OPERATING SYSTEMS

Storage Management - 10

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

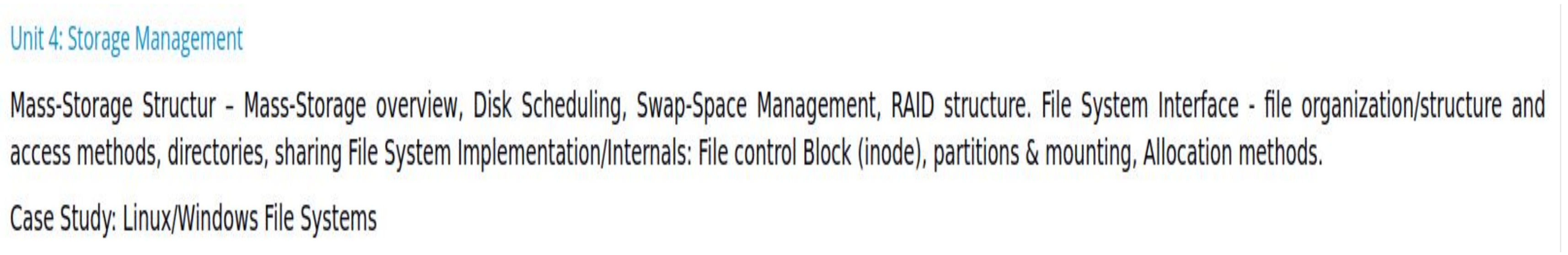
OPERATING SYSTEMS

Storage Management - 10: Case Study: Unix / Linux File System

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

OPERATING SYSTEMS

Course Syllabus - Unit 4



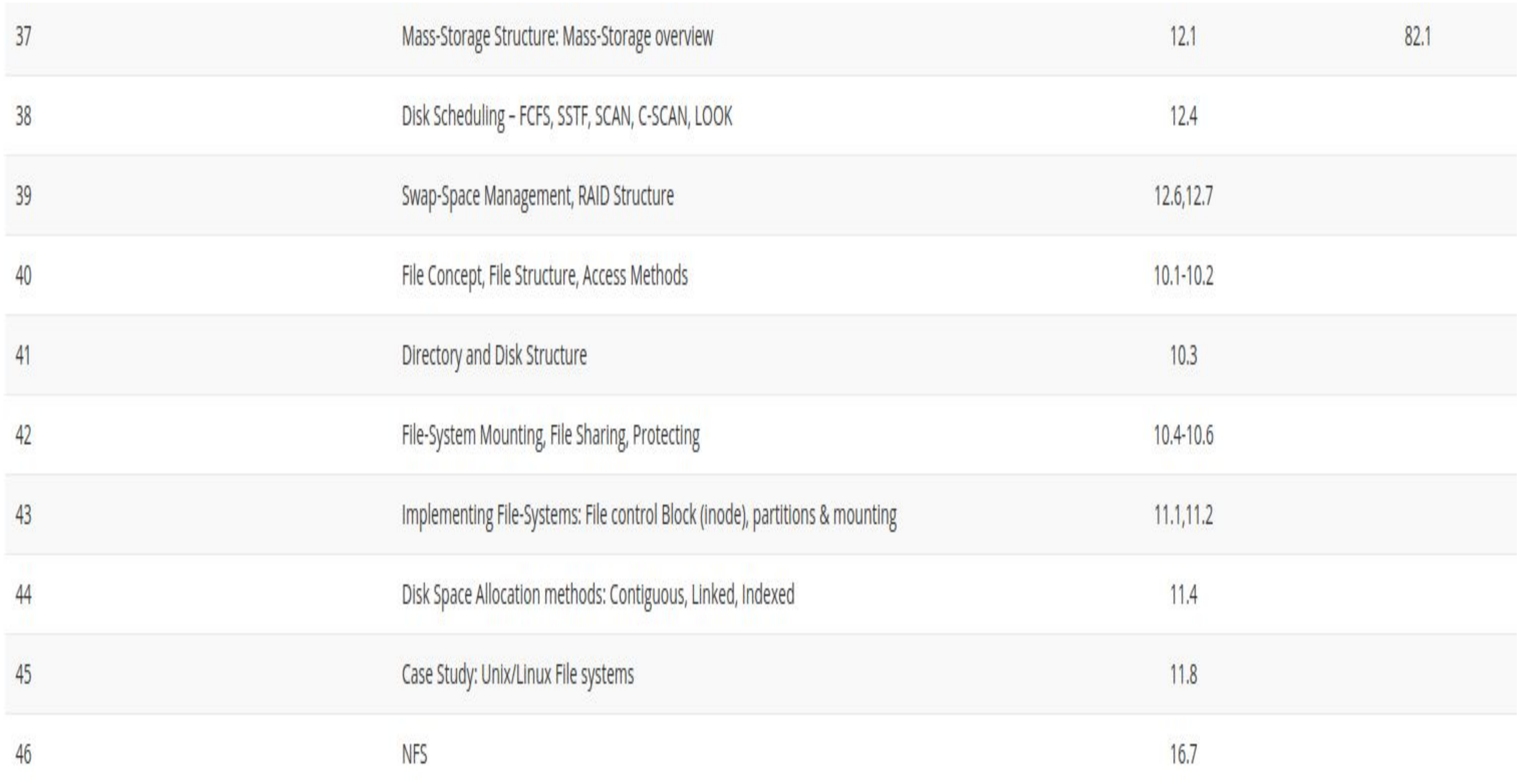
Unit 4: Storage Management

Mass-Storage Structur - Mass-Storage overview, Disk Scheduling, Swap-Space Management, RAID structure. File System Interface - file organization/structure and access methods, directories, sharing File System Implementation/Internals: File control Block (inode), partitions & mounting, Allocation methods.

