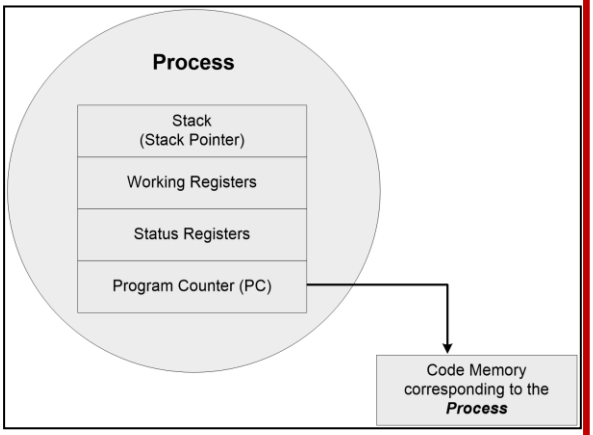
Tasks, Processes & Threads

Tasks, Processes & Job

- In the Operating System context, a task is defined as the program in execution and the related information maintained by the Operating system for the program
- Task is also known as '*Job*' in the operating system context
- A program or part of it in execution is also called a '*Process*'
- The terms '*Task*', '*job*' and '*Process*' refer to the same entity in the Operating System context and most often they are used interchangeably
- A process requires various system resources like CPU for executing the process, memory for storing the code corresponding to the process and associated variables, I/O devices for information exchange etc

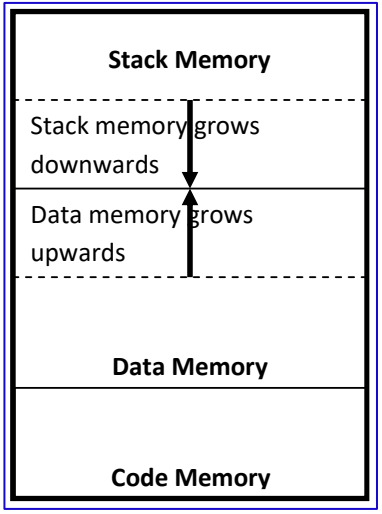
Structure of a Processes

- The concept of '*Process*' leads to concurrent execution (pseudo parallelism) of tasks and thereby the efficient utilization of the CPU and other system resources
- Concurrent execution is achieved through the sharing of CPU among the processes
- A process mimics a processor in properties and holds a set of registers, process status, a Program Counter (PC) to point to the next executable instruction of the process, a stack for holding the local variables associated with the process and the code corresponding to the process
- A process, which inherits all the properties of the CPU, can be considered as a virtual processor, awaiting its turn to have its properties switched into the physical processor
- When the process gets its turn, its registers and Program counter register becomes mapped to the physical registers of the CPU



Memory organization of a Processes

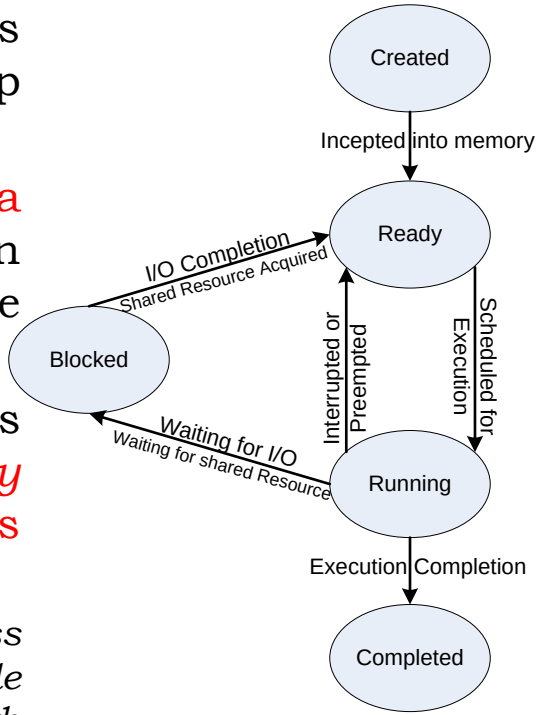
- The memory occupied by the *process* is segregated into three regions namely; Stack memory, Data memory and Code memory
- The 'Stack' memory holds all temporary data such as variables local to the process
- Data memory holds all global data for the process
- The code memory contains the program code (instructions) corresponding to the process
- On loading a process into the main memory, a specific area of memory is allocated for the process
- The stack memory usually starts at the highest memory address from the memory area allocated for the process (Depending on the OS kernel implementation)



Process States & State Transition

- The creation of a process to its termination is not a single step operation.
- The process traverses through a series of states during its transition from the newly created state to the terminated state.
- The cycle through which a process changes its state from '*newly created*' to '*execution completed*' is known as '*Process Life Cycle*'.

