SYLLABUS-SEMESTER 4TH (CBCS) - B.A/B.SC COMPUTER APPLICATIONS

(SKILL ENHANCEMENT COURSE- THEORY)

COURSE CODE: 1CAPSE0401 **COURSE TITLE: OPEN SOURCE SOFTWARE**

(Syllabus wise Topic Description)

UNIT-4: Application of Open Source Operating System: LINUX:

Introduction:

Linux is the best-known and most-used open source operating system. As an operating system, Linux is software that sits underneath all of the other software on a computer, receiving requests from those programs and relaying these requests to the computer's hardware.

How does Linux differ from other operating systems?

In many ways, Linux is similar to other operating systems you may have used before, such as Windows, OS X, or ioS. Like other operating systems, Linux has a graphical interface, and types of software you are accustomed to using on other operating systems, such as word processing applications, have Linux equivalents. In many cases, the software's creator may have made a Linux version of the same program you use on other systems. If you can use a computer or other electronic device, you can use Linux.

But Linux also is different from other operating systems in many important ways. First, and perhaps most importantly, Linux is open source software. The code used to create Linux is free and available to the public to view, edit, and—for users with the appropriate skills—to contribute to.

Linux is also different in that, although the core pieces of the Linux operating system are generally common, there are many distributions of Linux, which include different software options. This means that Linux is incredibly customizable, because not just applications, such as word processors and web browsers, can be swapped out. Linux users also can choose core components, such as which system displays graphics, and other user-interface components.

Who uses Linux?

Companies and individuals choose Linux for their servers because it is secure, and you can receive excellent support from a large community of users, in addition to companies like Canonical, SUSE, and Red Hat, which offer commercial support.

Many of the devices you own probably, such as Android phones, digital storage devices, personal video recorders, cameras, wearables, and more, also run Linux.

Who "owns" Linux?

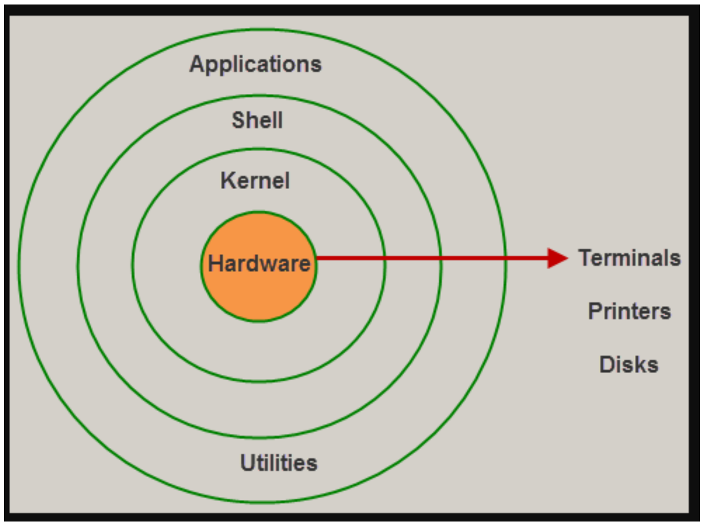
By virtue of its open source licensing, Linux is freely available to anyone. However, the trademark on the name "Linux" rests with its creator, Linus Torvalds. The source code for Linux is under copyright by its many individual authors, and licensed under the GPLv2 license. Because Linux has such a large number of contributors from across multiple decades of development, contacting each individual author and getting them to agree to a new license is virtually impossible, so that Linux remaining licensed under the GPLv2 in perpetuity is all but assured.

How was Linux created?

Linux was created in 1991 by Linus Torvalds, a then-student at the University of Helsinki.

Linux System Architecture

The Linux Operating System's architecture primarily has these components: the Kernel, Hardware layer, System library, Shell and System utility.



1. ***The kernel*** is the core part of the operating system, which is responsible for all the major activities of the LINUX operating system. This operating system consists of different modules and interacts directly with the underlying hardware. The kernel offers the required abstraction to hide application programs or low-level hardware details to the system.

