Linux Operating System

Submitted by **:** Arnab Das (CrS 2109)

Thatipamula Venkatesh (CrS 2121)

Sohan Das (CrS 2119)

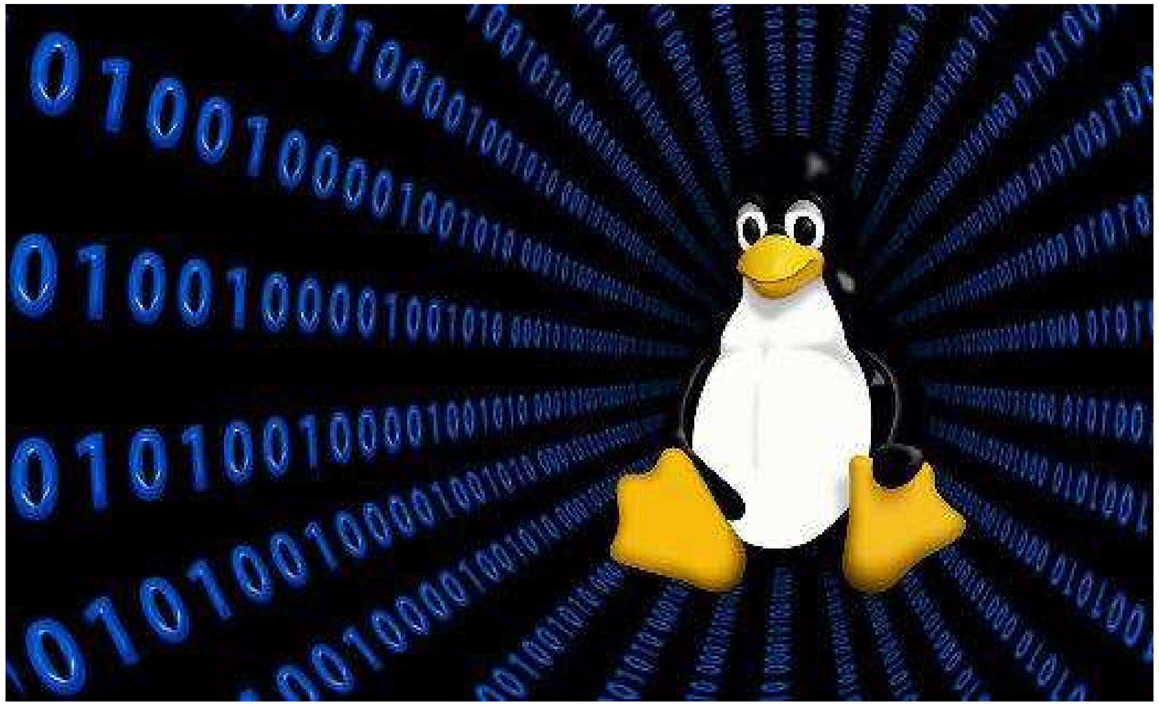
Under the Supervision of : Dr. Sushmita Mitra

Machine Intelligence Unit (MIU)

Indian Statistical Institute, Kolkata

INDEX

LINUX OPERATING SYSTEM



ARNAB DAS
CRS-2109

THATIPAMULA VENKATESH
CRS-2121

SOHAN DAS
CRS-2119

HISTORY OF LINUX

What do you think ?

- The operating system used by most of the stock exchanges?
- Which powers the servers delivering amazon , facebook, twitter ,ebay?
- By which OS, our most of the phones , T.V.'s, A.T.M's etc.?
- What OS used by most of the super computers?

Yes, we use LINUX in every one of them.

- We use LINUX every time we surf the INTERNET, LINUX is everywhere.
- The LINUX OS ,was developed by **LINUS TORVALDS** in 1991.
- The official mascot of LINUX is penguin named **TUX**.

MOBILE DEVICES
THE ENTERPRISE
WEB INFRASTRUCTURE
DATA CENTERS
SUPER COMPUTING
MORE...
LINUX IS DOMINATING

WHY LINUX IS SO SPECIAL?

- LINUX is built collaboratively across companies , geographies and markets. It's the Largest collaborative development project in the history of computing
- LINUX is open source OS and free for everyone, no need of copy right issues and license required and it can easily installed in any system.
- A new kernel comes out every 2-3 months compared to other OS's years for competing operating systems.

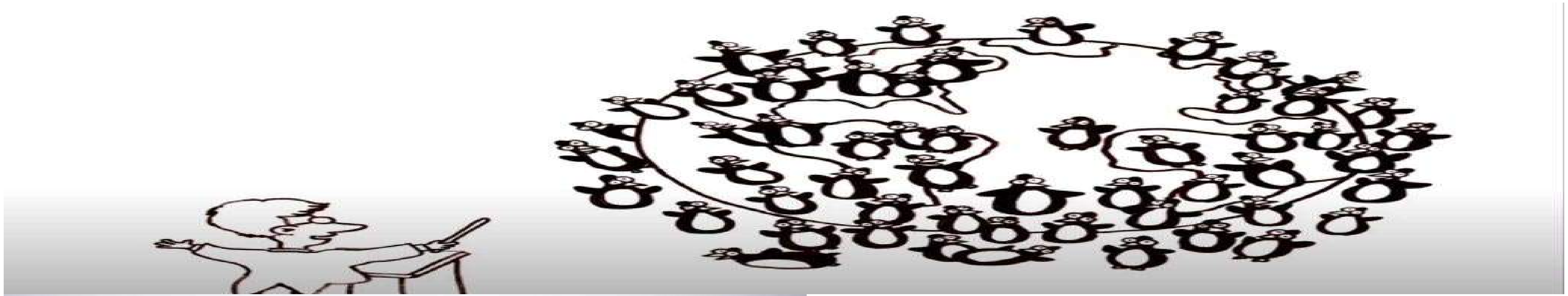
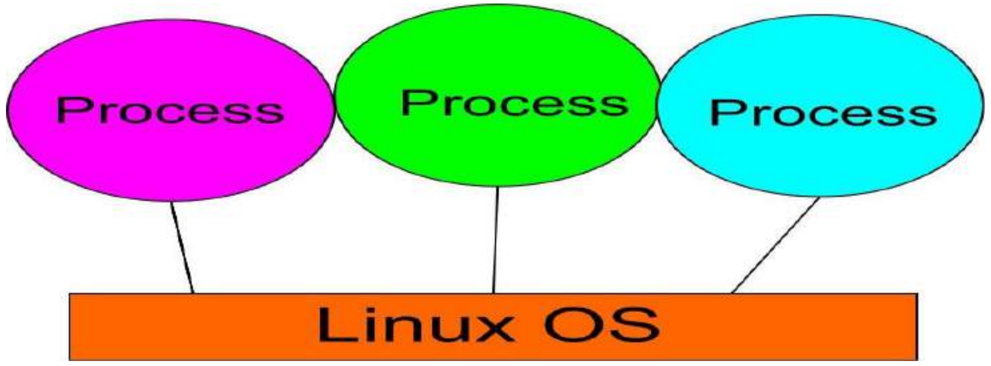
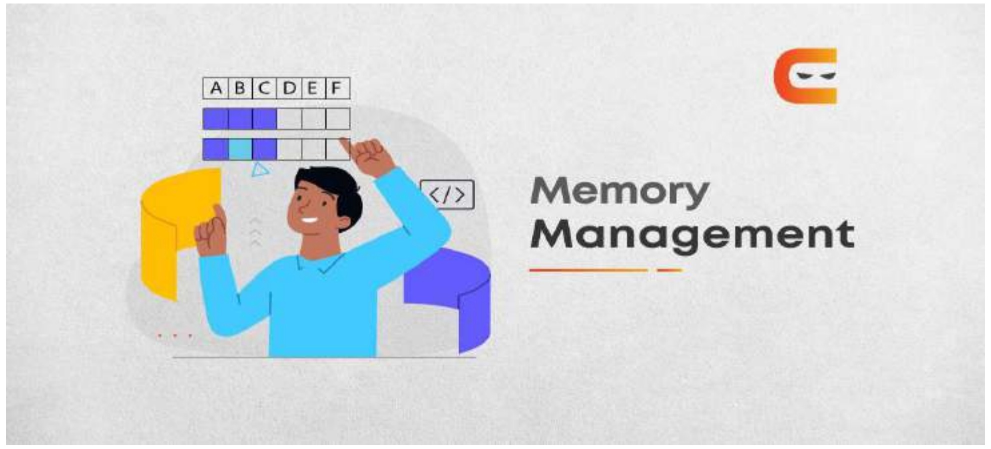
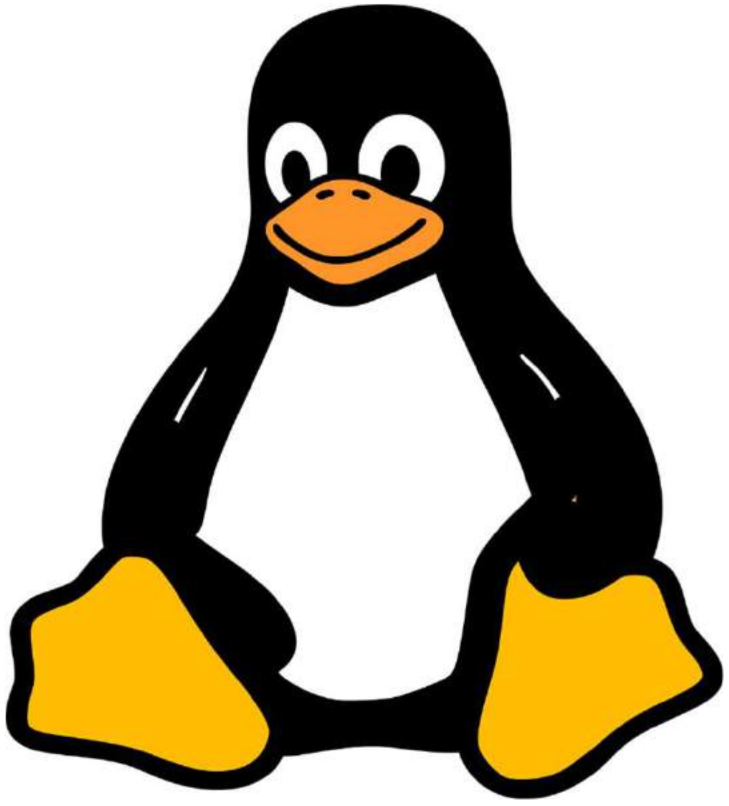


TABLE OF CONTENTS:





PROCESS MANAGEMENT in linux



WHAT IS PROCESS?

- Any program which is in execution is called process.
- So, process is the basic context in which all user requested activity is serviced within the OS
- When we run a program those instructions are copied into memory and space is allocated for variables and other stuff required to manage its execution.

DESCRIPTION OF PROCESS :

- Like human even process has the information like when it is created.
- Address space of allocated memory , security properties including ownership, credentials ,and privilage . One or more executions threads of program code and the process state.

PROCESS ID(PID)

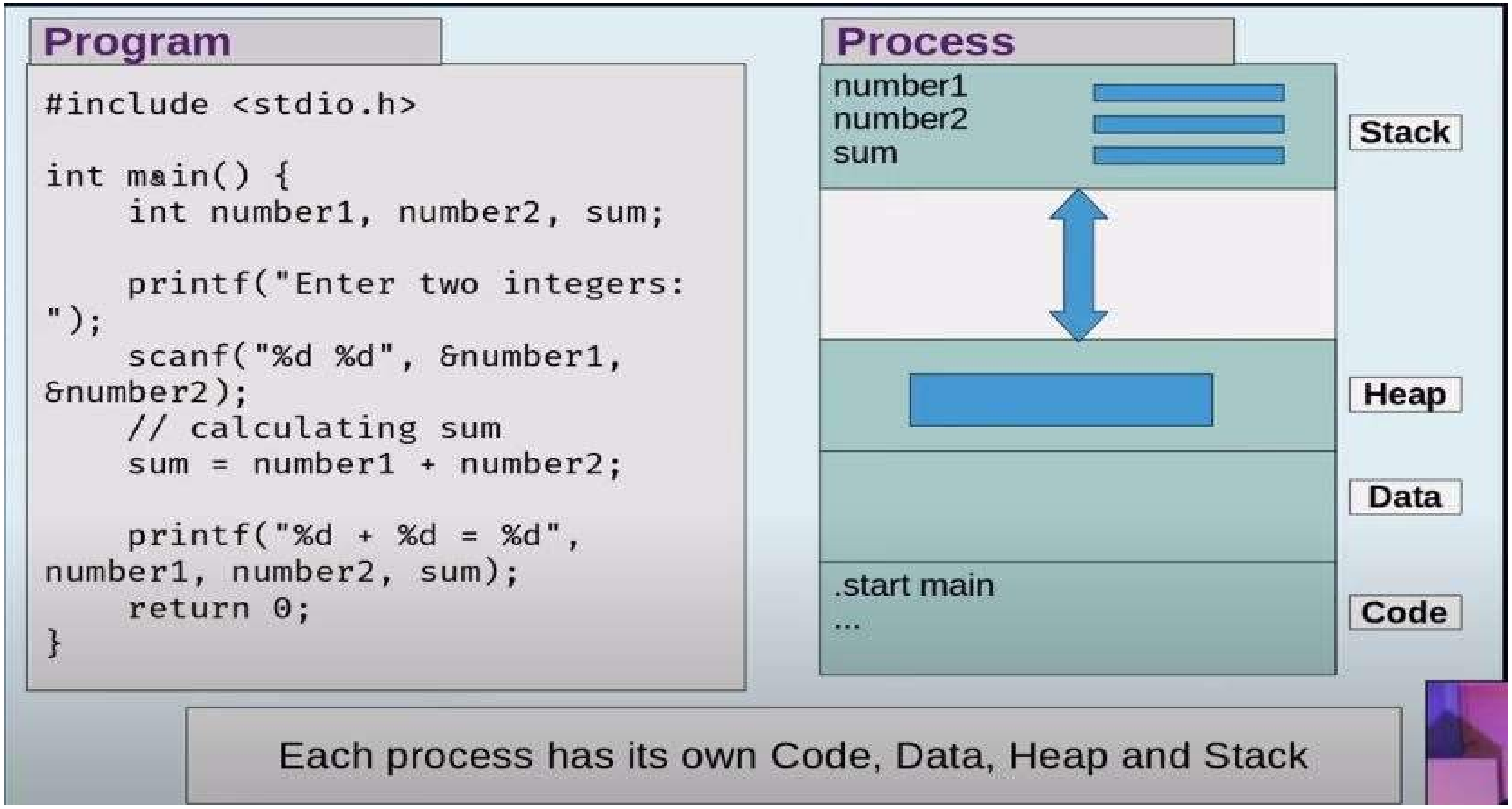
- Each LINUX process is identified by a unique process number called PID.
- Every process has a parent process ID(PPID),<except init>
- When a process loaded into memory there is a structure

  Stack

  Heap

  Data segment

  Code segment



```
#include <stdio.h>

int main() {
    int number1, number2, sum;

    printf("Enter two integers:
");
    scanf("%d %d", &number1,
&number2);
    // calculating sum
    sum = number1 + number2;

    printf("%d + %d = %d",
number1, number2, sum);
    return 0;
}
```

