# NEW L J INSTITUTE OF ENGINEERING & TECHNOLOGY

## SEMESTER- 4

## BRANCH- CSE/CSE[AIML]

## SUBJECT NAME: OPERATING SYSTEM

## SUBJECT CODE: 3140702

## GTU SYLLABUS

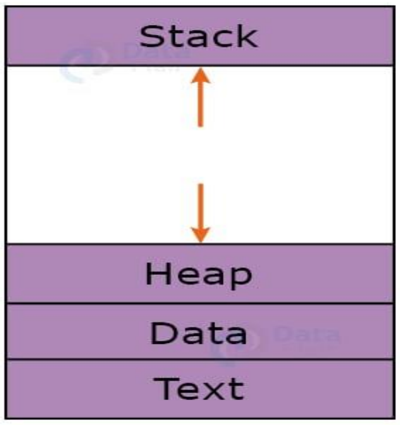
| Sr. No. | Content | % Weightage |
|---|---|---|
| 1 | **Introduction:** Computer system overview, Architecture, Goals & Structures of O.S, Basic functions, Interaction of O.S. & hardware architecture, System calls, Batch, multiprogramming. Multitasking, time sharing, parallel, distributed & realtime O.S. | 10 |
| | | |
| 2 | **Process and Threads Management:** Process Concept, Process states, Process control, Threads, Uni-processor Scheduling: Types of scheduling: Preemptive, Non preemptive, Scheduling algorithms: FCFS, SJF, RR, Priority, Thread Scheduling, Real Time Scheduling. System calls like ps, fork, join, exec family, wait. | 15 |
| | | |
| 3 | **Concurrency:** Principles of Concurrency, Mutual Exclusion: S/W approaches, H/W Support, Semaphores, Pipes, Message Passing, Signals, Monitors. | 8 |
| | | |
| 4 | **Inter Process Communication:** Race Conditions, Critical Section, Mutual Exclusion, Hardware Solution, Strict Alternation, Peterson's Solution, The Producer Consumer Problem, Semaphores, Event Counters, Monitors, Message Passing, Classical IPC Problems: Reader's & Writer Problem, Dinning Philosopher Problem etc., Scheduling, Scheduling Algorithms. | 15 |
| | | |
| 5 | **Deadlock:** Principles of Deadlock, Starvation, Deadlock Prevention, Deadlock Avoidance, Deadlock Detection, System calls | 8 |
| | | |
| 6 | **Memory Management:** Memory Management requirements, Memory partitioning: Fixed and Variable Partitioning, Memory Allocation: Allocation Strategies (First Fit, Best Fit, and Worst Fit), Swapping, Paging and Fragmentation. Demand Paging, Security Issues. Virtual Memory: Concepts, VM management, Page Replacement Policies (FIFO, LRU, Optimal, Other Strategies), Thrashing. | 15 |
| | | |
| 7 | **I/O Management & Disk scheduling:** I/O Devices, Organization of I/O functions, Operating System Design issues, I/O Buffering, Disk Scheduling (FCFS, SCAN, C-SCAN, SSTF), RAID, Disk Cache. | 10 |
| | | |
| 8 | **Security & Protection:** Security Environment, Design Principles Of Security, User Authentication, Protection Mechanism : Protection Domain, Access Control List | 7 |
| | | |
| 9 | **Unix/Linux Operating System:** Development Of Unix/Linux, Role & Function Of Kernel, System Calls, Elementary Linux command & Shell Programming, Directory Structure, System Administration Case study: Linux, Windows Operating System | 7 |
| | | |

| | | |
|---|---|---|
| 10 | **Virtualization Concepts:** Virtual machines; supporting multiple operating systems simultaneously on a single hardware platform; running one operating system on top of another. True or pure virtualization. | 5 |

## UNIT 2: Process and Threads Management

| | | |
|---|---|---|
| **Unit 1** | **Introduction:** Process Concept, Process states, Process control, Threads, Uni-processor Scheduling: Types of scheduling: Preemptive, Non preemptive, Scheduling algorithms: FCFS, SJF, RR, Priority, Thread Scheduling, Real Time Scheduling. System calls like ps, fork, join, exec family, wait. | |
| | **TOPIC:1: Process Concept, Process states, Process control** | |
| **Q** | What is Process? Give the difference between a process and a program. | |

- A process is the unit of work in a system.
- Such a system consists of a collection of concurrently executing processes, some of which are operating-system processes (those that execute system code) and the rest of which are user processes (those that execute user code).