2. ***System libraries*** are special functions, that are used to implement the functionality of the operating system and do not require code access rights of kernel modules.

3. ***System Utility programs*** are liable to do individual, and specialized-level tasks.

4. Hardware layer of the LINUX operating system consists of peripheral devices such as RAM, HDD, and CPU.

5. ***The shell*** is an interface between the user and the kernel, and it affords services of the kernel. It takes commands from the user and executes kernel's functions. The Shell is present in different types of operating systems, which are classified into two types:command line shells and graphical shells.

Features of Linux Operating System

The main features of Linux operating system are

- **Portable:** Linux operating system can work on different types of hardwares as well as Linux kernel supports the installation of any kind of hardware platform.
- **Open Source:** Source code of LINUX operating system is freely available and, to enhance the ability of the LINUX operating system, many teams work in collaboration.
- **Multiuser:** Linux operating system is a multiuser system, which means, multiple users can access the system resources like RAM, Memory or Application programs at the same time.
- **Multiprogramming:** Linux operating system is a multiprogramming system, which means multiple applications can run at the same time.

- **Hierarchical File System:** Linux operating system affords a standard file structure in which system files or user files are arranged.
- **Shell:** Linux operating system offers a special interpreter program, that can be used to execute commands of the OS. It can be used to do several types of operations like call application programs, and so on.
- **Security:** Linux operating system offers user security systems using authentication features like encryption of data or password protection or controlled access to particular files.

Applications of Linux Operating System

Nowadays, Linux is a multibillion dollar industry. Thousands of companies and governments around the world are using Linux OS due to affordability, lower licensing fee and time and money. Linux is used in a number of electronic devices, which are available for consumers worldwide. The list of some of popular Linux based electronic devices includes:

- Dell Inspiron Mini 9 and 12
- Garmin Nuvi 860, 880, and 5000
- Google Android Dev Phone 1
- HP Mini 1000
- Lenovo IdeaPad S9
- Motorola MotoRokr EM35 Phone
- One Laptop Per Child XO2
- Sony Bravia Television

User and Kernel Mode

Most operating systems have some method of displaying CPU utilization. In Windows, this is Task Manager. CPU usage is generally represented as a simple percentage of CPU time spent on non-idle tasks. But this is a bit of a simplification. In any modern operating system, the CPU is actually spending time in two very distinct modes:

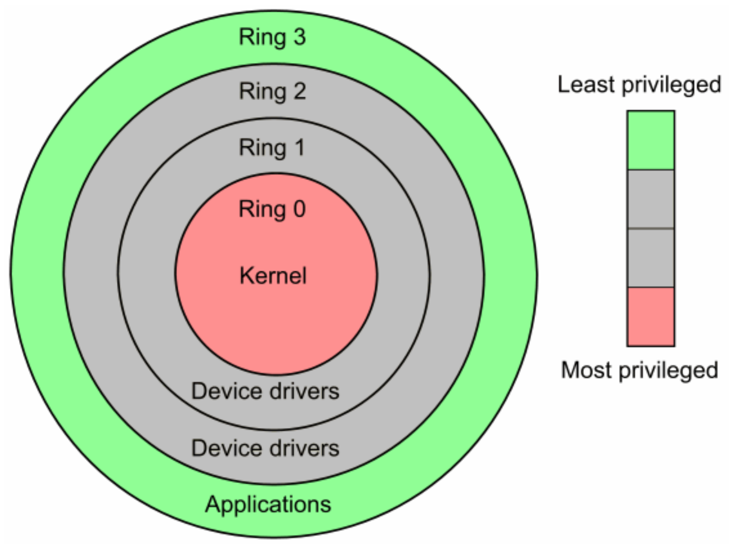
1. **Kernel Mode**
   In Kernel mode, the executing code has complete and unrestricted access to the underlying hardware. It can execute any CPU instruction and reference any memory address. Kernel mode is generally reserved for the lowest-level, most trusted functions of the operating system. Crashes in kernel mode are catastrophic; they will halt the entire PC.