Case Study: Linux/Windows File Systems

OPERATING SYSTEMS

Course Outline



37	Mass-Storage Structure: Mass-Storage overview	12.1	82.1
38	Disk Scheduling - FCFS, SSTF, SCAN, C-SCAN, LOOK	12.4	
39	Swap-Space Management, RAID Structure	12.6,12.7	
40	File Concept, File Structure, Access Methods	10.1-10.2	
41	Directory and Disk Structure	10.3	
42	File-System Mounting, File Sharing, Protecting	10.4-10.6	
43	Implementing File-Systems: File control Block (inode), partitions & mounting	11.1,11.2	
44	Disk Space Allocation methods: Contiguous, Linked, Indexed	11.4	
45	Case Study: Unix/Linux File systems	11.8	
46	NFS	16.7	

Topic Outline

- Case Study: Unix / Linux File System

Unix / Linux File System

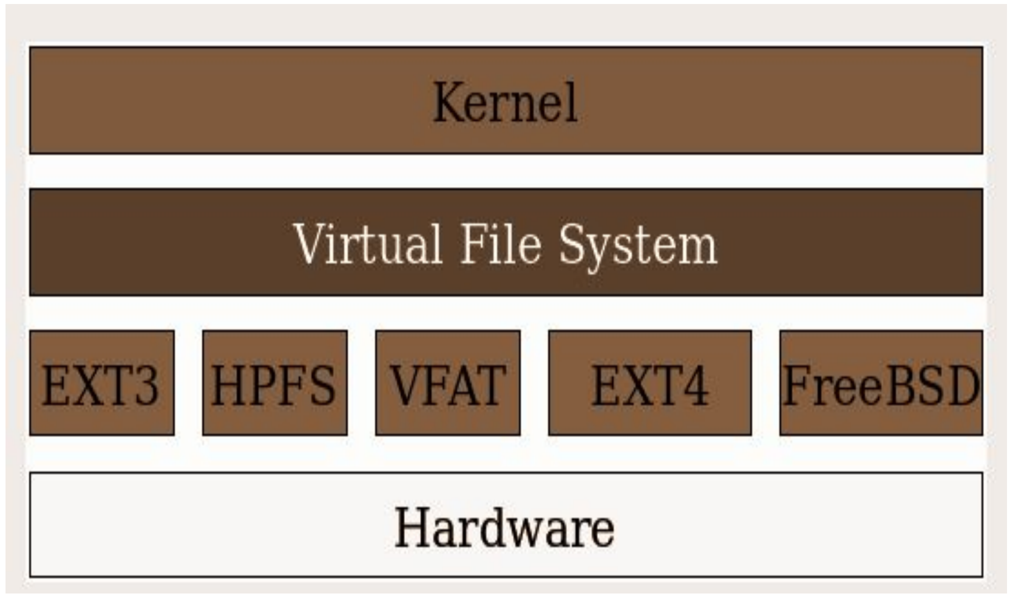
- To the user, Linux file system appears as a hierarchical directory tree obeying UNIX semantics

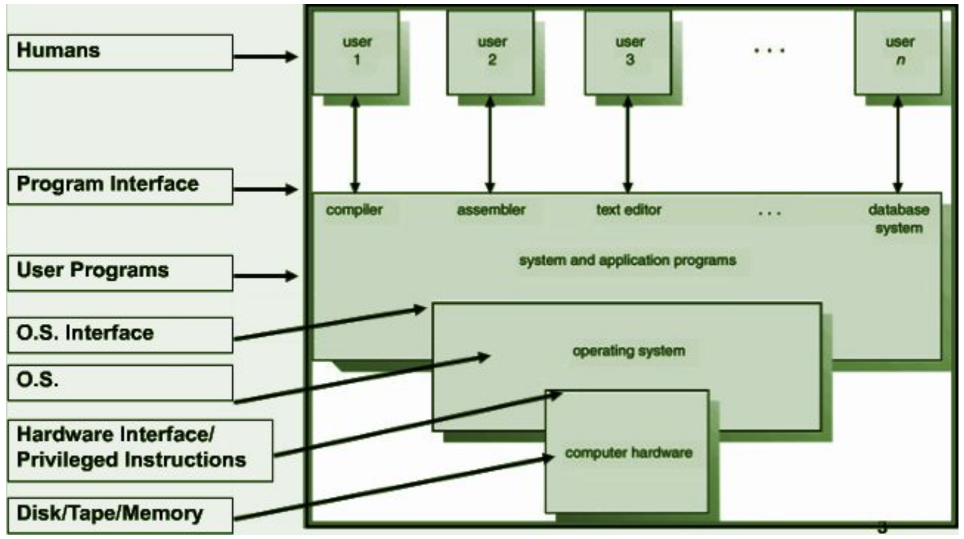
- Internally, the kernel hides implementation details and manages the multiple different file systems via an abstraction layer, that is, the virtual file system (VFS)

Unix / Linux File System

- The entire Linux directory structure starting at the top (/) root directory.
- A specific type of data storage format, such as EXT3, EXT4, BTRFS ("butter F S", "b-tree F S"), XFS, and so on. Linux supports almost 100 types of filesystems, including some very old ones as well as some of the newest.
- Each of these filesystem types uses its own metadata structures to define how the data is stored and accessed.
- A partition or logical volume formatted with a specific type of filesystem that can be mounted on a specified mount point on a Linux filesystem.