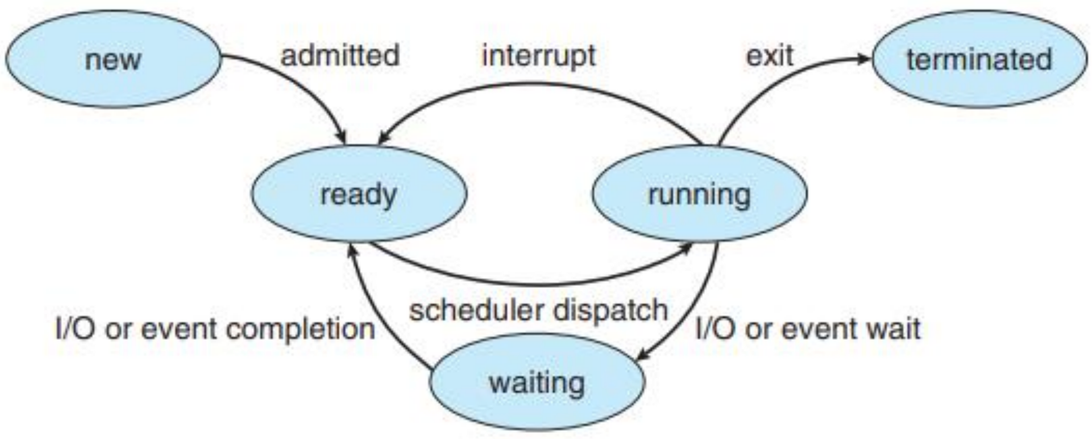


- A process is more than the program code, which is sometimes known as the text section.
- It also includes the current activity, as represented by the value of the program counter and the contents of the processor's registers.
- A process generally also includes the process stack, which contains temporary data (such as function parameters, return addresses, and local variables), and a data section, which contains global variables.
- A process may also include a heap, which is memory that is dynamically allocated during process run time.

| ASPECT | PROGRAM | PROCESS |
|---|---|---|
| Definition | A program is a set of instructions written in a programming language that defines a specific task or functionality. | A process is an instance of a program currently being executed by the operating system. It includes program code, data, and system resources. |

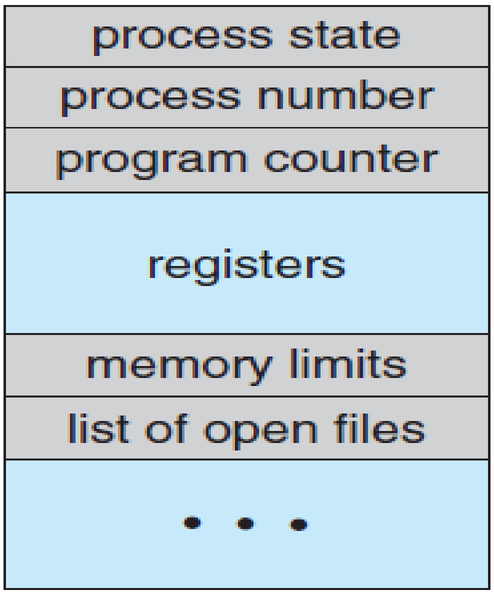
| | | |
|---|---|---|
| Nature | Static | Dynamic |
| Lifecycle | Fixed | Dynamic |
| Storage | Exists as a file on disk or in memory. | Exists in memory while running. |
| State | Inactive (until executed). | Active (running) or inactive (terminated). |
| Execution | Not directly executable by the computer. Needs an interpreter or compiler. | Executable by the computer's CPU. |
| Resources | Does not consume system resources while not running. | Consumes system resources during execution. |
| Interaction | Interacts with users or other programs through input/output operations. | Interacts with the operating system and other processes through system calls. |
| Creation | Created by a programmer or developer. | Created by the operating system when the program is executed. |
| Termination | Controlled by the user or the program. | Controlled by the operating system or can complete its execution naturally. |
| Parallel Execution | Multiple instances can run in parallel if started separately. | Multiple processes can run concurrently, utilizing multiple CPU cores. |
| Relationship | Can be a part of a larger software system or application. | A single program can give rise to multiple processes, each independent. |
| Examples | Microsoft Word, Photoshop, Web Browsers. | Instances of a web browser, background system tasks, printing documents. |

