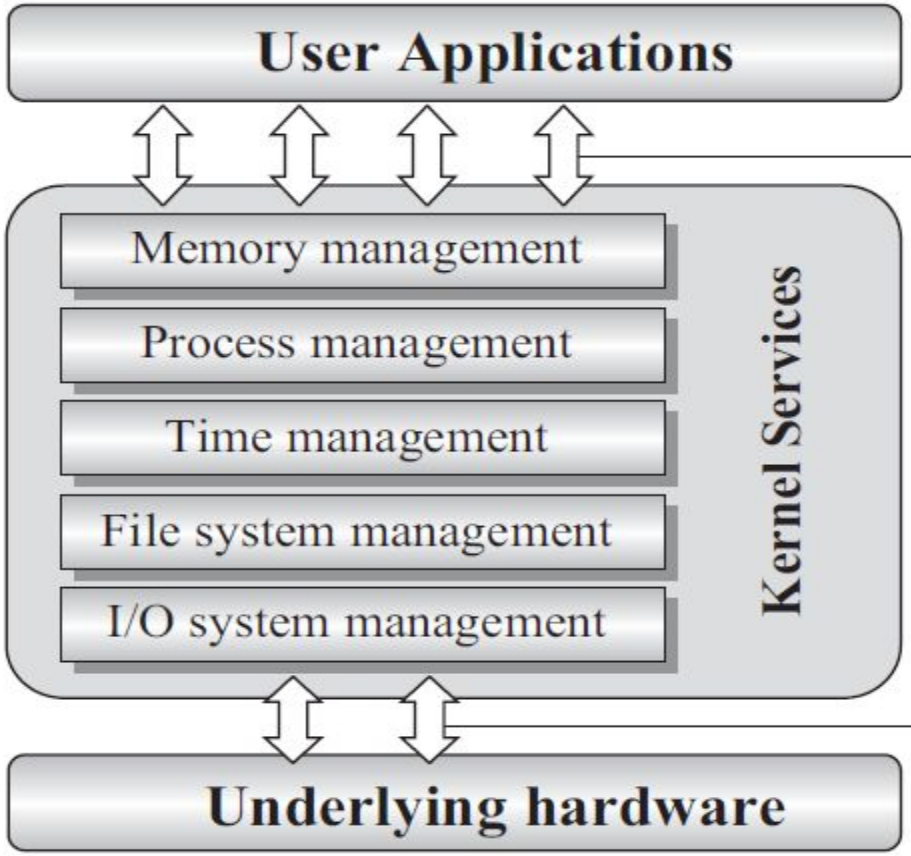
EC342 Embedded Systems

Dr. Sajesh Kumar U.

Operating System Architecture



Application programming interface (API)

