**Process Management:**

In Linux, unlike windows, all commands are run on terminal/shell. All administrative tasks can be accomplished through terminal/shell.

The process is a program in execution. The process is created when a command is to be executed so, it can be called a running instance of a program in execution. Tuning or controlling a process is called Process Management.

Any process can be run in two ways:

Foreground process: By default, All the processes are run in the foreground. When a process is run in foreground, no other process can be run on the same terminal until the process is finished or killed.

Background process: Adding ‘&’ to a foreground command makes it a background process. A background process runs on its own without input from the source and waits for input from the source. While the process runs in the background, other processes can be run in the foreground.

There are five types of Process in Linux

1. Parent process: All processes have a parent process, If it was created directly by user then the parent process will be the kernel process. All the processes are created when a process executes the fork() system call except the startup process. The process that executes the fork() system call is the parent process. A parent process is one that creates a child process using a fork() system call. … 0 is returned to the child process.

2. Child process: The process created by another process (by its parent process). All child processes have a parent process. A child process is created as its parent process’s copy and inherits most of its attributes. If a child process has no parent process, it was created directly by the kernel. If a child process exits or is interrupted, then a SIGCHLD signal is send to the parent process.

3. Orphan process: An orphan process in OS is one which is executing but its parent process has terminated is called an orphan process. Kernel allocates a new process as parent process to orphan process. Mostly the new parent is the init process (pid=1). Too many orphan processes are harmful.

4. Zombie process: A zombie process or defunct process is a process that has completed execution (via the exit system call) but still has an entry in the process table. This occurs for the child processes, where the entry is still needed to allow the parent process to read its child's exit status. Once the exit status is read via the wait system call, the zombie's entry is removed from the process table and said to be "reaped". A child process always first becomes a zombie before being removed from the resource table.

5. Daemon process: A daemon process is a background process that is not under the direct control of the user. This process is usually started when the system is bootstrapped and it terminated with the system shut down. Usually, the parent process of the daemon process is the init process.

**Thread Modelling:**

A thread is a basic unit of CPU utilization; it comprises a thread ID, a program Counter, a register set, and a stack. It shares with other threads belonging To the same process its code section, data section, and other operating-system Resources, such as open files and signals. A traditional (or heavyweight) process

Has a single thread of control. If a process has multiple threads of control, it Can perform more than one task at a time. Figure 4.1 illustrates the difference Between a traditional single-threaded process and a multithreaded process.

1. Responsiveness: Multithreading an interactive application may allow A program to continue running even if part of it is blocked or is Performing a lengthy operation, thereby increasing responsiveness to The user. This quality is especially useful in designing user interfaces. For Instance, consider what happens when a user clicks a button that results In the performance of a time-consuming operation. A single-threaded Application would be unresponsive to the user until the operation had Completed. In contrast, if the time-consuming operation is performed in A separate thread, the application remains responsive to the user.
2. Resource sharing: Processes can only share resources through techniques Such as shared memory and message passing. Such techniques must Be explicitly arranged by the programmer. However, threads share the Memory and the resources of the process to which they belong by default. The benefit of sharing code and data is that it allows an application to Have several different threads of activity within the same address space.
3. Economy: Allocating memory and resources for process creation is costly. Because threads share the resources of the process to which they belong, It is more economical to create and context-switch threads. Empirically Gauging the difference in overhead can be difficult, but in general it is Significantly more time consuming to create and manage processes than Threads.
4. Scalability: The benefits of multithreading can be even greater in a Multiprocessor architecture, where threads may be running in parallel On different processing cores. A single-threaded process can run on only One processor, regardless how many are available. We explore this issue Further in the following section.

User – Level Threads:

The user-level threads are implemented by users and the kernel is not aware of the existence of these threads. It handles them as if they were single-threaded processes. User-level threads are small and much faster than kernel level threads.

Kernel-Level Threads:

Kernel-level threads are handled by the operating system directly and the thread management is done by the kernel. The context information for the process as well as the process threads is all managed by the kernel. Because of this, kernel-level threads are slower than user-level threads.