Unix / Linux File System





- The first part of this two-part implementation is the Linux virtual filesystem.
- This virtual filesystem provides a single set of commands for the kernel, and developers, to access all types of filesystems.
- The virtual filesystem software calls the specific device driver required to interface to the various types of filesystems.
- The filesystem-specific device drivers are the second part of the implementation.
- The device driver interprets the standard set of filesystem commands to ones specific to the type of filesystem on the partition or logical volume.

Unix / Linux File System

- The Linux VFS is designed around object-oriented principles and is composed of four components:
  - A set of definitions that define what a file object is allowed to look like
    - The **inode object** structure represent an individual file
    - The **file object** represents an open file
    - The **superblock object** represents an entire file system
    - A **dentry object** represents an individual directory entry

Unix / Linux File System

- Internally, the kernel hides implementation details and manages the multiple different file systems via an abstraction layer, that is, the virtual file system (VFS)
- The Linux VFS is designed around object-oriented principles and layer of software to manipulate those objects with a set of operations on the objects
  - For example for the file object operations include (from struct file_operations in /usr/include/linux/fs.h

    int open(. . .) — Open a file

    ssize_t read(. . .) — Read from a file

    ssize_t write(. . .) — Write to a file

    int mmap(. . .) — Memory-map a file

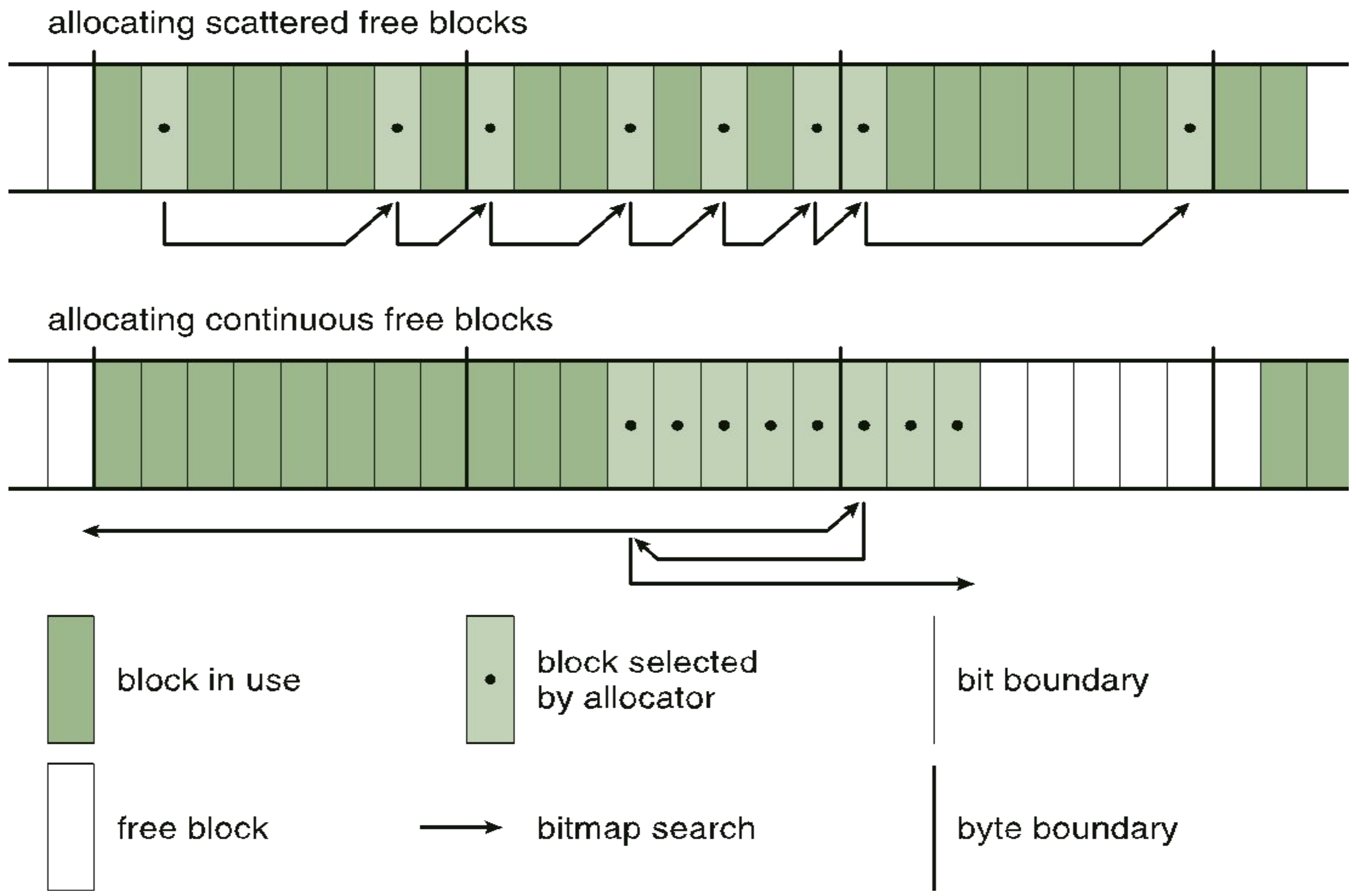
The Linux ext3 File System

- **ext3** is standard on disk file system for Linux
  - Uses a mechanism similar to that of BSD Fast File System (FFS) for locating data blocks belonging to a specific file
  - Supersedes older **extfs**, **ext2** file systems
  - Work underway on ext4 adding features like extents
  - of course, many other file system choices with Linux distros