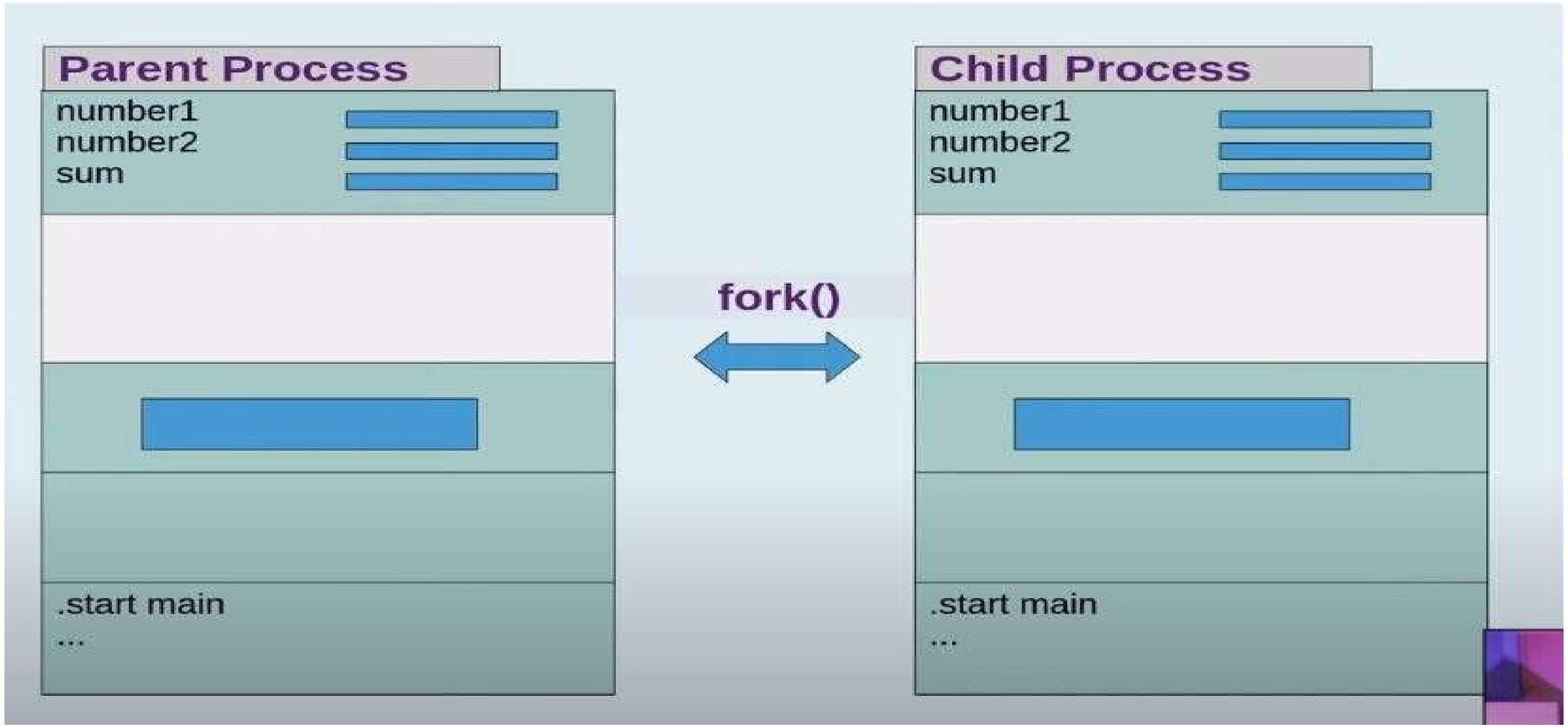
- If we see the above C-program, we have three different variables number1, number2 and sum.
- When the structure is built after we know this code has to be compiled into binary executable file and once that happens the above is the structure that gets built.
- **Stack** contains these variables.
- The white double arrowed part is an area of memory that is set aside incase your program needs more memory or you are calling a lot of functions or may be you are allocating memory.
- The next one is **Heap** , this is the memory segment that is dynamically allocated during process time
- **Data part** contains the information like static or global variables.
- Final one is **coding par**t, this contains the binary version of the code and it is also known as text segment.

CREATING PROCESS

- Creating a New process
- Running of new program

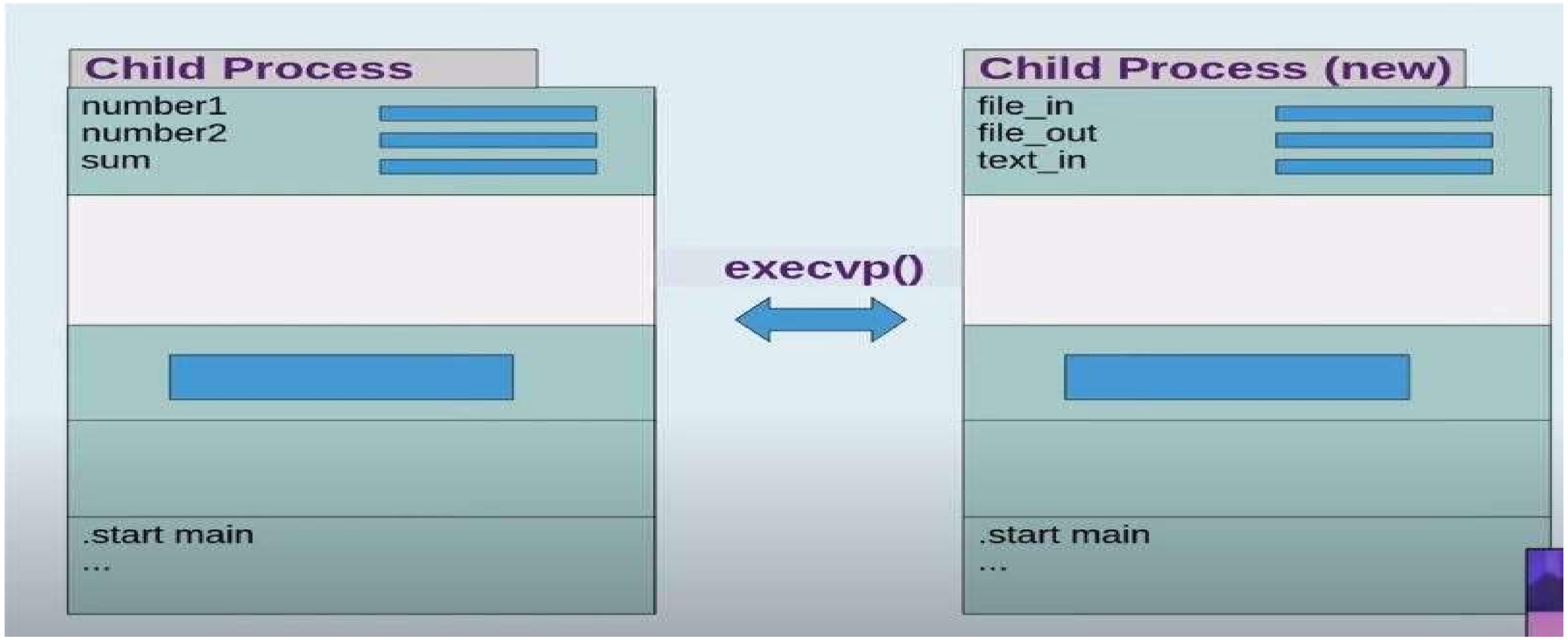
fork():

- Creates child processes by making an exact copy of it's parent process
- The parent process will immediately continue
- The parent process does not wait , it will immediately continue on.
- The child process starts executing in exactly the same place as the parent process.



REPLACING THE NEW PROCESS

- Okay, Now I have two processes running that are identical- Now,what?
- If you just want concurrency ,You are done . But if you want new process, then …..?
- **The exec() function** :
  - Immediately stops the child's process
  - The new program is loaded in it's place
  - Begins executing the new program
  - exec() will never returns unless an error occurs
  - The new process takes over the child processes PID, and also the parent PID(PPID).

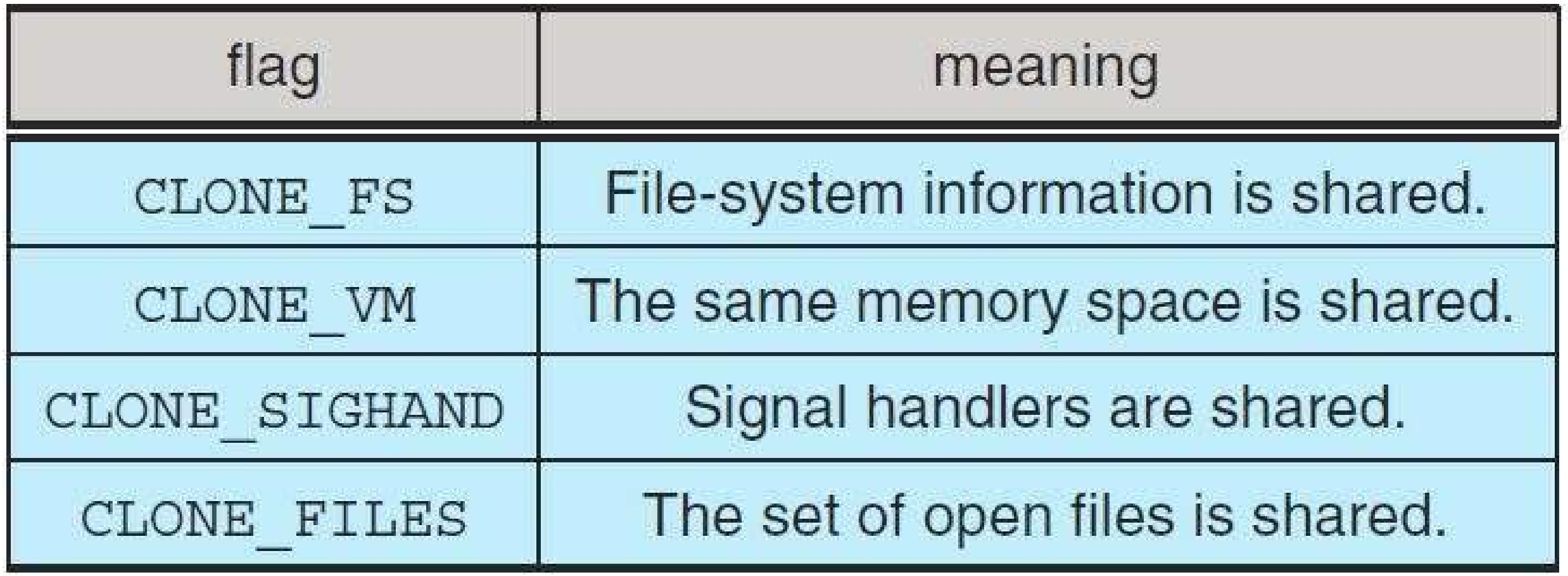


PROCESS CONTEXT

- Process context is the state of running program at any one time; it changes constantly
- **Scheduling Context:**
  - It is the information that a scheduler needs to suspend and restart the process. This information includes saved copies of all the process's.
- **Accounting:**
  - The kernel maintains accounting information about the resources currently being consumed by each process and the total resources consumed by the process in it's entire lifetime.
- **File table:**
  - It's an array of pointers to kernel file structures
  - When making file I/O system calls processes refer to files by their index into this table , the file descriptor(fd).
- **File System Context:**
  - It applies to request to open new files
- **Single Handler Table:**
  - Defines the routine in the processes address space
- **Virtual Memory Context:**
  - Describes the full contents of processes private address space

PROCESSES AND THREADS

- Both process and threads are called **<u>TASK</u>** by LINUX.
- fork() creates a new task with it's own entirely new task context
- clone() creates a new task with it's own identity but that is allowed to share the data structures of it's parent.



flag	meaning
CLONE_FS	File-system information is shared.
CLONE_VM	The same memory space is shared.
CLONE_SIGHAND	Signal handlers are shared.
CLONE_FILES	The set of open files is shared.

SCHEDULING

- Scheduling is the job of allocating CPU time to different task with in an OS.
- LINUX supports pre-emptive multi tasking.

- Except running or interrupting of user processes , it also includes running of various kernel tasks.
- It has two different process scheduling algorithms , one is
  Time sharing algorithm and other is designed for real tasks.

- LINUX scheduler is pre-emptive and priority based algorithm.
  - Real time range (0 to 99)
  - Nice value (-20 to 19)

  The smaller nice value means higher priority.

COMPLETELY FAIR SCHEDULER(CFS)

- LINUX 2.6.23 onwards uses completely fair scheduler, in previous versions O(n),O(1) scheduler were used.

- Calculates how long a process should run as a function of total number of tasks

- Ideal fair scheduling

  In this scheduling , all processes are allotted a proportion of the processor's time .

- Target latency

- Minimum granularity

  scheduling process for short length of time is insufficient. So, a minimum length of time for any process is allotted and it is called as minimum granularity.

Ideal Fair Scheduling

Process	burst time
A	8ms
B	4ms
C	16ms
D	4ms

Divide processor time equally among processes

Ideal Fairness : If there are N processes in the system, each process should have got (100/N)% of the CPU time

Ideal Fairness

A	1	2	3	4	6	8							
B	1	2	3	4									
C	1	2	3	4	6	8	12	16					
D	1	2	3	4									

4ms slice

execution with respect to time

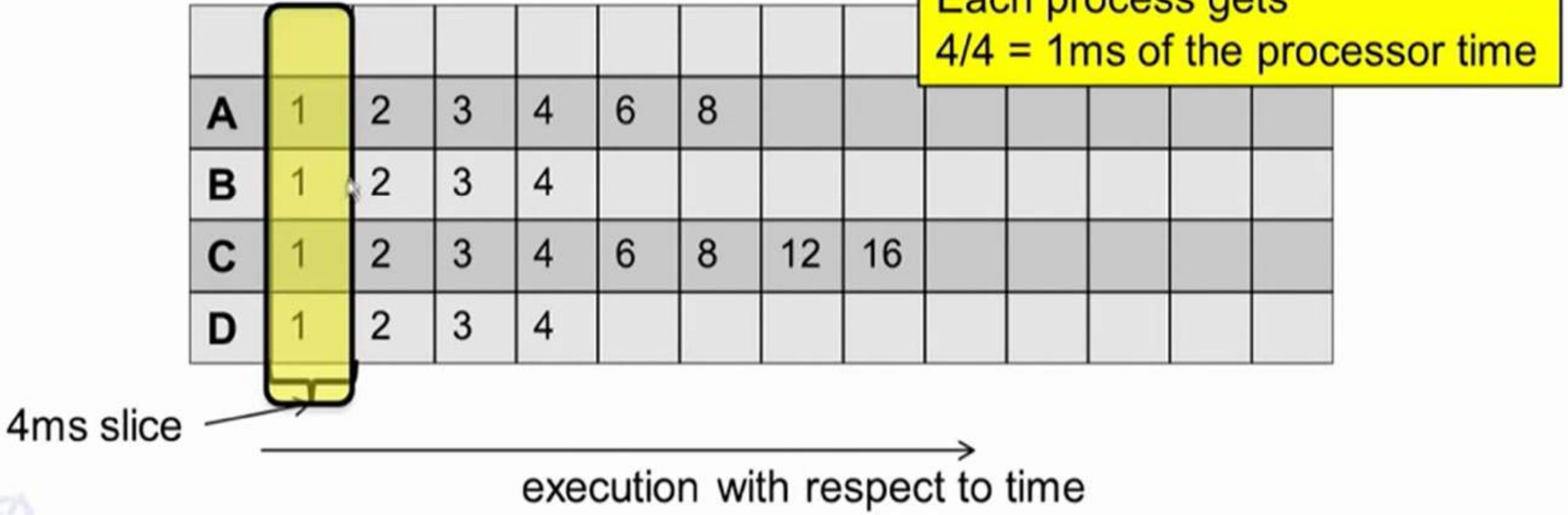
Ideal Fair Scheduling

Process	burst time
A	8ms
B	4ms
C	16ms
D	4ms

Divide processor time equally among processes

Ideal Fairness : If there are N processes in the system, each process should have got (100/N)% of the CPU time

Ideal Fairness


Each process gets
4/4 = 1ms of the processor time
4ms slice
execution with respect to time

