

# OPERATING SYSTEMS

---

## File Management

**Suresh Jamadagni**

Department of Computer Science

# OPERATING SYSTEMS

---

## File-Systems – Case Study: Linux

**Suresh Jamadagni**

Department of Computer Science

# OPERATING SYSTEMS

## Slides Credits for all the PPTs of this course

---

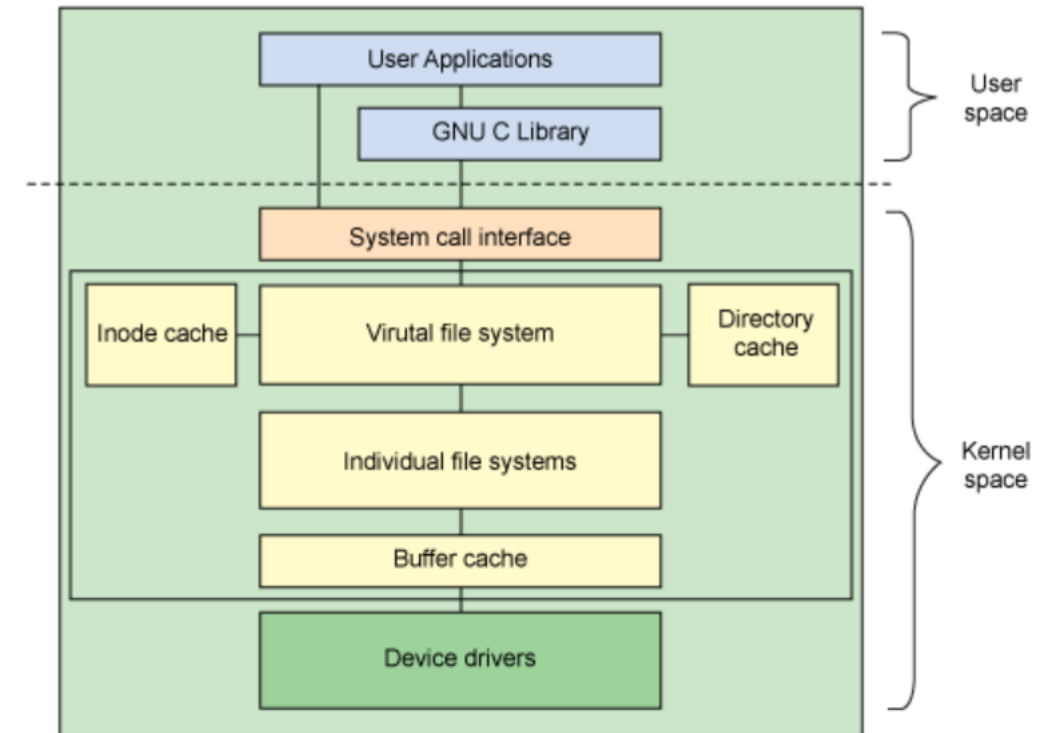


- The slides/diagrams in this course are an **adaptation, combination,** and **enhancement** of material from the following resources and persons:
  1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9<sup>th</sup> edition 2013 and some slides from 10<sup>th</sup> edition 2018
  2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9<sup>th</sup> edition 2018
  3. Some presentation transcripts from A. Frank – P. Weisberg
  4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

# OPERATING SYSTEMS

## Linux File System - Architecture

- User space contains the applications which provides the user interface for the file system calls (open, read, write, close).
- The system call interface acts as a switch, funneling system calls from user space to the appropriate endpoints in kernel space.
- The VFS is the primary interface to the underlying file systems. This component exports a set of interfaces and then abstracts them to the individual file systems, which may behave very differently from one another.
- Two caches exist for file system objects (inodes and dentries). Each provides a pool of recently-used file system objects.



- Each individual file system implementation, such as ext2, JFS, and so on, exports a common set of interfaces that is used by the VFS.
- The buffer cache buffers requests between the file systems and the block devices that they manipulate. For example, read and write requests to the underlying device drivers migrate through the buffer cache. This allows the requests to be cached there for faster access (rather than going back out to the physical device).
- The buffer cache is managed as a set of least recently used (LRU) lists. You can use the sync command to flush the buffer cache out to the storage media (force all unwritten data out to the device drivers and, subsequently, to the storage device).

