UNIT - 5

Process Management in Linux:

Learning Objective:
Process Management in Linux
process Identity, process environment and process context
Usage of fork and clone in linux.

process:

- process is a program in execution
- process comprises of text, data, stack, registers - PC, SP, kernel data structures - process table etc.
- Linux uses a process model similar to other versions of UNIX.
- Review of traditional UNIX process model and then the linux threading model.
- UNIX process management separates the creation of processes and the running of a new program into two distinct operation
  - The fork system call creates a new process.
    - A new program need not be run.
    - New process executes the same program as the first.
    - Continues to execute at the point the parent left.
  - A new program is run after a call to execve
    - does not require a process to be created just before the program is run.
    - Any process can run a program at any time.
    - A new binary object is loaded into the process's address space.
    - The new executable starts executing in the content of the existing process.
- Under Unix, a process encompasses all the information that the operating system must maintain to track the content of a single execution of a single program.
- Under Linux, process properties fall into 3 groups:
  1. The process's identity
  2. environment
  3. Content

- process ID (PID):
  - Unique identifier for the process
  - Used to specify processes to the operating system when an appn makes a system call to signal, modify, or wait for another process.

- Credentials:
  - Each process must have an associated user ID and one or more group IDs that determine the process rights to access system resources and files.

Process Identity

- personality:
  - Not traditionally found on UNIX systems.
  - sets the process execution domain.
  - Under Linux, each process has an associated personality identifier that can slightly modify the semantics of certain system calls.

- Namespace:
  - Each process has a specific view of the file-system, called namespace.
  - Most processes share a common namespace and thus operate on a shared file-system hierarchy (root directory, set of mounted file systems)
  - process and their children can have different namespaces.

Process Environment

- The process environment is inherited from its parent, and is composed of two null-terminated vectors:
  - The argument vector lists the command-line arguments used to invoke the running program; conventionally starts with the name of the program itself.
  - The environment vector is a list of "NAME=VALUE" pairs that associates named environment variables with arbitrary textual values.
  - TERM - to name the type of terminal connected to a user's login session.
- Arguments and environment vectors are not altered when a new process is created.
  - child inherits the environment of the parent:

- When a new program is invoked, a new environment can be set up.
- On calling exec(), a process can supply the environment for the new program.
- The kernel passes these environment variables to the next program, replacing the process's current environment.
- The environment-variable mechanism custom-tailors operating system on a per-process basis, rather than being configured for the system as a whole

Process Context:

- The (constantly changing) state of a running program at any point in time
- process context includes
  - Scheduling contexts
  - Accounting
  - File Table
  - File-system Context
  - Signal-handler table
  - Virtual memory context

Scheduling context:

- Information that the scheduler needs to suspend and restart the process.
- Saved copies of all process's registers.
- Information about scheduling priority.
- Information about any outstanding signals waiting to be delivered to the process.
- Kernel stack - used by the process while executing kernel code.

Accounting Information about:

- Resources currently consumed by each process.
- The total resources consumed by the process in its lifetime so far.

File Table:

- Array of pointers to kernel file structures.
- When making file I/O system calls, processes refer to files using a file descriptor (Kernel uses the file descriptor as an index into this table.

File-system context:

- Whereas the file table lists the existing open files, the file-system context applies to request to open new files.
- process's root directory, current working directory and namespace (default directories to be used for new file searches) are stored here.

Signal-handler table:

- defines the action to take in response to a specific signal
- Valid actions are ignoring the signal, terminating the process and invoking a routine in the process's address space

processes and Threads:

- Linux does not distinguish between a thread and a process
- Linux uses the term task to refer to a flow of control within a program.
  - fork duplicates a process without loading a new executable image.
  - clone behave similar to fork except that it accepts as argument a set of flags that dictate what resources are shared b/w the parent and the child.
  - Flags include
    - CLONE_FS, CLONE_FILES, CLONE_SIGHAND, CLONE_VM.
      - CLONE_FS → File system information (Current working directory is shared)
      - CLONE_FILES → The set of open files is shared
      - CLONE_SIGHAND → Signal handlers are shared
      - CLONE_VM → same memory space is shared
- Lack of distinction b/w process and a thread is because:
  - Linux does not hold a process entire content within the main process data structure.
  - It holds the content with independent subcontents.
  - A process file-system content, file descriptor table, signal-handler table and virtual-memory content are all held in separate data structures.
  - process data structure (struct task_struct) has a pointer to these structures
  - Any number of processes can easily share a subcontent by pointing to the subcontent and incrementing a reference count.

- Arguments to the clone() call tell it which subcontexts to copy and which to share
- Any new process is always given a new identity and a new scheduling context.
- According to the arguments passed, the kernel may share the subcontext data structures or make a copy of the subcontext data structures.
- The fork() system call is a special case of the clone() which copies all subcontexts, shares none.

Scheduling

- The job of allocating CPU time to different tasks within an operating system.
- Linux supports preemptive multitasking
- Process scheduler decides which process runs and when.
- Scheduling is not just running and interrupting of user processes, in Linux, scheduling also includes the running of the various kernel tasks.
- Kernel tasks - tasks that are requested by a running process and tasks that execute internally on behalf of the kernel (tasks spawned by Linux's I/O system).

Process Scheduling

- Linux uses 2 process-scheduling algorithms:
  - A time-sharing algorithm for fair preemptive scheduling b/w multiple process.
  - A real-time algorithm for tasks where absolute priorities are more important than fairness

Time-sharing Fair Algorithm (CFS)

- The Completely Fair Scheduler (CFS) was introduced in Linux kernel version 2.6.
- Preemptive, priority-based algorithm with 2 separate priority ranges

- Real-time range from 0-99
- nice value ranging from -20 to 19
- Smaller nice values indicate higher priorities.