Multithreading Model:

Multithreading allows the application to divide its task into individual threads. In multi-threads, the same process or task can be done by the number of threads, or we can say that there is more than one thread to perform the task in multithreading. With the use of multithreading, multitasking can be achieved.

There exists three established multithreading models classifying these relationships are:

* Many to one multithreading model
* One to one multithreading model
* Many to Many multithreading models

Many to one multithreading model:

* The many to one model maps many user levels threads to one kernel thread. This type of relationship facilitates an effective context-switching environment, easily implemented even on the simple kernel with no thread support.
* The disadvantage of this model is that since there is only one kernel-level thread schedule at any given time, this model cannot take advantage of the hardware acceleration offered by multithreaded processes or multi-processor systems. In this, all the thread management is done in the userspace. If blocking comes, this model blocks the whole system.

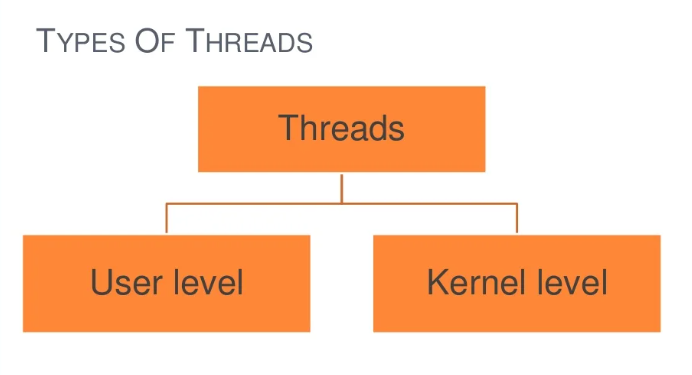
One to one multithreading model:

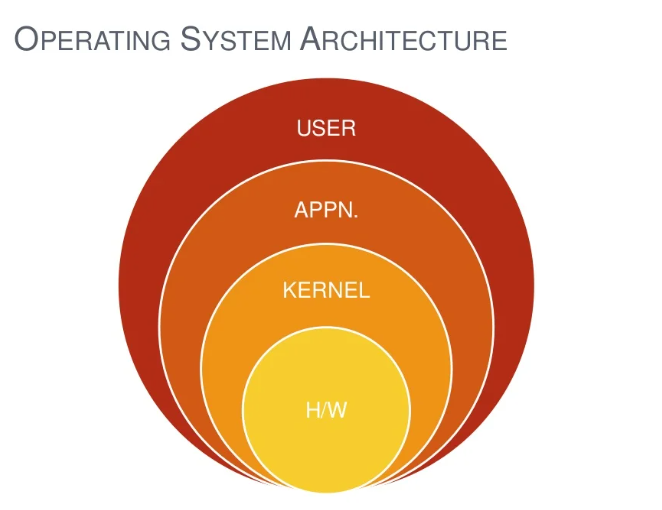
* The one-to-one model maps a single user-level thread to a single kernel-level thread. This type of relationship facilitates the running of multiple threads in parallel. However, this benefit comes with its drawback.
* The generation of every new user thread must include creating a corresponding kernel thread causing an overhead, which can hinder the performance of the parent process. Windows series and Linux operating systems try to tackle this problem by limiting the growth of the thread count.

