EECS3221 Section E
Operating System Fundamentals
Fall 2020 Project

Components in Linux Kernel Subsystems
Virtual File System
Kernel Version: 5.9.0
Linux Kernel Source Code: https://github.com/torvalds/linux/tree/v5.9

Kevin Shen
212298535

Table of Contents

Introduction

The Virtual File System (VFS) is designed for high speed, low latency access to files, data loss and corruption prevention, and access restrictions to unauthenticated users. The VFS is able to support multiple physical devices with different file systems under the Linux kernel. It does this by providing a file system interface to both the userspace and the kernel to allow multiple different file systems to coexist. The VFS includes a system call interface that is available at the user level which allows users to interact with files and directories. [1] There are several internal interface at the kernel level that interact with the kernel's data structures and provides functions for the other kernel subsystems. Another vital component in the VFS is the ability to mount external file systems. This interface allows different file systems to communicate with the kernel and enables it to be accessed by the kernel.

System Call Interface

The System Call interface provides users an interface to interact with files and directories within the VFS. It is designed to be compatible with POSIX compliant systems which allows for access without the need of external libraries or drivers. Because POSIX is a IEEE standard, most devices which follows the standard, will be compatible with the Linux system. POSIX provides a common definition for interfaces, system functions, subroutines and command shells. [4] This is illustrated in Figure 1. The system call interface uses the POSIX definitons for users to interact with its file systems and directories. It provides users the file operations: open, close, read, write, seek and tell, and the directory operations: readdir, creat, unlink, chmod and stat. [1]

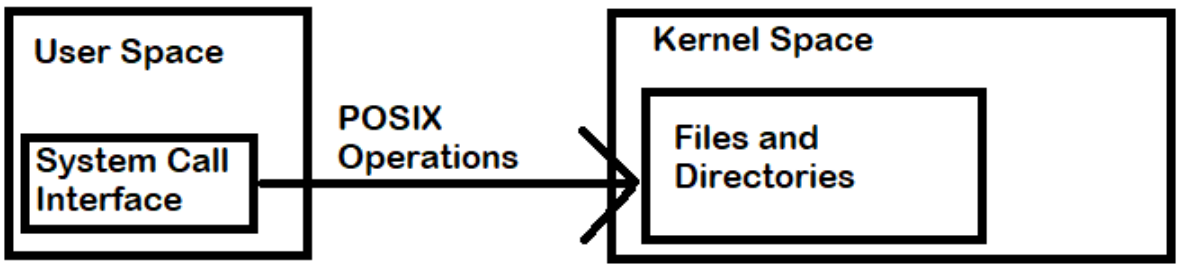

Figure 1: Diagram illustrating POSIX Operations from the System Call interface

Internal Interface – Inode

The Inode (also known as Index Node) interface exists on the kernel level and it provides the other kernel subsystems access to the data structures and functions within the VFS. It consists of the inode and the inode_operatons data structures, which contains the properties of a file, such as file size, ownership and index. [5] Shown in Figure 2.