- CFS calculates how long a process should run as a function of the total number of runnable processes
- If there are N runnable processes, each process should be allotted 1/N of the processors time.
- This allotment is adjusted by weighting each process allotment by its nice value.
- process with the default nice value have a weight of 1 - priority is unchanged.
- processes with a small nice value (higher priority) receive a higher weight.
- processes with a larger nice value (lower priority) receive a lower weight.
- CFS runs each process for a time slice proportional to the process's weight divided by the total weight of all runnable processes
- To calculate the actual length of time a process runs
  - A configuration variable called target latency is used.
  - Target latency is the interval of time during which every runnable task should run at least once.
  - Assume that the target latency is 10 milliseconds.
  - Assume that we have two runnable processes of the same priority.
  - Each of the processes has the same weight and therefore receive the same proportion of the processor's time
  - If the target latency is 10 milliseconds, the first process runs for 5 milliseconds and the second process runs for the next 5 milliseconds and so on.
  - If there are 10 runnable processes, each process will run for 1 millisecond before repeating.
  - If there are 1000 processes of each process can run only for 1 microsecond

- Switching costs are involved - scheduling processes for short lengths of time is inefficient.
- Therefore, another configuration variable called the minimum granularity is used.
- Minimum granularity is the minimum length of time any process is alloted the processors time.
- Regardless of the latency the process will run for atleast the minimum granularity.
- Thus CFS ensures that the switching costs donot grow unacceptably large when the number of runnable processes becomes too large.
- But fairness is violated.

Real-time scheduling

- Linux implements two real-time scheduling classes
  - FCFS, round robin.
- In both cases, each process has a priority in addition to its scheduling class.
- Always the process with the highest priority is run.
- Among process with the same priority, the one that was waiting for the longest is scheduled.
- FIFS processes continue to run until they exit or block
- Round robin processes will be preempted and are moved to the end of the scheduling queue.
- Round-Robin processes of equal priority will automatically time-share among themselves.
- Soft-real-time scheduling.

Kernel Synchronization

- Kernel schedule differs from user processes scheduling
- Kernel-mode execution can occur in 2 ways:
  * A running program may request an operating system service, either explicitly via a system call, or implicitly, for example when a page fault occurs.

(2) # Memory Management in Linux.

Learning Objectives:
- Management of physical Memory
  - Buddy Algorithm
  - slab Allocator
- Management of Virtual Memory:

Memory Management

- Linux memory Management
  - + allocating and freeing physical memory - pages, groups of pages, and small blocks of main memory.
  - + handling virtual memory, memory mapped into the address space of running processes.
  - + Linux is available for a wide range of architectures
  - + Needs to be an architecture independent way of describing memory
  - + Linux splits physical memory into 4 different zones due to hardware characteristics
    - ZONE_DMA,
    - ZONE_DMA32,
    - ZONE_NORMAL,
    - ZONE_HIGHMEM
  - After booting, 'dmesg' shows us the physical addresses of the zones.
  - ZONE_DMA in memory in the lower physical memory ranges which certain ISA devices require.
  - In x86-32 architecture, some devices access only lower 16MB of physical memory (ZONE_DMA) using DMA
  - Memory in the DMA zone can be used for transfers, for example with network cards, which can only address 24 bits, to 16 MB
  - Some cards/drivers can only utilize memory from the DMA zone
  - DMA32 is 4GB in size, used for data exchange with cards which can address 32 bits.

- Some devices supporting 64-bit addresses can access only first 4 GB (ZONE_DMA32)
- Physical memory not mapped into the kernel address space is called ZONE_HIGHMEM
  - In 32-bit Intel architecture (2^{32} = 4 GB address space), kernel is mapped to first 896 MB of the address space, remaining memory is high memory allocated from ZONE_HIGHMEM
- ZONE_NORMAL has everything else - the normal, regularly mapped pages - used for processes.
- Memory within ZONE_NORMAL is directly mapped by the kernel into the upper region of the linear address space.

Management of physical Memory

- Many kernel operations can only take place using ZONE_NORMAL. so it is most performance critical zone.
- All architectures may not have all zones.
- Modern Intel x86-64 bit architecture has a small 16MB ZONE_DMA and the rest of its memory is ZONE_NORMAL, no high memory

Relationship of Zones and Physical Addresses on 80x86

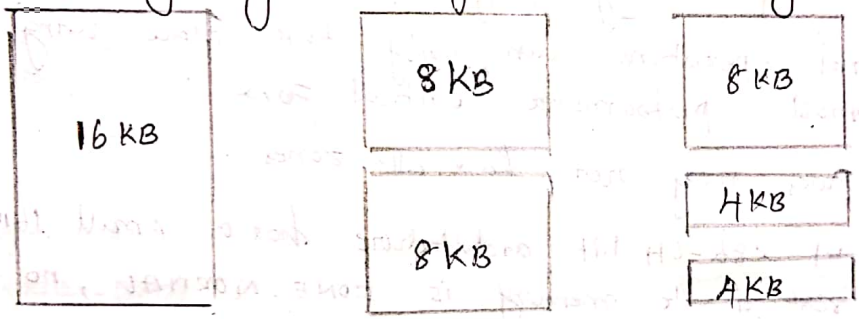
Zone	Physical Memory
1. ZONE_DMA	< 16 MB
2. ZONE_NORMAL	16 - 896 MB
3. ZONE_HIGHMEM	> 896 MB

- For each zone, the kernel maintains a list of free pages.
- When a request for physical memory arrives, the kernel satisfies the request using the appropriate zone.
- The physical memory manager in Linux is called the page allocator
- Each zone has its own allocator
  - Allocates and frees all physical pages for the zone
  - Capable of allocating ranges of physically contiguous pages on request
  - Uses a buddy system to keep track of available physical pages.

Buddy System

- Adjacent units of allocatable memory are paired together
- Each allocatable memory region has an adjacent partner (or buddy)
- Whenever two allocated partner regions are freed up, they are combined to form a larger region (buddy heap)
- The larger region also has a partner with which it can combine to form a still larger free region.
- If there is a request for a small region and if a region of that size is not available, a larger region is split into two partners.
- Smallest allocatable size is a single page.

