操作系统lab5实验报告

姓名：高远 学号：19374242 班级：202113

实验思考题

Thinking 5.1

/proc文件系统是一种特殊的，由软件创建的文件系统，内核使用它向外界导出信息，/proc系统只存在内存当中，而不占用外存空间。/proc下面的每个文件都绑定于一个内核函数，用户读取文件时，该函数动态地生成文件的内容。与其它常见的文件系统不同的是，/proc是一种伪文件系统（也即虚拟文件系统），存储的是当前内核运行状态的一系列特殊文件，用户可以通过这些文件查看有关系统硬件及当前正在运行进程的信息，甚至可以通过更改其中某些文件来改变内核的运行状态。

Windows通过分盘实现，每个驱动器有自己的根目录，形成的是多个树并列的结构。

/proc的好处：方便查看系统状态。缺点：proc文件系统的信息过载问题，有时候会猛烈地爆发。改进方案：1）结构化的方法，采用特定于问题的虚拟文件系统。一个很好的例子就是USB文件系统，将与USB子系统有关的许多状态信息导出到用户空间，而没有给/proc增加新的负担。2）Sysfs文件系统提供了一种层次化的视图，不仅包括设备树，还有重要的内核对象。proc可以查看系统数据类型，特定进程的数据，一般性系统信息，网络信息和系统控制信息。

Thinking 5.2

内核被放在kseg0区域，一般通过cache访问，如果对设备的写入缓存到cache中，就会导致以后想访问内核时却错误访问了写入设备的内容。

Thinking 5.3

在Unix/Linux操作系统中的inode区域用来保存文件的状态属性，以及指向数据块的指针，通常有多级指针域和多级间接磁盘块。而MOS操作系统的inode区域用来保存的指针分别只有一级和一个。

Thinking 5.4

1个磁盘块中最多能存储16个文件控制块，一个目录下最多能有1024✕16=16384个文件，我们 的文件系统支持的单个文件的最大大小为4KB✕1024=4MB

Thinking 5.5

0x6f3fd000

Thinking 5.6

不能正常工作，因为使用文件系统时会覆盖用户空间的数据，导致系统运行异常.

Thinking 5.7

```
12  /* Server's address                    */
13  #define DISKMAP     0x10000000
14
15  /* Maximum disk size we can handle (1GB) */
16  #define DISKMAX     0x40000000
17
```

如图为DISKMAP和DISKMAX的宏定义，其含义分别为缓冲区起始位置和缓冲区的大小。

如图，BY2BLK为某一磁盘块对应的大小，大小为页的大小

Thinking 5.8

因为在结构体Filefd中储存的第一个元素就是结构体Fd，因而对于相匹配的一对struct Fd和struct Filefd，他们的指针实际上指向了相同的虚拟地址，所以可以通过指针转化访问struct Filefd中的其他元素。

Thinking 5.9

fork前后的父子进程会共享一个文件描述符和定位指针

Thinking 5.10

Fd结构体用于表示文件描述符，fd_dev_id表示文件所在设备的id，fd_offset表示读或者写文件的时候，距离文件开头的偏移量。fd_omode用于描述文件打开的读写模式。它主要用于在打开文件之后记录文件的状态，以便对文件进行管理/读写，不对应物理实体，只是单纯的内存数据。

Filefd结构体是文件描述符和文件的组合形式，f_fd记录了文件描述符，f_fileid记录了文件的id，f_file则记录了文件控制块，包含文件的信息以及指向储存文件的磁盘块的指针，对应了磁盘的物理实体，也包含内存数据。

代码所示：

```
struct Fd {
        u_int fd_dev_id; // 指示了该文件所处的设备
        u_int fd_offset; // 指示了当前用户进程对该文件进行操作的指针偏移位置（从文件开头起）
        u_int fd_omode; // 指示了文件的访问权限
};
struct Filefd {
        struct Fd f_fd;
        u_int f_fileid;
        struct File f_file;
};
```

Open结构体在文件系统进程用用于储存文件相关信息，o_file指向了对应的文件控制块，o_fileid表示文件id用于在数组opentab中查找对应的Open，o_mode记录文件打开的状态，o_ff指向对应的Filefd结构体。

Thinking 5.11

实线箭头表示Object Message，也就是类间的消息发送。比如：方法调用和请求等等，这些都是从一方向另一方主动发出信息。

虚线箭头表示两个类之间存依赖关系,表示Return Message，一般用于表示方法调用后的返回信息。即：一个类引用另一个类。只存在单向依赖。

操作系统实现进程间通信：用户程序在发出文件系统操作请求时，将请求的内容放在对应结构体（fsreq）中进行消息的传递，fs_serv进程接收到IPC请求后，根据请求的类型执行相应的操作，并通过IPC将结果返回给用户程序。