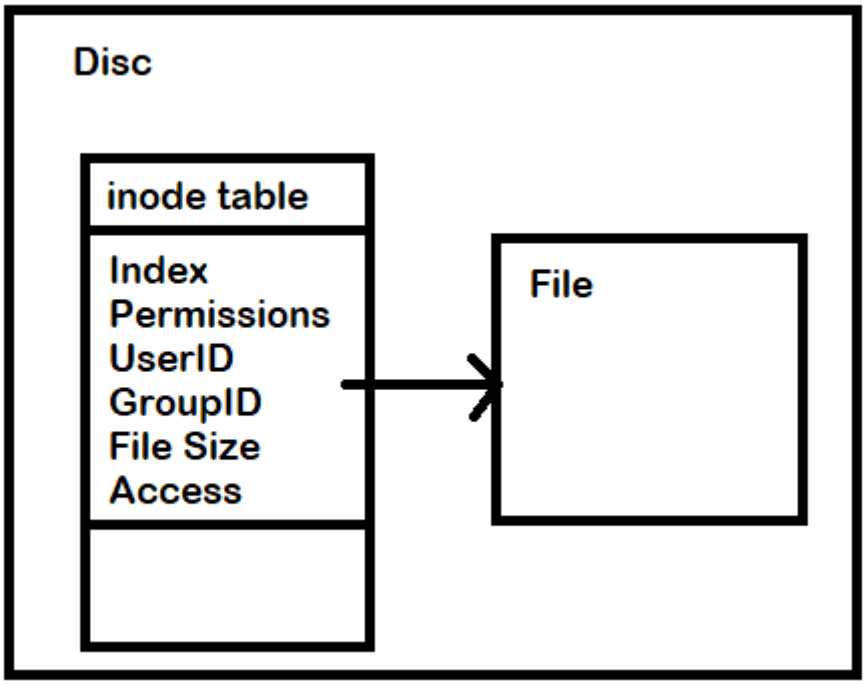

Figure 2: Diagram illustrating the inode structure

The inode_operations contains the functions for inodes. The inode structure is defined in **/linux/fs.h line 615** and the **inode_operations** structure is defined in **/linux/fs.h line 1843**. Every file on the system requires an inode object to be created for it, which contains all information associated to it in a single object. The inode objects are all stored onto disc (or block device) and are only copied to memory when called, and are then written back to disc afterwards. [5] Every inode will have a unique index number known as a the Inode Number. This inode number can be used by the file system when searching for a specific inode by mapping a pathname to an inode number with a dentry. [6]

Internal Interface – Superblock

The superblock is an object that stores the properties of an entire file system. The **super_block** structure is defined in **/linux/fs.h line 1415** and the **super_operations** structure is defined in **/linux/fs.h line 1921**. Some properties it stores are: block size, empty blocks, filled blocks, size and location of the inode tables.[2] It is similar to an inode where it acts as a virtual interface for file systems. When a request is sent to a file system, it will need to access the superblock; if the superblock cannot be accessed then neither the files will be accessible. Due to the importance of the superblock, backups of it are made periodically to prevent corruption. The Linux kernel will create a backup of the superblock and store it in memory. [2]

Internal Interface – Directory Entry

When an inode is created a Directory Entry (dentry) is also created along with it. A dentry is a part of the VFS that has a pointer to an inode and it is mainly used for file system traversal and path name lookups. The **dentry** structure is defined in **/linux/ dcache.h line 89** and the **dentry_operations** is defined in **/linux/dcache.h line 135**. The dentry will associate a pathname to a single inode. To speed up the inode lookup process, the VFS has a Directory Entry Cache (dcache) which stores dentrys in memory. [1] By storing the dentry in memory, it offers a much faster lookup, (due to the RAM being significantly faster than disc), but there are storage limitations so not every dentry can be stored onto the dcache. [7] This is illustrated in Figure 3.

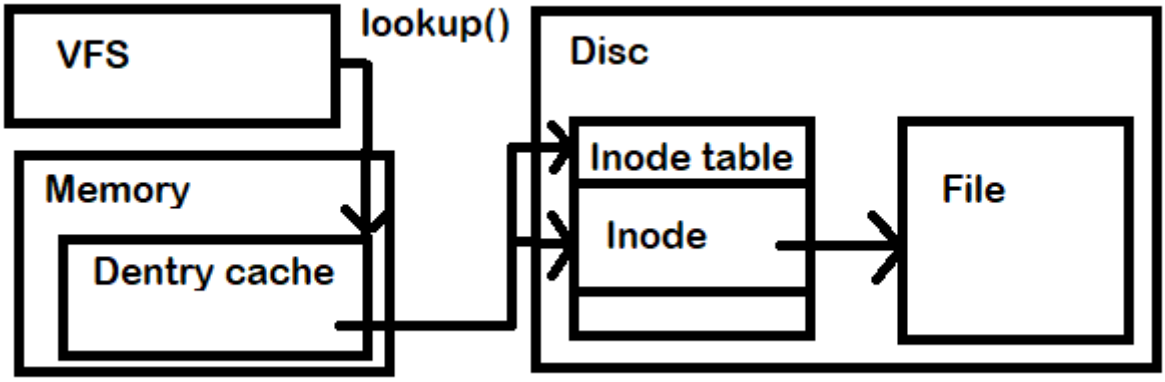

Figure 3: Diagram illustrating Inode lookup()

The dcache will store frequently or recently used dentrys near the top of the cache and when dentrys need to be removed the objects at the bottom are the first to be deleted. If there is no dentry in the dcache for a file, the VFS would have to iterate through all the inodes on disc and create a dentry for the file when found. [7]

Internal Interface – Mount point

The mount point is a special directory to allow for additional file systems to be added to the current file system. When a device is mounted, the mount point becomes the root directory of device's file system. [3] This is illustrated in Figure 4. When the VFS attempts to mount a device, the device provides a *file_system_type* where the VFS would have to call an appropriate *mount()* for that type. The VFS calls *register_filesystem(struct file_system_type *)* when it attempts mount a new file system, this function can be found in **/fs/filesystems.c line 72**. When the VFS wants to unmount a file system, it calls *unregister_filesystem(struct file_system_type *),* this function can be found in **/fs/filesystems.c line 108**. A list of file systems supported by the kernel can be found in **/proc/filesystems**. [5] Accessing a mount is similar to accessing a file, where the pathname is resolved by a dentry. [7]