2. **User Mode**
   In User mode, the executing code has no ability to directly access hardware or reference memory. Code running in user mode must delegate to system APIs to access hardware or memory. Due to the protection afforded by this sort of isolation, crashes in user mode are always recoverable. Most of the code running on your computer will execute in user mode.

It's possible to enable display of Kernel time in Task Manager, as I have in the above screenshot. The green line is total CPU time; the red line is Kernel time. The gap between the two is User time.

These two modes aren't mere labels; they're enforced by the CPU hardware. If code executing in User mode attempts to do something outside its purview-- like, say, accessing a privileged CPU instruction or modifying memory that it has no access to -- a trappable exception is thrown. Instead of your entire system crashing, only that particular application crashes. That's the value of User mode.

x86 CPU hardware actually provides four protection rings: 0, 1, 2, and 3. Only rings 0 (Kernel) and 3 (User) are typically used.



If we're only using two isolation rings, it's a bit unclear where device drivers should go-- the code that allows us to use our video cards, keyboards, mice, printers, and so forth. Do these drivers run in Kernel mode, for maximum performance, or do they run in User mode, for maximum stability? In Windows, at least, the answer is it depends. Device drivers can run in either user or kernel mode. Most drivers are shunted to the User side of the fence these days, with the notable exception of video card drivers, which need bare-knuckle Kernel mode performance. But even that is changing; in Windows Vista, video drivers are segmented into User and Kernel sections. Perhaps that's why gamers complain that Vista performs about 10 percent slower in games.

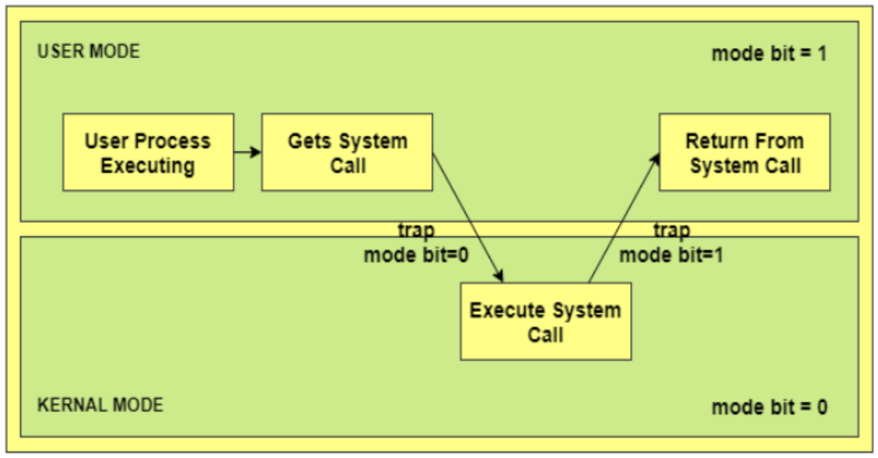
The exact border between these modes is still somewhat unclear. What code should run in User mode? What code should run in Kernel mode? Or maybe we'll just redefine the floor as the basement-- the rise of virtualization drove the creation of a new ring below all the others, Ring -1, which we now know as x86 hardware virtualization.

User mode is clearly a net public good, but it comes at a cost. Transitioning between User and Kernel mode is expensive. Really expensive. It's why software that throws exceptions is slow, for example. Exceptions imply kernel mode transitions. Granted, we have so much performance now that we rarely have to care about transition performance, but when you need ultimate performance, you definitely start caring about this stuff.

Probably the most public example of redrawing the user / kernel line is in webservers. Microsoft's IIS 6 moved a sizable chunk of its core functionality into Kernel mode, most notably after a particular open-source webserverleveraged Kernel mode to create a huge industry benchmark victory. It was kind of a pointless war, if you ask me, since the kernel optimizations (in both camps) only apply to static HTML content. But such is the way of all wars, benchmark or otherwise.