The various states through which a process traverses through during a Process Life Cycle indicates the current status of the process with respect to time and also provides information on what it is allowed to do next



Process States & State Transition

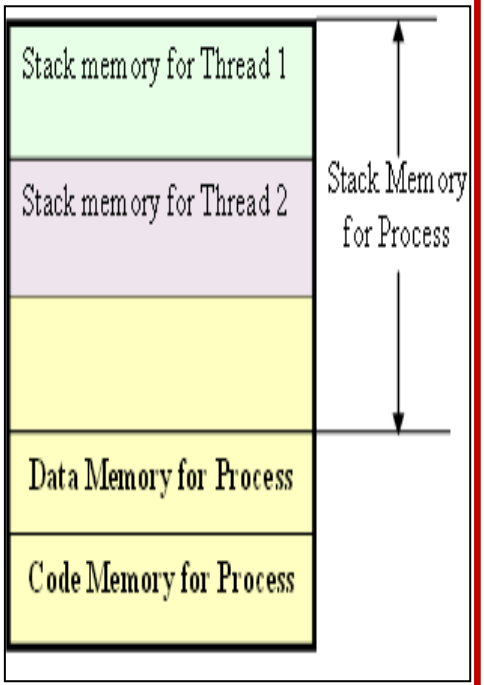
- **Created State:**
  - The state at which a process is being created is referred as 'Created State'.
  - The Operating System recognizes a process in the '*Created State*' but no resources are allocated to the process
- **Ready State:**
  - The state, where a process is incepted into the memory and awaiting the processor time for execution, is known as '*Ready State*'.
  - The process is placed in the '*Ready list*' queue maintained by the OS
- **Running State:**
  - The state where in the source code instructions corresponding to the process is being executed is called '*Running State*'.
  - Running state is the state at which the process execution happens.

Process States & State Transition

- **Blocked State/Wait State:**
  - Refers to a state where a running process is **temporarily suspended** from execution and **does not have immediate access to resources**.
  - The blocked state might have invoked by various conditions like- the process enters a wait state for an event to occur (E.g. Waiting for user inputs such as keyboard input) or waiting for getting access to a shared resource like semaphore, mutex etc
- **Completed State:**
  - A state where the process completes its execution
- The transition of a process from one state to another is known as '*State transition*'
- When a process changes its state from Ready to running or from running to blocked or terminated or from blocked to running, the CPU allocation for the process may also change

Threads

- A *thread* is the primitive that can execute code
- A *thread* is a single sequential flow of control within a process
- '*Thread*' is also known as ***lightweight process***
- A process can have many threads of execution
- Different threads, which are part of a process, share the same address space; meaning they share the data memory, code memory and heap memory area
- Threads maintain their own thread status (CPU register values), Program Counter (PC) and stack