Device driver interface

Secondary storage management

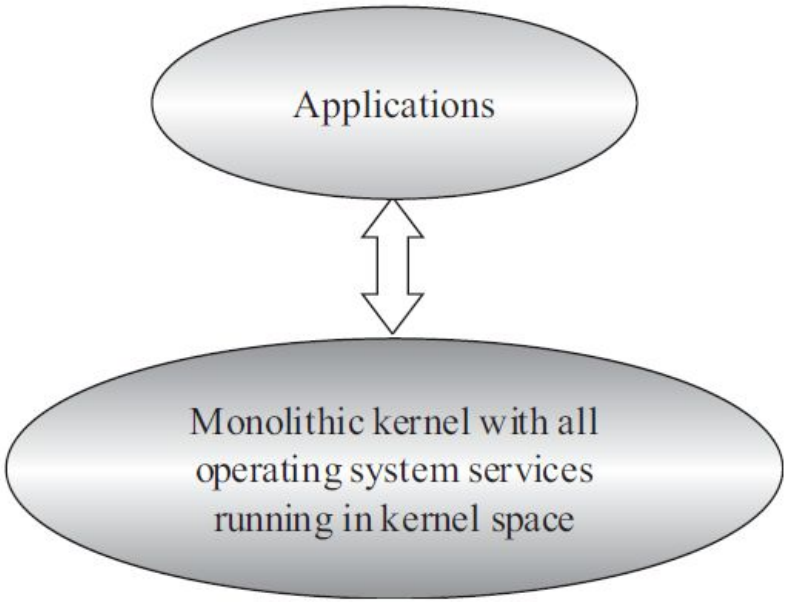
Interrupt Handler

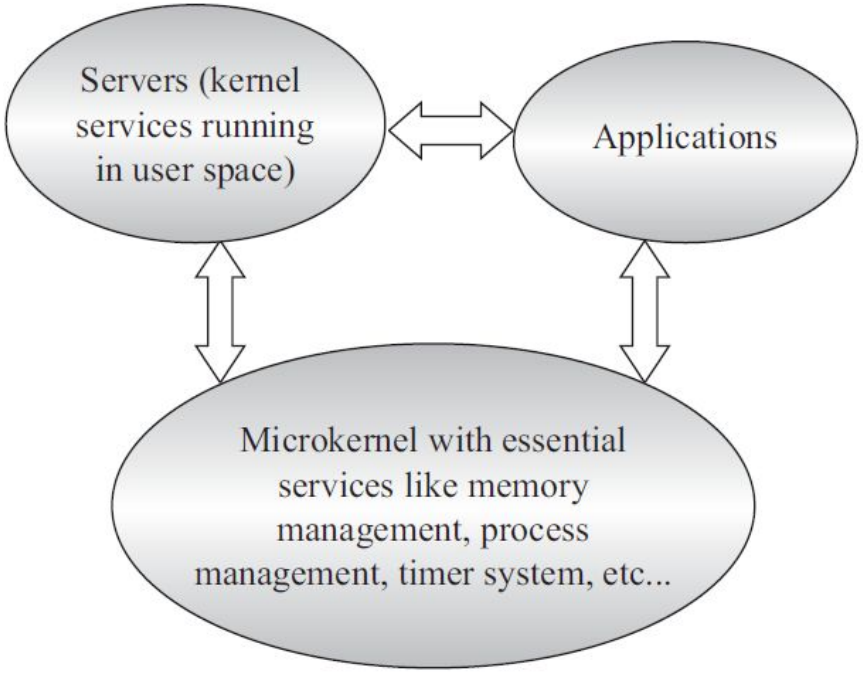
Protection systems

Kernal Space

User Space

Monolithic Kernal and Micro Kernal





Micro Kernal advantages

Robustness--- Error leads to reconfiguration of server

Configurability—Server services can be reconfigured

Types of Operating Systems

- **General Purpose Operating System**
- **Real Time Operating System**

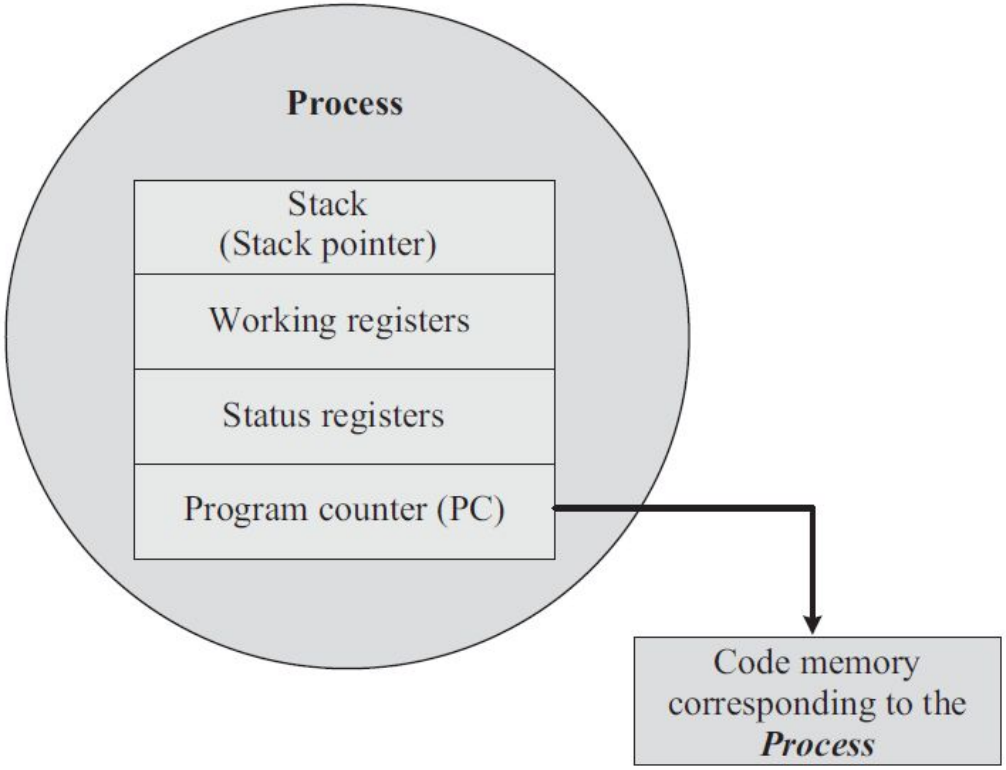
Real Time Kernal has the following services

