CHAPTER 2 CPU Scheduling

* Process Scheduling: It is done by process manage that handles the removal of the running process from the CPU and the selection of another process based on a particular strategy.

Categories of Scheduling

Preemptive	Non-Preemptive
Priority Scheduling	Shortest Job First (SJF)
Long Remaining Job First(LRJF)	Longest Job First (LJF)
Shortest " " "(SRTF)	First Come First Serve (FCFS)
Round Robin (RR)	Highest Response Ratio Next(HRRN)

* Types of Process Schedulers

1. Long Term/Job Scheduler: It brings the new process to the 'Ready State'. It controls the Degree of Multiprogramming i.e., the number of processes present in a ready state at any point in time.
2. Short Term/CPU Scheduler: It is responsible for selecting one process from the ready state for scheduling it on the running state.
Note: It only select the process to schedule it and doesn't load process on running. Here when all the scheduling algorithms are used. It ensure no starvation due to high burst time processes.
Dispatcher: It is responsible for loading the process selected by the short-term scheduler on the CPU (Ready to Running State). Context switching is done by the dispatcher only.
The dispatcher does the following: Context Switching, switching to user mode, jump to the proper location in the newly loaded program.
3. Medium-Term Scheduler: It is responsible for suspending and resuming the process. It mainly does swapping (moving processes from main memory to disk and vice versa.) It is helpful in maintaining a perfect balance between the I/O bound and the CPU bound.

Note: Long-term Scheduler can increase degree of multiprogramming.
Medium-term Scheduler can inc/dec degree of multiprogramming.
Short-term Scheduler does not change degree of multiprogramming.

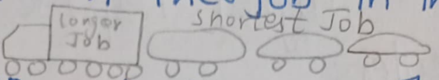
* Scheduling Times

1. Arrival Time: Time at which the process arrives in the ready queue.
2. Burst/Service Time: Amount of time a process runs on CPU.
3. Waiting Time: Amount of time a process waits in Ready State.
Waiting Time = Turn Arount Time − Burst Time

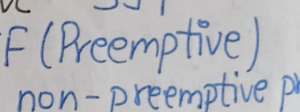
4. Completion Time : Time at which process completes its execution.
5. Turn Around Time : Amount of time process spends from arrival to completion.
Turn Around Time = Completion Time − Arrival Time
6. Response Time : Amount of time from arrival till first time execution of process. WT and RT are equal for non-preemptive.
7. Scheduling Length (L) : $Max(CT_i) - Min(AT_j)$
8. Throughput : Number of processes executed per unit time.

$$\text{Throughput} = \frac{\text{no. of process executed}}{\text{Scheduling Length}}$$

* Convoy Effect : If the CPU gets the processes of the higher burst time at the front end of the ready queue then the processes of lower burst time may get blocked which means they may never get the CPU if the job in the execution has a very high burst time.



For non-preemptive algorithm : no. of context switches = no. of processes − 1
* Aging : It is used to prevent starvation of process. It gradually increase the priority of processes that have been waiting for a long time. It increases the chance of getting the necessary resources to execute. This reduces the risk of starvation.
* Basis of Analysis of Scheduling Algorithms
Minimum average WT & TAT among non-preemptive SJF
Minimum average WT among all algorithms SRTF (Preemptive)
Non-preemptive always FCFS, SJF, LJF, HRRN, non-preemptive priority
Preemptive always —
SRTF behaves as SJF ① When all processes arrive together.
② When later arriving processes are not smaller than running process.
Preemptive Priority behaves as non-preemptive ① When all processes arrive together. ② When later arriving processes have lower priority.
RR behaves as non-preemptive Max (BT of all processes) ≤ Q
Convoy Effect FCFS, LJF, LRTF
Starvation SJF, SRTF, Priority algo both type, LJF, LRTF
When LRTF is non-preemptive When all processes have BT = 1.
* CPU Utilization
1. From gantt chart = $\frac{\text{Time unit for which CPU used}}{\text{Total execution time}}$



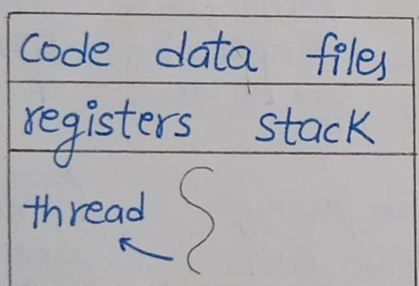
2. With IO Operations
for n number of processes having I/O requirement = P
CPU utilization = $1 - P^n$

***Threads**: A thread is the smallest unit of processing that can be scheduled and executed by the CPU. A process can contain multiple threads. Threads are also called lightweight processes as they possess some of the properties of processes. Each thread belongs to exactly one process. Threads run in parallel improving the application performance. Priority can be assigned to the threads just like process. Each thread has its own Thread Control Block (TCB). Just like process a context switch occurs for the thread.

