访问内核时错误地读取到设备内容。
2. 外设可能被随时更新，通过 cache 会访问到落后内容。

Thinking 5.2

- `BY2BLOCK` / `BY2FILE` $= 16$ 个文件控制块。
- `NINDIRECT` $\times 16 = 16\mathrm{K}$ 个文件
- `NINDIRECT` $\times$ `BY2BLOCK` $= 4\mathrm{M}$。

Thinking 5.3

```
/* Maximum disk size we can handle (1GB) */
#define DISKMAX      0x40000000
```

为 1GB

Thinking 5.4

大部分都带有注释，无注释的部分也都容易理解。

```
// fs/serv.h
#define PTE_DIRTY 0x0002 // file system block cache is dirty

/* IDE disk number to look on for our file system */
#define DISKNO 1

#define BY2SECT 512         /* Bytes per disk sector */
#define SECT2BLK (BY2BLK / BY2SECT) /* sectors to a block */
// 主要用于读写磁盘

/* Disk block n, when in memory, is mapped into the file system
 * server's address space at DISKMAP+(n*BY2BLK). */
#define DISKMAP 0x10000000

/* Maximum disk size we can handle (1GB) */
#define DISKMAX 0x40000000



// user/include/fs.h
// File nodes (both in-memory and on-disk)

// Bytes per file system block - same as page size
#define BY2BLK BY2PG // 磁盘块大小
#define BIT2BLK (BY2BLK * 8)
```

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

// Maximum size of a complete pathname, including null
#define MAXPATHLEN 1024

// Number of (direct) block pointers in a File descriptor
#define NDIRECT 10
#define NINDIRECT (BY2BLK / 4)
// 直接与间接指针

#define MAXFILESIZE (NINDIRECT * BY2BLK)

#define BY2FILE 256

#define FILE2BLK (BY2BLK / sizeof(struct File))

// File types
#define FTYPE_REG 0 // Regular file
#define FTYPE_DIR 1 // Directory

// File system super-block (both in-memory and on-disk)

#define FS_MAGIC 0x68286097 // 历史久远
```

Thinking 5.5

会。

编写代码如下：

```c
int main()
{
    int fd, r;
    char buf[512];
    fd = open("/motd", O_RDONLY);
    switch (r = fork()) {
    case -1:
        user_panic("fork error");
    case 0:
        if ((r = read(fd, buf, 10)) != 10) {
            user_panic("read error: %d", r);
        }
        printf("child's buf: %s\n", buf);
        break;
    default:
        wait(r);
        if ((r = read(fd, buf, 10)) != 10) {
            user_panic("read error: %d", r);
        }
        printf("father's buf: %s\n", buf);
        break;
    }
    return 0;
}
```

文件内容如下：

```
This is /motd, the message of the day.

Welcome to the MOS kernel, now with a file system!
```

输出如下：

```
$ fstest.b
child's buf: This is /m
[00003805] destroying 00003805
[00003805] free env 00003805
i am killed ...
father's buf: otd, the m
[00003004] destroying 00003004
[00003004] free env 00003004
i am killed ...
[00002803] destroying 00002803
[00002803] free env 00002803
i am killed ...
```

可见父子进程共享文件描述符和定位指针。

Thinking 5.6

File 结构体包含了文件的基本信息，对应磁盘上的物理实体。

```c
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]; // 直接块索引
    uint32_t f_indirect;        // 间接块索引

    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 *)]; // 保证 File 大小为 512 字节
}
```

Fd 结构体为文件描述符，仅为内存数据。

```c
struct Fd {
    u_int fd_dev_id; // 设备 id
    u_int fd_offset; // 偏移量
    u_int fd_omode;  // 打开模式
};
```

Filefd 结构体包含文件描述符和文件，对应了磁盘的物理实体，也包含内存数据。