A	1	2	3	4	6	8							
B	1	2	3	4									
C	1	2	3	4	6	8	12	16					
D	1	2	3	4									

Ideal Fair Scheduling

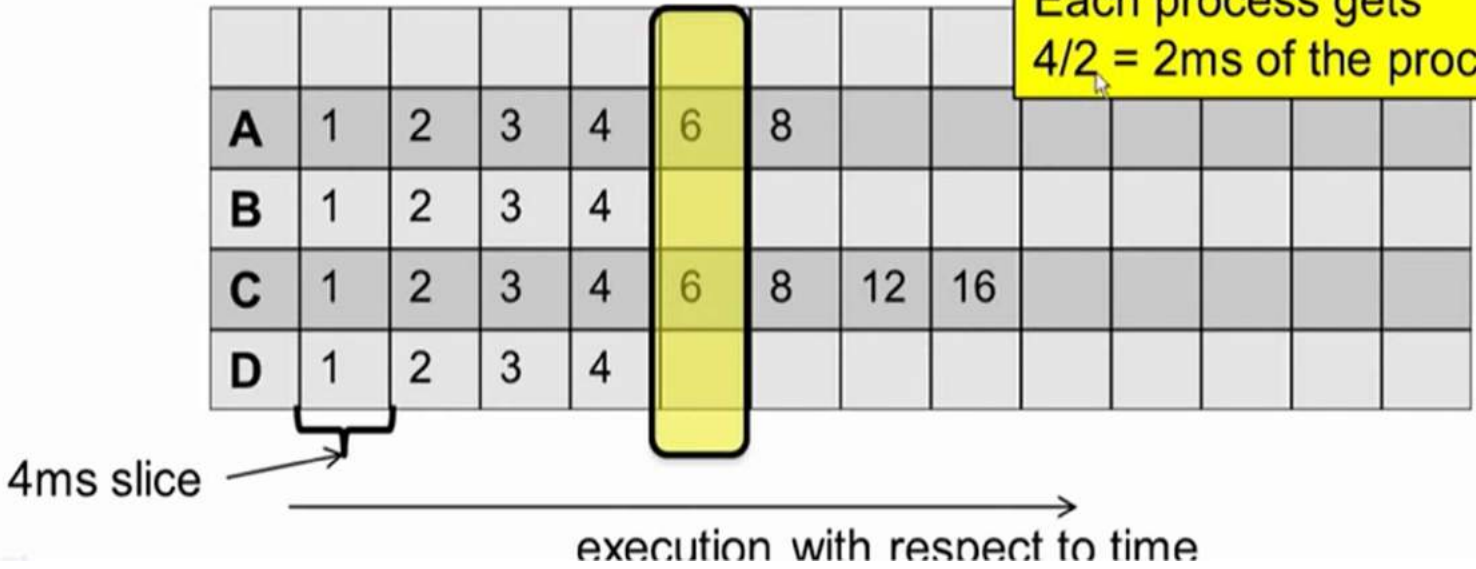
Process	burst time
A	8ms
B	4ms
C	16ms
D	4ms

Divide processor time equally among processes

Ideal Fairness : If there are N processes in the system, each process should have got (100/N)% of the CPU time

Ideal Fairness

Each process gets
4/2 = 2ms of the processor time



A	1	2	3	4	6	8							
B	1	2	3	4									
C	1	2	3	4	6	8	12	16					
D	1	2	3	4									

Ideal Fair Scheduling

Process	burst time
A	8ms
B	4ms
C	16ms
D	4ms

Divide processor time equally among processes

Ideal Fairness : If there are N processes in the system, each process should have got (100/N)% of the CPU time

Ideal Fairness

The single process gets the entire 4ms of the processor time

A	1	2	3	4	6	8							
B	1	2	3	4									
C	1	2	3	4	6	8	12	16					
D	1	2	3	4									

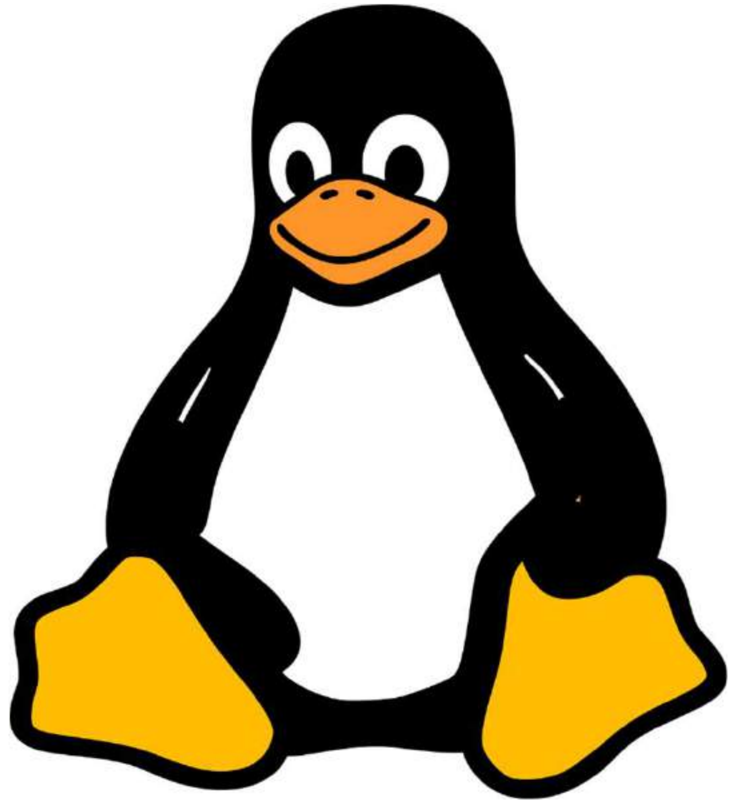
4ms slice

execution with respect to time

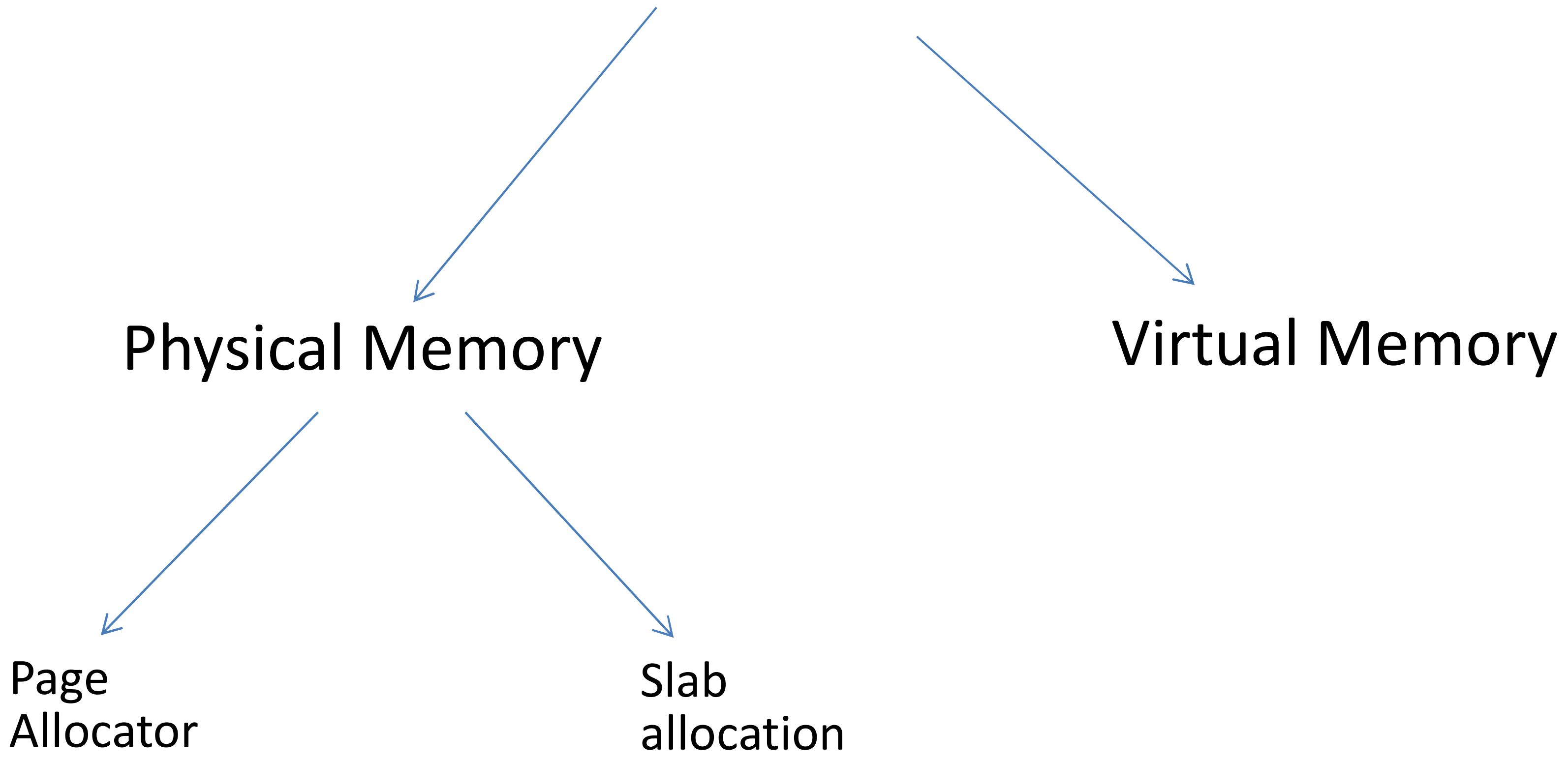
SYMMETRIC MULTIPROCESSING(SMP)

- LINUX 2.0 kernel was the first stable kernel to support SMP hardware.
- Simple multi processor(SMP) hardware that allows separate process to execute parallel on separate processors.
- In version 2.2 of the kernel is a single kernel spinlock(BKL for “big kernel lock”).This is carted to allow multiple processes to be active in the kernel concurrency.
- Later releases implement stability by splitting a spinlock into multiple locks
- Version 3.0 adds even more fine grained locking processor affinity and load balancing

MEMORY MANAGEMENT in linux



MEMORY MANGEMENT IN LINUX



- In Linux ,Memory Management has two components
- First one deals with Allocating and Freeing **Physical memory**-pages and small blocks of RAM
- Second handles virtual memory ,which is memory mapped into the address space of running processes

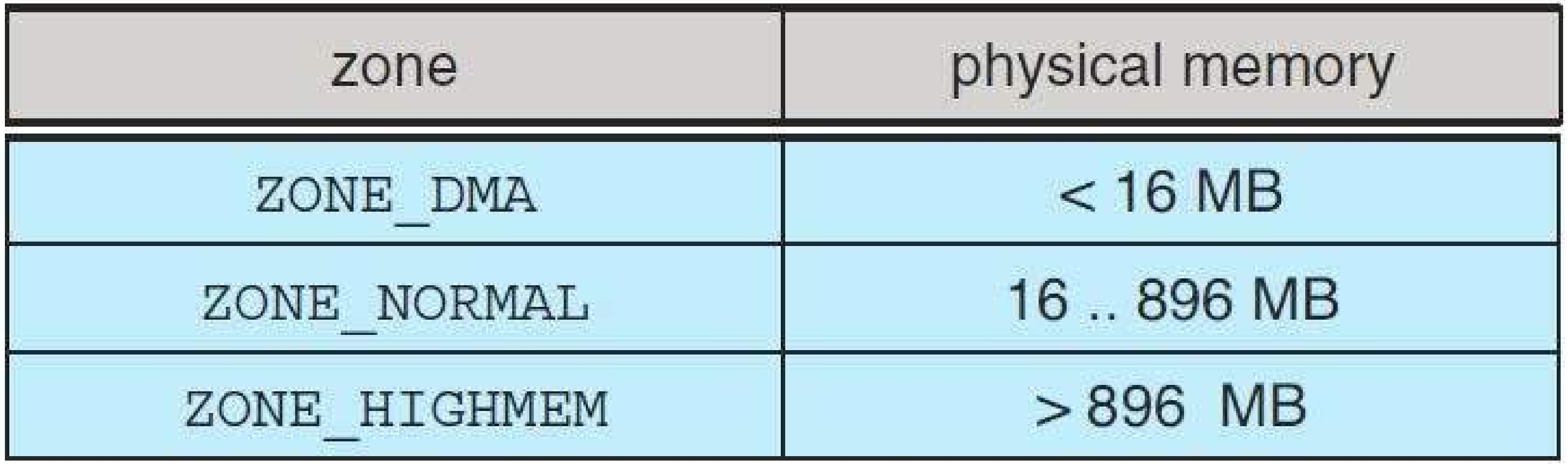
Management of Physical Memory

- Due to Hardware constraints , The physical memory is divided into four zones

. These Zones are Architecture specific
- When our request of Physical memory arrives, Kernel satisfies the request using appropriate zone.

1. ZONE_DMA
2. ZONE_DMA32
3. ZONE_NORMAL
4. ZONE_HIGHMEM

- In INTEL X86-32 devices can only access the lower 16MB of physical memory using DMA. This comprise ZONE_DMA
- On Other, can only access first 4GB of physical memory, despite supporting 64-bit address . This comprise ZONE_DMA32
- ZONE_HIGHMEM refers to physical memory that is not mapped into the kernel address space . In INTEL 32 bit, first 896MB by kernel space, remaining referred as high memory by ZONE_HIGHMEM
- Everything else-the normal, regularly mapped pages ZONE_NORMAL
- In 64-bit INTEL X86_64, first 16MB by ZONE_DMA, NO high memory, all other ZONE_NORMAL.



zone	physical memory
ZONE_DMA	< 16 MB
ZONE_NORMAL	16 .. 896 MB
ZONE_HIGHMEM	> 896 MB

e 18.3 Relationship of zones and physical addresses in Intel x86-32.

- Physical memory allocation will be done mainly in two ways
  1) **PAGE ALLOCATION** 2) **SLAB ALLOCATION**
- The Primary physical Memory Manager_ **PAGE ALLOCATOR**
- Page allocator allocates and frees physical pages for the zone and is capable of allocating ranges of physically contiguous pages on request.
- Each Zone has it’s own allocator.
- **Buddy System_**to keep track of available pages. Adjacent units of allocable memory are paired together . Each allocated memory has a adjacent pair called as **Budd**y.

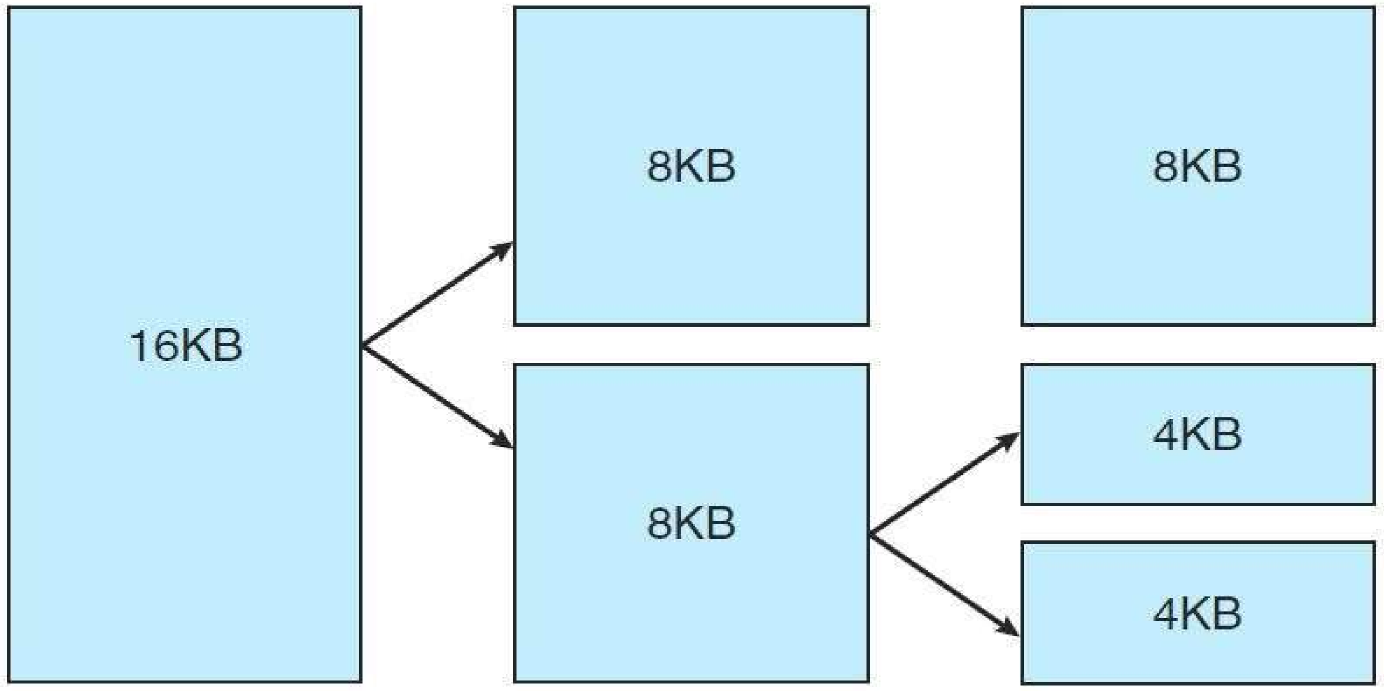


Figure 18.4 Splitting of memory in the buddy system.