The CPU's strict segregation of code between User and Kernel mode is completely transparent to most of us, but it is quite literally the difference between a computer that crashes all the time and a computer that crashes catastrophically all the time. This is what we extra-crashy-code-writing programmers like to call "progress".

An image that illustrates the transition from user mode to kernel mode and back again is:



Necessity of Dual Mode (User Mode and Kernel Mode) in Operating System

The lack of a dual mode i.e. user mode and kernel mode in an operating system can cause serious problems. Some of these are:

- A running user program can accidentally wipe out the operating system by overwriting it with user data.
- Multiple processes can write in the same system at the same time, with disastrous results.

These problems could have occurred in the MS-DOS operating system which had no mode bit and so no dual mode.

Process & Scheduling:

Process in Linux

A **process** is a program in execution in memory or in other words, an instance of a program in memory. A process is a running instance of a program. Any program executed creates a process. A program can be a command, a shell script, or any binary executable or any application. However, not all commands end up in creating process, there are some exceptions. Similar to how a file created has properties associated with it, a process also has lots of properties associated to it.

List the processes:
ps is the Unix / Linux command which lists the active processes and its status. By default, it lists the processes belonging to the current user being run from the current terminal.

The ps command output shows 4 things:
PID : The unique id of the process
TTY: The terminal from which the process or command is executed.
TIME: The amount of CPU time the process has taken
CMD: The command which is executed.

Two processes are listed in the above case:
1. -ksh : The login shell, which we are working on, is also a process which is currently running.
2. ps : The ps command which we executed to get the list also creates a process. And hence, by default, there will be atleast 2 processes when executing the ps command.

Types of Processes:

1. **Parent and Child process :** Every process in Linux has to be created by some other process. Hence, the ps command is also created by some other process. The 'ps' command is being run from the login shell, ksh. The ksh shell is a process running in the memory right from the moment the user logged in. So, for all the commands trigerred from the login shell, the login shell will be the parent process and the process created for the command executed will be the child process. In the same lines, the 'ksh' is the parent process for the child process 'ps'.

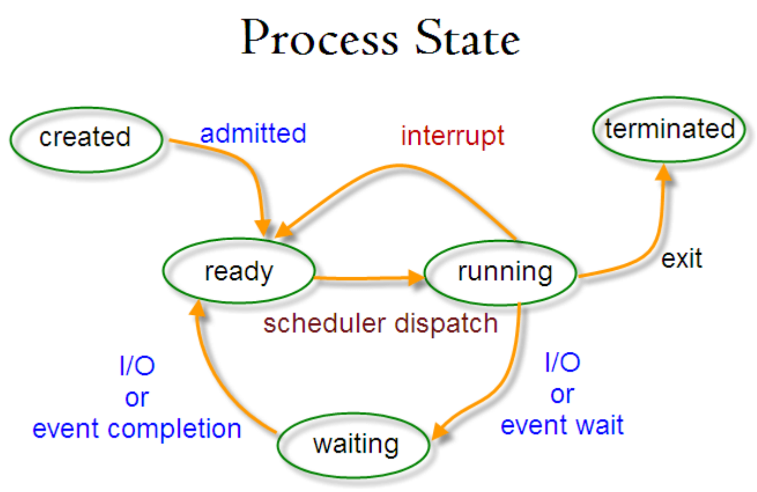
2. **Zombie and Orphan process :** After completing its execution a child process is terminated or killed and SIGCHLD updates the parent process about the termination and thus can continue the task assigned to it. But at times when the parent process is killed before the termination of the child process, the child processes becomes orphan processes, with the parent of all processes "init" process, becomes their new ppid. A process which is killed but still shows its entry in the process status or the process table is called a zombie process, they are dead and are not used.
3. **Daemon process :** They are system-related background processes that often run with the permissions of root and services requests from other processes, they most of the time run in the background and wait for processes it can work along with for ex print daemon. When ps –ef is executed, the process with ? in the tty field are daemon processes

The Life Cycle of Processes

From the time a process is created with a fork() until it has completed its job and disappears from the process table, it goes through many different states. The state a process is in changes many times during its "life." These changes can occur, for example, when the process makes a system call, it is someone else's turn to run, an interrupt occurs, or the process asks for a resource that is currently not available.