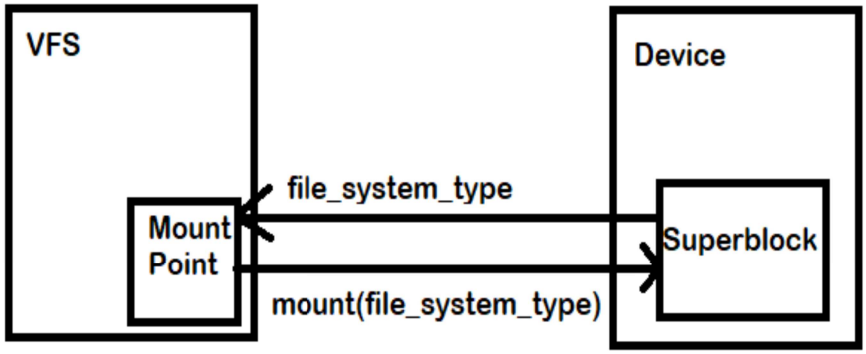

Figure 4: Diagram illustrating Mount point

Device Drivers

Device drivers allow for multiple different file systems to operate on a common interface and this allows the device to be access like a file. These devices have a control and status registers (CSR) which allow the device to send and receive requests to the file system. After the device sends a request, it checks whether its complete by polling, Direct Memory Access (DMA) or interrupts. [1] If a device uses polling, it would periodically check its CSR for whether the request has been fulfilled, then if its completed, it sends another request.

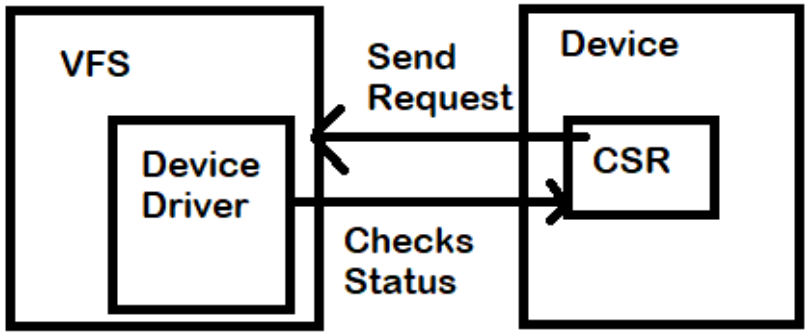

Figure 4: Diagram illustrating polling requests

If the device uses DMA, it would transfer data to a specfied memory region and sends the cpu an interrupt when completed. When the cpu receives an interrupt from a device, it would stop its current instruction and executes the kernel's interrupt handling code. [1]

Source Codes

struct inode : /linux/fs.h line 615
struct inode_operations : /linux/fs.h line 1843
struct super_block : /linux/fs.h line 1415
struct super_operations : /linux/fs.h line 1921
struct dentry : /linux/dcache.h line 89
struct dentry_operations : /linux/dcache.h line 135
int register_filesystem(struct file_system_type *) : /fs/filesystems.c line 72
int unregister_filesystem(struct file_system_type *) : /fs/filesystems.c line 108

References:

[1] Bowman, I., Siddiqi, S., Tanuan, M.C. *Concrete Architecture of the Linux Kernel*, 1998. Available at https://docs.huihoo.com/linux/kernel/a2/

[2] *Superblock Definition.* The Linux Information Project. August 2005, Available at http://www.linfo.org/superblock

[3] *Mount Point Definition.* The Linux Information Project, March 2006, Available at http://www.linfo.org/mount_point.html

[4] Zak, H., *Posix Standard.* Linuxhint, 2017. Available at https://linuxhint.com/posix-standard/

[5] *The Linux Kernel*. The Linux Kernel Archives, July 2020, Available at https://www.kernel.org/doc/html/latest/filesystems/vfs.html

[6] Zachariah, B. *Detailed Understanding of Linux Inodes with Example.* Linoxide. July 2020. Available at https://linoxide.com/linux-command/linux-inode/

[7] Love, R. *Linux Kernel Development Second Edition:The Dentry Object.* Sams Publishing. January 2015. Available at http://books.gigatux.nl/mirror/kerneldevelopment/0672327201/ch12lev1sec7.html