- Whenever two allocated regions are freed up, they combined to form a larger region called as **Buddy Heap**.
- If smaller region requests are there, then this larger region subdivided into two partners to satisfy the request.
- In the picture in previous slide we can see, for a 4KB request a 32KB page divided into two 16 KB then a 16KB to two 8 KB and then our request satisfied by dividing 8KB into two 4KB partners.
- Kernel provides additional allocation for arbitrary size requests
- Many components in LINUX OS need to allocate entire pages on demand, but often smaller blocks of memory requests will happen.
- **Kmalloc()**_Like in Dynamic memory allocation in C programming , In LINUX this command allocates entire physical pages on demand but then splits into smaller pieces.
- **Kfree()**_Memory regions claimed by kmalloc() are permanent until they are freed explicitly by the call kfree().

- **<u>Slab Allocation</u>**
- The slab allocation algorithm uses caches to store kernel objects. When a cache is created, a number of objects allocated to cache and number of objects depends on associated slab.
- Each cache is populated with objects.
- **Slab** - made of one or more physically contiguous pages
  - allocates memory for kernal data structures
- **Cache** - Consists of one or more slabs
  - Single cache for each unique kernal data structure
  - eg. Process descriptors , file objects , Inodes.
- **Objects** – Instantiations of kernal data structures that cache represents

- A slab has 3 possible states
  1. Full
  2. Empty
  3. Partial
  - A slab consisting of all allocated objects called as **Full Slab** , all free objects as **Empty slab** and mixture of allocated and free objects as **Partial slab**.
  - If a object request from kernel occurs, first asks for partial slab, If none exists, a free object assigned from empty slab, If no empty slabs are available, a new slab is allocated from contiguous physical pages and assigned to cache.

- For example, The cache representing inodes stores instances of inode structures, and the cache representing process descriptors stores instances of process descriptor structures.
- The below figure shows the relation between slabs, caches and objects . Here two 3KB size kernel objects and three of 7KB size are stored in their respective caches.
- Two other main sub systems which do their own management of physical memory are **page cache** and **virtual memory**

- **Page cache –** kernal's main cache for files and is the main mechanism through which I/O to block devices is performed

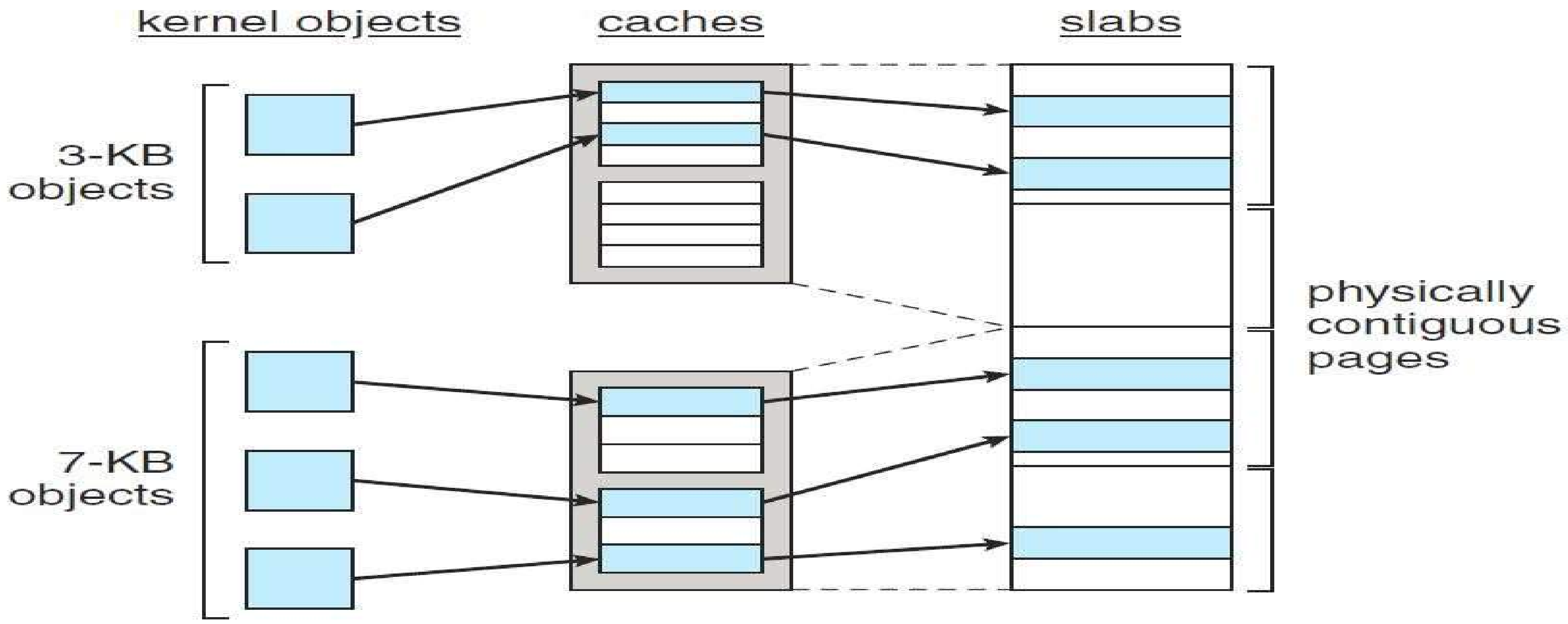


Figure 18.5 Slab allocator in Linux.

Virtual Memory

- Responsible for maintaining the address space accessible to each process.
- Creates pages for virtual memory on demand and manages loading those pages from disk and swapping them back out to disk as required.
- VMM maintains two separate views of a process address space
    1. Logical View
    2. Physical View
- In **Logical View** , the address space consists of a set of non overlapping regions, each region representing a continuous, page aligned subset of address space.
- Logical view, describing instructions that the virtual memory system has received concerning the layout of the address space.
- Each region described internally by vm_area_struct structure that defines the properties of the region, including processes read, write and execution permissions in the region as well as information about any files associated with the region.
- The region for each address space are linked into a binary tree to allow fast lookup of the region corresponding to any virtual address.

- **Physical View** stored in hardware page tables for the process.
- The page table entries identify the exact current location of each page of the virtual memory, whether it is on disk or physical memory.

Virtual Memory regions

- LINUX implements several types of virtual memory regions. One property that characterizes virtual memory is the backing store for region, which describes where the pages for the region come from.
- Demand Zero Memory-A region backed by nothing is the simplest type of virtual memory region.
- A virtual memory is also defined by it's reaction to writes. The mapping of a region into the processes address space can be either private or shared.

Life time Of a Virtual Address space

- Kernal creates a new virtual address space in two situations:

exec():when process runs a new program

when a new program executed , the process is given a new , completely empty virtual address space

fork(): When a new process is created

creating a new process with fork() involves creating a complete copy of the existing processes virtual address space.

Swapping and Paging

VM relocates pages of memory from physical memory out to disk.
UNIX systems uses Swapping out the contents of entire process at once.

- PAGING –The movement of individual pages of virtual memory between physical memory and disk.
- Paging mechanism in LINUX can be done by two parts one is Policy Algorithm and other is Paging Mechanism.

$$\begin{tikzcd}
 & \textbf{PAGING} \arrow[dl] \arrow[dr] & \\
\textbf{Policy Algorithm} & & \textbf{Paging Mechanism}
\end{tikzcd}$$

Policy Algorithm
(Which pages to write out to disk and when to write them)

Paging Mechanism
(carries out the transfer and backing of pages data into physi-cal memory when they needed)

Kernel Virtual Memory

- LINUX reserves for it’s own internal use a constant, architecture dependent region of the virtual address space of every process.
- The page table entries that map to these kernel pages are marked protected, so that these pages are not visible or modifiable when the processor is running in user mode.

Mapping of programs into Memory

- Under LINUX the binary loader does not load a binary file into physical memory. Rather, the pages of the binary file are mapped into regions of virtual memory
- It is responsible of the kernel’s binary loader to setup the initial memory mapping
- Older versions used binary files, newer use ELF files.
- An ELF binary file consists of a header followed by several page aligned sections.
- The ELF loader works by reading the header and mapping the sections of the file into separate regions of virtual memory.
- Loader initiates memory mapping to allow the execution of program to start . The region that need to be initialized include the stack and the program’s text and data regions.

- Each process has a pointer brk, that points to the current extent of this data region and process can extend or contract their brk region with a single system call **sbrk(**).
- The below diagram shows the memory layout of ELF programs how they can be mapped to memory regions and when stack and the program text and data regions initialized.

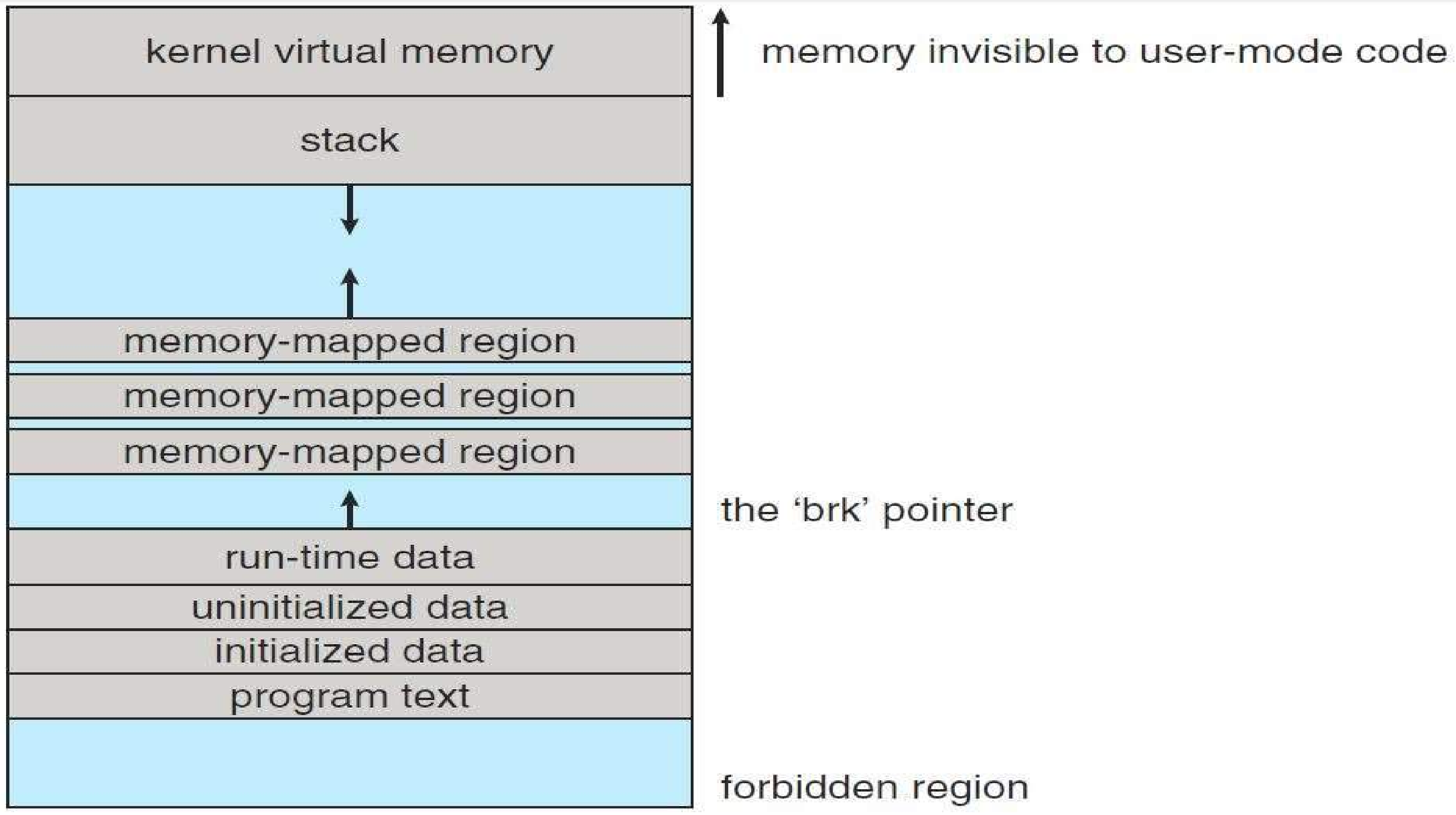


Figure 18.6 Memory layout for ELF programs.

Kernel Virtual Memory

- LINUX reserves for it’s own internal use a constant, architecture dependent region of the virtual address space of every process.
- The page table entries that map to these kernel pages are marked protected, so that these pages are not visible or modifiable when the processor is running in user mode.

Mapping of programs into Memory

- Under LINUX the binary loader does not load a binary file into physical memory. Rather, the pages of the binary file are mapped into regions of virtual memory
- It is responsible of the kernel’s binary loader to setup the initial memory mapping
- Older versions used binary files, newer use ELF files.
- An ELF binary file consists of a header followed by several page aligned sections.
- The ELF loader works by reading the header and mapping the sections of the file into separate regions of virtual memory.
- Loader initiates memory mapping to allow the execution of program to start . The region that need to be initialized include the stack and the program’s text and data regions.

- Each process has a pointer brk, that points to the current extent of this data region and process can extend or contract their brk region with a single system call **sbrk(**).
- The below diagram shows the memory layout of ELF programs how they can be mapped to memory regions and when stack and the program text and data regions initialized.