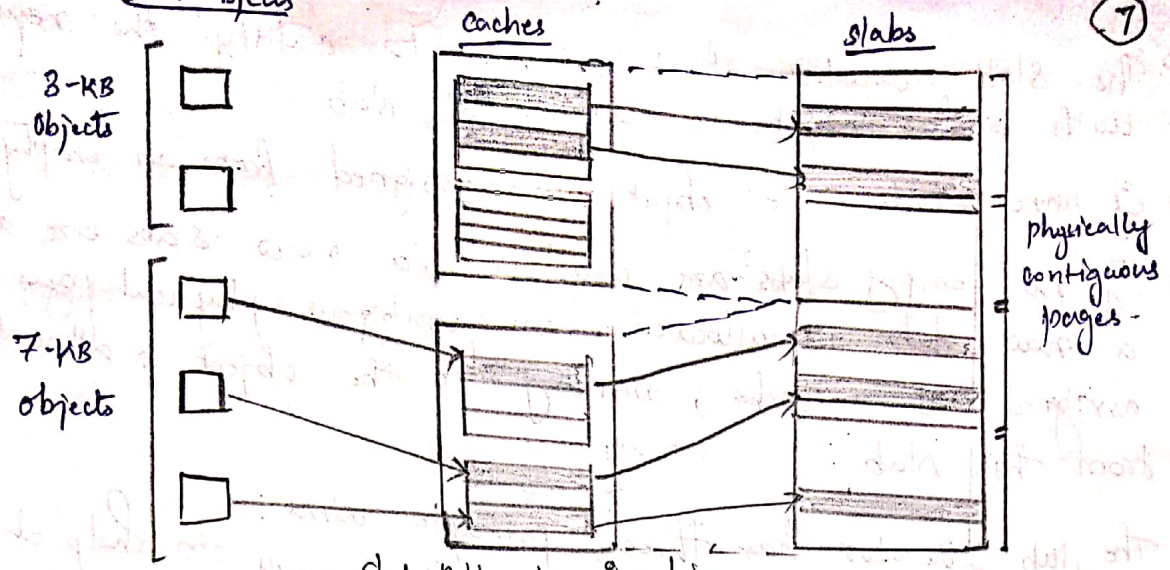
Splitting of Memory in a Buddy heap.



Management of Physical Memory - Slab Allocator

- Allocates memory for kernel data structures
- A slab is made up of one or more physically contiguous pages
- A cache consists of one / more slabs.
- The slab allocator consists of a variable number of caches that are linked together or a doubly linked circular list called a cache chain.
- For each unique kernel data structure, there is a single cache
  - cache for inodes, cache for file objects
- Each cache is populated with objects that are instantiations of the kernel data structure the cache represents
  - Inode cache stores instance of inode data structures.

Kernel objects



Slab Allocator in Linux.

- The slab-allocation algorithm uses caches to store kernel objects
- When a cache is created, a number of objects are allocated to the cache.
- The number of objects in the cache depends on the size of the associated slab.
  - A 12KB slab (3 contigous 4-KB pages) can store six 2 KB objects.
- All objects in the cache are initially marked free.
- When a new object is needed for a kernel data structure, the allocator can assign any free
- All objects in the cache are initially marked free
- When a new object is needed for a kernel data structure, the allocator can assign any free object from the cache to satisfy the request.
  - New task is created - new process descriptor object (struct task_struct) to be assigned.
  - Cache will fulfill the request using a free slab.
- A slab may be in one of 3 possible states:
  - Full: All objects in the slab are marked as used.
  - Empty: All objects in the slab are marked as free.
  - partial: The slab consists of both used and free objects.

- The slab allocator first attempts to satisfy the request with a free object in a partial slab.
- If none exists, the object is assigned from an empty slab
- If no empty slabs are available, a new slabs are available a new slab is allocated from contiguous physical pages and assigned to a cache; memory for the object is allocated from this slab.

- The slab allocator has three principle aims:
  - The allocation of small blocks of memory to help eliminate internal fragmentation that would be otherwise caused by the buddy system.
  - The caching on commonly used objects so that the system does not waste time allocating, initialising and destroying objects. Benchmarks on Solaris showed excellent speed improvement for allocations with the slab allocator.
  - The better utilization of hardware cache by aligning objects to the L1 or L2 caches.

Virtual Memory

- The VM system maintains the address space accessible to each process.
  - It creats pages of virtual memory on demand, and manages the loading of those pages from disk or their swapping back out of disk as required.
- The VM manager maintains two separate views of a process address space.
  - A logical view describing instructions concerning the layout of the address space.
    - The address space consists of a set of nonoverlapping regions, each representing a continuous, page-aligned subset of the address space.
  - A physical view of each address space which is stored in the hardware page table for the process.
    - page table entries have the current location of each page of virtual memory - whether it is in disk or in physical memory.

- Several types of virtual memory regions.
  - The backing store describes from where the pages for a region come from.
  - regions are usually backed by a file or by nothing (demand-zero memory)
  - A virtual memory region is also defined by the regions reaction to writes (shared-page sharing or private copy-on-write)
- The kernel creates a new virtual address space.
  - When a process runs a new program with the exec system call
  - upon creation of a new process by the fork system call.
- On executing a new program, the process is given a new, completely empty virtual-address space.
- The program-loading routines populate the address space with virtual-memory regions.
- Creating a new process with fork involves creating a complete copy of the existing process's virtual address space.
  - The kernel copies of the parent process VMA descriptors, then creates a new set of page tables for the child.
  - The parents page tables are copied directly into the child's with the reference count of each page covered being incremented.
  - After the fork, the parent and child share the same physical pages of memory in their address spaces.
  - If the virtual memory region is private, the pages for the regions are marked as read-only and copy-on-write is set.
- The VM paging system relocates pages of memory from physical memory out to disk when the memory is needed for something else
- no whole-process swapping.

- The VM paging system can be divided into 2 sections:
  - The pageout-policy algorithm decides which pages to write out to disk, and when (LFU)
  - The paging mechanism actually carries out the transfer and pages data back into physical memory as needed.

Summary

- Management of physical memory
  - Zones
  - page allocator
  - Slab Allocator
- Management of Virtual Memory.
  - Virtual Memory regions
  - Virtual address space - fork(), exec()
  - Swapping and paging

(3)

File Systems: in Linux

Linux retains UNIX's standard file-system model. In UNIX, files can be anything capable of handling the input/output of a stream of data, device drivers can appear as files, and interprocess communication channels or network connections also look like files to the user.

The Linux kernel handles all these types of files by hiding the implementation details of any single file behind a layer of software, the virtual file system. (VFS)

The Virtual File System:

The Linux VFS is designed around object-oriented principles.

It has 2 components:

1. a set of definitions that specify what file-system objects are allowed to look like and
2. a layer of s/w to manipulate the objects.

