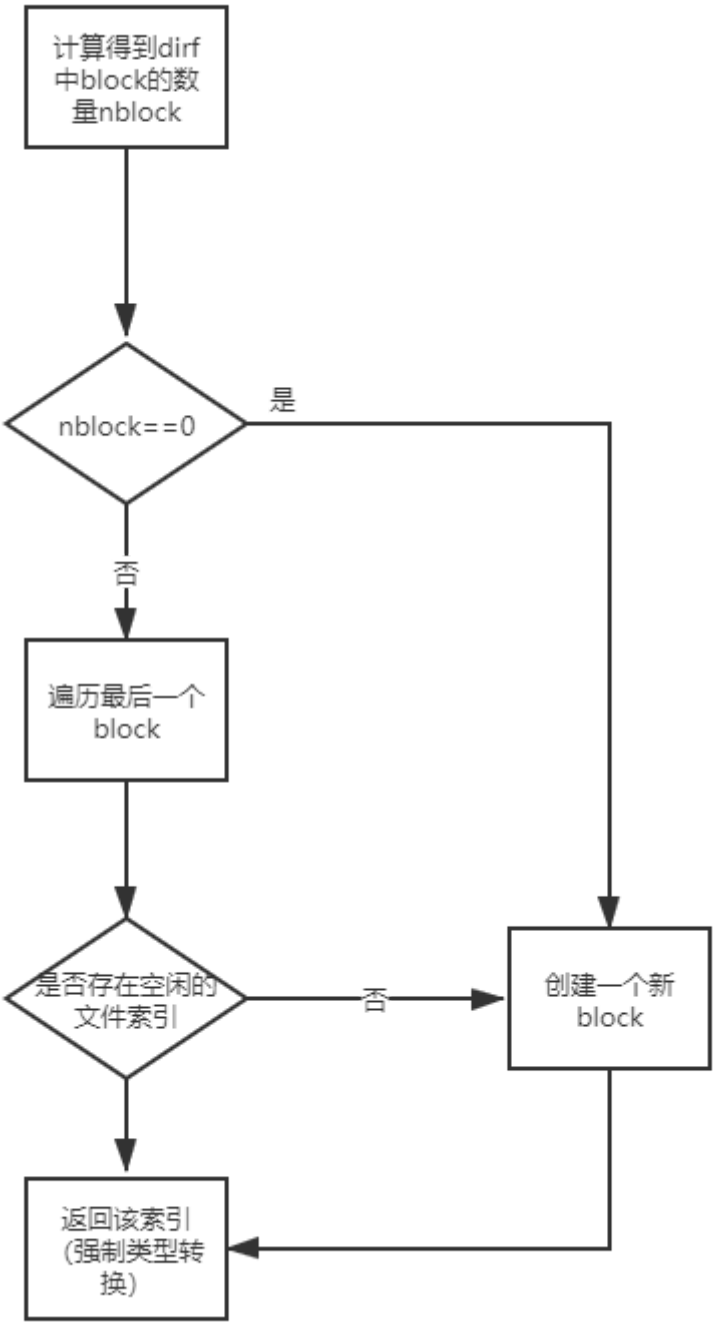
思考题

- 1
  - proc文件系统是一个伪文件系统，它只存在内存当中，而不占用外存空间。它以文件系统的方式为访问系统内核数据的操作提供接口。
  - 用户和应用程序可以通过proc得到系统的信息，并可以改变内核的某些参数。由于系统的信息，如进程，是动态改变的，所以用户或应用程序读取proc文件时，proc文件系统是动态从系统内核读出所需信息并提交的。
  - Windows通过Win32 API的调用实现类似的功能
  - 好处是使用户层面将内核视为文件，简化了交互过程
- 2
  - 会引起Cache空间的浪费
  - 我们需要与一些设备例如console进行实时交互，而如果经过Cache的话就不会有实时交互行为，写入Cache的数据需要等到数据替换的时候才会被写入console中
- 3
  - 10个直接指针，一个指针指向一个磁盘块，一个磁盘块4KB，能够表示40KB的文件
  - 间接指针指向一个磁盘块，一个磁盘块最多存储 1024 个指向其他磁盘块的指针，其中前10个指针不使用，所以能够表示1014 * 4KB = 4056KB的文件
  - 所以单个文件最大为 4096kKB = 4MB
- 4
  - 一个磁盘块4KB，一个FCB 256B， 最多能存储 4KB / 256B = 16个
  - 一个目录直接指针和间接指针共有1024个，16 * 1024 = 16K
- 5
  - 因为文件进程是一个用户进程，所以在用户态下可用的最大地址是ULIM，所以最大的磁盘大小是 ULIM - DISKMAP = 0x70000000 B = 1792MB
- 6
  - 不能，内核态空间被占用
- 7
  - Filefd结构体中包含了一个Fd结构体，而且Fd结构体就在Filefd的第一个位置，所以从地址上来说这样转换是没有错的；还可以认为Filefd是Fd的子类，Filefd在Fd的基础上进行了数据拓展
- 8
  - Fd结构体
    - fd_dev_id：设备的编号
    - fd_offset：读或写的偏移量
    - fd_omode： 打开权限，可读/可写等
  - Filefd结构体
    - f_fd：一个Fd类型的结构体
    - f_fileid：文件的id
    - f_file：一个File类型的结构体，文件
  - Open结构体
    - o_file：打开文件的映射描述符
    - o_fileid： 文件id
    - o_mode： 打开方式（权限）
    - o_ff：filefd页的地址

实验难点

在此次的lab5实验中，我认为最难的函数是creat_file，总结原因如下：第一这个函数没有任何的参照不向后面的read和write函数可以互相参照，简单修改就可以实现功能转换，；第二指导书和代码上的注释说的感觉也不是很清晰；第三刚进行到这里时还没有对文件系统的构成有一个清晰的认识，对需要用到的数据结构和被调用的函数也不了解。。。基于较多原因感觉这个函数的难度较大，花费的时间也比较长
下面是总结的函数基本架构：



回过头来发现这个函数其实并没有那么复杂

- 通过f_size / BY2BLK 计算dirf有多少个block
- 如果block大于0，那么遍历最后一个block，查找有没有空闲的文件索引（可以通过dirblk[index].f_name[0] == '\0'来判断），有则返回
- 如果block等于0 或者没有找到空闲的文件索引，那么需要通过make_link_block来新创建一个block返回
- 返回时要注意将返回值转换成File类型的指针 (struct File *)disk[bno].data
  完成这个函数之后感觉对文件系统有了一个比较清晰的认识

实验感想

- 此次的lab5难度感觉就刚刚好，有一些难点，但是稍加思考或者和同学交流之后就能很好的解决，但是又不会像lab4那样困难重重。
- 通过向指定的地址写入数据从而达到和设备交互的目的，这一点在lab1实现printf是就给我留下了很深的印象，此次又加入了console、磁盘等设备，更加深入理解了文件系统与设备之间交互的过程
- 实现了lab5才知道，文件系统自己本身也是一个进程，其他的用户进程需要通过进程间通信的方式来和文件系统进行通信从而实现对文件系统的一些操作

EXTRA

此次的lab5-extra的难度比较适中，花费了大半天的时间。通过实现console中断，我梳理了一下中断的实现方式，有了较深的印象。