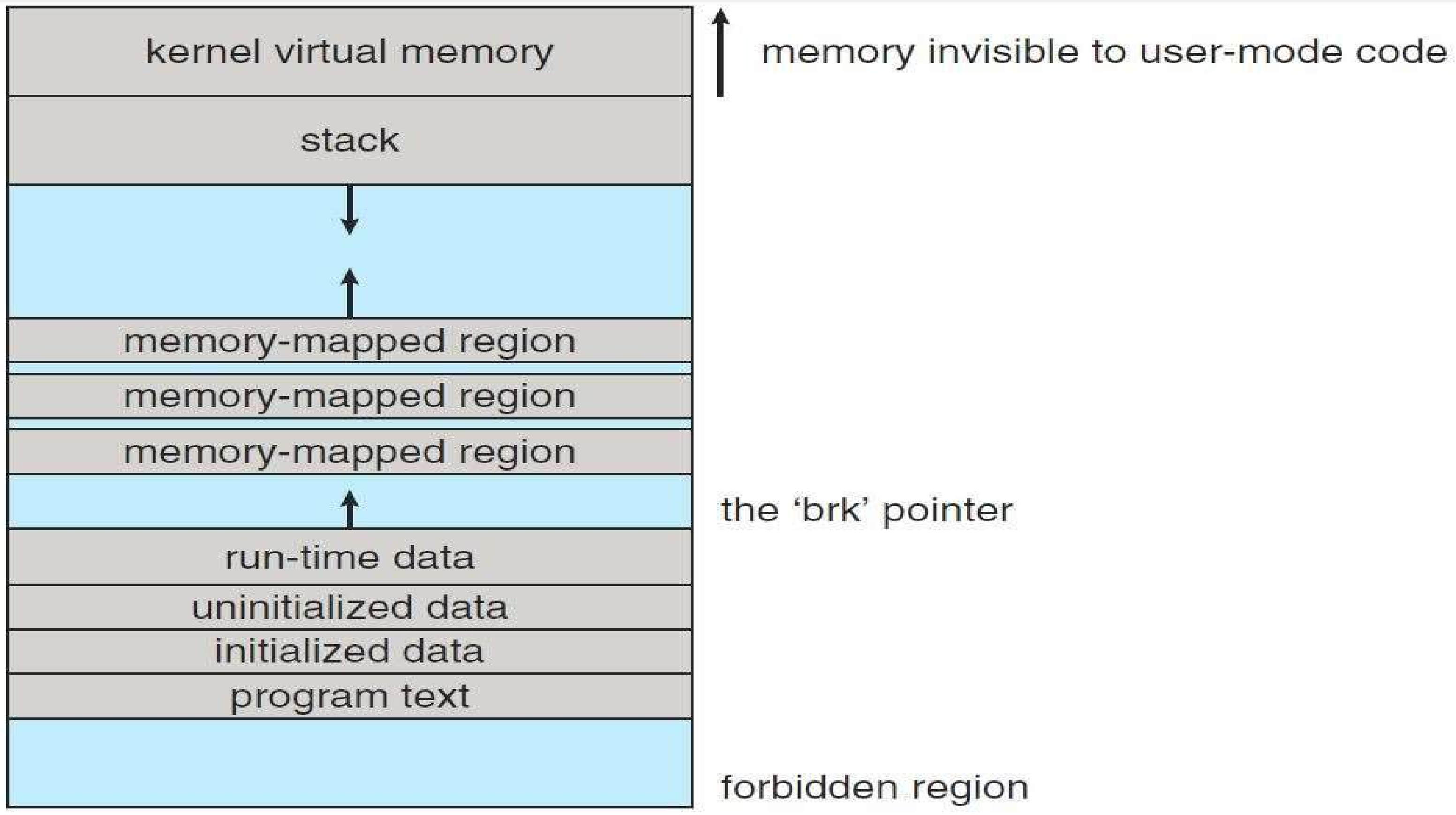


Figure 18.6 Memory layout for ELF programs.

FILE SYSTEM MANGEMENT IN LINUX

TOPICS

Sohan Das
M.Tech CrS2119
ISI ,Kolkata

Under the supervision of
Dr. Sushmita Mitra
Machine Intelligence Unit (MIU)
Indian Statistical Institute,Kolkata

❖ Introduction :

A Linux file system is a logical collection of files on a disk drive or a partition. A partition is a segment of memory and contains some specific data. In our machine, there can be various partitions of the memory. Generally, every partition contains a file system.

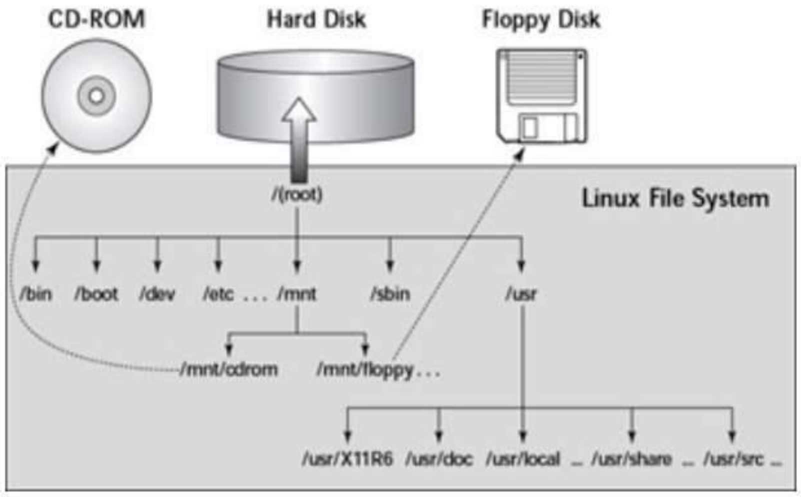
The general-purpose computer system needs to store data systematically so that we can easily access the files in less time. It stores the data on hard disks (HDD) or some equivalent non-volatile storage type . There may be below reasons for maintaining the file system:

- Primarily the computer saves data to the RAM storage; it may lose the data if it gets turned off. However, there is non-volatile RAM (Flash RAM and SSD) that is available to maintain the data after the power interruption.

- Data storage is preferred on hard drives as compared to standard RAM as RAM costs more than disk space. The hard disks costs are dropping gradually comparatively the RAM.

❖ What is Linux File System ?

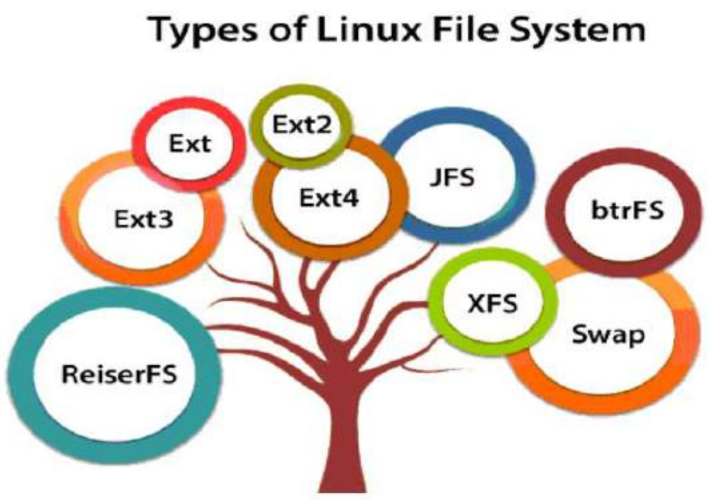
Linux file system is generally a built-in layer of a Linux file system used to handle the data management of the storage. It helps to arrange the file on the disk storage. It manages the file name, file size, creation date, and much more information about a file.



❖ Types of Linux File System

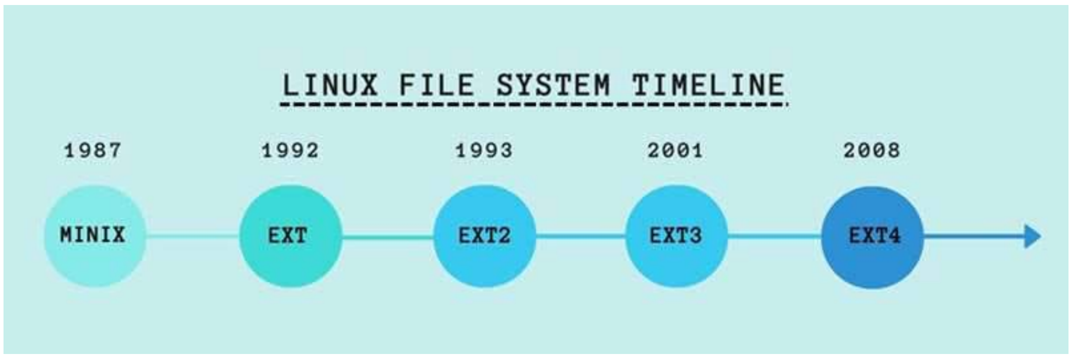
A standard Linux Distribution provides the choice of partitioning disk with the filttte formats listed below, each of which has special meaning associated with it.

Ext , Ext2 , Ext3 , Ext4 , JFS , ReiserFS , XFS , Btrfs , Swap FS etc.



1. Ext File System :

The extended file system, or ext, was implemented in April 1992 as the first file system created specifically for the Linux kernel. It has metadata structure inspired by traditional Unix filesystem principles, and was designed by Rémy Card to overcome certain limitations of the MINIX file system. The Ext file system is an older version, and is no longer used due to some limitations.



➢ Ext2 :

It is the upgraded version of Ext file system. Ext2 is the first Linux file system that allows managing two terabytes of data Ext2 is still recommended over journaling file systems on bootable USB flash drives and other solid-state drives.

Theoretical ext2 limits under Linux

Block size:	1 KiB	2 KiB	4 KiB	8 KiB
max. file size:	16 GiB	256 GiB	2 TiB	2 TiB
max. filesystem size:	4 TiB	8 TiB	16 TiB	32 iB

- **Ext3** :
  Ext3 is developed through Ext2; it is an upgraded version of Ext2 and contains backward compatibility. The major drawback of Ext3 is that it does not support servers because this file system does not support file recovery and disk snapshot.

  Ext3 adds the following features to ext2:
  - A journal
  - Online file system growth
  - HTree indexing for larger directories

- **Ext4 :**
  The Ext4 journaling file system or fourth extended filesystem is a journaling file system for Linux, developed as the successor to Ext3.

  Nowadays Ext4 is the default file system of many Linux distributions including Debian ,Ubuntu .

Features of Ext4:

i. **Large file system :** The ext4 filesystem can support volumes with sizes up to 1 exbibyte (EiB) and single files with sizes up to 16 tebibytes (TiB) with the standard 4 KiB block size.

ii. **Extents :** Extents replace the traditional block mapping scheme used by Ext2 and Ext3. An extent is a range of contiguous physical blocks, improving large-file performance and reducing fragmentation. A single extent in Ext4 can map up to 128 MiB of contiguous space with a 4 KiB block size. There can be four extents stored directly in the inode. When there are more than four extents to a file, the rest of the extents are indexed in a tree.

iii. **Multiblock allocation :** When Ext3 needs to write new data to the disk, there's a block allocator that decides which free blocks will be used to write the data. But the Ext3 block allocator only allocates one block (4KiB) at a time. That means that if the system needs to write the 100 MiB data mentioned in the previous point, it will need to call the block allocator 25600 times (and it was just 100 MiB!). Not only this is inefficient, it doesn't allow the block allocator to optimize the allocation policy because it doesn't know how many total data is being allocated, it only knows about a single block. Ext4 uses a "multiblock allocator" (mballoc) which allocates many blocks in a single call, instead of a single block per call, avoiding a lot of overhead. This improves the performance, and it's particularly useful with delayed allocation and extent.

iv. **Delaying block allocation :** Another file size-based optimization is delayed allocation. This performance optimization delays the allocation of physical blocks on the disk until they are to be flushed to the disk. The key to this optimization is that by delaying the allocation of physical blocks until they need to be written to the disk, more blocks are present to allocate and write in contiguous blocks. This is similar to persistent preallocation except that the file system performs the task automatically. But if the size of the file is known beforehand, persistent preallocation is the best route.

v. **Forward and backward compatibility :** As ext3 was one of the most popular file systems in use by Linux , migrating to ext4 should be simple and painless. For this reason, ext4 was designed to be both forward and backward (to an extent) compatible. Ext4 is forward compatible in that you can mount an ext3 file system as an ext4 file system. To fully take advantage of ext4, a file system migration is necessary to convert and exploit the new ext4 format.

We can also mount an ext4 file system as ext3 (backward compatible). In addition to the compatibility features, when we want to migrate an ext3 file system to ext4, you can do so gradually. This means that old files that we have not moved can remain in the older ext3 format, while new files (or older files that have been copied) will occupy the new ext4 data structures. In this way, we can migrate an ext3 file system online to an ext4 file system.

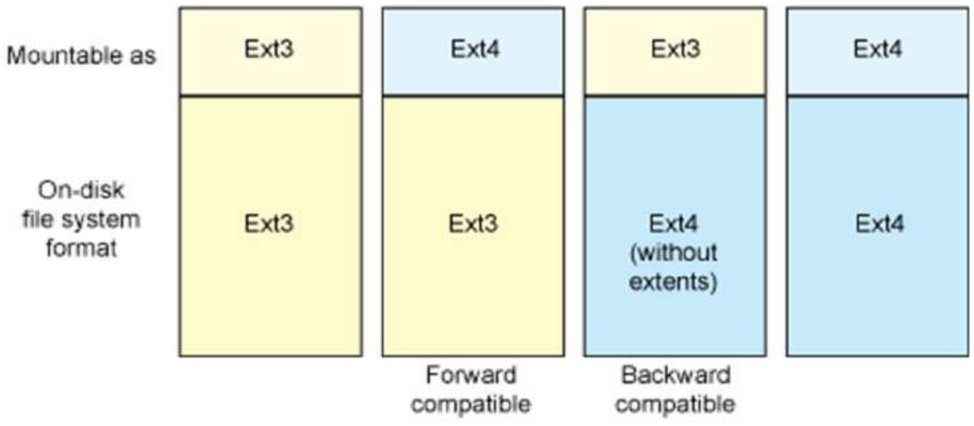


Figure : Forward and Backward compatibility

vi. **Online defragmentation :** Although ext4 incorporates features to reduce fragmentation within the file system (extents for sequential block allocation), some amount of fragmentation is impossible to avoid when a file system exists for a long period of time. For this reason, an online defragmentation tool exists to defragment both the file system and individual files for improved performance. The online defragmenter is a simple tool that copies files into a new ext4 inode that refers to contiguous extents.

The other aspect of online defragmentation is the reduced time required for a file system check (fsck). Ext4 marks unused groups of blocks within the inode table to allow the fsck process to skip them entirely to speed the check process. When the operating system decides to validate the file system because of internal corruption (which is inevitable as file systems increase in size and distribution), ext4's overall design means improved overall reliability.

vii. **Persistent pre-allocation :** When pre-allocating disk space for a file, most file systems must write zeroes to the blocks for that file on creation. Ext4 allows the use of **fallocate()** instead, which guarantees the availability of the space (and attempts to find contiguous space for it) without first needing to write to it. This significantly increases performance in both writes and future reads of the written data for streaming and database applications.

viii. **Journal checksumming :** The journal is the most used part of the disk, making the blocks that form part of it more prone to hardware failure. And recovering from a corrupted journal can lead to massive corruption. Ext4 checksums the journal data to know if the journal blocks are failing or corrupted. But journal checksumming has a bonus: it allows one to convert the two-phase commit system of Ext3's journaling to a single phase, speeding the filesystem operation up to 20% in some cases - so reliability and performance are improved at the same time.

2. Journaled File System (JFS) :

It is a 64-bit journaling file system created by IBM. JFS is a modern file system supporting many features, a few of which are listed here.

- fully 64-bit.
- dynamic space allocation for i-nodes, i.e. no running out of i-nodes on file systems with large number of small files.
- Directory structures designed for speed and efficiency:
- directories with eight or fewer entries have their contents storied inline within that directory's i-node.
- directories with more than eight entries have their contents stored in a B+ tree keyed on name.
- JFS utilizes extents for allocating blocks for large files.

3. **ReiserFS (Reiser File System)** :

It is a journaling file system for Linux. ReiserFS is known to be particulary good at handling the large numbers of small files. It is the default file system on a number of distributions, including: Elive, Xandros, Linspire, and GoboLinux.
In addition, the file system supports fast rebuilding of its tree and has extensive recovery capabilities. It is faster than Ext4 family file systems. And the most important thing is that ReiserFS can be used as the main file system for the root, as well as Ext4. But its disadvantage is that you cannot use background encryption.

ReiserFS supports filenames up to 4032 bytes, maximum file size of up to 8 TB, and file system of up to 16 TB.

At the moment, there are two stable versions of the file system. It’s Reiser3 or as it is called ReiserFS and the new version Reiser4. It has added encryption, improved performance, and much more.

4. **XFS** :

XFS is a high-performance 64-bit journaling file system created by Silicon Graphics, Inc (SGI) in 1993.
This file system was released under the GNU General Public License (GPL) in May 2000. A team led by Steve Lord at SGI ported it to Linux, and first support by a Linux distribution came in 2001. This support gradually became available in almost all Linux distributions.

- XFS is a high-performance 64-bit open source filesystem merged into the Linux kernel.

- XFS is supported by most Linux distributions, and is even used as the default filesystem for some Linux distributions.

- XFS supports large files and large file systems. Therefore it can be used as Linux filesystem solution to store file contents for a FileNet® repository (File Storage Area).

- XFS is a journaling filesystem and maintains a change log holding the filesystem modifications before they are committed to the filesystem assuring the consistency of the filesystem.

5. **Btrfs (B tree file system)** :

Btrfs is a copy on write filesystem for Linux aimed at implementing advanced features while focusing on fault tolerance, repair and easy administration. Jointly developed by several companies, licensed under the GPL and open for contribution from anyone.