A commonly used model shows processes operating in one of six separate states:

1. executing in user mode
2. executing in kernel mode
3. ready to run
4. sleeping
5. newly created, not ready to run, and not sleeping
6. issued exit system call (zombie)



CPU SCHEDULING

The *scheduler* is responsible for keeping the CPUs in the system busy. The Linux scheduler implements a number of *scheduling policies*, which determine when and for how long a thread runs on a particular CPU core. The **process scheduling** is the activity of the **process** manager that handles the removal of the running **process** from the CPU and the selection of another **process** on the basis of a particular strategy. **Process scheduling** is an essential part of a Multiprogramming operating systems.

Scheduling Policy: when to switch and what process to choose.
Some scheduling objectives:

- fast process response time
- avoidance of process starvation
- good throughput for background jobs
- support for soft real time processes

Scheduling policies are divided into two major categories:

1. ***Real time policies***
    - SCHED_FIFO
    - SCHED_RR
2. ***Normal policies***
    - SCHED_OTHER
    - SCHED_BATCH
    - SCHED_IDLE

- **Real-time scheduling policies:**

Real-time threads are scheduled first, and normal threads are scheduled after all real-time threads have been scheduled. The *real-time* policies are used for time-critical tasks that must complete without interruptions.

SCHED_FIFO

This policy is also referred to as *static priority scheduling*, because it defines a fixed priority (between 1 and 99) for each thread. The scheduler scans a list of SCHED_FIFO threads in priority order and schedules the highest priority thread that is ready to run. This thread runs until it blocks, exits, or is preempted by a higher priority thread that is ready to run.

Even the lowest priority real-time thread will be scheduled ahead of any thread with a non-real-time policy; if only one real-time thread exists, the SCHED_FIFO priority value does not matter.

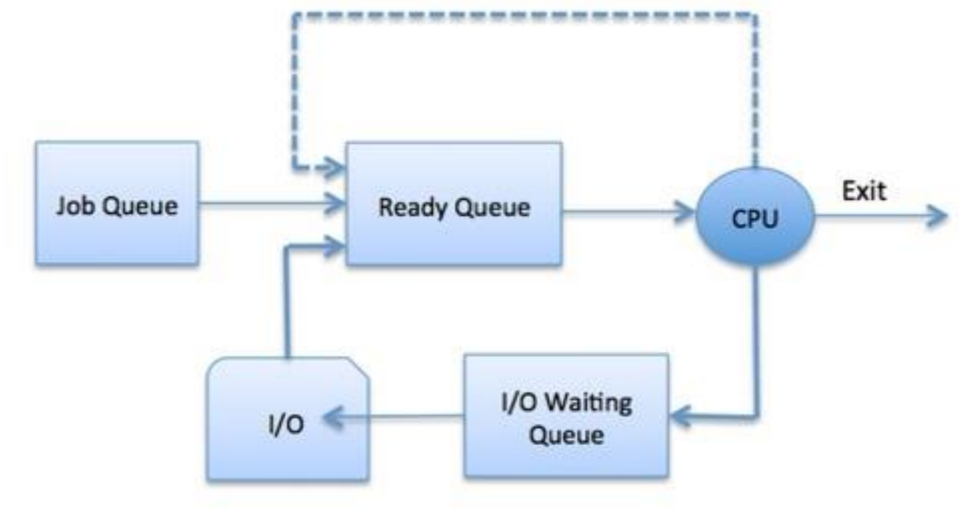
SCHED_RR
A round-robin variant of the SCHED_FIFO policy. SCHED_RR threads are also given a fixed priority between 1 and 99. However, threads with the same priority are scheduled round-robin style within a certain quantum, or time slice. The sched_rr_get_interval(2) system call returns the value of the time slice, but the duration of the time slice cannot be set by a user. This policy is useful if you need multiple thread to run at the same priority.