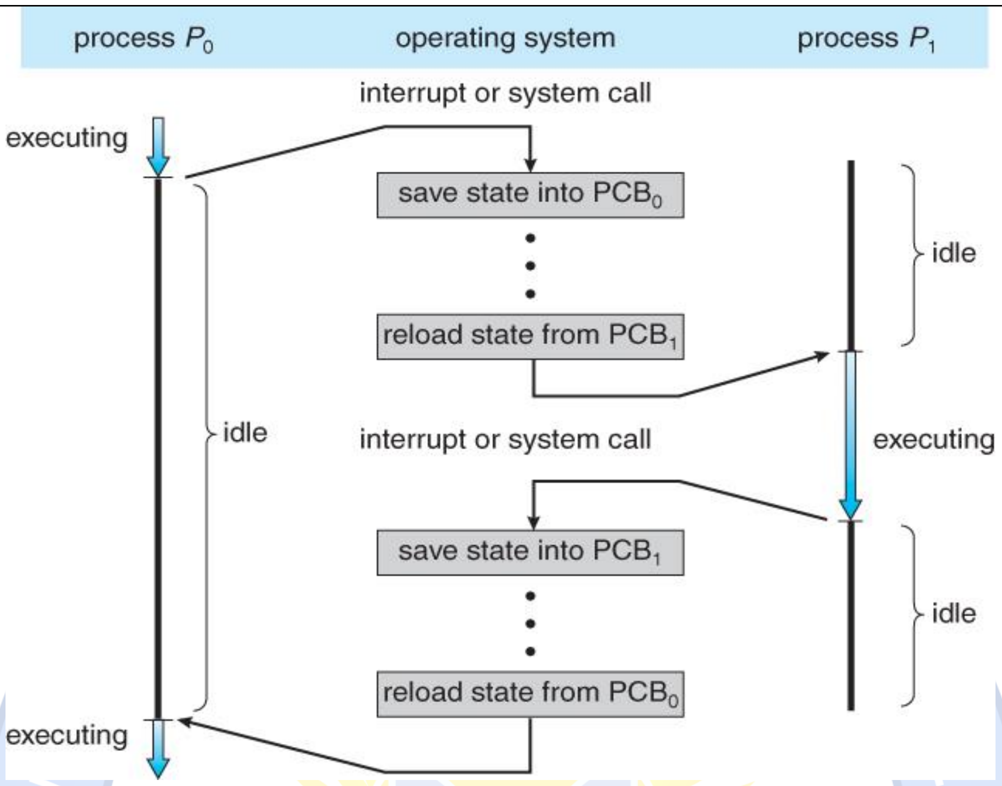
**Q** Explain different states of a process with a suitable diagram.



- **New**.Theprocessisbeingcreated.

- **Running**.Instructionsarebeingexecuted.
- **Waiting**. The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
- **Ready**.Theprocessiswaitingtobeassignedtoaprocessor.

| | | |
|---|---|---|
| | • **Terminated**.Theprocesshasfinishedexecution. | |
| **Q** | Write different process states. What is context switching? What are the contents of the process control block? | |
| |  process state<br>process number<br>program counter<br>registers<br>memory limits<br>list of open files<br>• • •<br>• **Process state**. The state may be new, ready, running, waiting, halted, and so on.<br>• **Programcounter**.Thecounterindicatestheaddressofthenextinstruction to be executed for this process.<br>• **CPU registers**. The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code information. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward<br>• **CPU-schedulinginformation**.Thisinformationincludesaprocesspriority, pointers to scheduling queues, and any other scheduling parameters.<br>• **Memory-management information**. This information may include such items as the value of the base and limit registers and the page tables, or the segment tables, depending on the memory system used by the operating system. | |

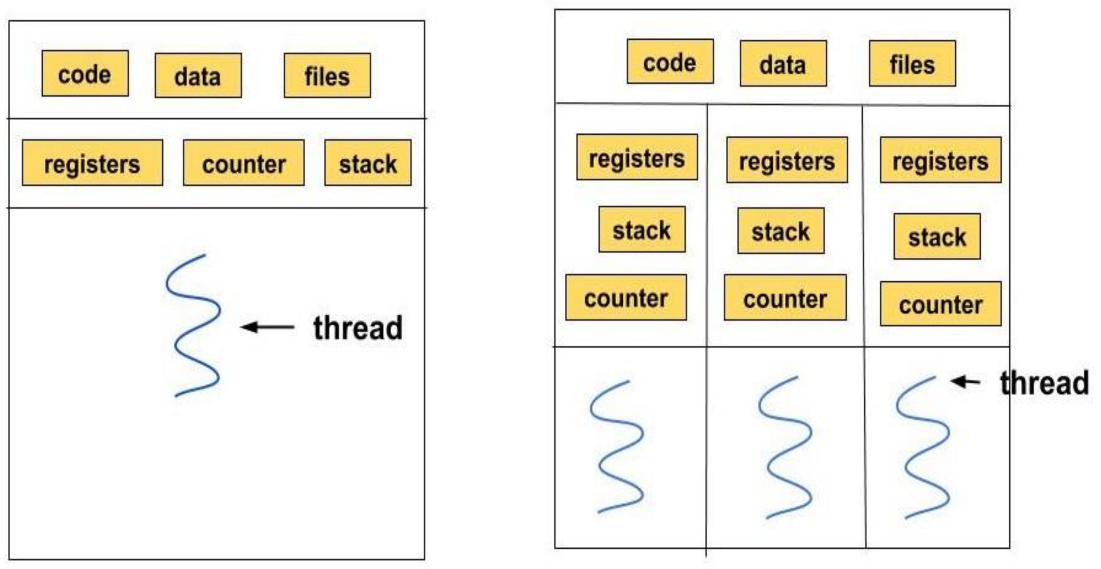


• **Accounting information**. This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.