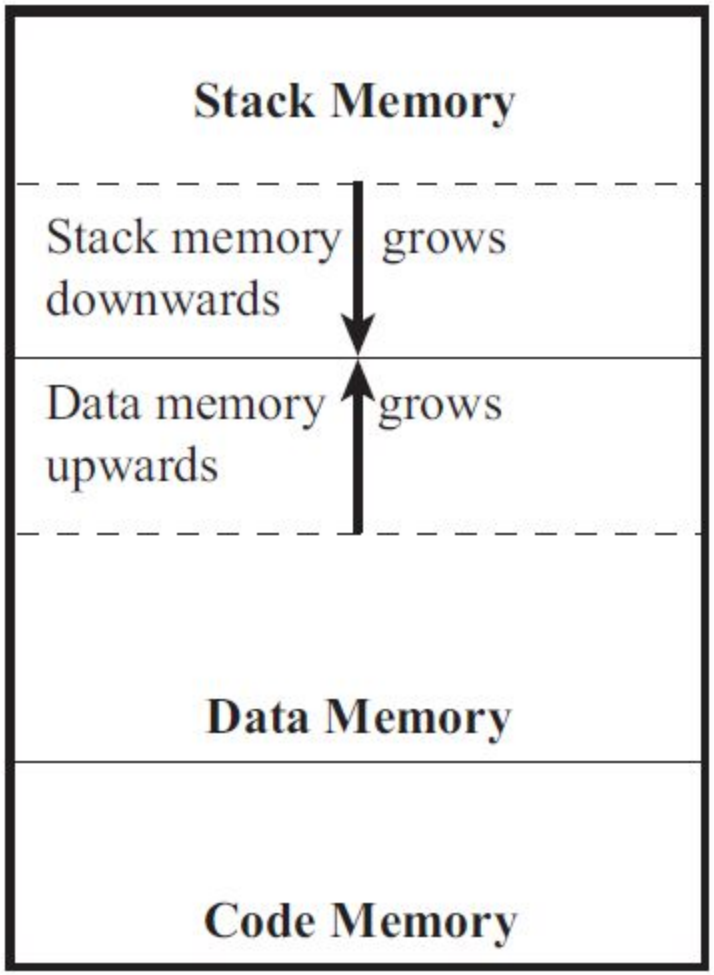
- Task/Process management –TCB for task control
- Task/Process scheduling—Priority scheduling
- Task/Process Synchronization—concurrent access of the resources
- Error/Exception handling—errors, deadlock, timeout
- Memory management---block based instead of Dynamic Memory Allocation
- Interrupt Handling---synchronous, Asynchronous
- Time management—Timer interrupt and Timer tick

Hard Real Time and Soft Real Time

- Hard real time---must meet deadlines, automatic, no human in the middle loop
- Examples are Airbag deployment, Aero plane navigation
- Soft real time----meeting the deadline not mandatory, can be human in the middle
- Examples are ATM, vending machines