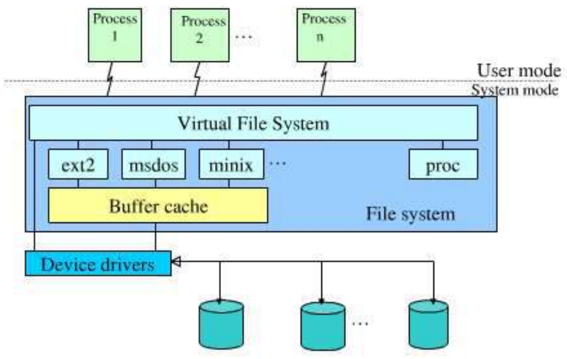
The main Btrfs features include:

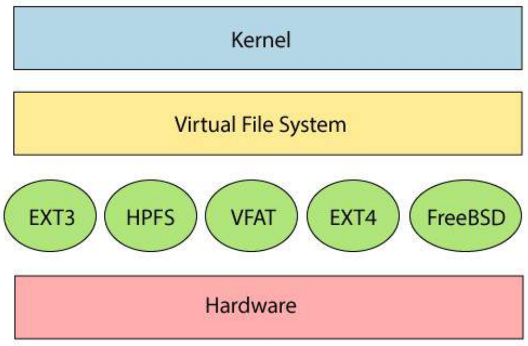
- Extent based file storage (2^64 max file size)
- Space efficient packing of small files
- Space efficient indexed directories
- Dynamic inode allocation
- Writable snapshots
- Subvolumes (separate internal filesystem roots)
- Object level mirroring and striping
- Checksums on data and metadata (multiple algorithms available)
- Compression
- Integrated multiple device support, with several raid algorithms
- Offline filesystem check
- Efficient incremental backup and FS mirroring
- Online filesystem defragmentation.

❖ **VFS (Virtual File System)**

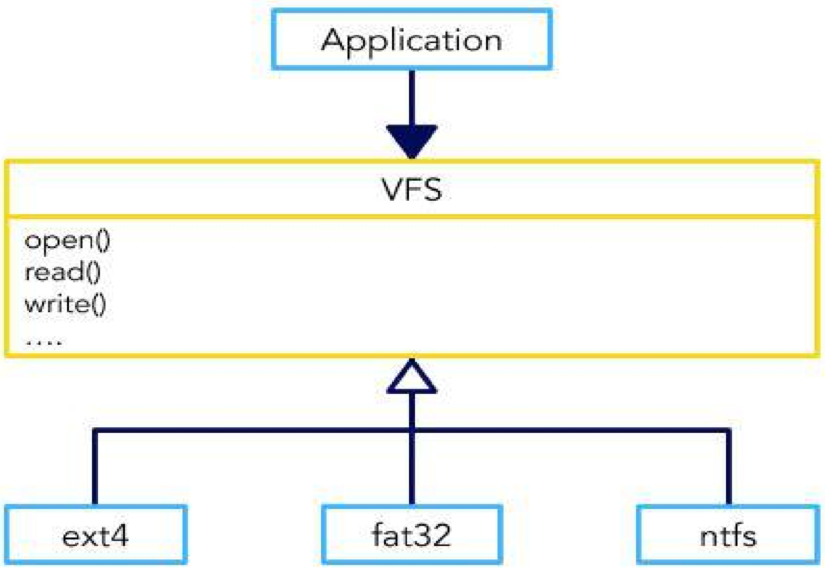
Linux file system contains two-part file system software implementation architecture. The first two parts of the given file system together called a Linux virtual file system. It provides a single set of commands for the kernel and developers to access the file system. This virtual file system requires the specific system driver to give an interface to the file system.

This file system requires an API (Application programming interface) to access the function calls to interact with file system components like files and directories. API facilitates tasks such as creating, deleting, and copying the files. It facilitates an algorithm that defines the arrangement of files on a file system.





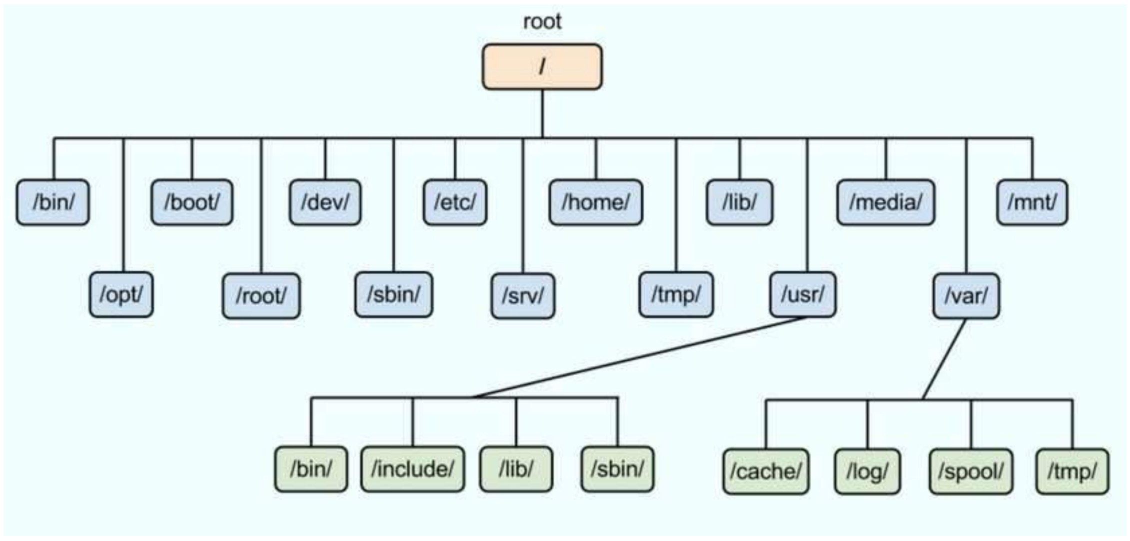
The VFS is sandwiched between two layers: the upper and the lower. The upper layer is the system call layer where a user space process traps into the kernel to request a service. The lower layer is a set of function pointers, one set per filesystem implementation, which the VFS calls when it needs an action performed that requires information specific to a particular filesystem. Taken another way, the VFS could be thought of as the glue between the user space applications’ requests for file-related services and the filesystem-specific functions which the VFS invokes as needed. From a high-level view, this can be modeled as a standard abstract interface:



By having the VFS require filesystems to define certain callback functions, such as read and write, we can operate on files without worrying about the implementation details. This way, the user space programmer is only concerned with reading from a file or writing to it and not with how the file is physically stored in a USB flash drive, disk drive, over the network, or on any other conceivable medium.

❖ Linux File System Structure

Linux file system has a hierarchal file structure as it contains a root directory and its subdirectories. All other directories can be accessed from the root directory. A partition usually has only one file system, but it may have more than one file system.

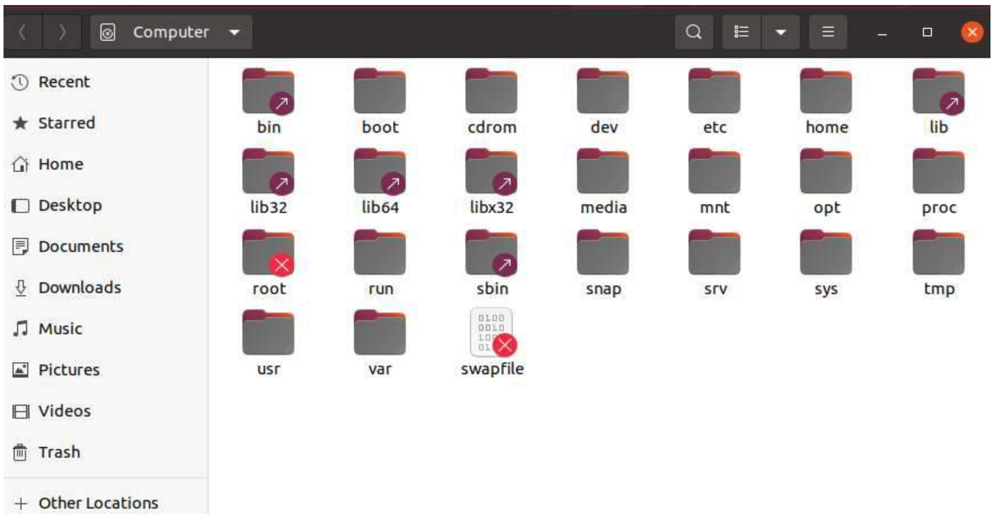


1. **/ (Root):** Primary hierarchy root and root directory of the entire file system hierarchy.

   - Every single file and directory starts from the root directory
   - The only root user has the right to write under this directory
   - /root is the root user’s home directory, which is not the same as /

2. **/bin – User Binaries**

   - Contains binary executables.
   - Common linux commands you need to use in single-user modes are located under this directory.
   - Commands used by all the users of the system are located here.
   - For example: ps, ls, ping, grep, cp.



3. /boot – Boot Loader Files

- Contains boot loader related files.
- Kernel initrd, vmlinux, grub files are located under /boot
- For example: initrd.img-2.6.32-24-generic, vmlinuz-2.6.32-24-generic

4. /dev : Essential device files.

- These include terminal devices, usb, or any device attached to the system.
- Example: /dev/tty1, /dev/usbmon0

5. /etc : Host-specific system-wide configuration files.

- Contains configuration files required by all programs.
- This also contains startup and shutdown shell scripts used to start/stop individual programs.
- Example: /etc/resolv.conf, /etc/logrotate.conf.

6. /home : Users' home directories, containing saved files, personal settings, etc.

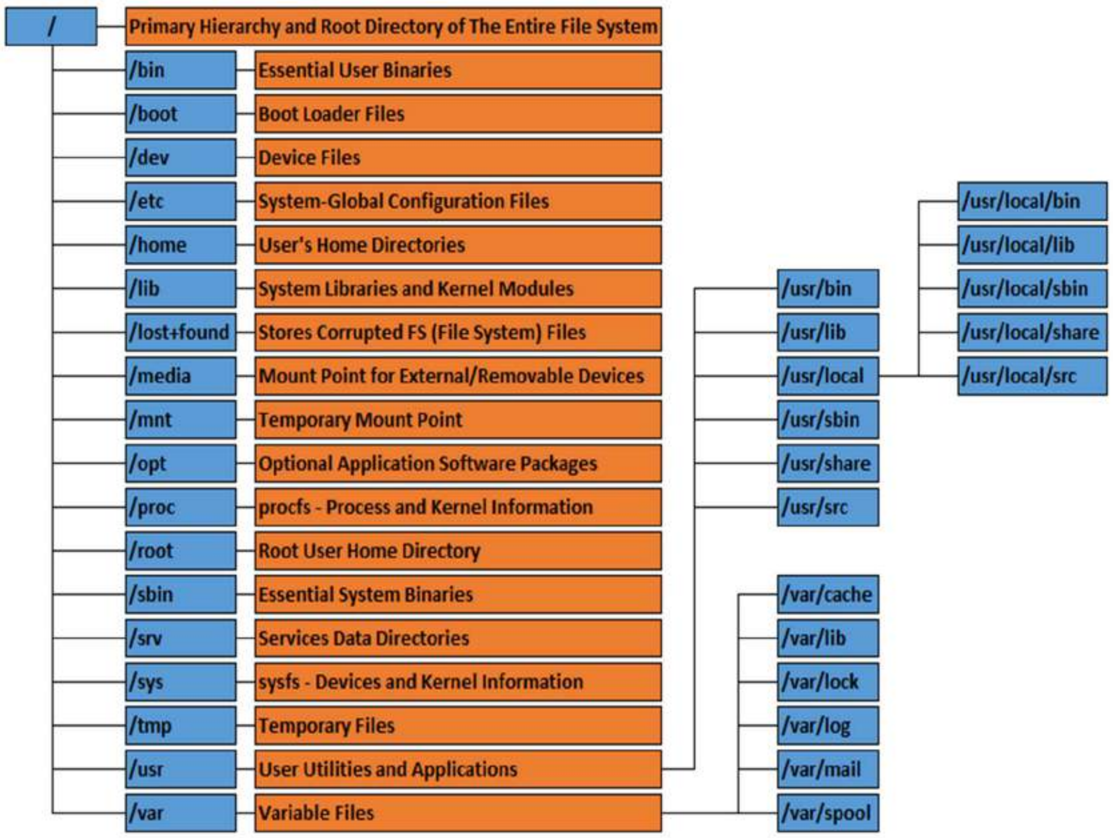
- Home directories for all users to store their personal files.
- example: /home/kishlay, /home/kv

7. **/lib : Libraries essential for the binaries in /bin/ and /sbin/.**
    - Library filenames are either ld* or lib*.so.*
    - Example: ld-2.11.1.so, libncurses.so.5.7

8. **/media : Mount points for removable media such as CD-ROMs (appeared in FHS-2.3).**
    - Temporary mount directory for removable devices.
    - Examples, /media/cdrom for CD-ROM; /media/floppy for floppy drives; /media/cdrecorder for CD writer

9. **/mnt : Temporarily mounted filesystems.**
    - Temporary mount directory where sysadmins can mount filesystems.

10. **/opt : Optional application software packages.**
    - Contains add-on applications from individual vendors.
    - Add-on applications should be installed under either /opt/ or /opt/ sub-directory.



11. /sbin : Essential system binaries, e.g., fsck, init, route.

- Just like /bin, /sbin also contains binary executables.
- The linux commands located under this directory are used typically by system administrator, for system maintenance purpose.
- Example: iptables, reboot, fdisk, ifconfig, swapon.

12. /srv : Site-specific data served by this system, such as data and scripts for web servers,
data offered by FTP servers, and repositories for version control systems.

- srv stands for service.
- Contains server specific services related data.
- Example, /srv/cvs contains CVS related data.

13. /tmp : Temporary files. Often not preserved between system reboots, and may be severely size restricted.

- Directory that contains temporary files created by system and users.
- Files under this directory are deleted when system is rebooted.

14. /usr : Secondary hierarchy for read-only user data; contains the majority of (multi-)user utilities and applications.

- Contains binaries, libraries, documentation, and source-code for second level programs.
- /usr/bin contains binary files for user programs. If you can't find a user binary under /bin, look under /usr/bin. For example: at, awk, cc, less, scp
- /usr/sbin contains binary files for system administrators. If you can't find a system binary under /sbin, look under /usr/sbin. For example: atd, cron, sshd, useradd, userdel
- /usr/lib contains libraries for /usr/bin and /usr/sbin
- /usr/local contains users programs that you install from source. For example, when you install apache from source, it goes under /usr/local/apache2
- /usr/src holds the Linux kernel sources, header-files and documentation.

15. /proc : Virtual filesystem providing process and kernel information as files. In Linux, corresponds to a procfs mount. Generally automatically generated and populated by the system, on the fly.

- Contains information about system process.
- This is a pseudo filesystem contains information about running process. For example: /proc/{pid} directory contains information about the process with that particular pid.
- This is a virtual filesystem with text information about system resources. For example: /proc/uptime

❖ FILE

➢ What is file ?

A file is a container in a computer system for storing information. Files used in computers are similar in features to that of paper documents used in library and office files. There are different types of files such as text files, data files, directory files, binary and graphic files, and these different types of files store different types of information. In a computer operating system, files can be stored on optical drives, hard drives or other types of storage devices.

A file must have a unique name within a given file directory. However, while creating a filename, certain characters are considered illegal, and hence cannot be used. A filename is comprised of a name with a suffix, which is also known as a file extension. The file extension is two to four characters following the period in the complete filename. The file extension helps in identifying the type of file, file format and the attributes associated with the file.

Linux provides security or protection measures against file corruption or damage. The data contained in the files could range from system-generated information to user-specified information.

➢ Everything in linux is a file

In Linux system, everything is a file and if it is not a file, it is a process. A file doesn't include only text files, images and compiled programs but also include partitions, hardware device drivers and directories. The whole Linux operating system is just a collection of files. Linux consider everything as a file.