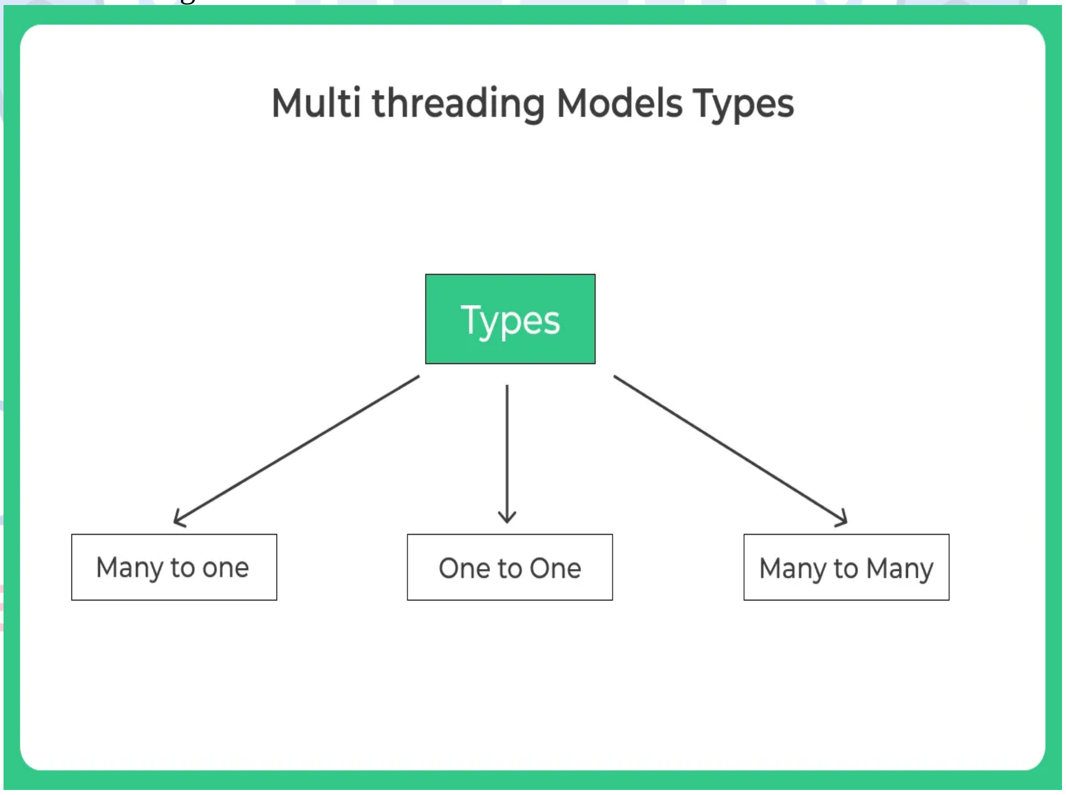
• **I/O status information**. This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

