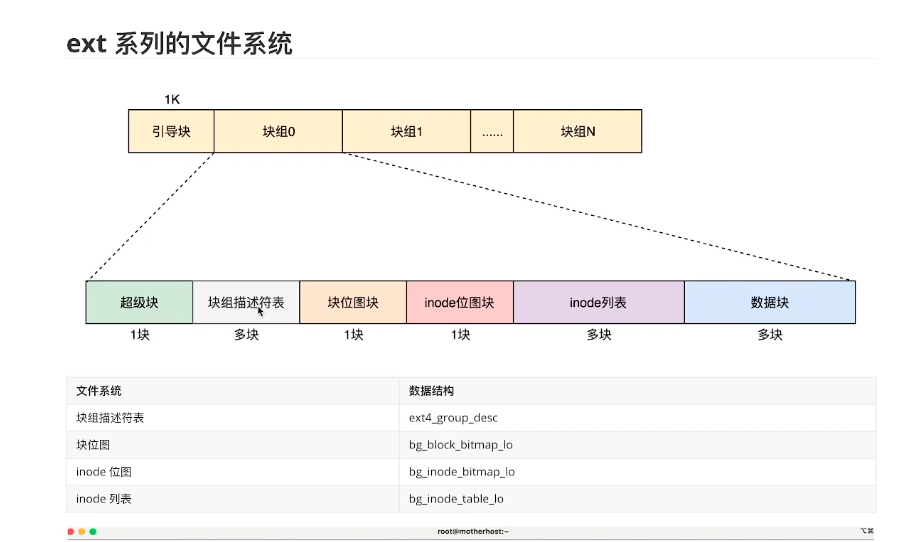
EXT4文件系统的标准磁盘布局如下：



一个的Ext4文件系统被分成一系列块组。为减少磁盘碎片产生的性能瓶颈，块分配器尽量保持每个文件的数据块都在同一个块组中，从而减少寻道时间。以4KB的数据块为例，一个块组可以包含32768个数据块，也就是128MB。

超级块，GDT，块位图，索引节点位图都是整个文件系统的元数据。

文件系统注册：

module\_init(ext4\_init\_fs)