```
struct Filefd {
    struct Fd f_fd;      // 文件描述符
    u_int f_fileid;      // 文件 id
    struct File f_file; // 文件结构体
};
```

Thinking 5.7

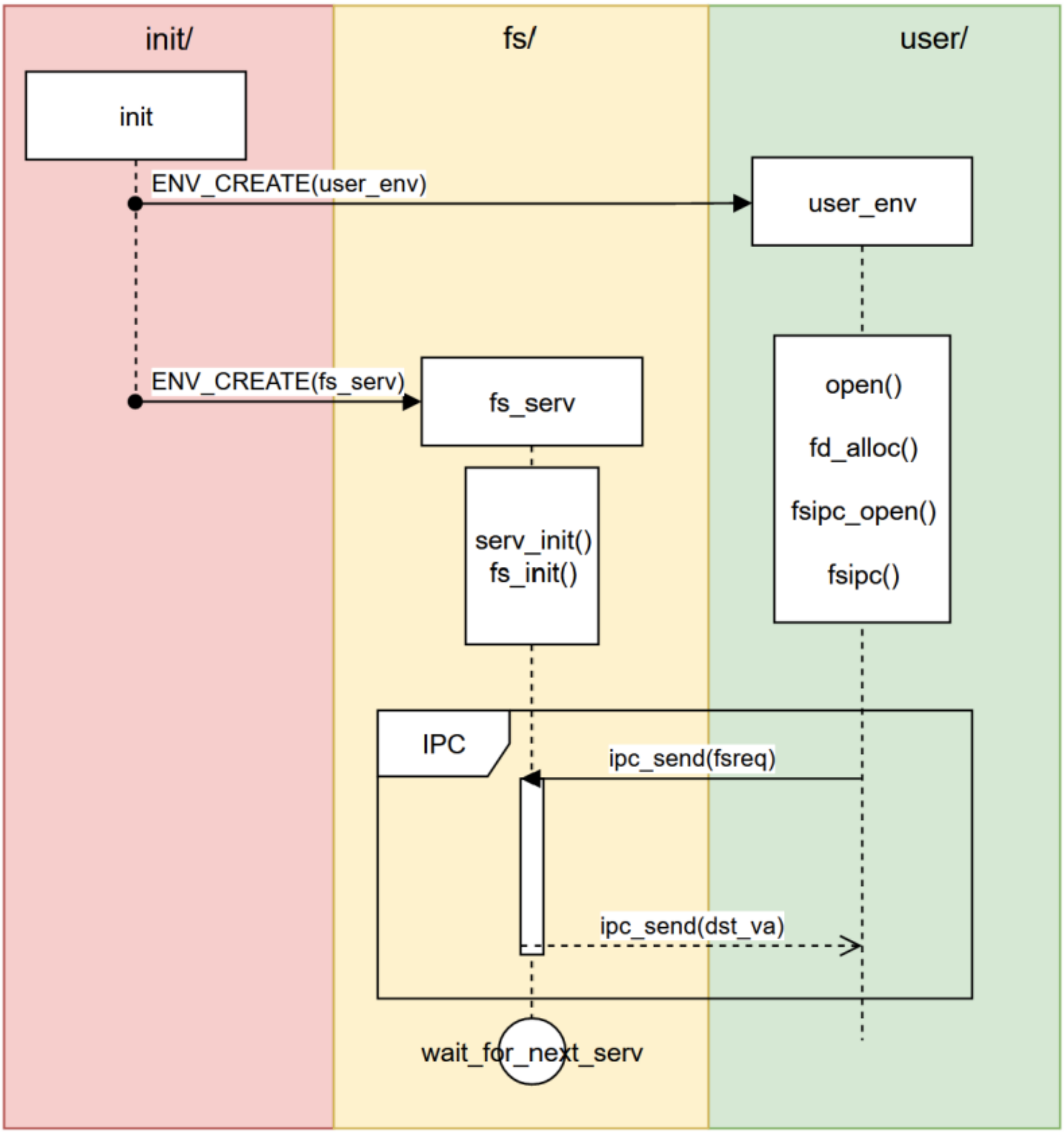

图 5.7: 文件系统服务时序图

为请求消息，用户向文件系统发送请求。采用指针的类型转换实现传输不同种类请求所需的数据。

为返回消息，文件系统对于用户请求提供返回值。

难点分析

主要难点在文件系统结构的理解，分 user 部分与 serv 部分分别梳理会简单很多。

实验体会

Lab5 课上与课下内容均较为简单，但仅完成课下填空无法建立起对文件系统的认知。在梳理后对文件理解仍有不足，课上将 fd 号与文件 id 混淆，导致出现问题，所幸得以解决。