The VFS defines 4 main object types:

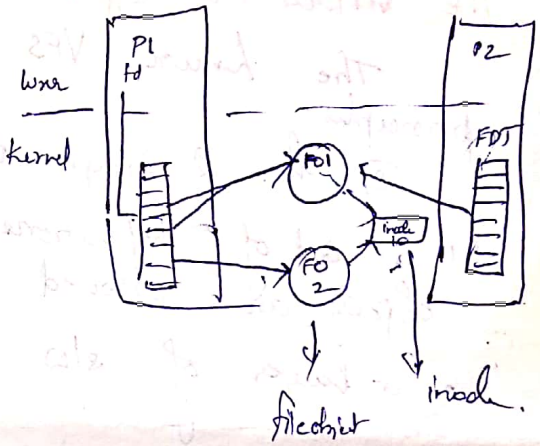
- An inode object represent an individual file.
- A file object represents an open file.
- A superblock object represents an entire file system.
- A dentry object represent an individual directory entry

The VFS defines a set of operations for each of the object types. Example:

1. int open() — open a file.
2. ssize_t read(...) - Read from a file
3. ssize_t write(...) - write to a file.
4. int mmap(..) - Memory - map a file.

The VFS software layer can perform an operation on one of the file-system objects by calling the appropriate function from the object's function table, without having to know in advance exactly what kind of object it is dealing with.

The inode & file objects are the mechanisms used to access file data. An inode object is a data structure containing pointers to the disk blocks that contain actual file contents and a file object represents a point of access to the data in an open file. A process cannot access an inode's contents without first obtaining a file object pointing to the inode.



Linux ext3 file system :

The standard on-disk file system used by Linux is called ext3fs.

Linux was originally programmed with a Minix-compatible file system, to ease exchanging data with the Minix development system, but that file system was severely restricted by 14-character file-name limits and a maximum file-system size of 64 MB.

The Minix file system was superseded by a new file system, which was christened the Extended File System (extfs).

A later redesign to improve performance & scalability and to add a few missing features led to the second extended file system (ext2).

Further development added journaling capabilities, and the system was renamed the third extended file system (ext3).

Linux kernel developers are working on augmenting ext3 with modern file-system features such as extents. This new file system is called the fourth extended file system (ext4).

Similarities and differences b/w ext3fs and BSD ffs (Fast File System)

Similarities : Mechanisms for searching the blocks of data, saving blocks of data pointer within Indirect block.

Difference : Disk allocation policies.

Fast file system allocation is performed in blocks which are of 8KB size. Fragmentation is performed on these blocks to make each block of 1KB size that are used for storing file of less size.

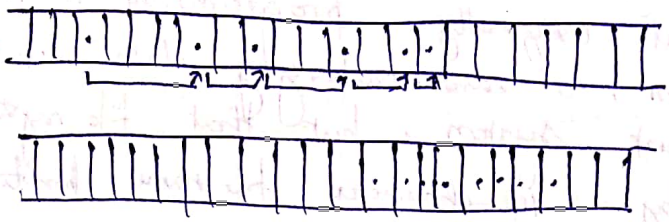
In ext3fs, there is no fragmentation required because allocation is performed in blocks which are of 1KB size.

Allocating a file in ext2fs is to select the block group. An ext2fs file system is partitioned into number of block groups.

Ext3 block allocation policies

~~Journaling~~:

The



[ ] blocks inuse [·] blocks selected for allocator | bit boundary

[ ] Free block → bitmap search | byte boundary

Journaling:

The ext3 file system supports a popular feature called Journaling whereby modification to the file system are written sequentially to a Journal.

A set of operations that perform a specific task is a transaction. Once a transaction is written to the Journal, it is considered to be committed.

Journal is represented as circular buffer, it is present in different file system sections. Even if the system crashes, all transactions that are committed but not completed can be executed until they are completely executed.

Linux process file system:

The Linux process file system known as /proc file system implements file system that do not store data permanently, but provides an interface is used for computing the functionality on demand depending on various I/O requests made by user.

There are 2 things that are implemented by /proc file system

1. Structure of directory
2. Content of file within the directory.

There is a unique inode number specified by /proc file system for identifying operation requests. These requests are used for reading the content of file, collecting the info, translating it into text file and placing it into the read buffer of requested process.

PID	Information
16bit	16 bit

structure of inode number.

Inter Process Communication in Linux System

1. Interprocess Communication:

refers to way of describing how communication takes place b/w one process to the other and letting other process know that an event has been triggered.

Synchronization & Signals:

- Unix initiate the inter-process communication with the help of signal that informs the process that event has triggered.
- Linux does not make use of signal for communicating b/w the processes that are present in kernel mode.
- It uses scheduling state and wait-queue structure that allows the kernel to communicate about the events that are not synchronous.
- Using this structure the device driver as well as running system can generate the events which are informed to other process present in the kernel mode.
- When a process waits for completion it is placed on the wait-queue. It means that this process is not ready for execution. Once the event is completed all the processes present in wait-queue are now ready for execution.

* Another method that is used by Linux for inter process communication is the use of semaphore that are used in System V Unix.

* There are 2 advantages of using semaphore instead of signal;

1. Many processes that are not dependent on each other can share large number of semaphores.
2. Operations can be performed atomically on multiple semaphores.

Wait-queue is responsible for synchronizing the process that are communicating with semaphores internally.

Passing of Data among Process:

- The Method that are used for passing the data among processes are

1. Linux makes use of pipe that have wait-queue pair responsible for synchronizing the read and write opn. [The pipe mechanism allows a child process to inherit a communication channel to its parent, data written to end of the pipe can be read the other].

2. Through the use of shared Memory that offer very fast communication, when one process write data to shared memory it can be read easily by all the other available process.

- The drawback of using shared memory method is that it does not provide synchronization. This drawback can be overcome if shared memory mechanism is merged with other IPC mechanism that provides synchronization.

Passing of Data among Process:

Shared memory is considered as a persistent object that can be created/deleted by process. These objects are present in small address spaces that are independent. These objects of shared acts as a backup store for shared memory region when a page fault occurs, the shared memory maps the page from persistent shared memory objects.

Input and output in Linux (1)

I/O system in Linux looks much similar to Unix. All device drivers appear as Normal files. A device user can open an access channel to a device in the same way as she can open any other file. Devices can appear as objects within the file system. The system administrator can create special file within a file system that contain references to a specific device driver, and a user opening such a file will be able to read from and write to the device referenced.

Linux splits all device into 3 classes.

1. Block Devices
2. Character Devices
3. N/w Devices.

Block Devices:

- include all devices that allow random access to completely independent, fixed-sized blocks of data, including hard disks, floppy disks and CD-ROMs.
- Block devices provide the main interface to all disk devices in a system.
- performance is particularly important for disks, and the block-device system must provide functionality to ensure that disk access is as fast as possible. This functionality is achieved through the scheduling of I/O operations.
- A block represents the unit with which the kernel performs I/O. When a block is read into memory, it is stored in a buffer.
- The request manager is the layer of software that manages the reading and writing of buffer contents to and from a block-device driver.