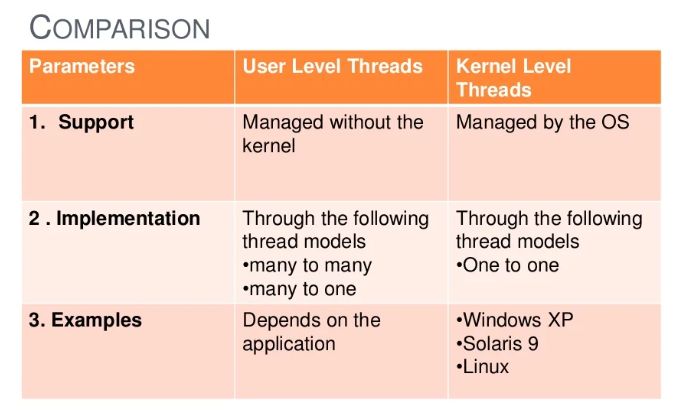
Many to Many Model multithreading model:

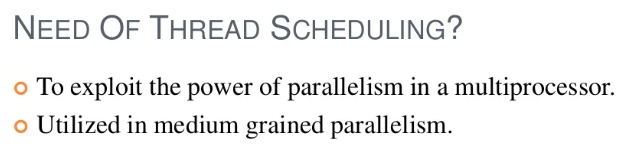
* In this type of model, there are several user-level threads and several kernel-level threads. The number of kernel threads created depends upon a particular application. The developer can create as many threads at both levels but may not be the same.
* The many to many model is a compromise between the other two models. In this model, if any thread makes a blocking system call, the kernel can schedule another thread for execution. Also, with the introduction of multiple threads, complexity is not present as in the previous models.
* Though this model allows the creation of multiple kernel threads, true concurrency cannot be achieved by this model. This is because the kernel can schedule only one process at a time.

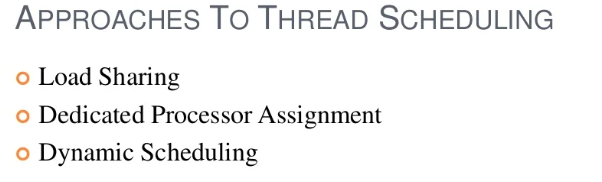
**THREAD SCHEDULING:**



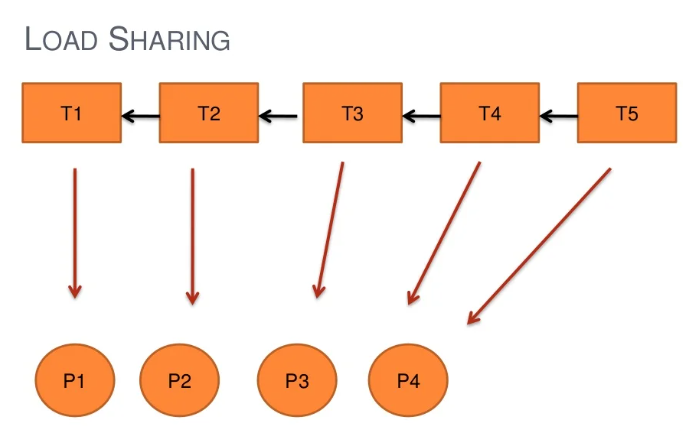


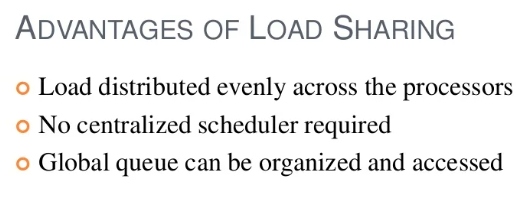


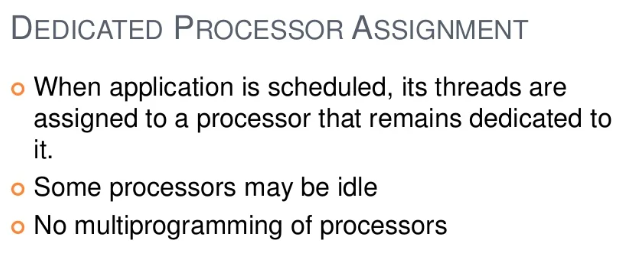




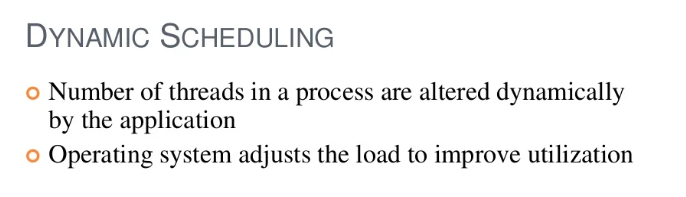
Load sharing basically denotes the process of forwarding a router to share the forwarding of traffic, in case of multiple paths if available in the routing table. In case there are equal paths then the forwarding process will follow the load-sharing algorithm.