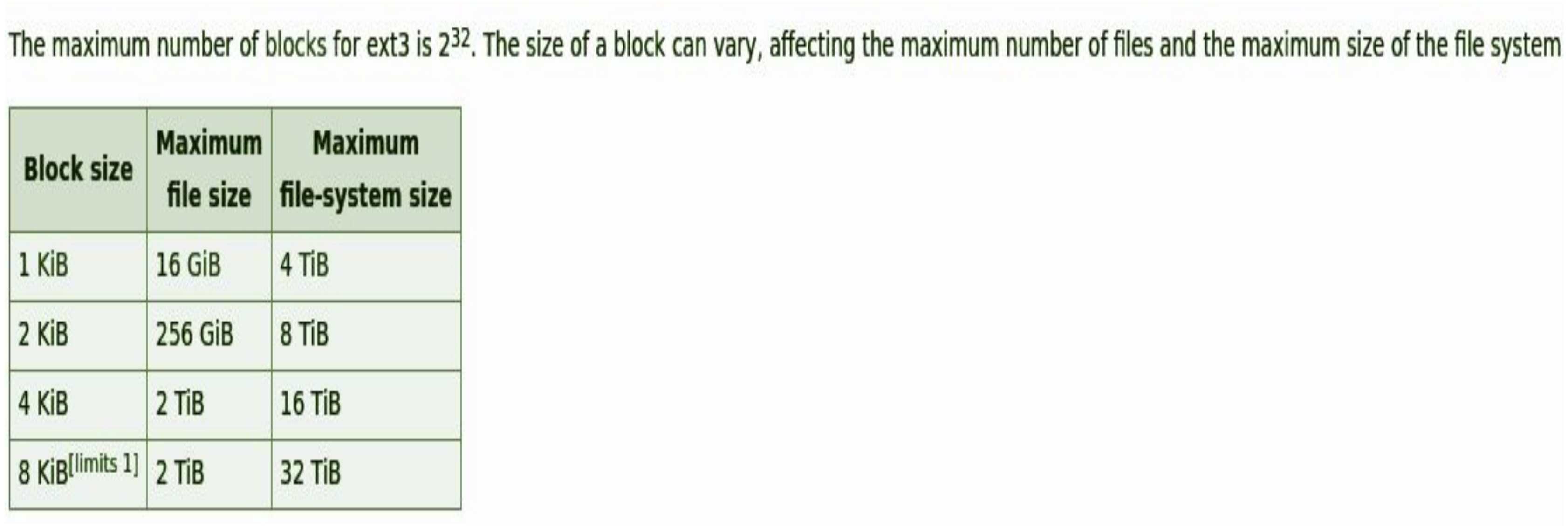
The Linux ext3 File System

- The main differences between ext2fs and FFS concern their disk allocation policies
    - In ffs, the disk is allocated to files in blocks of 8Kb, with blocks being subdivided into fragments of 1Kb to store small files or partially filled blocks at the end of a file
    - ext3 does not use fragments; it performs its allocations in smaller units
    - The default block size on ext3 varies as a function of total size of file system with support for 1, 2, 4 and 8 KB blocks
    - ext3 uses cluster allocation policies designed to place logically adjacent blocks of a file into physically adjacent blocks on disk, so that it can submit an I/O request for several disk blocks as a single operation on a **block group**
    - Maintains bit map of free blocks in a block group, searches for free byte to allocate at least 8 blocks at a time

Ext3 Block-Allocation Policies


allocating scattered free blocks
allocating continuous free blocks
block in use
block selected by allocator
bit boundary
free block
bitmap search
byte boundary

Ext3 Block-Allocation Policies



The maximum number of blocks for ext3 is 2^{32}. The size of a block can vary, affecting the maximum number of files and the maximum size of the file system

Block size	Maximum file size	Maximum file-system size
1 KiB	16 GiB	4 TiB
2 KiB	256 GiB	8 TiB
4 KiB	2 TiB	16 TiB
8 KiB[limits 1]	2 TiB	32 TiB

Journaling

- ext3 implements **journaling**, with file system updates first written to a log file in the form of **transactions**
  - Once in log file, considered committed
  - Over time, log file transactions replayed over file system to put changes in place
- On system crash, some transactions might be in journal but not yet placed into file system
  - Must be completed once system recovers
  - No other consistency checking is needed after a crash (much faster than older methods)
- Improves write performance on hard disks by turning random I/O into sequential I/O

Journaling

Three levels of journaling are available in the Linux kernel implementation of ext3: journal, ordered, and writeback.

- **Journal** is the lowest risk mode, writing both data and metadata to the journal before committing it to the filesystem. This ensures consistency of the file being written to, as well as the filesystem as a whole, but can significantly decrease performance.

- **Ordered** is the default mode in most Linux distributions; ordered mode writes metadata to the journal but commits data directly to the filesystem. As the name implies, the order of operations here is rigid: First, metadata is committed to the journal; second, data is written to the filesystem, and only then is the associated metadata in the journal flushed to the filesystem itself. This ensures that, in the event of a crash, the metadata associated with incomplete writes is still in the journal, and the filesystem can sanitize those incomplete writes while rolling back the journal. In ordered mode, a crash may result in corruption of the file or files being actively written to during the crash, but the filesystem itself—and files not actively being written to—are guaranteed safe.