Block Devices: A separate list of requests is kept for each block-device driver. Traditionally, these requests have been scheduled according to a unidirectional-elevator (C-SCAN) algorithm, that exploits the order in which requests are inserted in and removed from the lists.

The request lists are maintained in sorted order of increasing starting-sector-number.

- When a request is accepted for processing by a block-device driver, it is not removed from the list. It is removed only after the I/O is complete, at which point the driver continue with the next request in the list (even if new requests have been inserted in the list before the active request). As new I/O requests are made, the request manager attempts to merge requests in the lists.

- Linux Kernel version 2.6 introduced a new I/O scheduling algorithm - default one.

- CFQ maintains a set of lists - by default one for each process. Request originating from a process go in that process list. For example, if 8 processes are issuing I/O request, CFQ will maintain 8 separate lists of requests, one for each process. The list are maintained according to the C-SCAN algorithm.

Character Devices:

- A character-device driver can be almost any device driver that does not offer random access to fixed blocks of data.
- It includes devices such as mouse and keyboard.
- These devices can access only serially / sequentially.
- They read the data character by character. The kernel performs almost no pre-processing of a file read / write request to the device in question and lets the device deal with request

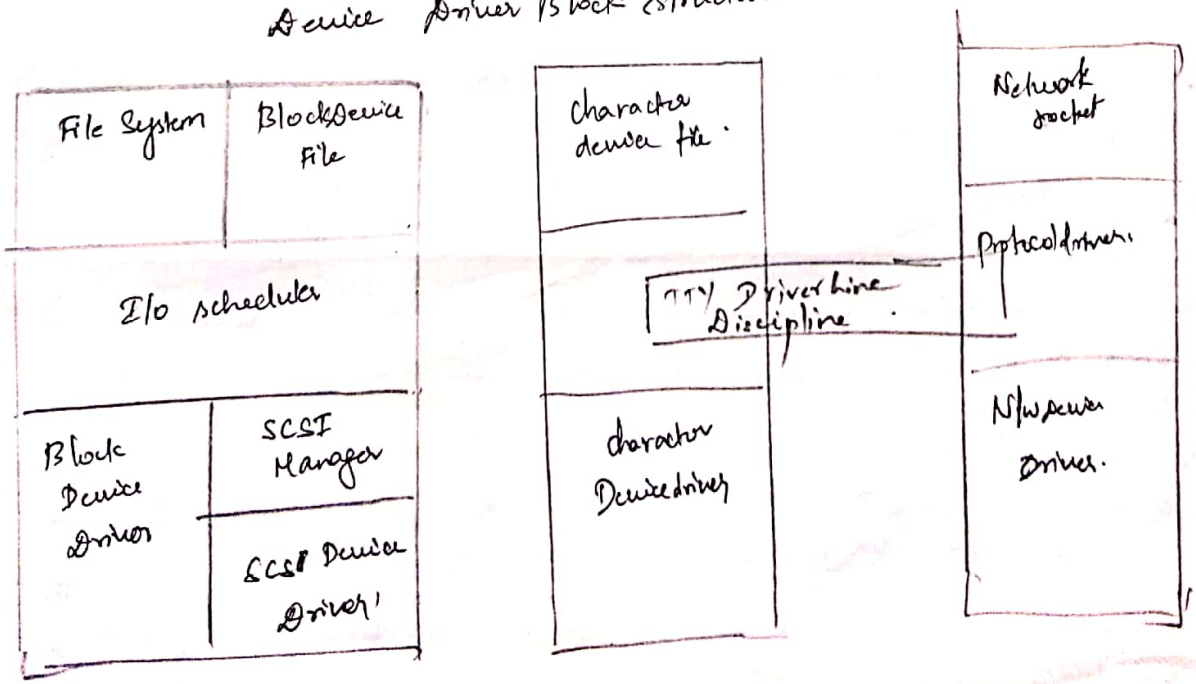
printers are character devices and after the kernel sends data to the printer, the responsibility for that data passes to the printer, the kernel cannot backup and re-examine the data.

- The exception to this rule is the special subset of character device drivers that implement terminal devices.
- The kernel maintains a standard interface to these drivers by means of a set of tty-struct structures. Each of these structures provides buffering and flow control on the data stream from the terminal and feeds that data to a line discipline.
- A line discipline is an interpreter for the information from the terminal device. The most common line discipline is the tty discipline. A tty discipline decides which process's data should be attached or detached from the terminal device.

Network devices:

Users cannot directly transfer data to N/w devices, instead, they must communicate indirectly by opening a connection to the Kernel N/w subsystem.

Device Driver Block structure.



Memory Management in Linux

Memory management under Linux has 2 components.

① The memory management deals with allocating and freeing physical memory - pages, groups of pages, and small blocks of RAM. (Physical memory)

② The second handles virtual memory, which is memory-mapped into the address space of running processes.

Management of Physical Memory :

Linux separates physical memory into 4 different zones/regions:

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

These zones are architecture specific.

Example: Intel x86-32 architecture :-

(Industry standard architecture)

ISA devices can only access the lower 16MB of physical memory using DMA. The first 16MB of physical memory comprise ZONE_DMA.

Certain devices can only access the first 4GB of physical memory, despite supporting 64-bit addresses. The first 4GB of physical memory comprise ZONE_DMA32.

ZONE_HIGHMEM refers to physical memory that is not mapped into the kernel address space. In 32 bit architecture, 232 [4-GB address space], the kernel is mapped into the first 896 MB of the address space. The remaining memory is referred to as high memory and is allocated from ZONE_HIGHMEM.

ZONE_NORMAL comprises the normal, regularly mapped pages.

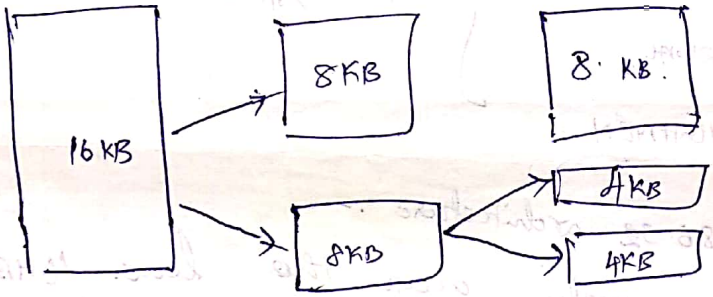
The kernel maintains a list of free pages for each zone. When a request for physical memory arrives, the kernel satisfies the request using the appropriate zone.