**TOPIC 2: Thread**

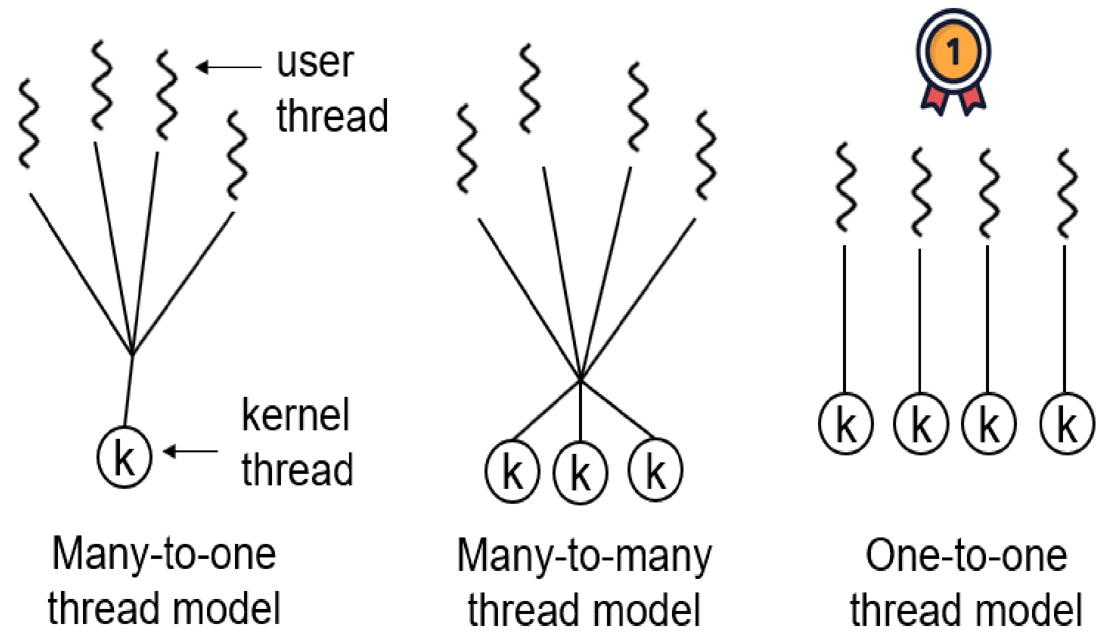
- A thread is a basic unit of CPU utilization; it comprises a thread ID, a program counter, a register set, and a stack .
- It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.
- A traditional (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.
- Threads are also known as light-weight processes.



- Single-threaded process Multi-threaded process
- A single-threaded process is a process with a single thread. A multi-threaded process is a process with multiple threads. As the diagram clearly shows that the multiple threads in it have its own registers, stack, and counter but they share the code and data segment.
- Multithreading Models



-



- 
- **Many To One Model**
- Many User Level Threads are mapped to one Kernel level Thread.
- Does Not support parallel processing of threads in Multicore systems, Because only 1 thread can access kernel at a particular time.
- If 1 thread makes a blocking call, then entire process will stop.
- This is rarely used, mainly cause it does not support Multicore system.

- **One To One Model**
- Each of User thread is mapped to kernel thread.
- Provides more concurrency.
- Whole process does not gets blocked when 1 thread makes a blocking call as other threads can run.
- Creates Overhead as every User thread requires to create the corresponding Kernel Thread for it.
- Supports a particular number of thread only, cause of the overhead caused due to creation of Kernel Level Thread.
- Implemented By Linux and Windows Operating System.

- **Many to Many Model**
- Multiple User Threads are multiplexed to same or less number of Kernel threads.
- This is the best among the three models, as it improves on shortcoming of one to one model and many to one model.
- We can create as number of User Threads as we want, unlike One to One Model.
- There is no issue of process blocking and threads can run parallelly in multiprocess system unlike Many To One Model.