Tasks, Processes and Threads





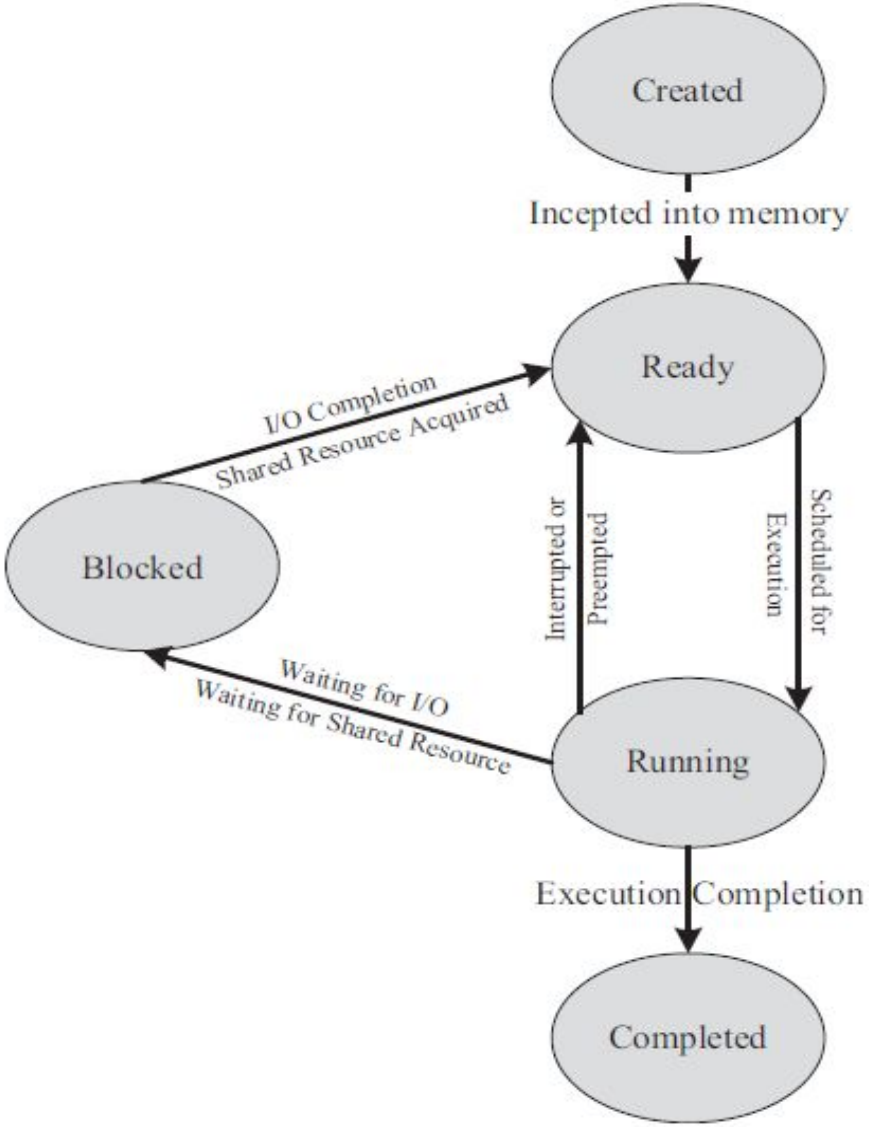
Process is a program or part of it in execution

Process is a part of task in execution

Process States

Process management includes

- Creating a process
- Setting up the memory space
- Loading the process code into memory
- Allocating system resources
- Setting up process control block for process, its termination/deletion

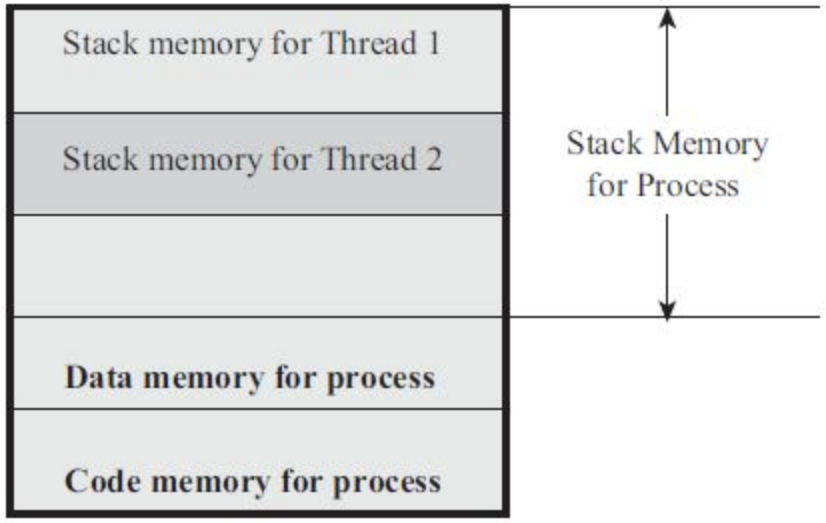


Threads

- Process is divided into multiple threads
- Thread is a primitive that can execute a code
- Single sequential flow within a process