Relationship of Zones and Physical Addresses in Intel x86-32

Zone	physical memory
ZONE_DMA	16MB
ZONE_NORMAL	16...896MB
ZONE_HIGHMEM	> 896 MB

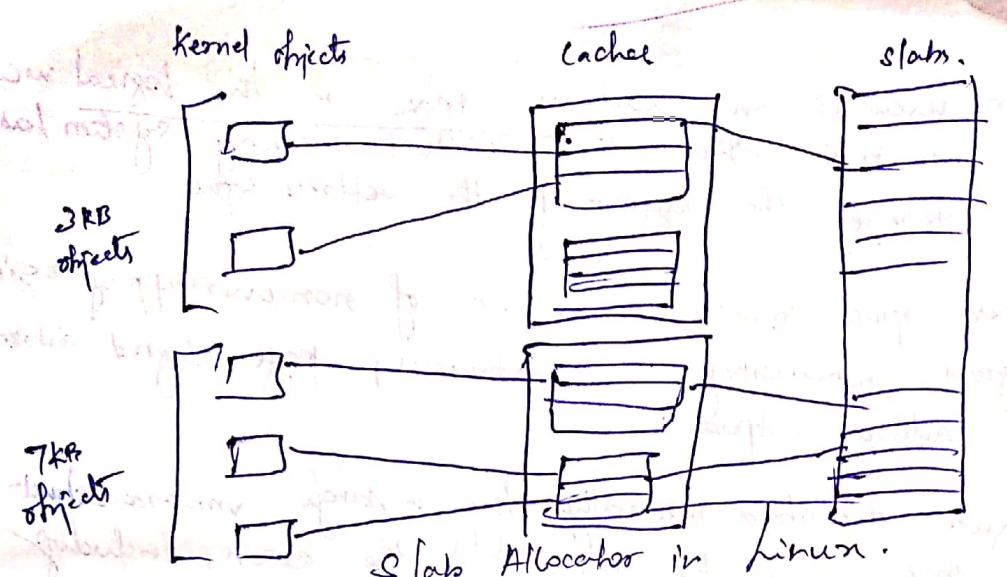
The primary physical-memory manager in the Linux kernel is the page allocator.

Strategy: Each zone has its own allocator, which is responsible for allocating and freeing all physical pages for the zone and is capable of allocating ranges of physically contiguous pages on request. The allocator uses a buddy system to keep track of available physical pages.



Splitting of memory in the buddy system

Another strategy adopted by Linux for allocating kernel memory is known as slab allocation. A slab is used for allocating memory for kernel data structures and is made up of one or more physically contiguous pages. A page cache consists of one or more slabs. There is a single cache for each unique kernel data structure - for eg a cache for the data structure representing process descriptors, a cache for file objects, a cache for inodes and so forth.



Slab Allocator in Linux.

In Linux, a slab may be in one of 3 possible states:

1. Full: All objects in the slab are marked as Used.
2. Empty: All objects in the slab are marked as free.
3. Partial: The slab consists of both used and free objects.

The slab allocator first attempts to satisfy the request with a free object in a partial slab.

If none exist, a free object is assigned from an empty slab. If no empty slabs are available, a new slab is allocated from contiguous physical pages and assigned to a cache; memory for the object is allocated from this slab.

Virtual Memory

- The Linux virtual memory system is responsible for maintaining the address space accessible to each process.
- It creates pages of virtual memory on demand and manages loading those pages from disk and swapping them back out to disk as required.

The virtual memory manager maintains two separate views of a process's address space: as a set of separate regions and as a set of pages.

* The first view of an address space is the logical view, describing instructions that the virtual memory system has received concerning the layout of the address space.

The address space consists of a set of non-overlapping regions, each regions representing a continuous, page-aligned subset of the address space.

Each region described internally by a single vm_area_struct structure that defines the properties of the region, including the process's read, write & execute permissions in the region as well as information about any file associated with the region.

* The kernel also maintains a second, physical view of each address space. The page-table entries identify the exact current location of each page of virtual memory, whether it is on disk or in physical memory. The physical view is managed by a set of routines vm_area_struct.

All requests to read/write an unavailable page are eventually dispatched to the appropriate handler in the function table for the vm_area_struct.

(5)

(16)

★Mobile OS

- Introduction

→ A mobile operating system is software that allows smartphones, tablets PCs and other devices to run applications and programs.

→ The OS is responsible for determining the functions and features available on your device, such as thumbwheel, keyboard, WAP, synchronization with application, email, text messaging and more.

→ Some of the more commonly and well-known Mobile OS includes the following

1) Android
2) iOS (Apple/iPhone)

1) ANDROID OS

- Introduction

→ Android is an OS for mobile devices.

→ It's an integrated open source software stack for mobile devices:

i) OS
ii) Middleware
iii) key mobile applications

→ Android SDK (software development kit) provides the tool and API's necessary to being development applications on the Android platform using the Java programming language.

- Features of Android OS

→ some of the current features and specifications of Android are:

↓

a. Application framework :-

→ It enabling reuse and replacement of components

b. Dalvik virtual machine :-

→ Its optimized for mobile device.