Each program is given a set of processors equal to the number of threads it contains. why You have many processors, CPU utilization is not that important. You do not have to do process (context) switching.



**Deadlock Handling:**

What is Deadlock in OS?

* All the processes in a system require some resources such as central processing unit(CPU), file storage, input/output devices, etc to execute it. Once the execution is finished, the process releases the resource it was holding. However, when many processes run on a system they also compete for these resources they require for execution. This may arise a deadlock situation.
* A deadlock is a situation in which more than one process is blocked because it is holding a resource and also requires some resource that is acquired by some other process. Therefore, none of the processes gets executed.

Necessary Conditions for Deadlock:

The four necessary conditions for a deadlock to arise are as follows:

* Mutual Exclusion: Only one process can use a resource at any given time i.e. the resources are non-sharable.
* Hold and wait: A process is holding at least one resource at a time and is waiting to acquire other resources held by some other process.
* No preemption: The resource can be released by a process voluntarily i.e. after execution of the process.
* Circular Wait: A set of processes are waiting for each other in a circular fashion. For example, lets say there are a set of processes

Methods of Handling Deadlocks in Operating System:

The first two methods are used to ensure the system never enters a deadlock:

Deadlock Prevention:

* This is done by restraining the ways a request can be made. Since deadlock occurs when all the above four conditions are met, we try to prevent any one of them, thus preventing a deadlock.

Deadlock Avoidance:

* When a process requests a resource, the deadlock avoidance algorithm examines the resource-allocation state. If allocating that resource sends the system into an unsafe state, the request is got granted.
* Therefore, it requires additional information such as how many resources of each type is required by a process. If the system enters into an unsafe state, it has to take a step back to avoid deadlock.

Deadlock Detection and Recovery:

We let the system fall into a deadlock and if it happens, we detect it using a detection algorithm and try to recover. Some ways of recovery are as follows:

* Aborting all the deadlocked processes
* Abort one process at a time until the system recovers from the deadlock.

Resource Preemption:

Resources are taken one by one from a process and assigned to higher priority processes until the deadlock is resolved.

Deadlock Ignorance:

In the method, the system assumes that deadlock never occurs. Since the problem of deadlock situation is not frequent, some systems simply ignore it. Operating systems such as UNIX and Windows follow this approach. However, if a deadlock occurs we can reboot our system and the deadlock is resolved automatically.

Advantage of Deadlock Method:

* No preemption is needed for deadlocks.
* It is a good method if the state of the resource can be saved and restored easily.
* It is good for activities that perform a single burst of activity.
* It does not need run-time computations because the problem is solved in system design.

Disadvantages of Deadlock Method:

* The processes must know the maximum resource of each type required to execute it.
* Preemptions are frequently encountered.
* It delays the process initiation.
* There are inherent pre-emption losses.
* It does not support incremental request of resources.

**Disk Scheduling:**

Disk scheduling is done by operating systems to schedule I/O requests arriving for the disk. Disk scheduling is also known as I/O scheduling.

Disk scheduling is important because:

* Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller. Thus other I/O requests need to wait in the waiting queue and need to be scheduled.
* Two or more request may be far from each other so can result in greater disk arm movement.
* Hard drives are one of the slowest parts of the computer system and thus need to be accessed in an efficient manner.
* There are many Disk Scheduling Algorithms but before discussing them let’s have a quick look at some of the important terms:

Seek Time:

Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or write. So the disk scheduling algorithm that gives minimum average seek time is better.