This means that not only partitions are mounted as files but directories of specific devices such as RAM, smartphones, external disks, and optical media discs are all files. Besides these, sockets and pipes are also files!

➢ Types of FILES in Linux

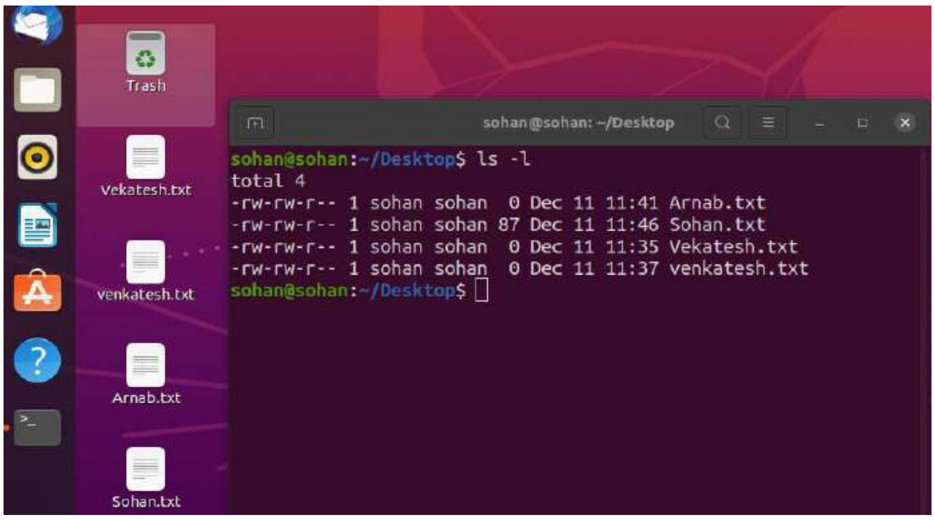
A file type helps us in identifying the type of content that is saved in the file. Linux supports seven different types of files. These file types are the Regular file, Directory file, Link file, Character special file, Block special file, Socket file, and Named pipe file.

- The following table provides a brief description of these file types.

File type	Description
Ordinary or regular files	Contain data of various content types such as text, script, image, videos, etc.
Directory files	Contain the name and address of other files.
Block or character special files	Represent device files such as hard drives, monitors, etc.
Link files	Point or mirror other files
Socket files	Provide inter-process communication
Named pipe files	Allow processes to send data to other processes or receive data from other processes.

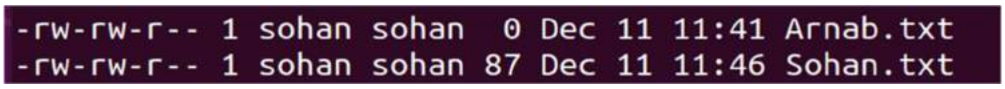
➢ Attributes of a File

Every file has some attributes or meta-data properties which tells about the file. We will understand it with the following example :

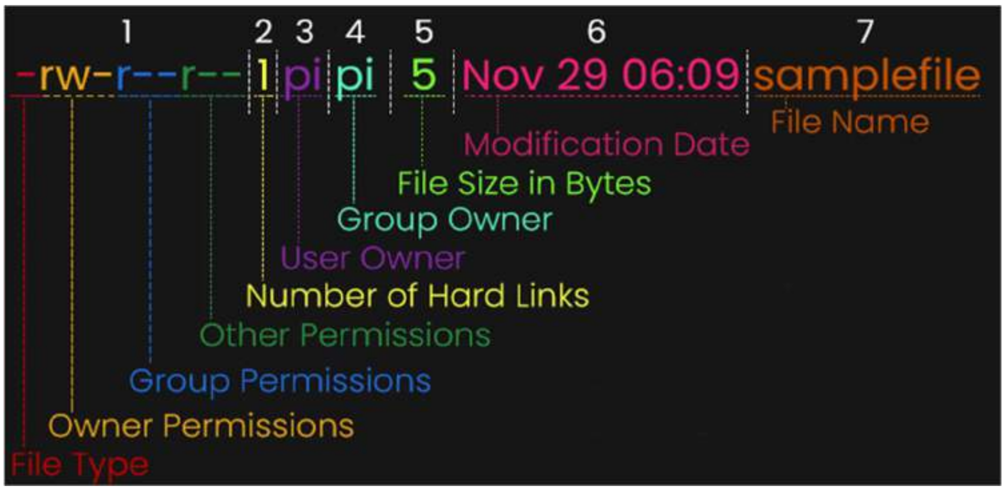


```
sohan@sohan:~/Desktop$ ls -l
total 4
-rw-rw-r-- 1 sohan sohan  0 Dec 11 11:41 Arnab.txt
-rw-rw-r-- 1 sohan sohan 87 Dec 11 11:46 Sohan.txt
-rw-rw-r-- 1 sohan sohan  0 Dec 11 11:35 Vekatesh.txt
-rw-rw-r-- 1 sohan sohan  0 Dec 11 11:37 venkatesh.txt
sohan@sohan:~/Desktop$ 
```

Using ls -l command we can see all the files with their attributes at the present directory. Here we can see that there are four text files present at the /Desktop directory.



```
-rw-rw-r-- 1 sohan sohan  0 Dec 11 11:41 Arnab.txt
-rw-rw-r-- 1 sohan sohan 87 Dec 11 11:46 Sohan.txt
```



Linux is a clone of UNIX, the multi-user operating system which can be accessed by many users simultaneously. Linux can also be used in mainframes and servers without any modifications. But this raises security concerns as an unsolicited or malign user can corrupt, change or remove crucial data. For effective security, Linux divides authorization into 2 levels.

1. **Ownership**
2. **Permission**

➢ Linux File Ownership

Every file and directory on Linux system is assigned 3 types of owner, given below.

1. **User :**

   A user is the owner of the file. By default, the person who created a file becomes its owner. Hence, a user is also sometimes called an owner.

2. **Group :**

   A user- group can contain multiple users. All users belonging to a group will have the same Linux group permissions access to the file. Suppose you have a project where a number of people require access to a file. Instead of manually assigning permissions to each user, you could add all users to a group, and assign group permission to file such that only this group members and no one else can read or modify the files.

3. **Other :**

Any other user who has access to a file. This person has neither created the file, nor he belongs to a usergroup who could own the file. Practically, it means everybody else. Hence, when you set the permission for others, it is also referred as set permissions for the world.

Now, the big question arises how does Linux distinguish between these three user types so that a user 'A' cannot affect a file which contains some other user 'B's' vital information/data. It is like you do not want your colleague, who works on your Linux computer, to view your images. This is where Permissions set in, and they define user behavior.

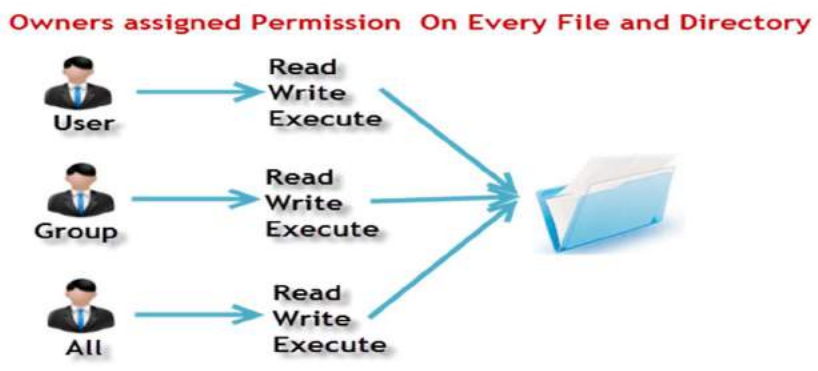
➢ Linux File Permissions

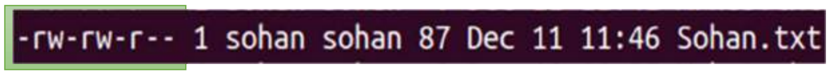
Every file and directory in the Linux system has following 3 permissions defined for all the 3 owners discussed above.

- **Read:** This permission gives us the authority to open and read a file. Read permission on a directory gives us the ability to lists its content.

- **Write:** The write permission gives us the authority to modify the contents of a file. The write permission on a directory gives us the authority to add, remove and rename files stored in the directory. Consider a scenario where we have to write permission on file but do not have write permission on the directory where the file is stored. We will be able to modify the file contents. But we will not be able to rename, move or remove the file from the directory.

- **Execute:** In Windows, an executable program usually has an extension ".exe" and which you can easily run. In Linux, we cannot run a program unless the execute permission is set. If the execute permission is not set, we might still be able to see/modify the program code(provided read & write permissions are set), but not run it.





```
-rw-rw-r-- 1 sohan sohan 87 Dec 11 11:46 Sohan.txt
```

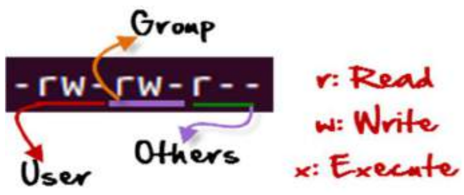
Here, we have highlighted **'-rw-rw-r--'** and this weird looking code is the one that tells us about the Unix permissions given to the owner, user group and the world.

Here, the first '**–**' implies that we have selected an ordinary file.

If it were a directory, **d** would have been shown.

The characters are pretty easy to remember.

- **r** = read permission
- **w** = write permission
- **x** = execute permission
- **–** = no permission



In the above example the first part of the code is **'rw-'** . This suggests that the owner 'Home' can:

- Read the file
- Write or edit the file
- He cannot execute the file since the execute bit is set to '-'.

The second part is **'rw-' .** It for the user group 'Home' and group-members can:

- Read the file
- Write or edit the file
- Cann't execute the file

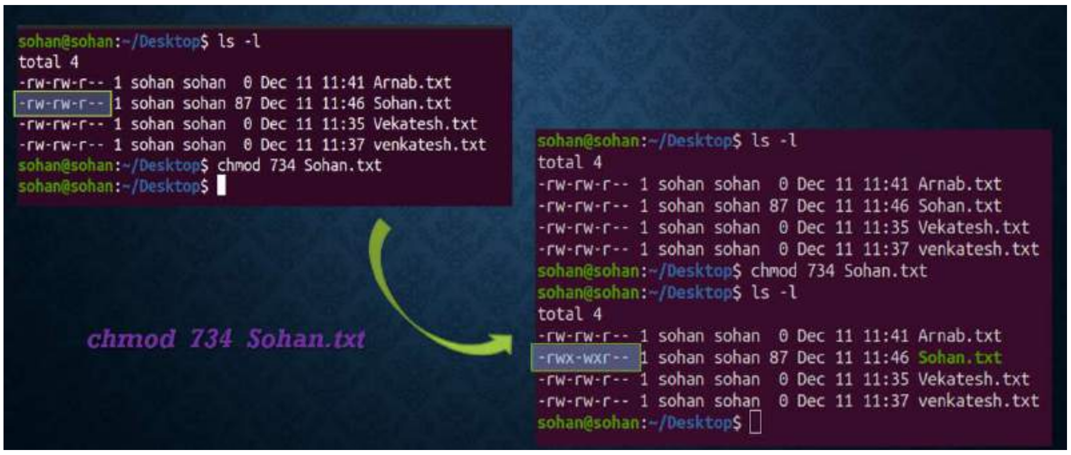
The third part is for the world which means any user. It says **'r--' .** This means the user can only:

- Read the file
- Cann't write or edit the file
- Cann't execute the file

➢ Changing file/directory permissions in Linux Using ‘chmod’ command

Say we do not want our friends to see our personal images. This can be achieved by changing file permissions.

We can use the ‘**chmod’** command which stands for ‘change mode’. Using the command, we can set permissions (read, write, execute) on a file/directory for the owner, group and the world.



```
sohan@sohan:~/Desktop$ ls -l
total 4
-rw-rw-r-- 1 sohan sohan  0 Dec 11 11:41 Arnab.txt
-rw-rw-r-- 1 sohan sohan 87 Dec 11 11:46 Sohan.txt
-rw-rw-r-- 1 sohan sohan  0 Dec 11 11:35 Vekatesh.txt
-rw-rw-r-- 1 sohan sohan  0 Dec 11 11:37 venkatesh.txt
sohan@sohan:~/Desktop$ chmod 734 Sohan.txt
sohan@sohan:~/Desktop$
```

```
sohan@sohan:~/Desktop$ ls -l
total 4
-rw-rw-r-- 1 sohan sohan  0 Dec 11 11:41 Arnab.txt
-rw-rw-r-- 1 sohan sohan 87 Dec 11 11:46 Sohan.txt
-rw-rw-r-- 1 sohan sohan  0 Dec 11 11:35 Vekatesh.txt
-rw-rw-r-- 1 sohan sohan  0 Dec 11 11:37 venkatesh.txt
sohan@sohan:~/Desktop$ chmod 734 Sohan.txt
sohan@sohan:~/Desktop$ ls -l
total 4
-rw-rw-r-- 1 sohan sohan  0 Dec 11 11:41 Arnab.txt
-rwx-wxr-- 1 sohan sohan 87 Dec 11 11:46 Sohan.txt
-rw-rw-r-- 1 sohan sohan  0 Dec 11 11:35 Vekatesh.txt
-rw-rw-r-- 1 sohan sohan  0 Dec 11 11:37 venkatesh.txt
sohan@sohan:~/Desktop$
```

Here we have changed the permissions **‘ rw-rw-r- -’** of the text file **‘ Sohan.txt ‘** to **‘ rwx-wxr- - ‘** using the simple command **‘ chmod 734 Sohan.txt ‘** .

Now we will discuss how we are getting those numbers like 734. Forget the old permissions. Just think about the new permissions **‘ rwx-wxr- - ‘**. In this new permission owner have permissions **‘rwx’** i.e owner can read, write or edit and execute the file. So for **rwx** we will construct a binary number **111** (since all those r,w,x are present. If one of them is not present i.e. when ‘-‘ is present then we will put 0 as that particular pemission is denied).

The Binary number **111** is equivalent to the decimal number **7**.

Similarly group has permissions **‘-wx’** i.e. group cann’t read but write and execute the file.

So we will form the binary number **011** which is equivalent to the decimal number **3**.

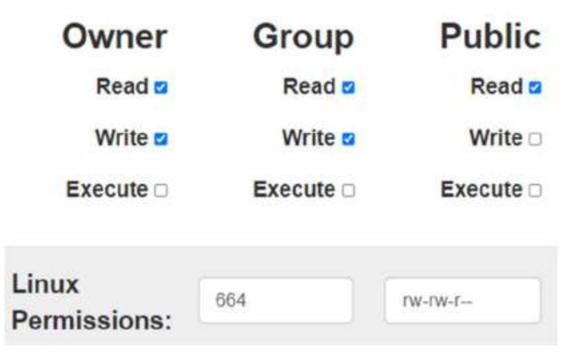
And at the last others have the permission **‘r- -‘** i.e. others can only read the file ,so we will assign the binary number **100** which is equivalent to the decimal number **4**.

In this way we are getting the number **734**.

<u>Owner</u> : r w x = $(111)_2$ = 1*2^2 +1*2^1 + 1*2^0 = 4+2+1 = $(7)_{10}$

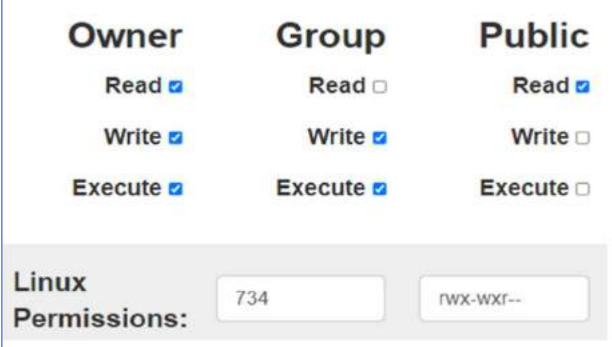
<u>Group</u> : - w x = $(011)_2$ = 0*2^2 +1*2^1 + 1*2^0 = 0+2+1 = $(3)_{10}$

<u>Public</u> : r - - = $(100)_2$ = 1*2^2 + 0*2^1 + 0*2^0 = 4+0+0 = $(4)_{10}$



	Owner	Group	Public
Read	☑	☑	☑
Write	☑	☑	☐
Execute	☐	☐	☐

Linux Permissions: 664 rw-rw-r--



	Owner	Group	Public
Read	☑	☐	☑
Write	☑	☑	☐
Execute	☑	☑	☐

Linux Permissions: 734 rwx-wxr--

➢ File Creating in Linux

Linux file system considers everything as a file in Linux; whether it is text file images, partitions, compiled programs, directories, or hardware devices. If it is not a file, then it must be a process. To manage the data, it forms a tree structure.