c. Integrated browser :-

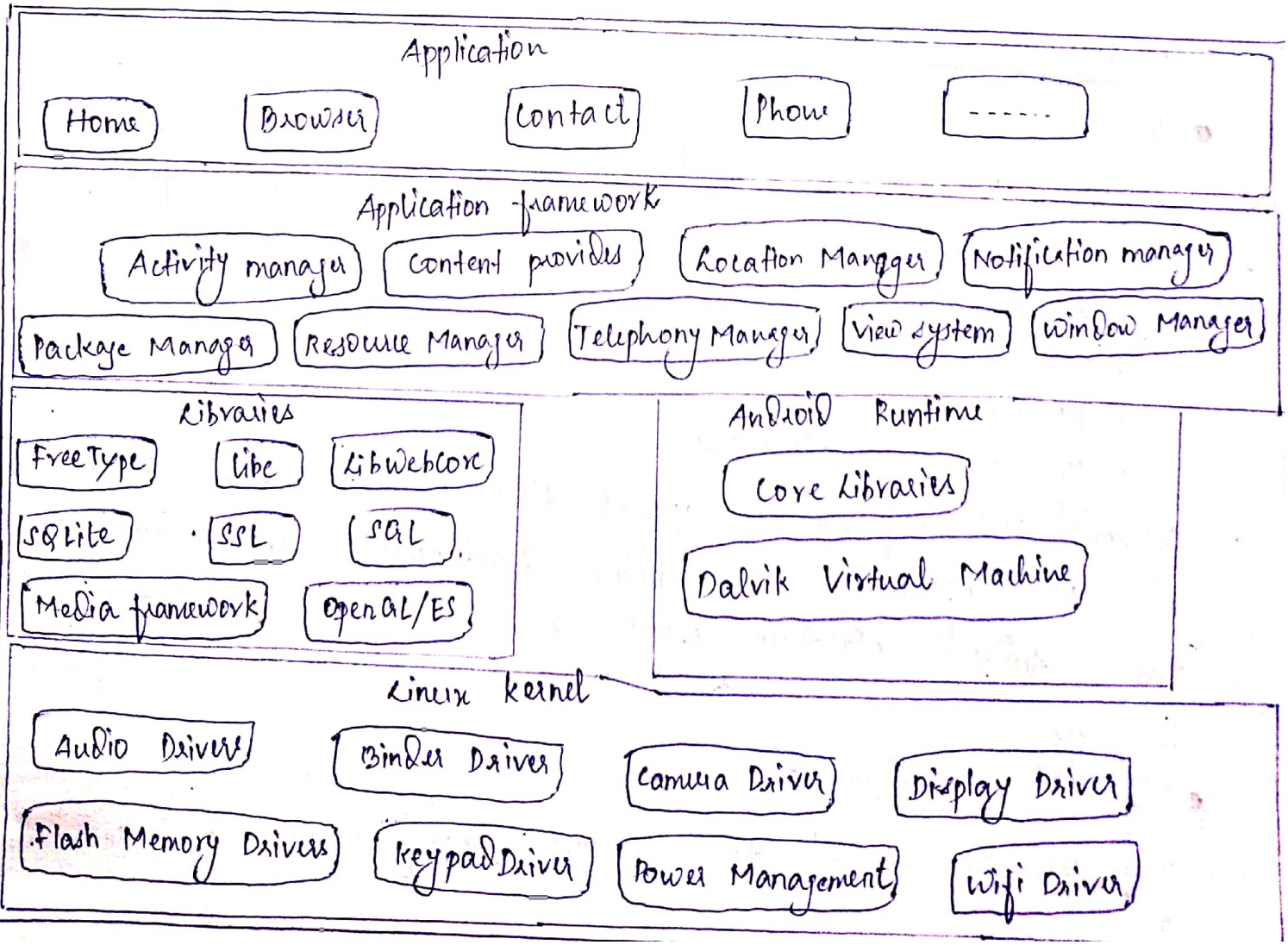
→ It is based on the open source Webkit engine

d. SQLite :-

→ Used for structured data storage.

- Android Architecture

→ Android software stack consists of Application at the top, middleware (consisting of an application framework, libraries, and Android runtime) in the middle and a Linux kernel with various drivers at the bottom.



→ The Diagram which is given previous that shows the layered architecture.

→ It consist some components

a) Activity Manager :-
→ It provides an app's lifecycle and maintains a shared activity stack for navigating within and among apps.

b) Resource Manager :- This component lets an app access its resources.

c) View System :-
→ It manges user interface elements and user interface-oriented event generation.

d) Package Manager :-
→ It lets an app learn about other app. package that are currently installed on the device

e) Location Manager :-
→ It makes possible for an Android device to be aware of its physical location.

2) iOS (Apple/iPhone)

→ The term 'iOS' was originally known as 'iPhone OS' and was introduced in 2007 along with the first iPhone hardware device Apple released.

→ The user interface of iOS is based on the concept of direct manipulation, using multi-touch gestures. Interface control elements consist of sliders, switches, and buttons.

→ The 'iOS' platform is a mobile-device-based software system that works like a computer system, but on mobile devices like portable phone.

→ It is designed to be smaller, faster and use less power. It also has a "touch" friendly user interface.

- Differences b/w Android and iOS

	ANDROID	iOS
1) First Version	Android 1.0, Alpha	iPhone os 1, before named iOS
2) Customizability	A lot can change almost anything	Limited unless jailbroken
3) Source model	Open source	closed, with open source components.
4) File transfer	Easier than iOS.	More difficult.
5) Widgets	Yes, except on lockscreen	No, except in Notification center.
6) Available language (s)	100+ Languages	34 languages.
7) Virtual assistant	Google Assistant	Siri

Kernel Modules:

(7) The Linux Kernel has the ability to load and unload arbitrary sections of Kernel code on demand.

- These loadable kernel modules run in privileged kernel mode and as a consequence have full access to all the hardware capabilities of the machine on which they run.

- Linux Kernel modules are convinient for several reasons: (2 marks) ?

1. Linux's source code is free, so anybody wanting to write kernel code is able to compile a modified kernel and to reboot to load that new functionality.

   Example: Once a new driver is written, it can be distributed as a module so that other users can benefit from it without having to rebuild their kernels.

2. Linux Kernel is covered by the GPL license, the Linux Kernel cannot be released with proprietary components added to it, unless those new components are also released under the GPL and the source code for them is made available on demand.

   The kernel's module interface allows third parties to write and distribute, on their own terms, device drivers or file systems that could not be distributed under the GPL.

Kernel modules allow a Linux system to be set up with a standard, minimal kernel, without any extra device drivers built in. Any device drivers that the user needs can be either loaded explicitly by the system at startup, or loaded automatically by the system on demand and unloaded when not in use.

The module support under Linux has three components.

1. The module mgmt allows modules to be loaded into memory and talk to the rest of the kernel.
2. The driver registration allows modules to tell the rest of the kernel that a new driver has become available.
3. A conflict-resolution mechanism allows different device drivers to reserve hardware resources and to protect those resources from accidental use by another driver.

Module Management:

(i) It supports loading modules into memory and letting them talk to the rest of the kernel.

(ii) Linux maintains an internal symbol table in the kernel.

(iii) Module loading is split into 2 separate sections:
   a) Management sections of module code in kernel memory.
   b) Handling symbols that modules are allowed to reference

(iv) Symbol table does not contain the full set of symbol defined in the kernel during latter compilation.