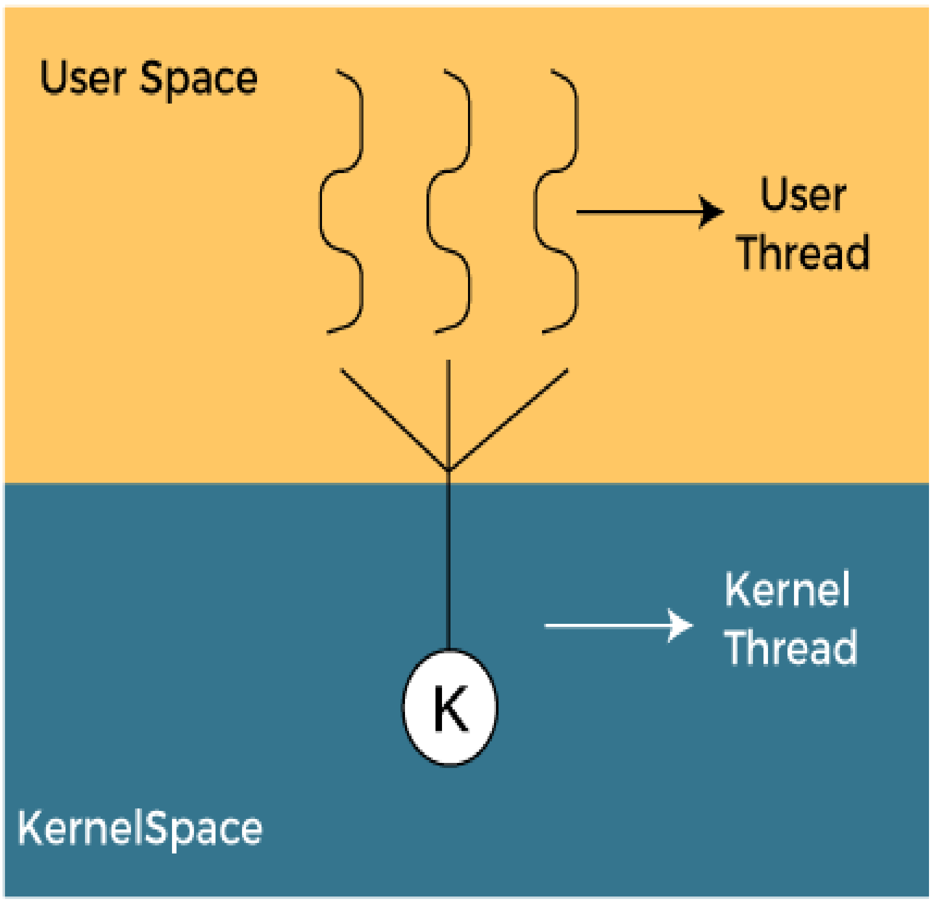
- Types of Thread

- **User-Level Thread**

- The user-level threads are managed by users and the kernel is not aware of it.
- These threads are faster to create and manage.
- The kernel manages them as if it was a single-threaded process.
- It is implemented using user-level libraries and not by system calls. So, no call to the operating system is made when a thread switches the context.
- Each process has its own private thread table to keep the track of the threads.

- **Kernel-Level Thread**

- The kernel knows about the thread and is supported by the OS.
- The threads are created and implemented using system calls.
- The thread table is not present here for each process. The kernel has a thread table to keep the track of all the threads present in the system.
- Kernel-level threads are slower to create and manage as compared to user-level threads.



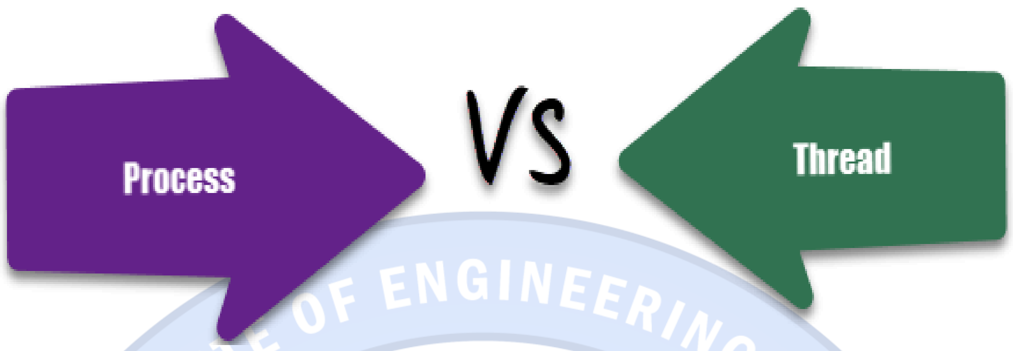
○

| USER LEVEL THREAD | KERNEL LEVEL THREAD |
|---|---|
| User thread are implemented by users. | kernel threads are implemented by OS |
| OS doesn't recognized user level threads. Implementation of User threads is easy. | Kernel threads are recognized by OS.Implementation of Kernel thread is complicated. |
| Context switch time is less. | Context switch time is more. |
| Context switch requires no hardware support. | Hardware support is needed. |
| If one user level thread perform blocking operation then entire process will be blocked. | If one kernel thread perform blocking operation then another thread can continue execution. |
| Example: Java thread, POSIX threads. | Example: Window Solaris. |

- Advantages of threads

- **Performance:** Threads improve the overall performance(throughput, computational speed, responsiveness) of a program.
- **Resource sharing:** As the threads can share the memory and resources of any process it allows any application to perform multiple activities inside the same address space.
- **Utilization of Multiple Processor Architecture:** The different threads can run parallel on the multiple processors hence, this enables the utilization of the processor to a large extent and efficiency.
- **Reduced Context Switching Time:** The threads minimize the context switching time as in Thread Switching, the virtual memory space remains the same.
- **Concurrency:** Thread provides concurrency within a process.
- **Parallelism:** Parallel programming techniques are easier to implement.