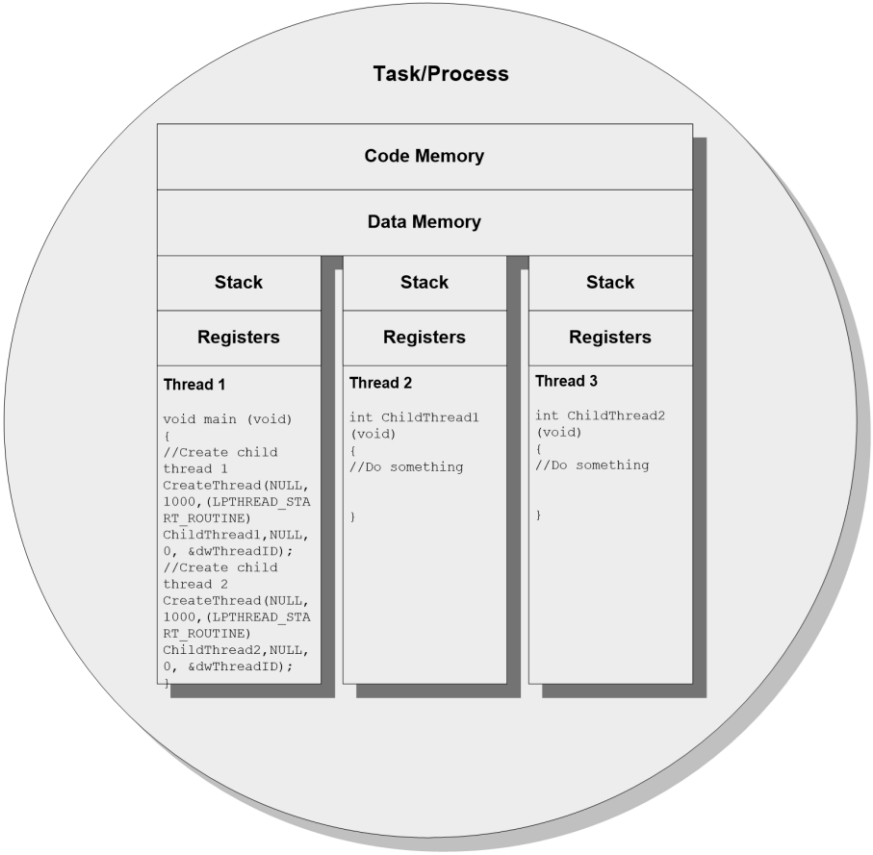


The Concept of multithreading

Use of multiple threads to execute a process brings the following advantage.

- Better memory utilization. Multiple threads of the same process share the address space for data memory. This also reduces the complexity of inter thread communication since variables can be shared across the threads.
- Since the process is split into different threads, when one thread enters a wait state, the CPU can be utilized by other threads of the process that do not require the event, which the other thread is waiting, for processing. This speeds up the execution of the process.
- Efficient CPU utilization. The CPU is engaged all time.

Multithreading



Thread V/s Process

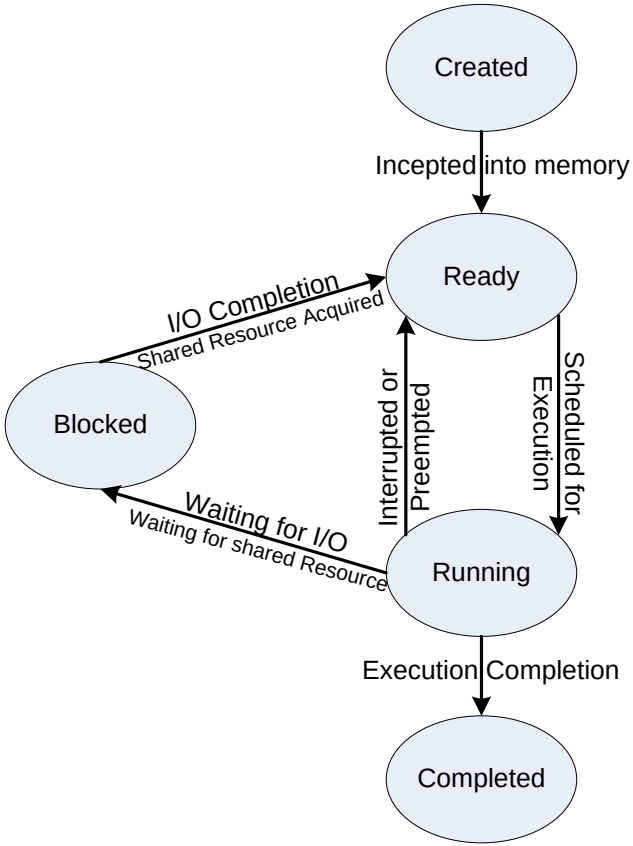
Thread	Process
Thread is a single unit of execution and is part of process.	Process is a program in execution and contains one or more threads.
A thread does not have its own data memory and heap memory. It shares the data memory and heap memory with other threads of the same process.	Process has its 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.

?????