- **Process Scheduling Queues**

The OS maintains all PCBs in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

The Operating System maintains the following important process scheduling queues −

- **Job queue** − This queue keeps all the processes in the system.
- **Ready queue** − This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
- **Device queues** − The processes which are blocked due to unavailability of an I/O device constitute this queue.



The OS can use different policies to manage each queue (FIFO, Round Robin, Priority, etc.). The OS scheduler determines how to move processes between the ready and run queues which can only have one entry per processor core on the system; in the above diagram, it has been merged with the CPU.

Schedulers

Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types −

- Long-Term Scheduler
- Short-Term Scheduler
- Medium-Term Scheduler

➢ Long Term Scheduler

It is also called a **job scheduler**. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.

The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

On some systems, the long-term scheduler may not be available or minimal. Time-sharing operating systems have no long term scheduler. When a process changes the state from new to ready, then there is use of long-term scheduler.

➢ Short Term Scheduler

It is also called as **CPU scheduler**. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

➢ Medium Term Scheduler

Medium-term scheduling is a part of **swapping**. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes.

A running process may become suspended if it makes an I/O request. A suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called **swapping**, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

Development with Linux

Linux is one of the most popular platforms used for development. Its widespread adoption stems from core design principles behind its kernel: to be extensible, efficient, modular, simple, robust and open source. The Linux open source system created a framework for software development that was decentralized, open and peer-reviewed – a model that persists today in modern software development practices. Other open source projects, like the Apache web server, brought Linux mainstream visibility and success. Users adopted Apache on Linux servers, accounting for most of the websites on the internet. Linux's modularity made it easier to secure and its open source code meant an entire community could work on removing vulnerabilities. Today, Linux and its many distributions provide the flexibility and reliability needed for the budding developer and the enterprise level website or application.

Terminal Commands

Linux at heart is a command-line operating system with many commands for controlling and configuring the system, running applications, and so on. You can control everything from the command line. To be fair to those GUIs, they are far more user-friendly than a terminal, and let you run multiple terminals at once; you can also browse around the file system far more easily than issuing commands in a terminal. Irrespective of the GUI used, if you are familiar with the terminal commands and one of the shells, you can find your way around any Linux deployment. There are many popular shells, though everyone knows Bash (a.k.a. Bourne-again Shell); they differ in features such as history, saving commands as scripts, command line completion and the like.

Prefer Linux to Mac or Windows. Here, following are the reasons:

- Linux is free and open source, released under the GNU General Public License (GPL). This means you can install and use a Linux distribution on your machine for free.
- Lightweight Linux distributions have low minimum system resource requirements. These can be used to rescue aging computers from obsolescence.
- New server technologies are often made available on Linux first, so you can start building with the newest and latest tooling.
- Customize and personalize your Linux distribution with configuration files, multiple shell choices and desktop environments that range from minimal window managers to full-featured GUIs.
- It offers greater breadth and depth of open-source software.
- It's less fiddly running open-source software on Linux.
- It runs on anything, especially an old Windows PC.
- Once you've learned the terminal commands, you can be extremely efficient and productive.
- Rebuilding a Windows Development PC after a major crash can take hours or days to reinstall everything. Linux, by contrast, is a lot quicker and also easier to containerize.
- The development environment is similar to production. It's also likely to stay stable longer, as there's less commercial pressure to release new OS versions.
- In the world of servers, it reigns supreme. It's also a major player in mobile, thanks to Android.