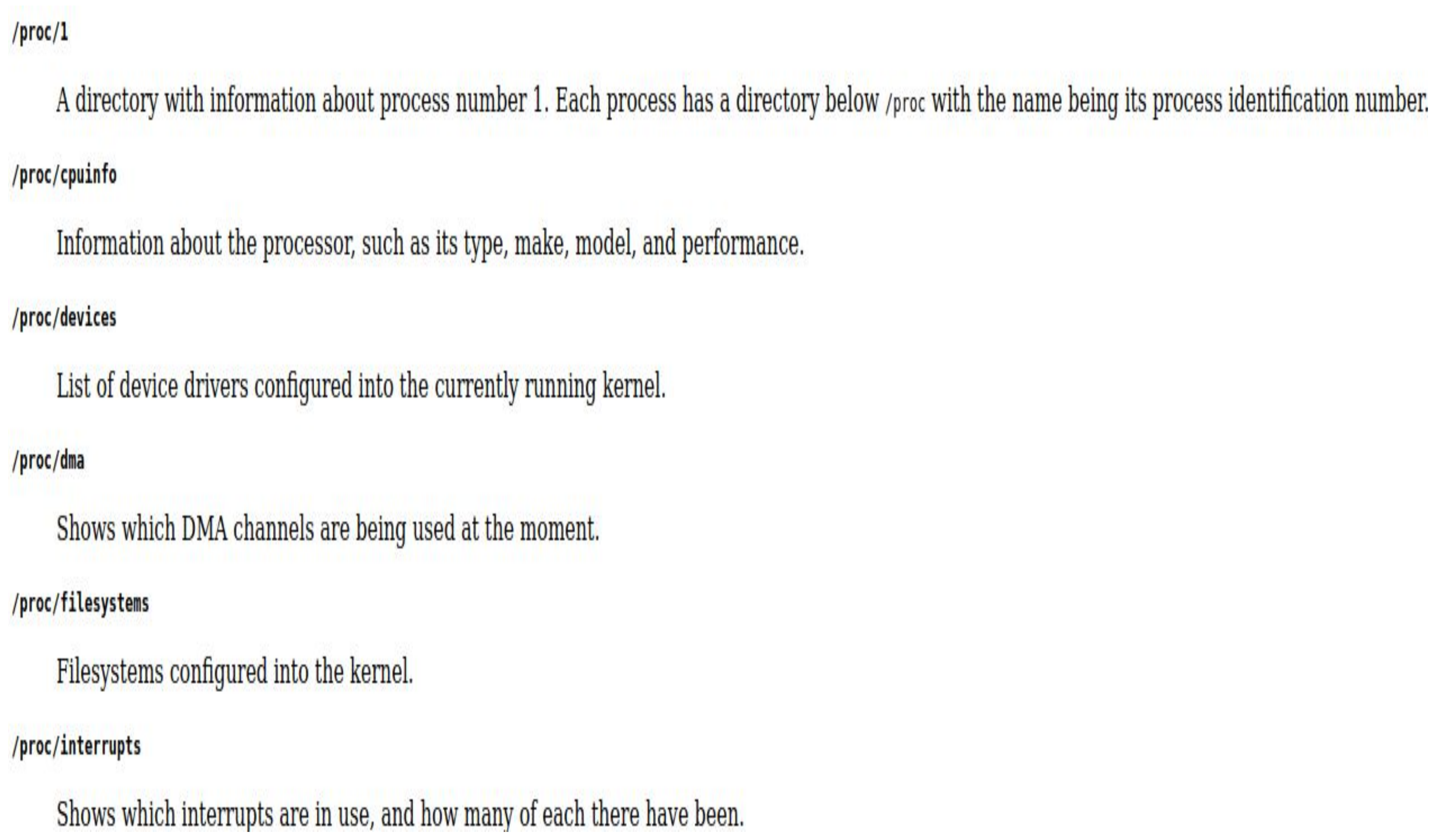
- **Writeback** is the third—and least safe—journaling mode. In writeback mode, like ordered mode, metadata is journaled, but data is not. Unlike ordered mode, metadata and data alike may be written in whatever order makes sense for best performance. This can offer significant increases in performance, but it's much less safe. Although writeback mode still offers a guarantee of safety to the filesystem itself, files that were written to during or before the crash are vulnerable to loss or corruption.

The Linux Proc File System

- The **proc file system** does not store data, rather, its contents are computed on demand according to user file I/O requests
- **proc** must implement a directory structure, and the file contents within; it must then define a unique and persistent inode number for each directory and files it contains
  - It uses this inode number to identify just what operation is required when a user tries to read from a particular file inode or perform a lookup in a particular directory inode
  - When data is read from one of these files, **proc** collects the appropriate information, formats it into text form and places it into the requesting process's read buffer

- The /proc filesystem contains a illusionary filesystem. It does not exist on a disk. Instead, the kernel creates it in memory. It is used to provide information about the system (originally about processes, hence the name).

The Linux Proc File System



`/proc/1`

A directory with information about process number 1. Each process has a directory below `/proc` with the name being its process identification number.

`/proc/cpuinfo`

Information about the processor, such as its type, make, model, and performance.

`/proc/devices`

List of device drivers configured into the currently running kernel.

`/proc/dma`

Shows which DMA channels are being used at the moment.