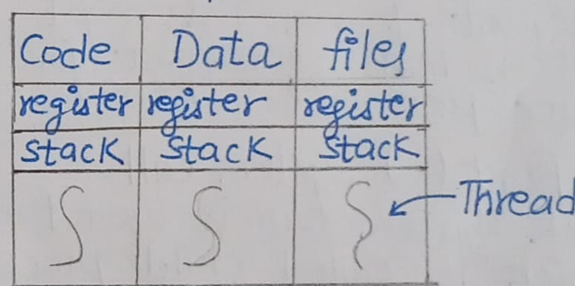
Shared Among Threads	Unique for Each Thread
Code Section	Thread id
Data Section	Register Set
OS Resources	Stack
Open files & Signals	Program Counter
Heap	

*** Single Threaded Process** contain the execution of instructions in a single sequence. In other words, one command is processes at a time.

*** Multi Threaded Process** allows multiple threads to share the same resources of a single process, allow the execution of multiple parts of a program at the same time. Ex: CPU, memory & I/O devices. In browser, multiple tabs can be different threads. In MS word, one thread to format the text, another thread to process inputs etc.

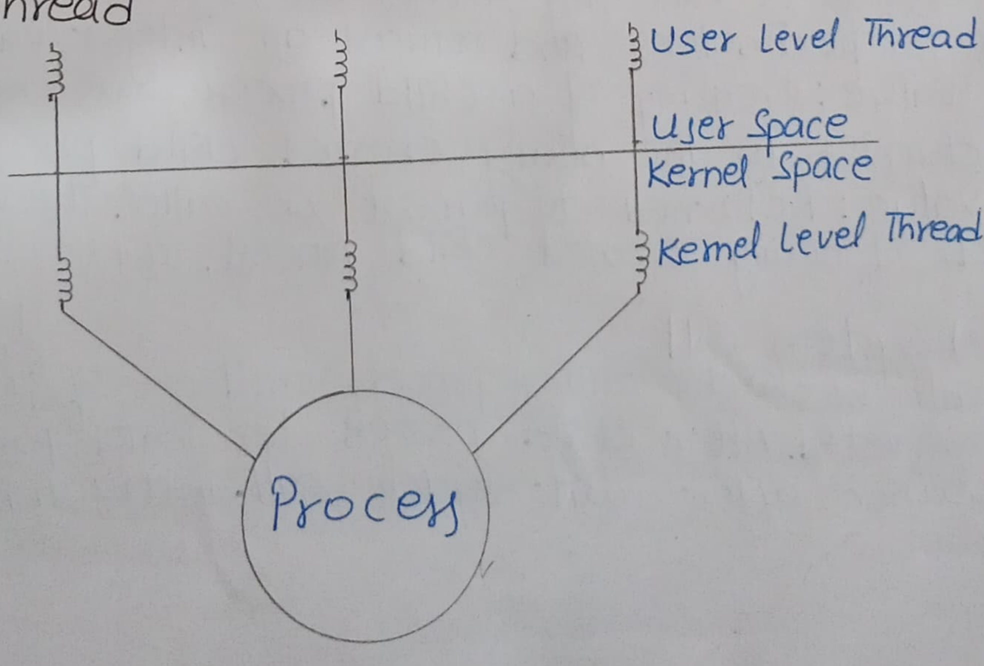


Single Threaded Process



Multi-Threaded Process

*** Types of Thread**



1. User Level Threads: It is a type of thread that is not created using system calls. It is implemented by the user. The Kernel has no work in the management of user-level threads.

2. Kernel Level Threads: It is directly handled by OS. It does not have any information about user-level threads. Context Switch is slow.

User Threads	Kernel Thread
Multithreading in user process. Created without kernel intervation.	Multithreading in Kernel process. Kernel itself is multi-threaded.
Context Switch is very fast.	Context Switch is Slow.
If one thread is blocked, OS blocks entire process.	Individual thread can be blocked
Generic and can run on any OS.	Specific to OS
Faster to Create & manage.	Slower to create and manage.

* System Call: Programmatic way in which a computer program requests a service from kernel.

How System Call Works

Request of process from user space → Interrupt request for Kernel. → Control transfer to Kernel → Kernel checks if operation can be done → Kernel performs operation ↓ Kernel return the output to user process.

* Fork( ) System Call

Fork system call is used for creating a new process, which is called child process. Child process runs concurrently with the process that makes the fork( ) call (parent process). Child process starts executing after the fork( ) call while created it. It takes no perimeters and returns an integer value.

Negative value: Creating of a child process was unsuccessful.

Zero: Returned to the newly created child process.

Positive Value: Returned to parent or caller. The value contains process ID of newly created child process.

Total no. of processes = 2^n

1 Parent | $2^n - 1$ (Child

* Wait( ) System Call

A wait call blocks the calling process until one of its child process completed. After child process terminates parent continues its execution after wait system call instruction.