Rotational Latency:

Rotational Latency is the time taken by the desired sector of disk to rotate into a position so that it can access the read/write heads. So the disk scheduling algorithm that gives minimum rotational latency is better.

Transfer Time:

Transfer time is the time to transfer the data. It depends on the rotating speed of the disk and number of bytes to be transferred.

Disk Access Time:

Disk Access Time = Seek Time + Rotational Latency + Transfer Time

Disk Scheduling Algorithms:

FCFS:

FCFS is the simplest of all the Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue.

Advantages:

* Every request gets a fair chance
* No indefinite postponement

Disadvantages:

* Does not try to optimize seek time
* May not provide the best possible service

SSTF:

* In SSTF (Shortest Seek Time First), requests having shortest seek time are executed first.
* So, the seek time of every request is calculated in advance in the queue and then they are scheduled according to their calculated seek time. As a result, the request near the disk arm will get executed first.
* SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of system.

Advantages:

* Average Response Time decreases
* Throughput increases

Disadvantages:

* Overhead to calculate seek time in advance
* Can cause Starvation for a request if it has higher seek time as compared to incoming requests
* High variance of response time as SSTF favors only some requests

SCAN:

* In SCAN algorithm the disk arm moves into a particular direction and services the requests coming in its path and after reaching the end of disk, it reverses its direction and again services the request arriving in its path.
* So, this algorithm works as an elevator and hence also known as elevator algorithm. As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

Advantages:

* High throughput
* Low variance of response time
* Average response time

Disadvantages:

* Long waiting time for requests for locations just visited by disk arm

CSCAN:

* In SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction.
* So, it may be possible that too many requests are waiting at the other end or there may be zero or few requests pending at the scanned area.
* These situations are avoided in CSCAN algorithm in which the disk arm instead of reversing its direction goes to the other end of the disk and starts servicing the requests from there.
* So, the disk arm moves in a circular fashion and this algorithm is also similar to SCAN algorithm and hence it is known as C-SCAN (Circular SCAN).

Advantages:

* Provides more uniform wait time compared to SCAN

LOOK:

* It is similar to the SCAN disk scheduling algorithm except for the difference that the disk arm in spite of going to the end of the disk goes only to the last request to be serviced in front of the head and then reverses its direction from there only.
* Thus it prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

Advantages:

* High throughput
* Low variance of response time
* Average response time

Disadvantages:

* Long waiting time for requests for locations just visited by disk arm

CLOOK:

* As LOOK is similar to SCAN algorithm, in similar way, CLOOK is similar to CSCAN disk scheduling algorithm.
* In CLOOK, the disk arm in spite of going to the end goes only to the last request to be serviced in front of the head and then from there goes to the other end’s last request.
* Thus, it also prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

Advantages:

* Provides more uniform wait time compared to SCAN

**Memory Management:**

* Memory management is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution.
* Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free.
* It checks how much memory is to be allocated to processes. It decides which process will get memory at what time.

Memory Allocation:

Main memory usually has two partitions :

* Low Memory – Operating system resides in this memory.
* High Memory – User processes are held in high memory.

Operating system uses the following memory allocation mechanism:

Memory Allocation & Description:

Single-Partition Allocation:

* In this type of allocation, relocation-register scheme is used to protect user processes from each other, and from changing operating-system code and data.
* Relocation register contains value of smallest physical address whereas limit register contains range of logical addresses. Each logical address must be less than the limit register.

Multiple-Partition Allocation:

* In this type of allocation, main memory is divided into a number of fixed-sized partitions where each partition should contain only one process.
* When a partition is free, a process is selected from the input queue and is loaded into the free partition.
* When the process terminates, the partition becomes available for another process.

Fragmentation:

* As processes are loaded and removed from memory, the free memory space is broken into little pieces.
* It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused.
* This problem is known as Fragmentation.

Fragmentation is of two types :

* External fragmentation:

Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.

* Internal fragmentation:

Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.