- Difference between Process and Thread

| | | |
|---|---|---|
| |  | |

Process
VS
Thread

| Process | Thread |
|---|---|
| A process is an instance of a program that is being executed or processed. | Thread is a segment of a process or a lightweight process that is managed by the scheduler independently. |
| Processes are independent of each other and hence don't share a memory or other resources. | Threads are interdependent and share memory. |
| Each process is treated as a new process by the operating system. | The operating system takes all the User-level threads as a single process. |
| If one process gets blocked by the operating system, then the other process can continue the execution. | If any user-level thread gets blocked, all of its peer threads also get blocked because OS takes all of them as a Single process. |
| Context switching between two processes takes much time as they are heavy compared to thread. | Context switching between the threads is fast because they are very Lightweight. |
| The data segment and code segment of each process are independent of the other. | Threads share data segment and code segment with their peer threads;Hence are the same for other threads also. |
| The operating system takes more time to terminate a process. | Threads can be terminated in very little time. |
| New process creation is more time taking as each new process takes all the resources. | A thread needs less time for creation. |

## TOPIC:3: Uni-processor Scheduling: Types of scheduling: Preemptive, Non preemptive

**Q** CPU scheduling algorithm:

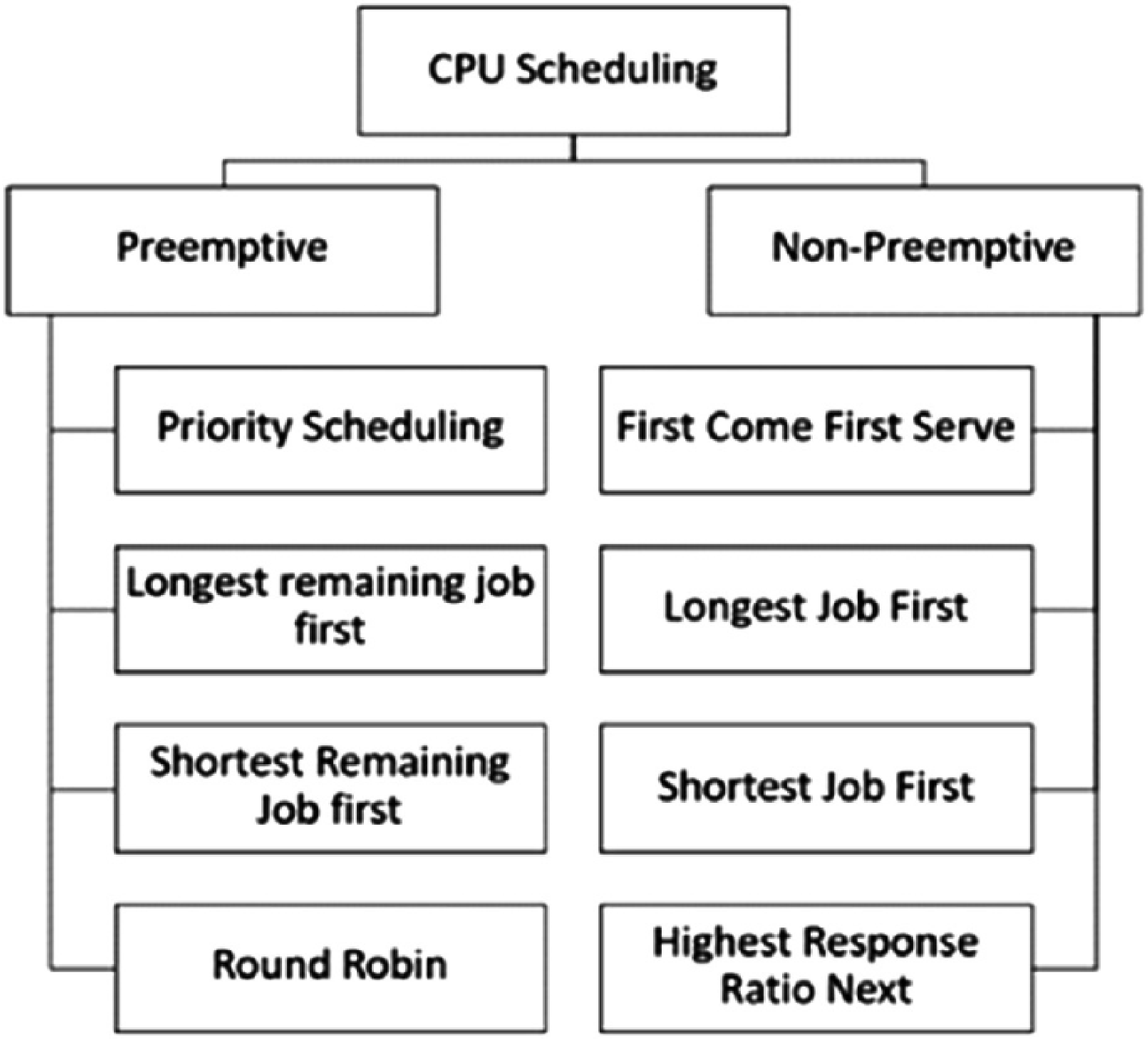
- **CPU scheduling** is the process of deciding which process will own the CPU to use while another process is suspended. The main function of the CPU scheduling is to ensure that whenever the CPU remains idle, the OS has at least selected one of the processes available in the ready-to-use line.
- In Multiprogramming, if the long-term scheduler selects multiple I / O binding processes then most of the time, the CPU remains an idle. The function of an effective program is to improve resource utilization.
- If most operating systems change their status from performance to waiting then there may always be a chance of failure in the system. So in order to minimize this excess, the OS needs to schedule tasks in order to make full use of the CPU and avoid the possibility of deadlock.