Linux下一切皆文件，共有七种文件类型

-：普通文件        d：目录

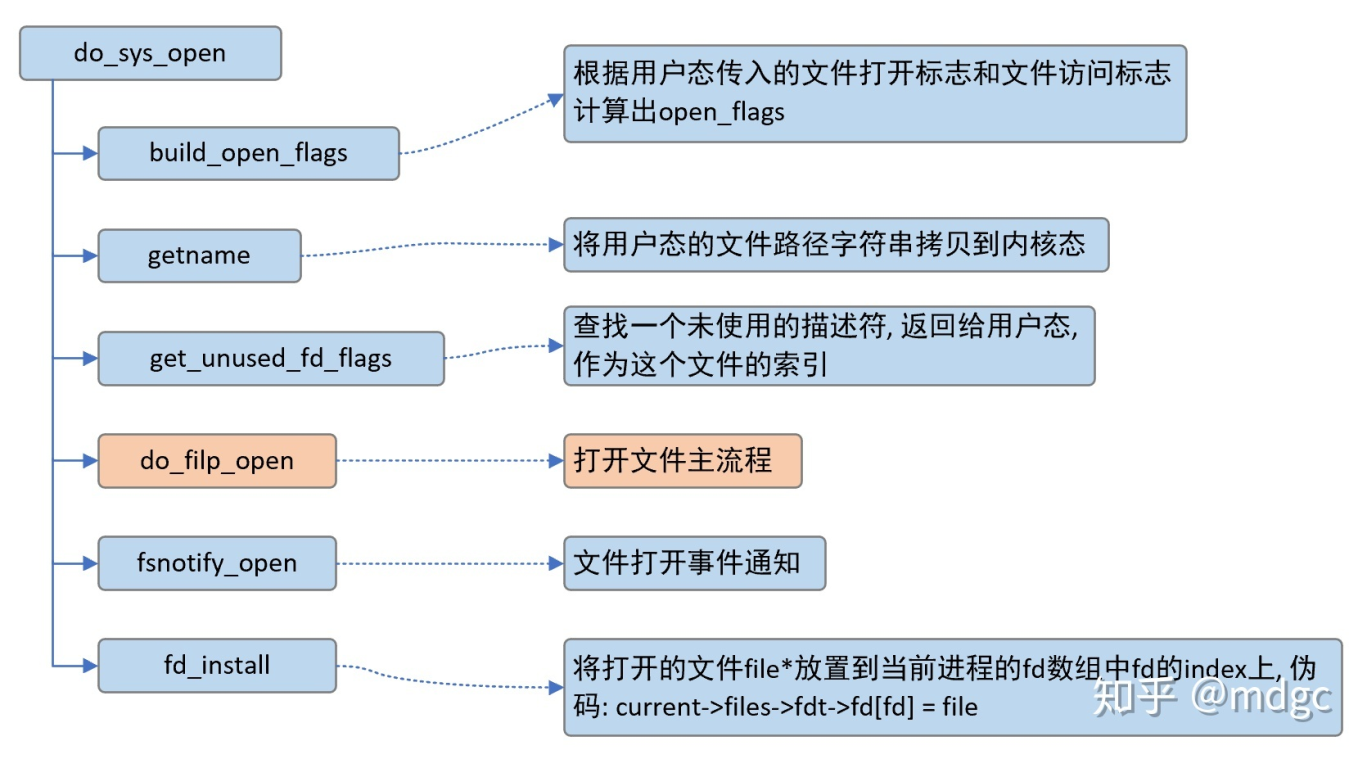
c：字符设备      b：块设备

s：套接字          p：管道          l：软链

1. 打开文件(open函数)

在操作文件的时候，在用户态看到的是**文件描述符fd**（一个整数）。

* 1. **do\_sys\_open的流程**

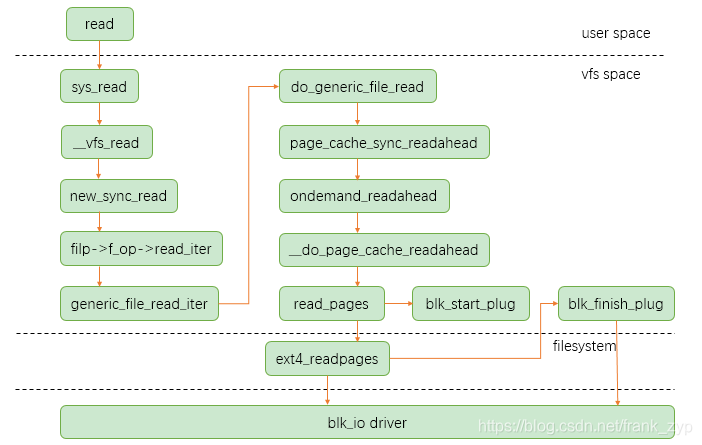


* 1. **do\_filp\_open的流程**

Open函数：

SYSCALL\_DEFINE3(open, const char \_\_user \*, filename, int, flags, umode\_t, mode)

read函数系统调用图谱：



SYSCALL\_DEFINE3(read, unsigned int, fd, char \_\_user \*, buf, size\_t, count)

loff\_t pos = file\_pos\_read(f.file); //获取文件的位置

ret = vfs\_read(f.file, buf, count, &pos);

ret = rw\_verify\_area(READ, file, pos, count); // read or write verify

ret = \_\_vfs\_read(file, buf, count, pos);

new\_sync\_read(file, buf, count, pos)

file->f\_op->read\_iter(&kiocb, &iter) //ext4\_file\_operations

走ext4 的generic\_file\_read\_iter 函数：

generic\_file\_read\_iter(&kiocb, &iter)

mapping->a\_ops->direct\_IO(iocb, &data, pos); //direct io

do\_generic\_file\_read(file, ppos, iter, retval)

page = find\_get\_page(mapping, index); //在radix树中查找相应的page

page\_cache\_sync\_readahead(mapping,ra, filp,index, last\_index - index); //没有page cache中，进行预读