Paging:

* Paging is a memory management technique in which process address space is broken into blocks of the same size called pages (size is power of 2, between 512 bytes and 8192 bytes). The size of the process is measured in the number of pages.
* Similarly, main memory is divided into small fixed-sized blocks of (physical) memory called frames and the size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to avoid external fragmentation.

Address Translation:

Page address is called logical address and represented by page number and the offset.

* Logical Address = Page number + page offset

Frame address is called physical address and represented by a frame number and the offset.

* Physical Address = Frame number + page offset

Process:

* A data structure called page map table is used to keep track of the relation between a page of a process to a frame in physical memory.
* When the system allocates a frame to any page, it translates this logical address into a physical address and create entry into the page table to be used throughout execution of the program.
* When a process is to be executed, its corresponding pages are loaded into any available memory frames.
* This process continues during the whole execution of the program where the OS keeps removing idle pages from the main memory and write them onto the secondary memory and bring them back when required by the program.

Advantages and Disadvantages of Paging:

* Paging reduces external fragmentation, but still suffer from internal fragmentation.
* Paging is simple to implement and assumed as an efficient memory management technique.
* Due to equal size of the pages and frames, swapping becomes very easy.
* Page table requires extra memory space, so may not be good for a system having small RAM

Segmentation:

* Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions. Each segment is actually a different logical address space of the program.
* Segmentation memory management works very similar to paging but here segments are of variable-length where as in paging pages are of fixed size.
* A program segment contains the program’s main function, utility functions, data structures, and so on. The operating system maintains a segment map table for every process and a list of free memory blocks along with segment numbers, their size and corresponding memory locations in main memory.
* For each segment, the table stores the starting address of the segment and the length of the segment. A reference to a memory location includes a value that identifies a segment and an offset.

How to check hard disk, CPU, and memory usage in elementary OS?

File Systems:

The File Systems tab gives the hard disk usage of all the front-end accessible partitions. In addition, it shows the available free hard disk space and the used space.

System Resources:

The Resources tab gives CPU Usage History, Memory and SWAP consumption, and Network usage details.

Processes:

The processes tab has a list of all running processes in the session.

**File Management in OS:**

File Concept:

* The file is a collection of related information that is recorded on storage device.
* The files stores information and data.
* It is a way of data collection which is used as medium of giving input and collecting output from any application.

File Types:

1. Ordinary Files (Regular Files)

– It contains user information

– It contains text or executable programs

– It is either text file or binary file

– User can apply various operations

Directory Files:

– It is like a folder that contains files or another folder

– It doesn’t contain data

– But they have references to file contain within them

Device File:

– It contains device information

– It is used to communicate with hardware

– Device file is of two types

I. Character special file It is a hardware file which read or write data character by character e.g.: mouse, printer

ll. Block file It contains data related to hard disk

FIFO Files:

– It is a file which provides temporary buffer for two or more process to communicates by writing data & reading data from buffer

– The buffer associated with this file is allocated when any process opens the file & buffer is discarded when all processes which are connected to this file close their reference

File Operations:

1. Create 2. Delete 3. Open 4. Close 5. Read 6. Write 7. Append 8. Seek 9. Get Attributes 10. Set Attributes 11. Rename

Directory Structure:

• To keep track of files, file systems normally have directories of folders.

• Directories are system files for maintaining the structure of the file system.

1. Single Level Directory Structure

2. Hierarchical Directory

l. Two Level Directory Structure

ll. Tree Level Directory Structure

lll. Acyclic Graph Directory

File System Structure:

* The concept of file system is to access the disk efficiently. It allows data to store & retrieve from disk.

