**Linux虚拟文件系统四大对象**

**超级块、inode、dentry、file之间关系**

**一、文件系统**

1、什么是文件系统

操作系统中负责管理和存储文件信息的软件机构称为文件管理系统，简称文件系统。

通常文件系统是用于存储和组织文件的一种机制，便于对文件进行方便的查找与访问。

文件系统是对文件存储设备的空间进行组织和分配，负责文件存储并对存入的文件进行保护和检索的系统。它负责为用户建立文件，存入、读出、修改、转储文件，控制文件的存取，当用户不再使用时撤销文件等。随着文件种类的增多，扩增了更多的文件系统，为了对各种文件系统进行统一的管理与组织。

2、Linux文件系统

Linux将文件系统分为了两层：VFS（虚拟文件系统）、具体文件系统，如下图所示：



VFS（Virtual Filesystem Switch）称为虚拟文件系统或虚拟文件系统转换，是一个内核软件层，在具体的文件系统之上抽象的一层，用来处理与Posix文件系统相关的所有调用，表现为能够给各种文件系统提供一个通用的接口，使上层的应用程序能够使用通用的接口访问不同文件系统，同时也为不同文件系统的通信提供了媒介。

VFS并不是一种实际的文件系统，它只存在于内存中，不存在任何外存空间，VFS在系统启动时建立，在系统关闭时消亡。

VFS由**超级块、inode、dentry、vfsmount**等结构来组成。

Linux系统中存在很多的文件系统，例如常常见的：ext2、ext3、ext4、sysfs、rootfs、proc…等。

**二、VFS**

1、VFS在linux架构中的位置

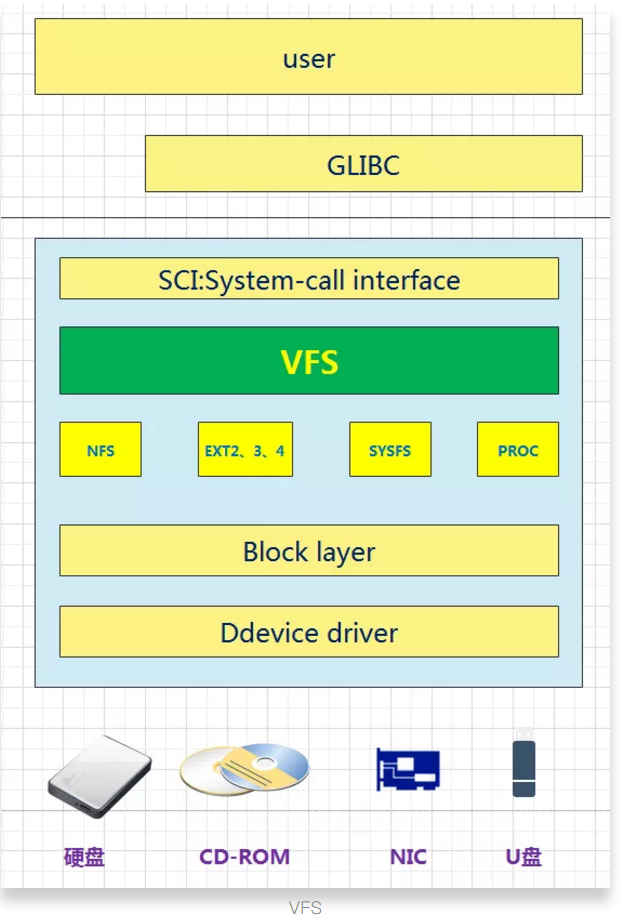
从用户的使用角度，Linux下的文件系统中宏观上主要分为三层：

1.上层的文件系统的系统调用（System-call ）；

2.虚拟文件系统VFS(Virtual File System)层，

3.挂载到VFS中的各种实际文件系统。

VFS在整个Linux系统中的架构视图如下：



Linux系统的User使用GLIBC（POSIX标准、GUN C运行时库）作为应用程序的运行时库，然后通过操作系统，将其转换为系统调用SCI（system-call interface），SCI是操作系统内核定义的系统调用接口，这层抽象允许用户程序的I/O操作转换为内核的接口调用。

2、用户如何透明的去处理文件？

我们知道每个文件系统是独立的，有自己的组织方法，操作方法。那么对于用户来说，不可能所有的文件系统都了解，那么怎么做到让用户透明的去处理文件呢？

例如：我想写文件，那就直接read就OK，不管你是什么文件系统，具体怎么去读！这里就需要引入虚拟文件系统。

所以虚拟文件系统就是：对于一个system，可以存在多个“实际的文件系统”，例如：ext2，ext3，fat32，ntfs...例如我现在有多个分区，对于每一个分区我们知道可以是不同的“实际文件系统”。