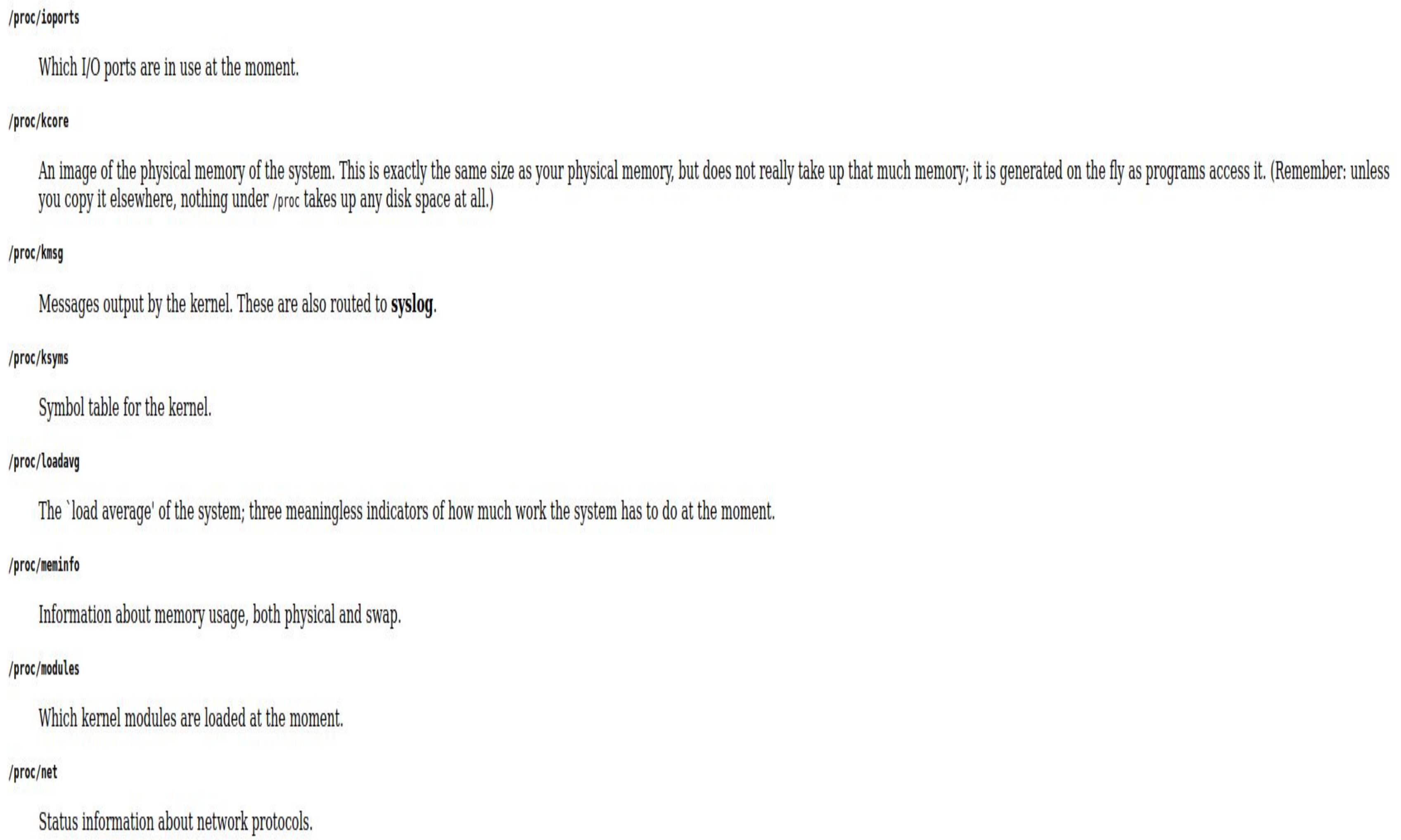
`/proc/filesystems`

Filesystems configured into the kernel.

`/proc/interrupts`

Shows which interrupts are in use, and how many of each there have been.

The Linux Proc File System



`/proc/ioports`

Which I/O ports are in use at the moment.

`/proc/kcore`

An image of the physical memory of the system. This is exactly the same size as your physical memory, but does not really take up that much memory; it is generated on the fly as programs access it. (Remember: unless you copy it elsewhere, nothing under `/proc` takes up any disk space at all.)

`/proc/kmsg`

Messages output by the kernel. These are also routed to **syslog**.

`/proc/ksyms`

Symbol table for the kernel.

`/proc/loadavg`

The `load average' of the system; three meaningless indicators of how much work the system has to do at the moment.

`/proc/meminfo`

Information about memory usage, both physical and swap.