Multithreading

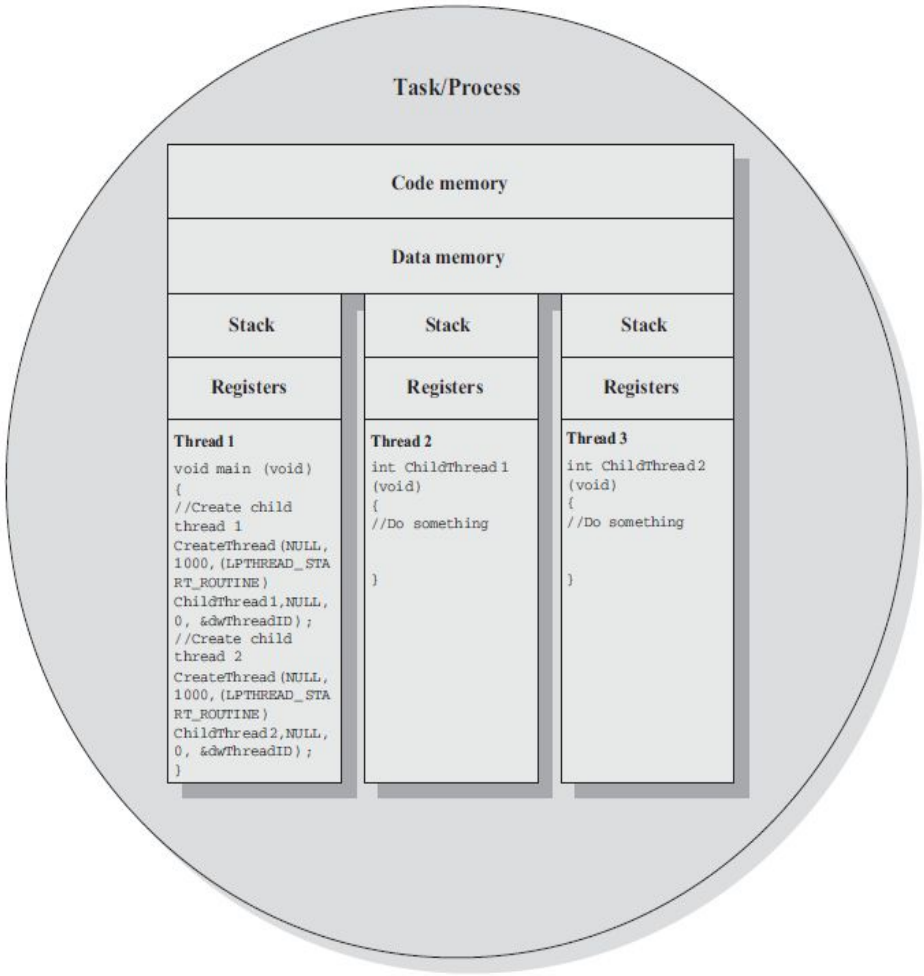
- Better memory utilization as data memory are shared between threads
- Speeds up the execution of a process---- if a thread is entered into wait state, another thread can run
- Effective CPU utilization



Process with multi threads

Thread pre emption

- User level thread
- Kernal level thread
- Many to one model
- One to one model
- Many to many model



Thread and Process

Thread	Process
Thread is a single unit of execution and is part of process.	Process is a program in execution and contains one or more threads.
A thread does not have its own data memory and heap memory. It shares the data memory and heap memory with other threads of the same process.	Process has its own code memory, data memory and stack memory.
A thread cannot live independently; it lives within the process.	A process contains at least one thread.
There can be multiple threads in a process. The first thread (main thread) calls the main function and occupies the start of the stack memory of the process.	Threads within a process share the code, data and heap memory. Each thread holds separate memory area for stack (shares the total stack memory of the process).
Threads are very inexpensive to create	Processes are very expensive to create. Involves many OS overhead.
Context switching is inexpensive and fast	Context switching is complex and involves lot of OS overhead and is comparatively slower.
If a thread expires, its stack is reclaimed by the process.	If a process dies, the resources allocated to it are reclaimed by the OS and all the associated threads of the process also dies.