# OPERATING SYSTEMS

## Linux File System – Common set of objects

---



- Linux views all file systems from the perspective of a common set of objects.
- These objects are the **superblock**, **inode**, **dentry**, and **file**.
- At the root of each file system is the superblock, which describes and maintains state for the file system.
- Every object that is managed within a file system (file or directory) is represented in Linux as an inode.
- The inode contains all the metadata to manage objects in the file system (including the operations that are possible on it).
- Another set of structures, called dentries, is used to translate between names and inodes, for which a directory cache exists to keep the most-recently used around.
- The dentry also maintains relationships between directories and files for traversing file systems.
- A VFS file represents an open file (keeps state for the open file such as the write offset, and so on).

- The superblock is a structure that represents a file system.
- It includes the necessary information to manage the file system during operation.
- It includes the file system name (such as ext2), the size of the file system and its state, a reference to the block device, and metadata information (such as free lists and so on).
- The superblock is typically stored on the storage medium
- Superblock structure is available in `./linux/include/linux/fs.h`.

`current->namespace->list->mnt_sb`      *see Figure 3*

```
struct super_block {  
    struct list_head  
    unsigned long  
    struct file_system_type *s_type;  
    struct super_operations *s_op;  
    struct semaphore  
    int  
    struct list_head  
    struct block_device  
    ...  
};
```

`s_list;` → doubly linked list of all mounted filesystems  
`s_blocksize;`  
`*s_type;` → see Figure 2  
`*s_op;`  
`s_lock;`  
`s_need_sync_fs;`  
`s_dirty;`  
`*s_bdev;`


```
struct super_operations {  
    struct inode *(*alloc_inode)(struct super_block *sb);  
    void (*destroy_inode)(struct inode *);  
    void (*read_inode)(struct inode *);  
    void (*write_inode)(struct inode *, int);  
    int (*sync_fs)(struct super_block *sb, int wait);  
    ...  
};
```

- The inode represents an object in the file system with a unique identifier.
- The individual file systems provide methods for translating a filename into a unique inode identifier and then to an inode reference.
- inode\_operations and file\_operations structures refers to the individual operations that may be performed on the inode, file and directories

```
struct inode {
    unsigned long    i_ino;
    umode_t          i_mode;
    uid_t            i_uid;
    struct timespec  i_atime;
    struct timespec  i_mtime;
    struct timespec  i_ctime;
    unsigned short   i_bytes;
    struct inode_operations *i_op;
    struct file_operations *i_fop;
    struct super_block *i_sb;
    ...
}

struct inode_operations {
    int (*create)(struct inode *, struct dentry *,
                  struct nameidata *);
    struct dentry *(*lookup)(struct inode *,
                             struct dentry *,
                             struct nameidata *);
    int (*mkdir)(struct inode *, struct dentry *, int);
    int (*rename)(struct inode *, struct dentry *,
                  struct inode *, struct dentry *);
    ...
}

struct file_operations {
    struct module *owner;
    ssize_t (*read)(struct file *, char __user *,
                   size_t, loff_t *);
    ssize_t (*write)(struct file *, const char __user *,
                    size_t, loff_t *);
    int (*open)(struct inode *, struct file *);
    ...
}
```





- Buffer cache keeps track of read and write requests from the individual file system implementations and the physical devices (through the device drivers).
- For efficiency, Linux maintains a cache of the requests to avoid having to go back out to the physical device for all requests. Instead, the most-recently used buffers (pages) are cached here and can be quickly provided back to the individual file systems.
- Linux supports a wide range of file systems such as MINIX, MS-DOS, and ext2. Linux also supports the new journaling file systems such as ext3, JFS, and ReiserFS. Additionally, Linux supports cryptographic file systems such as CFS and virtual file system such as /proc.
- Linux also supports **Filesystem in USErspace (FUSE)** filesystem that lets non-privileged users create their own file systems without editing kernel code. This is achieved by running file system code in user space while the FUSE module provides only a bridge to the actual kernel interfaces



**THANK YOU**

---

**Suresh Jamadagni**

Department of Computer Science and Engineering

**[sureshjamadagni@pes.edu](mailto:sureshjamadagni@pes.edu)**