Things to take care while designing a CPU Scheduling algorithm

- **CPU utilization:** The main purpose of any CPU algorithm is to keep the CPU as busy as possible. Theoretically, CPU usage can range from 0 to 100 but in a real-time system, it varies from 40 to 90 percent depending on the system load.
- **Throughput:** The average CPU performance is the number of processes performed and completed during each unit. This is called throughput. The output may vary depending on the length or duration of the processes.
- **Turn round Time:** For a particular process, the important conditions are how long it takes to perform that process. The time elapsed from the time of process delivery to the time of completion is known as the conversion time. Conversion time is the amount of time spent waiting for memory access, waiting in line, using CPU, and waiting for I / O.
- **Waiting Time:** The Scheduling algorithm does not affect the time required to complete the process once it has started performing. It only affects the waiting time of the process i.e. the time spent in the waiting process in the ready queue.
- **Response Time:** In a collaborative system, turn around time is not the best option. The process may produce something early and continue to computing the new results while the previous results are released to the user. Therefore another method is the time taken in the submission of the application process until the first response is issued. This measure is called response time.

Types of CPU Scheduling Algorithms:

- **Preemptive** scheduling is used when a process switches from running state to ready state or from the waiting state to the ready state.
- **Non-Preemptive** scheduling is used when a process terminates , or when a process switches from running state to waiting state.



## TOPIC 4: System calls

**Q** What is system call in OS? Explain fork () system call to create new process in UNIX OS.

System Call: System call is a way in which a program requests a service from the kernel. It is a way to for user program interact with operating system. Example of system calls are: open, read, write, close, wait, exec, fork, exit, and kill fork() System Call:

• fork() system call is used to create new process

• During execution of process, it may require to create several new processes for different task

• E.g. Suppose user wants to print file then operating system creates a new process to handle task of printing. o Here one process is already running (MS word - .doc file) and new process is created for printing

• Process which is creating new process is called parent process and new process is known as child process.

• When process wants to create new process, it creates child process using fork () system call.

• Child process is same as parent process but process id (PID) will be different

| Process | Windows | Unix |
|---|---|---|
| **Process Control** | CreateProcess()<br>ExitProcess()<br>WaitForSingleObject() | Fork()<br>Exit()<br>Wait() |
| **File Manipulation** | CreateFile()<br>ReadFile()<br>WriteFile()<br>CloseHandle() | Open()<br>Read()<br>Write()<br>Close() |
| **Device Management** | SetConsoleMode()<br>ReadConsole()<br>WriteConsole() | Ioctl()<br>Read()<br>Write() |
| **Information Maintenance** | GetCurrentProcessID()<br>SetTimer()<br>Sleep() | Getpid()<br>Alarm()<br>Sleep() |
| **Communication** | CreatePipe()<br>CreateFileMapping()<br>MapViewOfFile() | Pipe()<br>Shmget()<br>Mmap() |
| **Protection** | SetFileSecurity()<br>InitializeSecurityDescriptor()<br>SetSecurityDescriptorgroup() | Chmod()<br>Umask()<br>Chown() |

**exec():**

When an executable file replaces an earlier executable file in an already executing process, this system function is invoked. As a new process is not built, the old process identification stays, but the new process replaces data, stack, data, head, etc.

**Wait():**

In some systems, a process may have to wait for another process to complete its execution before proceeding. When a parent process makes a child process, the parent process execution is suspended until the child process is finished. The **wait()** system call is used to suspend the parent process. Once the child process has completed its execution, control is returned to the parent process.

**Ps () :**

The ps (i.e., process status) command is used to provide information about the currently running processes, including their process identification numbers (PIDs). A process, also referred to as a task, is an executing (i.e., running) instance of a program. Every process is assigned a unique PID by the system. The basic syntax of ps is ps [options] When ps is used without any options, it sends to standard output, which is the display monitor by default, four items of information for at least two processes currently on the system: the shell and ps.

**Join ():**

The join command in UNIX is a command line utility for joining lines of two files on a common field.
Syntax : $join[option] file1 file 2

File1.text:
1.name
2.branch
3.subject
File2.text:
1.ram
2.cse
3.os

Output:

```
$join file1 file2.text
1.name ram
2.branch cse
3.subject os
```

| | | |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| 7 | | |
| | | |