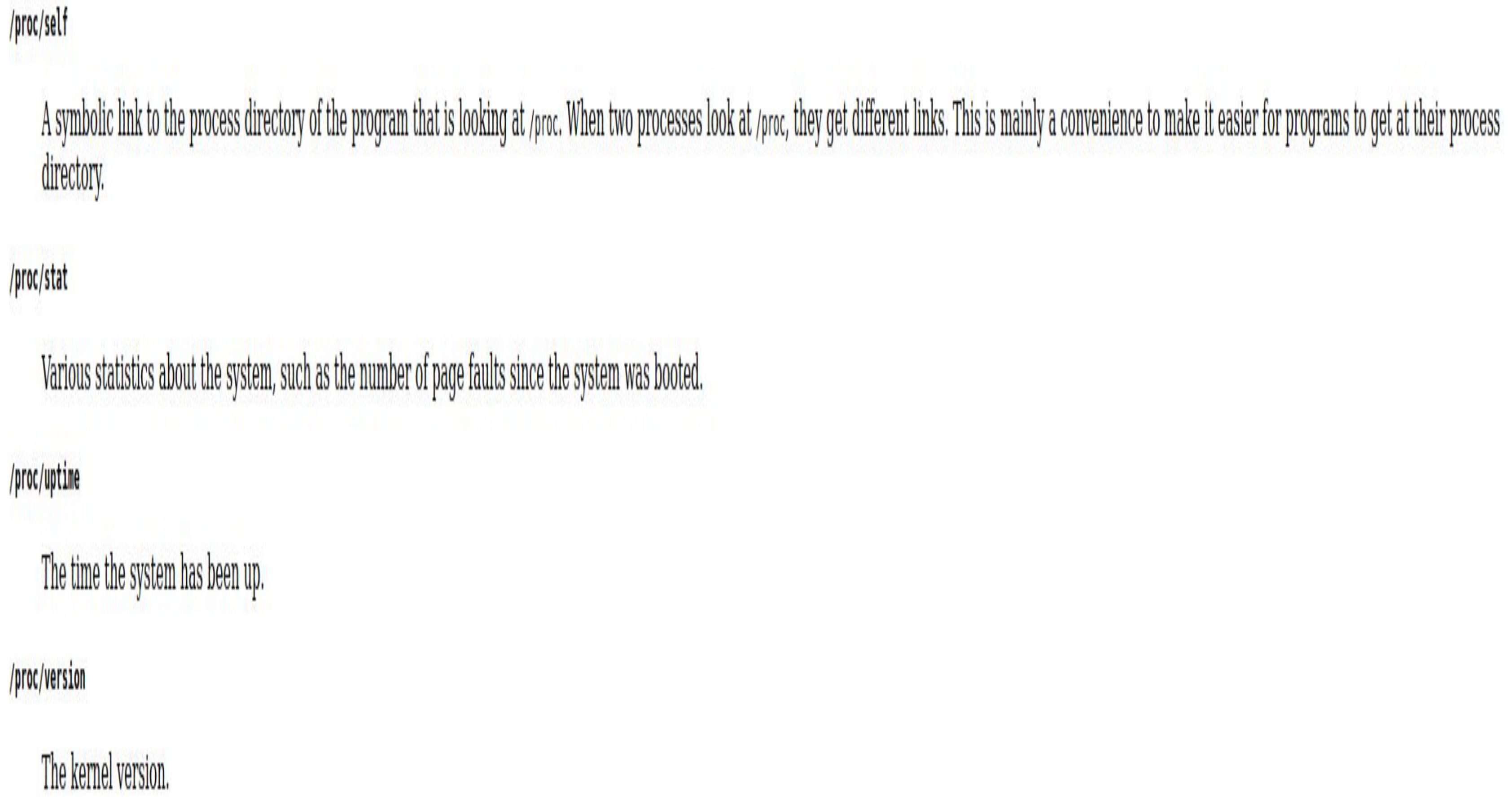
`/proc/modules`

Which kernel modules are loaded at the moment.

`/proc/net`

Status information about network protocols.

The Linux Proc File System



`/proc/self`

A symbolic link to the process directory of the program that is looking at `/proc`. When two processes look at `/proc`, they get different links. This is mainly a convenience to make it easier for programs to get at their process directory.

`/proc/stat`

Various statistics about the system, such as the number of page faults since the system was booted.

`/proc/uptime`

The time the system has been up.

`/proc/version`

The kernel version.

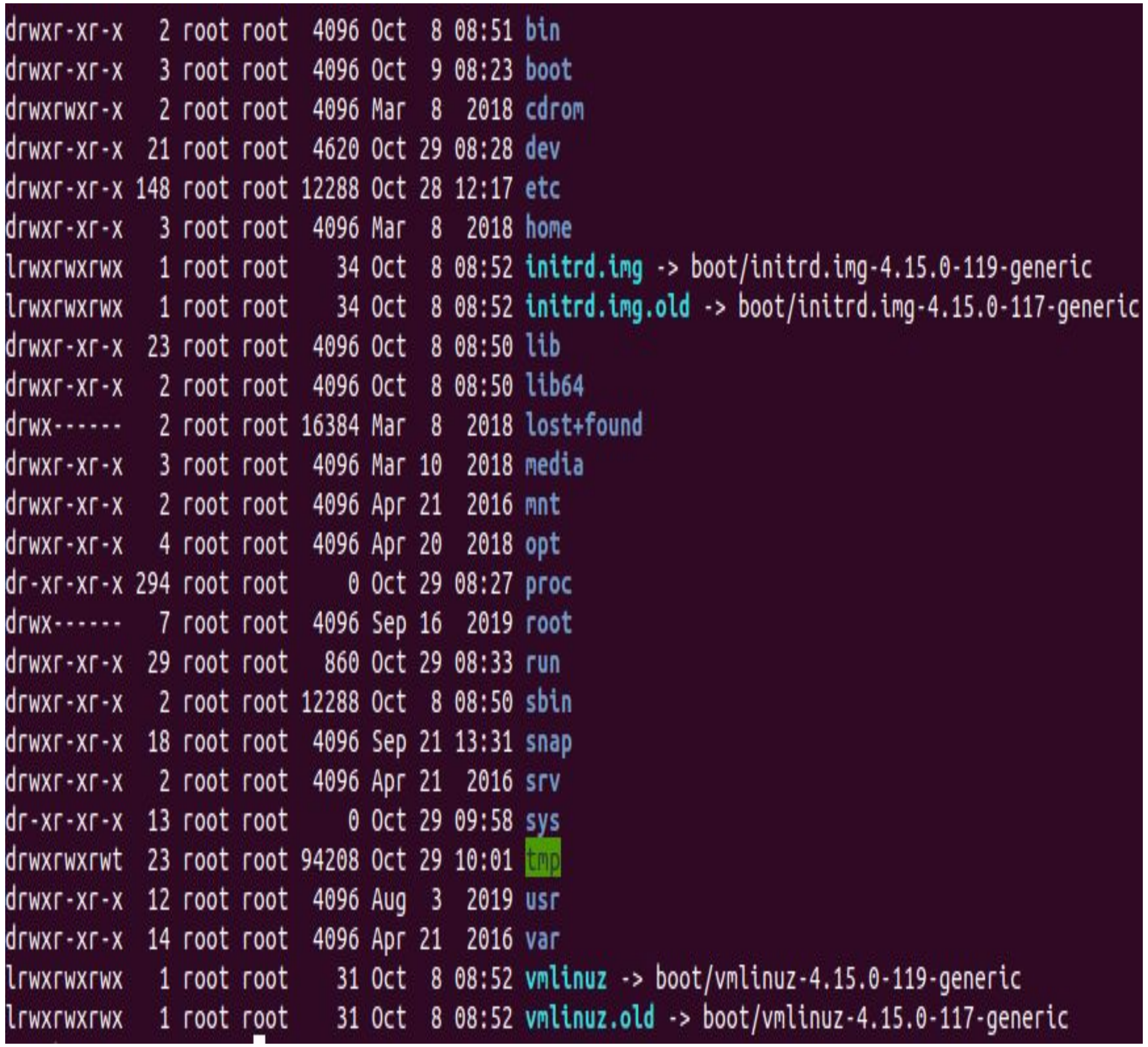
Unix / Linux File System - Additional Input