(v) The module requestor manages loading requested, but currently unloaded modules.

(vi) It also regularly queries the kernel to see whether a dynamically loaded module is still in use and will unload it when it is no longer actively needed.

(vii) Original service request will complete once the module is loaded.

Driver Registration:

(i) Allows modules to tell the rest of the kernel that a new driver has become available.

(ii) Kernel maintains dynamic tables of all known drivers.

(iii) Kernel also provides a set of routines to allow drivers to be added or removed.

(iv) Registration table contains following:
a) Device Drivers.
b) File Systems.
c) Network Protocols.
d) Binary format.

(v) Device drivers may be block / character devices

(vi) Files system contains network file system, virtual file system etc

(vii) Network protocols include IPX / packet filtering rules for a N/w.

Conflict Resolution:

(i) A mechanism that allows different device drivers to reserve hardware resources and to protect those resources from accidental use by another drivers.

(ii) Linux provides a central Conflict resolution mechanism.

(iii) Conflict Resolution module aims are
(a) To prevent modules from clashing over access to hardware resources.
(b) Prevent auto probes from interfering with existing device drivers.
(c) Resolve conflicts among multiple drivers trying to access

6 What is a Linux system? List its design principles. Also explain its components & in detail.. architecture

- Linux is an open source operating system which is powerful and easy to implement. It can be easily installed onto the system.
- Linux observes POSIX (Portable Operating System Interface) specifications and is provided with extensions similar to UNIX system V and BSD.
- Linux is multi-user, multi-tasking, time-sharing and monolithic kernel etc.

Design principles: Linux is a multiuser, multitasking system with a full set of UNIX compatible tools.

- Its file system adheres to traditional UNIX semantics and it fully implements the standard UNIX networking model.

Main design goals are speed, efficiency and standardization.

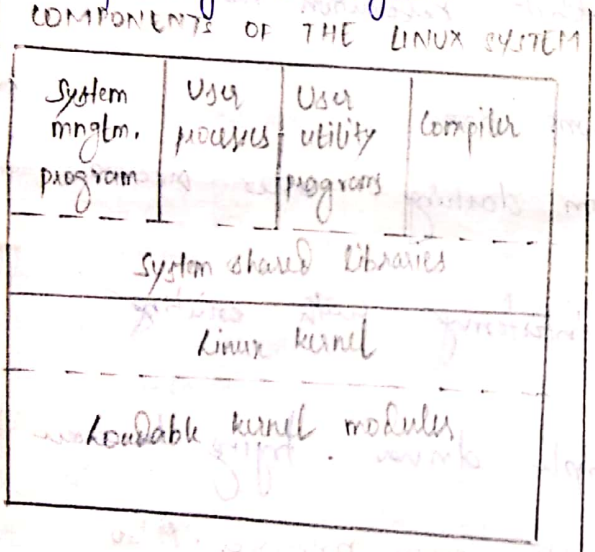
- Linux is designed to be compliant with the relevant POSIX documents; at least two Linux distributions have achieved official POSIX certification.

Components of Linux System:

1. Kernel: It is responsible for maintaining all the important abstractions of the operating system, including virtual memory and processes.

2. System Libraries: The system libraries define a standard set of functions through which applications can interact with the kernel. These functions implement much of the operating-system functionality that does not need the full privileges of kernel code.

3. System Utilities: The system utilities are programs that perform individual, specialized management task. Some system utilities are invoked just once to initialize and configure some aspect of the system.



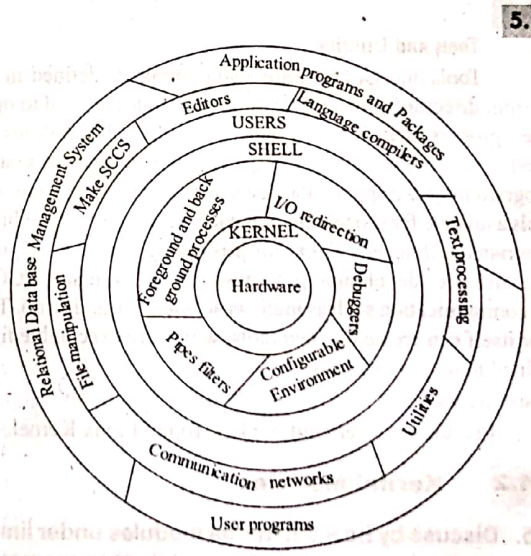


Figure: Linux Architecture

- Linux refers to the privileged mode as kernel mode.
- Under Linux, no user code is built into the kernel.
- Any operating-system-support code that does not need to run in kernel mode is placed into the system libraries and runs in user mode.
- Unlike kernel mode, user mode has access only to a controlled subset of the system resources.
- The kernel is responsible for maintaining the important abstractions of the operating system.
- Kernel code executes in kernel mode with full access to all the physical resources of the computer.

- All kernel code and data structures are kept in the same single address space address space.

Linux Architecture :

- Linux operating system consists of a hardware, kernel shell and various tools and utilities. All the parts are arranged around the computer hardware.

1. Hardware : consists of a peripheral devices such as
   (i) Random Access Memory (RAM)
   (ii) Hard disk Drive (HDD)
   (iii) Central processing Unit (CPU).
   (iv) Input / output devices.

2. kernel :

Kernel is the central core of Unix Operating System. It can directly interact with the hardware. The main functions performed by the kernel are.

(i) It controls computer hardware resources such as memory, disc, printers and so on.
(ii) It schedules processes, control and execute various user jobs.
(iii) It manages data storage and access.
(iv) It controls the access to the computer by several users.

3. Shell :

- The shell which is also command interpreter is the most important component of the unix structure because it receives and understands the commands issued by the user.
- It acts as an interface between the user and the kernel. The user interacts with the shell by issuing shell commands.

There are 2 major parts of a shell.
First, the interpreter that interprets the commands given by the user and gets them executed by the kernel.
Second, The programming capability that allows the users to write a shell script. A shell script, also known as a shell program, is a file that contains shell commands which perform a specific task.

Tools and utilities :
Tools and Utilities, are also commands defined in the system directories /bin and /usr/bin. The tools are used to open the

process them and then return the output.

- The utilities are used to ease the job of the user, particularly in efficient system programming and application development.

There are numerous tools available for processing files such as wc, pipes; for editing programs such as ed, vi, ex; for processing text such as grep, cut, paste; for developing programs such as my, make, sccs.