例如现在三个磁盘分区分别是：ext2，ext3，fat32，那么每个“实际的文件系统”的操作和数据结构肯定不一样，那么，用户怎么能透明使用它们呢？

这个时候就需要VFS作为中间一层！用户直接和VFS打交道。

VFS是一种软件机制，只存在于内存中，每次系统初始化期间Linux都会先在内存中构造一棵VFS的目录树（也就是源码中的namespace）。

VFS主要的作用是对上层应用屏蔽底层不同的调用方法，提供一套统一的调用接口，二是便于对不同的文件系统进行组织管理。

VFS提供了一个抽象层，将POSIX API接口与不同存储设备的具体接口实现进行了分离，使得底层的文件系统类型、设备类型对上层应用程序透明。

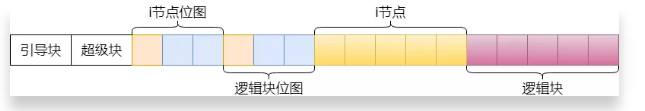
例如read，write，那么映射到VFS中就是sys\_read，sys\_write，那么VFS可以根据你操作的是哪个“实际文件系统”(哪个分区)来进行不同的实际的操作！这个技术也是很熟悉的“钩子结构”技术来处理的。

其实就是VFS中提供一个抽象的struct结构体，然后对于每一个具体的文件系统要把自己的字段和函数填充进去，这样就解决了异构问题（内核很多子系统都大量使用了这种机制）。

**三、Linux虚拟文件系统四大对象**

为了对文件系统进行统一的管理与组织，Linux创建了一个公共根目录和全局文件系统树。要访问一个文件系统中的文件，必须先将这个文件系统挂载在全局文件系统树的某个根目录下，这一挂载过程被称作文件系统的挂载，所挂载的目录称为挂载点。

传统的文件系统在磁盘上的布局如下：



由上图可知，文件系统的开头通常是由一个磁盘扇区所组成的引导块，该部分的主要目的是用于对操作系统的引导。一般只在启动操作系统时使用。

随后是超级块，超级块主要存放了该物理磁盘中文件系统结构的相关信息，并且对各个部分的大小进行说明。

最后由i节点位图，逻辑块位图、i节点、逻辑块这几部分分布在物理磁盘上。

Linux为了对超级块，i节点，逻辑块这三部分进行高效的管理，Linux创建了几种不同的数据结构，分别是**文件系统类型、inode、dentry**等几种。

其中，文件系统类型规定了某种文件系统的行为，利用该数据结构可以构造某种文件系统类型的实例，另外，该实例也被称为超级块实例。

超级块则是反映了文件系统整体的控制信息。超级块能够以多种的方式存在，对于基于磁盘的文件系统，它以特定的格式存在于磁盘的固定区域（取决于文件系统类型）上。在挂载文件系统时，该超级块中的内容被读入磁盘中，从而构建出位于内存中的新的超级块。

inode则反映了文件系统对象中的一般元数据信息。

dentry则是反映出某个文件系统对象在全局文件系统树中的位置。

1、超级块

超级块：一个超级块对应一个文件系统(已经安装的文件系统类型如ext2，此处是实际的文件系统，不是VFS)。

之前我们已经说了文件系统用于管理这些文件的数据格式和操作之类的，系统文件有系统文件自己的文件系统，同时对于不同的磁盘分区也有可以是不同的文件系统。那么一个超级块对于一个独立的文件系统。保存文件系统的类型、大小、状态等等。

(“文件系统”和“文件系统类型”不一样！一个文件系统类型下可以包括很多文件系统即很多的super\_block)。既然我们知道对于不同的文件系统有不同的super\_block，那么对于不同的super\_block的操作肯定也是不同的。

2、索引节点

索引节点inode：保存的其实是实际数据的一些信息，这些信息称为“元数据”(也就是对文件属性的描述)。

例如：文件大小，设备标识符，用户标识符，用户组标识符，文件模式，扩展属性，文件读取或修改的时间戳，链接数量，指向存储该内容的磁盘区块的指针，文件分类等等。( 注意数据分成：元数据+数据本身 )

同时注意：inode有两种，一种是VFS的inode，一种是具体文件系统的inode。前者在内存中，后者在磁盘中。所以每次其实是将磁盘中的inode调进填充内存中的inode，这样才是算使用了磁盘文件inode。

inode怎样生成的?

每个inode节点的大小，一般是128字节或256字节。inode节点的总数，在格式化时就给定(现代OS可以动态变化)，一般每2KB就设置一个inode。

一般文件系统中很少有文件小于2KB的，所以预定按照2KB分，一般inode是用不完的。所以inode在文件系统安装的时候会有一个默认数量，后期会根据实际的需要发生变化。

注意inode号：inode号是唯一的，表示不同的文件。其实在Linux内部的时候，访问文件都是通过inode号来进行的，所谓文件名仅仅是给用户容易使用的。

当我们打开一个文件的时候，首先，系统找到这个文件名对应的inode号；然后，通过inode号，得到inode信息，最后，由inode找到文件数据所在的block，现在可以处理文件数据了。

inode和文件的关系?