File system structure is composed of many layers as following:

1. Input-Output Control Interface:

* It consist of device driver and interrupt handler
* Both are used for data transfer between memory & disk
* The device driver translates file level command & output is hardware instruction

2. Basic File System Layers:

* It generates commands for device driver
* Device driver read or write physical block on disk

3. File Organization Module Layer:

* This layer translates logical block address to physical block address

4. Logical System Layer:

* It manages metadata of any file like location & permission of file

5. Application Program Layer:

* This is a layer in which user creates an application program

Access Methods:

* The information stored in the file needs to be accessed and read into the computer memory.
* There are different methods available to do it. 1) Sequential Access 2) Indexed Access 3) Direct Access

Sequential Access:

* Most common form of file structure
* A fixed format is used for records
* Key field uniquely identifies the record & determines storage order
* Typically used in batch applications
* Only organization that is easily stored on tape as well as disk

Indexed Access:

* Records are accessed only through their indexes
* Variable-length records can be employed  Exhaustive index contains one entry for every record in the main file
* Partial index contains entries to records where the field of interest exists
* Used mostly in applications where timeliness of information is critical

Direct Access:

* Access directly any block of a known address
* Makes use of hashing on the key value
* Often used where:
* very rapid access is required
* fixed-length records are used
* records are always accessed one at a time

**Comparision between elementary OS and Ubuntu:**

* Since elementary OS is based in Ubuntu, many of the core features between the distros overlap. We have mainly highlighted 5 key areas where the OSes differ from each other and factors of high interest.
* Also, we are pitting elementary OS against Ubuntu 20.04 LTS for this comparison overview. However, most of the points still stand valid even if you are using a slightly older or newer build of the distros.

User interface & user experience:

* One of the biggest reasons elementary OS has amassed such a huge popularity is its user interface.
* The distro features its own home-grown desktop environment – Pantheon focusing on clean aesthetics and a user-friendly UI.
* The design takes slight cues from the macOS, which is also why this distro is recommended to new Linux users coming from Apple.
* Your entire workflow revolves around the bottom dock from where you can pin and launch your favorite apps.
* Then there’s the application menu at the top-left from where you can browse and launch all the apps installed on your system.
* Next, you have the Time and Date information in the middle, and clicking on it will also open the notification area. And finally, on the top-right, you have all the system quick settings.

Software Repositories and Package Managers:

* Ubuntu has one of the largest official software repositories among all Linux distros.
* So, if a developer has released an app for Linux, you can bet that it’s made available for Ubuntu.
* And if an app is not available in the official repo, you will most likely find a PPA (Personal Page Archive) for it. PPAs are mostly maintained by software developers themselves and are generally safe to install.

Ubuntu:

* Ubuntu has two release branches – an LTS (Long Term Support) release and a non-LTS release.
* A new LTS version gets released every 2 years and is supported for the next 5 years. During the 5 years, you don’t technically need to upgrade your system and can keep using your installation. However, for the sake of stability, the software repo will also hold on to older versions of apps, and you will not be able to access the latest software until you upgrade your system.
* The current LTS release is Ubuntu 20.04.
* On the other end, we have the non-LTS version designed for users who prioritizes access to new and up-to-date software. Here, a new version is released every 6 months and is supported for 9 months. Non-LTS releases do notice slightly more bugs that get ironed out with the LTS release.
* The current non-LTS release is Ubuntu 21.04.

Elementary OS:

* Elementary OS is based on the LTS branch of Ubuntu. This means that you can expect a new build announcement following a new Ubuntu LTS release.
* Elementary OS 5.0 Juno was based on Ubuntu 18.04 LTS and was released 6 months after Ubuntu 18.04. 5.1 Hera, an upgrade over Juno, is also based on Ubuntu 18.04 LTS and was released around 1 year after Juno.

Privacy & Security Features:

* All things considered, if you are using Linux, you are in safe hands.
* That being said, you also need to consider that security and privacy are almost the same things as far as computers are concerned. You want high security because you want to keep your private information safe.
* Now that being said, you do have the option to opt-out from giving canonical access to your data. But the very fact that the option is there, to begin with, brings in a lot of criticism.
* On the flip side, elementary OS is privacy respecting through and through! This doesn’t collect user data, nor does it push ads or deals based on user behavior. So if privacy is your utmost concern, then elementary OS is the better distro over Ubuntu.