Multi processing and Multi Tasking

- In Multi processing multiple processes are executed simultaneously using multiple processors
- In Multi tasking multiple processes are executed using single processor by task switching
- Virtual processor, Context Switching (saving and retrieval)
- Task Scheduler

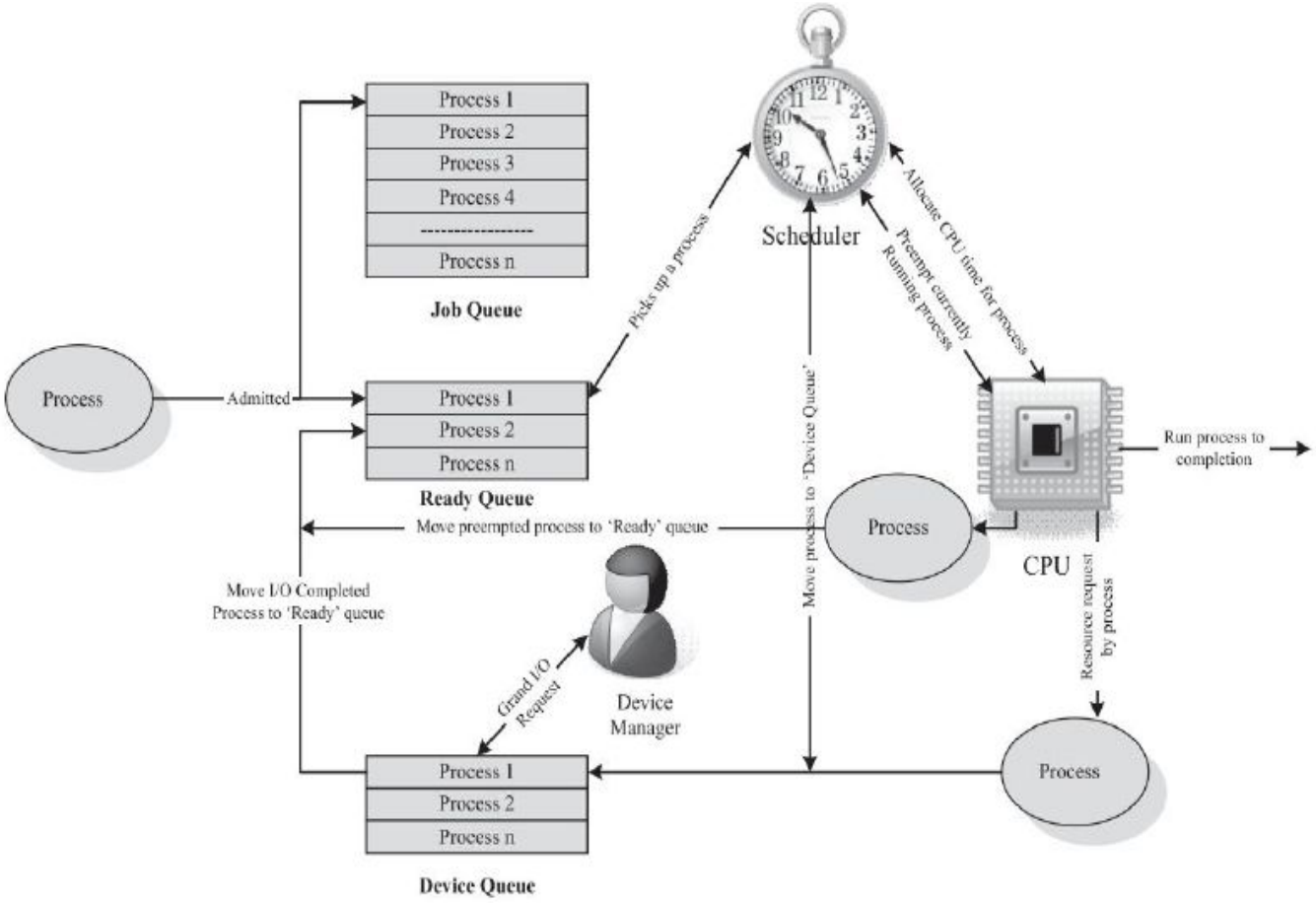
Types of Multitasking

- **Co operative**--- Any task can hold the CPU as much time as it wants
- **Pre emptive**--- Every task gets a chance to execute. Current task is pre empted to give chance to other tasks
- **Non Pre emptive**– Task is switched only when the current task enters into blocked/wait state or is waiting for I/O or system resource.
- Difference between cooperative and non pre emptive is that, in cooperative the task need not relinquish CPU when it is in wait/blocked state or waiting for resources