/ (root filesystem)	The root filesystem is the top-level directory of the filesystem. It must contain all of the files required to boot the Linux system before other filesystems are mounted. It must include all of the required executables and libraries required to boot the remaining filesystems. After the system is booted, all other filesystems are mounted on standard, well-defined mount points as subdirectories of the root filesystem.
/bin	The /bin directory contains user executable files.
/boot	Contains the static bootloader and kernel executable and configuration files required to boot a Linux computer.
/dev	This directory contains the device files for every hardware device attached to the system. These are not device drivers, rather they are files that represent each device on the computer and facilitate access to those devices.
/etc	Contains the local system configuration files for the host computer.
/home	Home directory storage for user files. Each user has a subdirectory in /home.
/lib	Contains shared library files that are required to boot the system.
/media	A place to mount external removable media devices such as USB thumb drives that may be connected to the host.

Unix / Linux File System - Additional Input

/mnt	A temporary mount point for regular file systems (as in not removable media) that can be used while the administrator is repairing or working on a filesystem.
/opt	Optional files such as vendor supplied application programs should be located here.
/root	This is not the root (/) filesystem. It is the home directory for the root user.
/sbin	System binary files. These are executables used for system administration.
/tmp	Temporary directory. Used by the operating system and many programs to store temporary files. Users may also store files here temporarily. Note that files stored here may be deleted at any time without prior notice.
/usr	These are shareable, read-only files, including executable binaries and libraries, man files, and other types of documentation.
/var	Variable data files are stored here. This can include things like log files, MySQL, and other database files, web server data files, email inboxes, and much more.
Snap is a software packaging and deployment system developed by Canonical for the operating systems that use the Linux kernel.	

Unix / Linux File System - Additional Input



```
drwxr-xr-x   2 root root  4096 Oct  8 08:51 bin
drwxr-xr-x   3 root root  4096 Oct  9 08:23 boot
drwxrwxr-x   2 root root  4096 Mar  8  2018 cdrom
drwxr-xr-x  21 root root  4620 Oct 29 08:28 dev
drwxr-xr-x 148 root root 12288 Oct 28 12:17 etc
drwxr-xr-x   3 root root  4096 Mar  8  2018 home
lrwxrwxrwx   1 root root    34 Oct  8 08:52 initrd.img -> boot/initrd.img-4.15.0-119-generic
lrwxrwxrwx   1 root root    34 Oct  8 08:52 initrd.img.old -> boot/initrd.img-4.15.0-117-generic
drwxr-xr-x  23 root root  4096 Oct  8 08:50 lib
drwxr-xr-x   2 root root  4096 Oct  8 08:50 lib64
drwx------   2 root root 16384 Mar  8  2018 lost+found
drwxr-xr-x   3 root root  4096 Mar 10  2018 media
drwxr-xr-x   2 root root  4096 Apr 21  2016 mnt
drwxr-xr-x   4 root root  4096 Apr 20  2018 opt
dr-xr-xr-x 294 root root     0 Oct 29 08:27 proc
drwx------   7 root root  4096 Sep 16  2019 root
drwxr-xr-x  29 root root   860 Oct 29 08:33 run
drwxr-xr-x   2 root root 12288 Oct  8 08:50 sbin
drwxr-xr-x  18 root root  4096 Sep 21 13:31 snap
drwxr-xr-x   2 root root  4096 Apr 21  2016 srv
dr-xr-xr-x  13 root root     0 Oct 29 09:58 sys
drwxrwxrwt  23 root root 94208 Oct 29 10:01 tmp
drwxr-xr-x  12 root root  4096 Aug  3  2019 usr
drwxr-xr-x  14 root root  4096 Apr 21  2016 var
lrwxrwxrwx   1 root root    31 Oct  8 08:52 vmlinuz -> boot/vmlinuz-4.15.0-119-generic
lrwxrwxrwx   1 root root    31 Oct  8 08:52 vmlinuz.old -> boot/vmlinuz-4.15.0-117-generic
```

- `-` - Regular file
- `b` - Block special file
- `c` - Character special file
- `d` - Directory
- `l` - Symbolic link
- `n` - Network file
- `p` - FIFO
- `s` - Socket

Topic Outline

- Case Study: Unix / Linux File System

THANK YOU

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

nitin.pujari@pes.edu

For Course Deliverables by the Anchor Faculty click on www.pesuacademy.com