VFS中的数据结构（superblock、dentry、inode、file）

2018年4月14日 | Linux

VFS简介

VFS的理念是使用统一的数据结构在内核中保存不同类型文件系统的信息（含操作）。

VFS是一个介于用户程序和文件系统实现之间的一个抽象层，VFS既给了不同类型的文件系统支持Linux系统的公共接口，也给用户程序提供了一个统一的调用接口。

VFS背后的核心idea是引入了通用文件模型（common file model），该模型其实就是Unix原生文件系统的一个镜像（这样可以在Unix原生文件系统了具有最小的开销，最高的性能）。其他非原生文件系统（比如FAT、MS-DOS filesystem）则需要做一些转换，从而支持该通用文件模型

通用文件模型（common file model）

通用文件模型由superblock、inode、dentry、file四种数据结构构成，这四种数据结构都是保存在内核空间中的。

- superblock数据结构存储了挂载的文件系统的元信息，对于基于磁盘的文件系统，通常在磁盘上有一个对应的superblock（比如ext2的superblock）。
- inode数据结构用于保存文件的元信息，一个文件对应一个inode，每个inode都有一个inode number唯一标识。inode通常对应磁盘上保存的文件控制块（file control block）
- dentry数据结构用于将inode（表示文件）和目录项（表示文件路径）关联起来。inode中是没有文件路径的，所以需要dentry来讲文件路径和文件关联起来。不同的磁盘文件系统使用各自的方式来存储dentry
- file数据结构用于存储进程和打开的文件之间交互的信息，这个信息只会存在于内核空间中，在磁盘上没有对应的信息。

下面将分别介绍这4种数据结构，所列举的信息使用的是2.6内核的数据。

superblock数据结构

superblock是文件系统的metadata，保存了文件系统的各种信息以及可以对其执行的操作。

包含的域（仅列举部分）：

类型	域名	描述
int	s_type	文件系统类型
unsigned long	s_blocksize	块大小（block size）
struct dentry *	s_root	指向文件系统根目录对应的dentry
struct list_head	s_inodes	文件系统中所有文件的inode（使用list_head双向链表存储）
void *	s_fs_info	指向具体文件系统实现（如ext2）的特有的数据结构
struct superblock*	s_op	superblock的操作函数（结构体里都是函数指针）

superblock operations（存储在s_op中，仅列举部分）：

函数	功能
alloc_inode(sb)	为一个inode对象分配空间
destroy_inode(inode)	销毁一个inode对象
read_inode(inode)	从磁盘中读取inode数据，填充作为传入参数的inode对象
write_inode(inode, flag)	使用内存中的inode信息更新磁盘中的inode信息
delete_inode(inode)	删除内存中的inode对象同时删除磁盘上的inode

inode数据结构

inode数据结构中保存了文件系统处理文件所需要的全部信息以及可以对其执行的操作。一个inode对应磁盘上一个实际的文件（目录是一种特殊的文件）

inode中没有直接存储文件的每个块的位置，根据ext2的的Data Blocks Addressing，下表中的i_blocks和i_bytes应该是共同规定了文件的file block number（从0开始计数）和文件结束位置。下层的文件系统（如ext2）负责将file block number转换为logical block numbe（在磁盘上的block地址）

包含的域（仅列举部分）：

类型	域名	描述
struct super_block *	i_sb	指向inode所在的superblock对象

类型	域名	描述
struct list_head	i_dentry	这是一个双向链表的头节点，链表中保存的是指向该inode的dentry对象
unsigned long	i_ino	inode编号
umode_t	i_mode	文件类型和访问权限域
unsigned int	i_nlink	指向该inode的硬链接数量，为0时意味着该inode要销毁了
uid_t	i_uid	inode所有者的id
struct timespec	i_atime	上次访问的时间戳
unsigned long	i_blocks	文件的块数目
unsigned short	i_bytes	文件最后一个块的字节大小
struct inode_operations *	i_op	inode operations,inode的操作函数

inode operations(存储在i_op中，仅列举部分)

函数	功能
create(dir, dentry, mode, nameidata)	创建一个inode
lookup(dir, dentry, nameidata)	在一个目录文件中查找和dentry包含的文件名匹配的inode
link(old_dentry, dir, new_dentry)	创建一个指向new_dentry的硬链接，保存在old_dentry中，该old_dentry和new_dentry指向同一个inode，即同一个文件。
symlink(dir, dentry, symname)	创建一个新的inode，该inode是一个软连接文件，指向参数dentry
mkdir(dir, dentry, mode)	为dentry创建一个目录文件的inode

dentry数据结构

dentry数据结构用于将inode（表示文件）和目录项（表示文件路径）关联起来。inode中是没有文件路径的，所以需要dentry来将文件路径和文件关联起来。dentry在磁盘中没有对应的镜像，所以不需要考虑该dentry是否需要更新。

内核在进行路径查找的时候会为每一级都建立一个dentry，比如/home/damon/cppfile/test.cpp。就需要建立5个dentry（第一个是根目录/，第二个是home,第5个是test.cpp）。

dentry cache将已经查找过的路径缓存在内存中，这样下次查找的时候就不需要重新读取每一级的目录文件进行查找，可以直接通过dentry cache获得对应的inode

包含的域（仅列举部分）

类型	域名	描述
atomic_t	d_count	dentry对象的使用计数器
struct inode *	d_inode	dentry指向的inode
struct dentry *	d_parent	指向上级目录的dentry
struct qstr	d_name	文件名
struct list_head	d_subdirs	如果当前dentry是目录的dentry，那么双向链表保存的是所有子目录的dentry
struct super_block *	d_sb	dentry对应的super block
struct dentry_operations*	d_op	Dentry methods，dentry的操作函数

dentry operations(存储在d_op中，仅列举部分)

函数	功能
d_revalidate(dentry, nameidata)	判断当前dentry对象是仍然有效，应该是dentry cache中使用的
d_hash(dentry, name)	计算哈希值，应该也是dentry cache中使用的
d_delete(dentry)	在d_count为0时，删除dentry，默认的VFS函数什么都不做

file数据结构

file数据结构用于存储进程和打开的文件之间交互的信息，这个信息只会存在于内核空间中，在磁盘上没有对应的信息。

包含的域（仅列举部分）

类型	域名	描述
struct dentry *	f_dentry	和该file对应的dentry
struct vfsmount *	f_vfsmnt	file所在的文件系统（文件系统挂载的数据结构）
unsigned int	f_flags	打开文件时使用的flag

类型	域名	描述
mode_t	f_mode	进程access mode
struct file_operations *	f_op	file operations，文件操作函数

dentry operations(存储在d_op中，仅列举部分)

函数	功能
open(inode, file)	打开文件
llseek(file, offset, origin)	移动文件指针
read(file, buf, count, offset)	读文件
write(file, buf, count, offset)	写文件

参考资料

《Understanding the Linux Kernel 3rd Edition》

What is a Superblock, Inode, Dentry and a File?

从文件 I/O 看 Linux 的虚拟文件系统

See Also

- Linux进程状态码总结
- 《C和C++程序员面试秘笈》中存在的错误
- 最小生成树(MST)：Prim算法与Kruskal算法
- 二叉树的遍历（先序遍历，中序遍历，后序遍历，层次遍历）
- Trie，单词查找树

Linux 文件系统

0 Comments - *powered by utteranc.es*