Task Scheduling

- **Non Pre emptive Scheduling**
- **Pre emptive Scheduling**

CPU utilization
Throughput
Turn around time
Waiting time
Response time



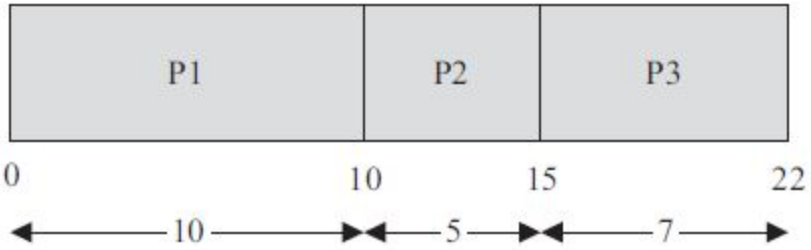
Non Pre emptive scheduling

- First come first served/FIFO scheduling
- Last come first served/LIFO scheduling
- Shortest Job first scheduling
- Priority based scheduling

Pre emptive scheduling

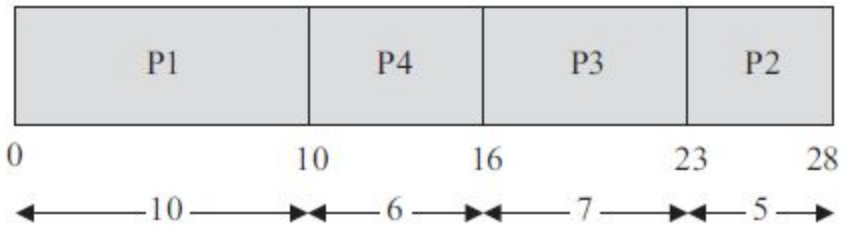
- Pre emptive SJF scheduling/ Shortest Remaining Time
- Round Robin Scheduling
- Priority based scheduling

First come first served/FIFO scheduling



Waiting Time for P1 = 0 ms (P1 starts executing first)
Waiting Time for P2 = 10 ms (P2 starts executing after completing P1)
Waiting Time for P3 = 15 ms (P3 starts executing after completing P1 and P2)
Average waiting time = (Waiting time for all processes) / No. of Processes
= (Waiting time for (P1+P2+P3)) / 3
= (0+10+15)/3 = 25/3
= 8.33 milliseconds
Turn Around Time (TAT) for P1 = 10 ms (Time spent in Ready Queue + Execution Time)
Turn Around Time (TAT) for P2 = 15 ms (-Do-)
Turn Around Time (TAT) for P3 = 22 ms (-Do-)
Average Turn Around Time = (Turn Around Time for all processes) / No. of Processes
= (Turn Around Time for (P1+P2+P3)) / 3
= (10+15+22)/3 = 47/3
= 15.66 milliseconds
Average Turn Around Time (TAT) is the sum of average waiting time and average execution time.
Average Execution Time = (Execution time for all processes)/No. of processes
= (Execution time for (P1+P2+P3))/3
= (10+5+7)/3 = 22/3
= 7.33
Average Turn Around Time = Average waiting time + Average execution time
= 8.33 + 7.33
= 15.66 milliseconds

Last come first served/LIFO scheduling



The waiting time for all the processes is given as
Waiting Time for P1 = 0 ms (P1 starts executing first)
Waiting Time for P4 = 5 ms (P4 starts executing after completing P1. But P4 arrived after 5 ms of execution of P1. Hence its waiting time = Execution start time – Arrival Time = 10 – 5 = 5)
Waiting Time for P3 = 16 ms (P3 starts executing after completing P1 and P4)
Waiting Time for P2 = 23 ms (P2 starts executing after completing P1, P4 and P3)
Average waiting time = (Waiting time for all processes) / No. of Processes
= (Waiting time for (P1+P4+P3+P2)) / 4
= (0 + 5 + 16 + 23)/4 = 44/4
= 11 milliseconds
Turn Around Time (TAT) for P1 = 10 ms (Time spent in Ready Queue + Execution Time)
Turn Around Time (TAT) for P4 = 11 ms (Time spent in Ready Queue + Execution Time = (Execution Start Time – Arrival Time) + Estimated Execution Time = (10 – 5) + 6 = 5 + 6)
Turn Around Time (TAT) for P3 = 23 ms (Time spent in Ready Queue + Execution Time)
Turn Around Time (TAT) for P2 = 28 ms (Time spent in Ready Queue + Execution Time)
Average Turn Around Time = (Turn Around Time for all processes) / No. of Processes
= (Turn Around Time for (P1+P4+P3+P2)) / 4
= (10+11+23+28)/4 = 72/4
= 18 milliseconds