当创建一个文件的时候，就给文件分配了一个inode。一个inode只对应一个实际文件，一个文件也会只有一个inode。inodes最大数量就是文件的最大数量。

3、目录项(dentry)

目录项:是描述文件的逻辑属性，只存在于内存中，并没有实际对应的磁盘上的描述，更确切的说是存在于内存的目录项缓存，为了提高查找性能而设计。

注意不管是文件夹还是最终的文件，都是属于目录项，所有的目录项在一起构成一颗庞大的目录树。

例如：open一个文件/home/xxx/yyy.txt，那么/、home、xxx、yyy.txt都是一个目录项，VFS在查找的时候，根据一层一层的目录项找到对应的每个目录项的inode，那么沿着目录项进行操作就可以找到最终的文件。

注意：目录也是一种文件(所以也存在对应的inode)。打开目录，实际上就是打开目录文件。

一个有效的dentry结构必定有一个inode结构，这是因为一个目录项要么代表着一个文件，要么代表着一个目录，而目录实际上也是文件。所以，只要dentry结构是有效的，则其指针d\_inode必定指向一个inode结构。但是inode却可以对应多个。

整个结构其实就是一棵树，目录其实就是文件（kobject、inode）再加上一层封装，这里所谓的封装主要就是增加两个指针，一个是指向父目录，一个是指向该目录所包含的所有文件（普通文件和目录）的链表头。

这样才能有我们的目录操作（比如回到上次目录，只需要一个指针步骤【..】，而进入子目录需要链表索引需要多个步骤）

dentry相关的操作（inode里面已经包含了mkdir，rmdir，mknod之类的操作了）。

4、文件对象(file)

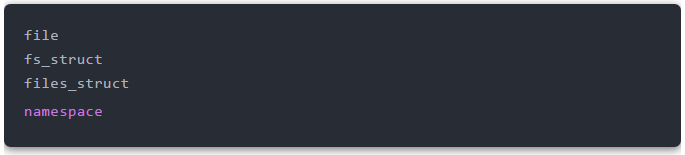
文件对象:描述的是进程已经打开的文件。因为一个文件可以被多个进程打开，所以一个文件可以存在多个文件对象。但是由于文件是唯一的，那么inode就是唯一的，目录项也是定的！

进程其实是通过文件描述符来操作文件的，每个文件都有一个32位的数字来表示下一个读写的字节位置，这个数字叫做文件位置。

一般情况下打开文件后，打开位置都是从0开始，除非一些特殊情况。Linux用file结构体来保存打开的文件的位置，所以file称为**打开的文件描述**。file结构形成一个双链表，称为系统打开文件表。

四、进程与四者之间的关系

内核中用于管理进程的结构体是task\_struct。进程打开文件就涉及到上述4个重要的数据结构:



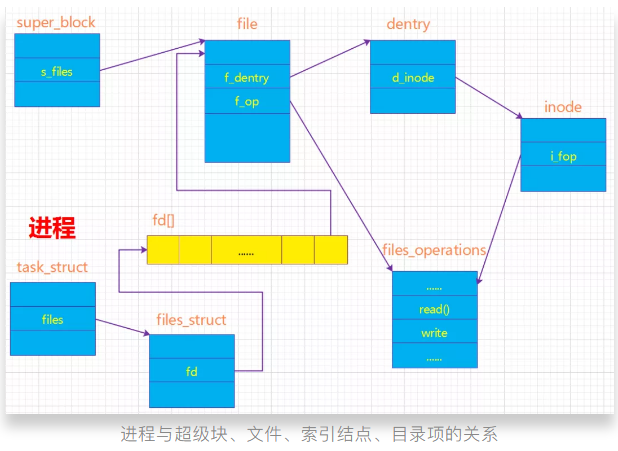
每个进程都有自己的namespace。

fs\_struct用于表示进程与文件系统之间的结构关系,比如当前的工作目录,进程的根目录等等。

files\_struct 用于表示当前进程打开的文件。

而对于每一个打开的文件,由file对象来表示。

Linux中,常常用文件描述符(file descriptor)来表示一个打开的文件,这个描述符的值往往是一个大于或等于0的整数。而这个整数,其实就是在files\_struct中file数组fd的下标。对于所有打开的文件, 这些文件描述符会存储在open\_fds的位图中。



从图中可知：

进程通过task\_struct中的一个域files->files\_struct 来了解它当前所打开的文件对象；而我们通常所说的文件描述符其实是进程打开的文件对象数组的索引值。

文件对象通过域f\_dentry找到它对应的dentry对象，再由dentry对象的域d\_inode找到它对应的索引节点（通过索引节点又可以得到超级块的信息，也就可以得到最终操作文件的方法，在open文件的时候就是使用这样一个过程），这样就建立了文件对象与实际的物理文件的关联。

文件对象所对应的文件操作函数列表是通过索引节点的域i\_fop得到的，而i\_fop最终又是通过struct super\_operations  \*s\_op来初始化的。

VFS文件系统中的inode和dentry与实际文件系统的inode和dentry有一定的关系，但不能等同。

真实磁盘文件的inode和dentry是存在于物理外存上的，但VFS中的inode和dentry是存在于内存中的，系统读取外存中的inode和dentry信息进行一定加工后，生成内存中的inode和dentry。

虚拟的文件系统也具有inode和dentry结构，只是这是系统根据相应的规则生成的，不存在于实际外存中。

**五、磁盘与系统**

假设一块磁盘被分为好几个分区，每个分区都是不同的文件系统。