Thinking 5.12

因为再调用ipc_recv的时候这个进程会等待其他进程向他发送请求文件系统调用再继续执行，如果没有接收到请求就会在这里一直等待，因此不会自己执行死循环。

实验难点

难点1：位图法

在文件系统中，我们将使用位图 法来管理空闲的磁盘资源，用一个二进制位 bit 标识磁盘中的每个磁盘块的使用情况（实验中，1 表示空闲）。如图：

```
if (blockno == 0 || (super != NULL && blockno >= super->s_nblocks))
    return;
bitmap[blockno / 32] |= 1 << (blockno & 0x1f);
```

在bitmap中，如果bitmap[blockno/32]&(1<<(blockno&0x1f))的值是1，则说明处于空闲状态，反之，处于非空闲状态。

难点2：create_file函数

```
/*** exercise 5.4 ***/
struct File *create_file(struct File *dirf) {
    struct File *dirblk;
    int i, bno, found;
    int nblk = dirf->f_size / BY2BLK;

    // Your code here
    // Step1: According to different range of nblk, make classified discussion to
    //        calculate the correct block number.

    int j;
    for (i = 0; i < nblk; ++i)
    {
        if (i < NDIRECT)
            bno = dirf->f_direct[i];
        else
            bno = ((int *)(disk[dirf->f_indirect].data))[i];
        dirblk = (struct File *)(disk[bno].data);
        for (j = 0; j < FILE2BLK; ++j)
        {
            if (dirblk[j].f_name[0] == '\0')
                return dirblk + j;
        }
    }
    bno = make_link_block(dirf, nblk);
    return (struct File *)(disk[bno].data);
    // Step2: Find an unused pointer

```

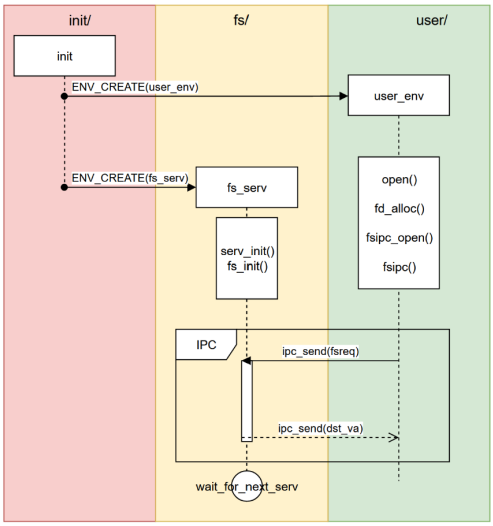
个函数从一个目录文件出发，目的是寻找第一个能够放下新的文件控制块的位置。当它找到一个指向已经被删除了的文件的文件控制块指针时，它直接返回，以求后续操作将这个空间覆盖掉。而当没有找到时，其直接进行拓展一个Block，并返回这个新的空白空间的起始地址。这里我们需要注意到，一个未被占用的空间被解释为文件控制块指针时，其行为和一个指向已经被删除了的文件的文件控制块指针一致，因此能够被统一处理。

如果发现有的文件控制块名称为终止符，说明这个文件已经被删除了/这个文件控制块的位置还没被占用，将该文件控制块起始地址返回

如果遍历了所有的Block后都没找到一个空的能放文件控制块的地方，直接拓展dirf的大小，并使第nblk个磁盘块链接到一个新的磁盘块（块号bno），最后返回这个新的磁盘块内的起始地址。

难点3：文件系统服务

在文件系统服务中，分为两个方面，一是fs文件夹中的函数，负责接收并处理用户的需求，其中，file.c中函数可调用fsipc.c中的函数。二是user文件夹中的函数，负责发出用户请求给fs，如下图所示：



用户程序在发出文件系统操作请求时，将请求的内容放在对应的结构体（fsreq）中进行消息的传递，fs_serv 进程收到其他进行的IPC 请求后，IPC 传递的消息包含了请求的类型（定义在include/fs.h中）和其他必要的参数，根据请求的类型执行相应的文件操作（文件的增、删、改、查等），将结果重新通过 IPC 反馈给用户程序。

体会与感想

单单对于我而言，lab5比起之前除lab0外任何一个实验都更加简单，无论是代码量方面还是思考深度方面。但这并不意味着lab5可以非常容易理解。在lab5中蕴含着大量新出现的结构体和宏定义，这需要我花费较大的时间进行阅读与理解。

总而言之，lab5让我理解了操作系统中文件系统的运行方法，使我受益匪浅。

指导书反馈

在指导书中，对于文件系统服务的内容讲解较少，且难以理解，花费了我大量的时间，希望可以有所改进。

残留难点

对read和write函数的理解还是不够充分。