There are multiple ways to create a file in Linux. Some conventional methods are as follows:

- using cat command
- using touch command
- using redirect '>' symbol
- using echo command
- using printf command
- using a different text editor like vim, nano, vi

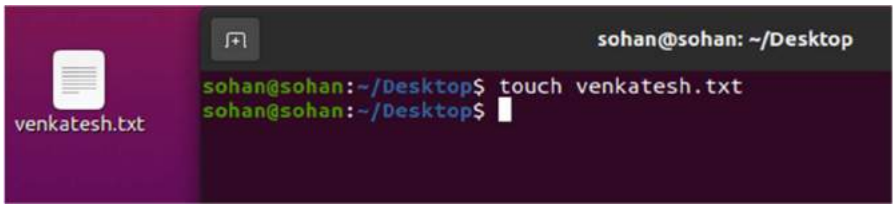
Apart from all of the above methods, we can also create a file from the desktop file manager. Let's understand the above methods in detail:

1. <u>Using **cat** Command</u> :
   Open the terminal and go to the directory where you want to create a new text file using **cd** command (**cd** stands for Change Directory).Then execute the **cat** command as follows:
   cat > test.txt

2. Using **touch** command :

   It is the simplest way to create a new file . We can create multiple files by executing this command at once.
   To create a file, execute the touch command followed by the file name as given below:

   touch venkatesh.txt



```
sohan@sohan:~/Desktop$ touch venkatesh.txt
sohan@sohan:~/Desktop$ █
```

   To create multiple file execute the command **touch test1.txt test2.odt test3.txt**

3. Using the **redirect (>)** symbol :

   To create a file with redirect (>) operator, execute the command as follows:

   >test4.txt

4. Using **echo** command :

   The **echo** command is used to create a file, but we should specify the file content on the command line.

   To create the file with the echo command, execute the command as follows:

   echo " File content" > test5.txt

5. Using **printf** command :

   We can also create a file using **printf** command. For this we need to specify the file content on the command line.
   To create a file with the **printf** command, execute the command as follows:

   printf " File content" > test6.txt

6. Using Text Editor :

   We can also create a file using the different text editors like **vim, nano, vi**, and more.

- Using **Vim** text editor :
  To create a file using the **vim** text editor (it must have to be installed , otherwise we have to install it), execute the below command:

  vim test7.txt

  The above command will open the text editor, press i key to go to the insert mode of the editor.
  Enter the file content, press Esc key preceded by :wq to save and exit the file.

- Using **Nano** editor :
  We can create a file using the **nano** text editor. To create a file, execute the below command:

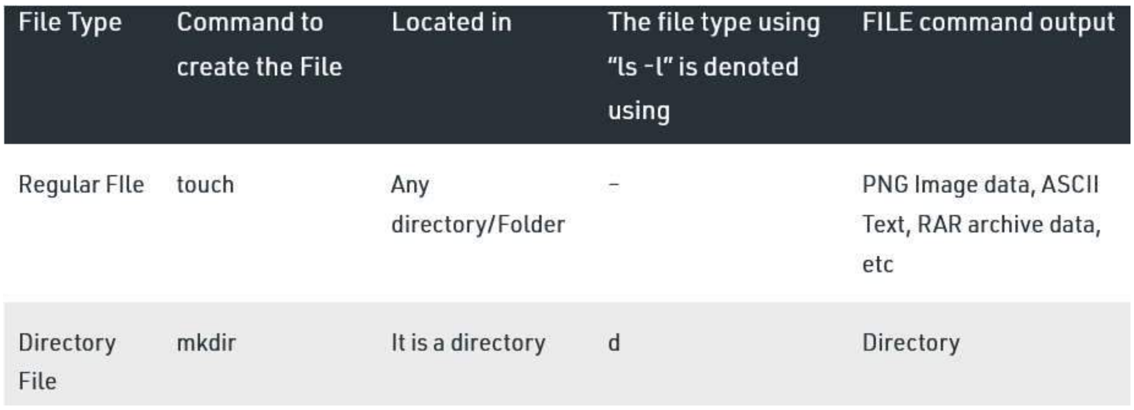
  nano test8.txt

  The above command will open the nano text editor. Enter the desired text and press CTRL + X then type y for confirmation of the file changes. Press Enter key to exit from the editor.

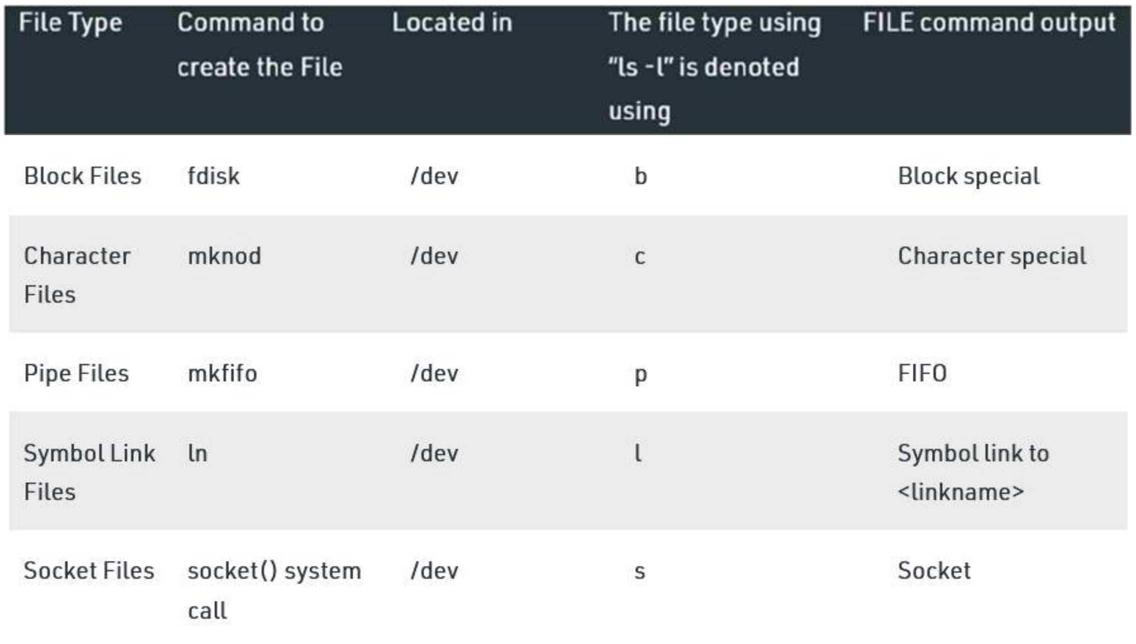
- Using **Vi** editor :
  To create a file with Vi editor, execute the below command :
  vi test9.txt

  The above command will open the Vi editor. Press i key for the insert mode and enter the file content. Press Esc key and :wq to save and exit the file from the editor.

The following table shows the types of files in Linux and what will be output using ls and file command



File Type	Command to create the File	Located in	The file type using "ls -l" is denoted using	FILE command output
Regular FIle	touch	Any directory/Folder	-	PNG Image data, ASCII Text, RAR archive data, etc
Directory File	mkdir	It is a directory	d	Directory

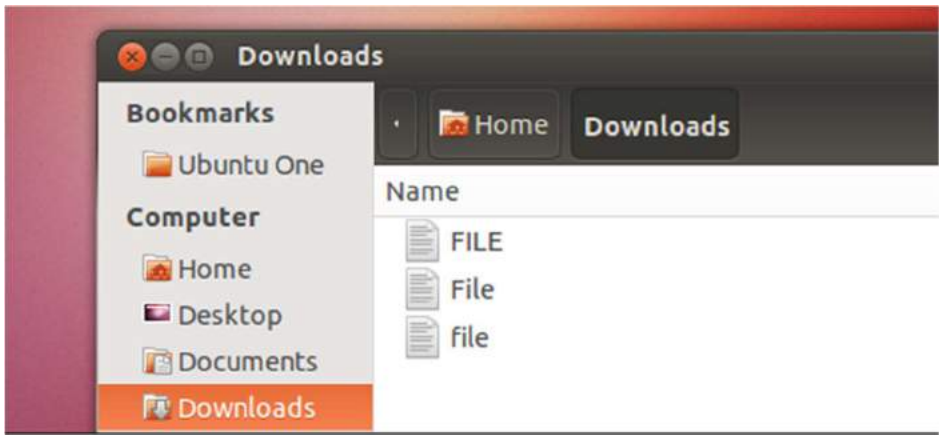


File Type	Command to create the File	Located in	The file type using "ls -l" is denoted using	FILE command output
Block Files	fdisk	/dev	b	Block special
Character Files	mknod	/dev	c	Character special
Pipe Files	mkfifo	/dev	p	FIFO
Symbol Link Files	ln	/dev	l	Symbol link to <linkname>
Socket Files	socket() system call	/dev	s	Socket

Case Sensitivity

On Windows, you can't have a file named file and another file named FILE in the same folder. The Windows file system isn't case sensitive, so it treats these names as the same file.

But on Linux, the file system is case sensitive. This means that you could have files named file, File, and FILE in the same folder. Each file would have different contents – Linux treats capitalized letters and lower-case letters as different characters.



❖ Some useful commands

The following commands are used to work with files and directories.

- **pwd :**
  This command displays the present working directory where you are currently in.
  Ex.
  ```
  $ pwd
  /home/sohan
  ```

- **ls :**
  This command will list the content of a directory.
  Ex.
  ```
  $ ls
  Arnab.txt  Sohan.txt  Venkatesh.txt
  ```

- **ls -la :**
  This command will list all the content of a directory including the hidden files and directories.

- **mkdir :**
  This command will create a new directory, provided it doesn't exists.

  Ex : `$ mkdir Project`

- **mkdir -p :**
  This command will create nested directories.

  Ex. `$ mkdir -p example/hello/world`

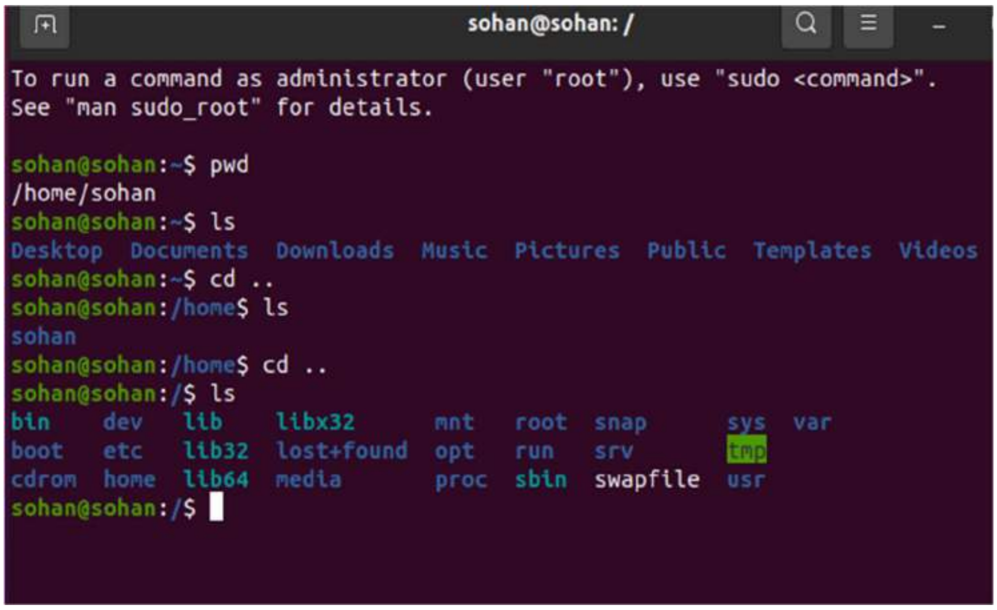
- **rmdir :**
  This command will remove/delete an existing directory, provided it is empty.

  Ex. `$ rmdir example`

- **cd :**
  This command is used to change directory.
  Ex. `$ cd /`

- **cd .. :**
  This command will take us one level up the directory tree.
  Ex. `$ cd ..`

- **rm filename :**
  This command will delete a file.
  Ex. $ rm hello.txt



```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

sohan@sohan:~$ pwd
/home/sohan
sohan@sohan:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
sohan@sohan:~$ cd ..
sohan@sohan:/home$ ls
sohan
sohan@sohan:/home$ cd ..
sohan@sohan:/$ ls
bin    dev   lib    libx32      mnt   root  snap      sys  var
boot   etc   lib32  lost+found  opt   run   srv       tmp
cdrom  home  lib64  media       proc  sbin  swapfile  usr
sohan@sohan:/$
```

- **cp file1 file2 :**
  This command copies the content of file file1 into file file2.
  Ex. $ cp file1.txt file2.txt

- **cp -r dir1 dir2 :**
  This command copies the content of directory dir1 into directory dir2.

  If directory dir2 doesn't exists then it is created. If it exists then its content is

  overwritten.

  Ex. $ cp -r Project1 project2

- **mv** - rename files and directories
  We can use mv command to rename files and directories.

  Ex. $ mv hello.txt hi.txt

- **cat filename :**
  This will print the content of a file.

  Ex. $ cat hello.txt
  The quick brown fox jumps over the lazy dog.

❖ Conclusions :

Until a few years ago, Linux was used mainly for servers and was not considered suitable for desktops. But its user interface and ease of use has been steadily improving over the last few years. Linux has today become user-friendly enough to replace Windows on desktops. It is being used by hundreds of thousands of people across the globe. Here are some most important reasons to use Linux :

- High security : Installing and using Linux on your system is the easiest way to avoid viruses and malware.
- high stability : The Linux system is very stable and is not prone to crashes.
- Runs on any hardware : Linux makes very efficient use of the system's resources. Linux installation can be customised for users and for specific hardware requirements. Even on old hardware Linux installation is possible.
- Free : Linux is completely free and users do not need to pay for anything. All the basic software required by a typical user and even an advanced user are available.
- Open Source : The most important aspect of Linux is that its source code is available as it falls under the FOSS category (Free and Open Source Software).
- Customization : Users have tremendous flexibility in customising the system as per their requirements. There are numerous choices for wallpapers, desktop icons and panels. There are more than half-a-dozen desktop environments to choose from, like GNOME, KDE, etc. For any task, right from the GUI interface and file managers, to DVD burners and browsers, around four to six options are available for any particular software.
- Education : This is the most useful aspect for students, as they can use the software to study how it works, before modifying and extending the code to suit their needs.
- Support : There is strong community support for Linux over the Internet through various forums.

BIBLIOGRAPHY

- Operating System Concepts, 10th edition –Abhraam Silberchatz , Greg Gagna , Peter Galvin
- https://www.javatpoint.com/linux-file-system
- https://en.wikipedia.org/wiki/Linux
- https://www.youtube.com/watch?v=Ip9P1XJt5PI
- https://www.youtube.com/watch?v=scfDOof9pww&t=66s
- https://www.youtube.com/watch?v=yVpbFMhOAwE
- https://www.youtube.com/watch?v=5ocq6_3-nEw
- https://www.youtube.com/watch?v=akU1Ji8Vzdk&list=PLZ5dJPlUQexlMzytxuLk2uVHttBKV-1HH
- https://www.tutorialspoint.com/unix/unix-file-system.htm
- https://www.linux.com/training-tutorials/linux-filesystem-explained/