Multiprocessing

Vs

Multitasking


Created
Incepted into memory
Ready
I/O Completion
Shared Resource Acquired
Blocked
Interrupted or
Preempted
Scheduled for
Execution
Waiting for I/O
Waiting for shared Resource
Running
Execution Completion
Completed

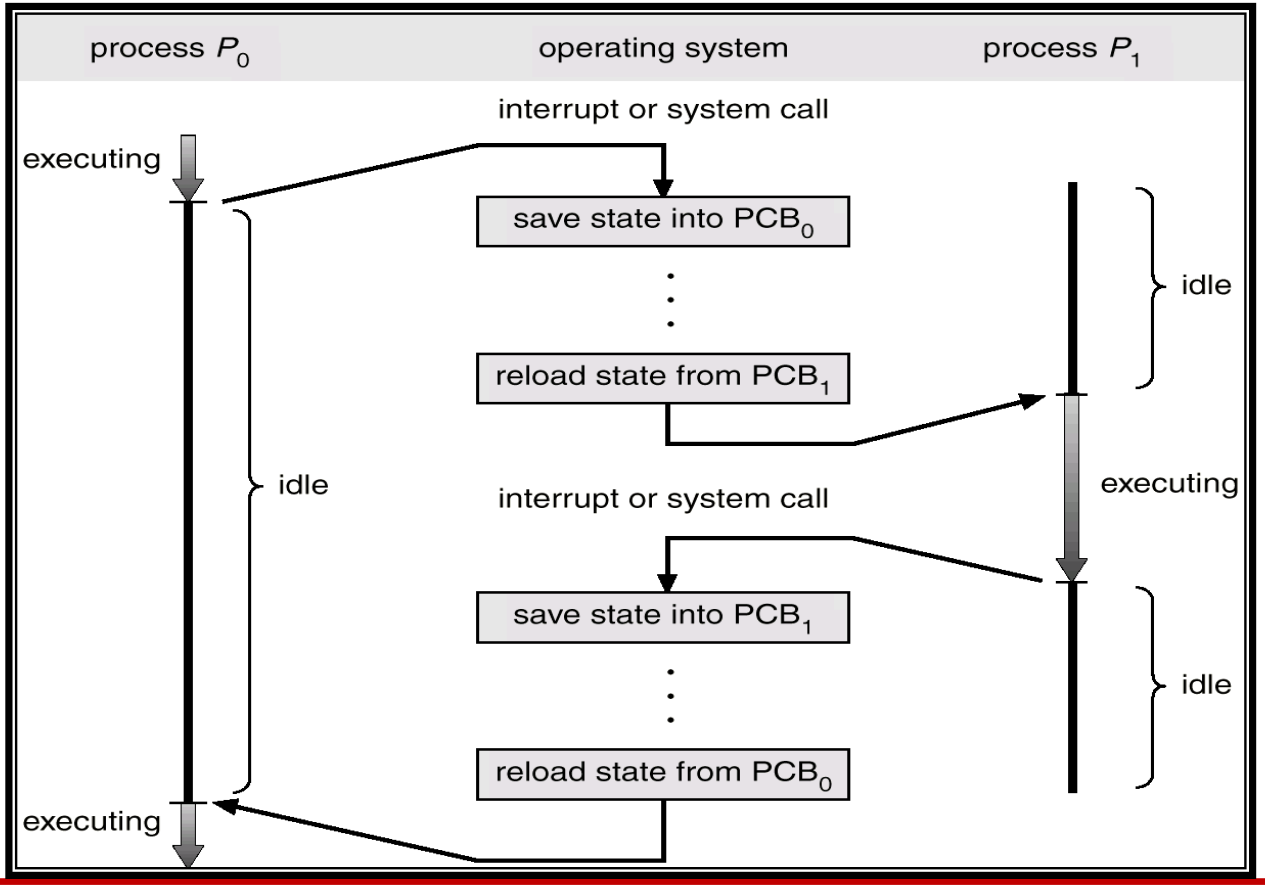
Multiprocessing & Multitasking

- The ability to execute multiple processes simultaneously is referred as *multiprocessing*
- Systems which are capable of performing multiprocessing are known as *multiprocessor* systems
- *Multiprocessor* systems possess multiple CPUs and can execute multiple processes simultaneously
- The ability of the Operating System to have multiple programs in memory, which are ready for execution, is referred as *multiprogramming*

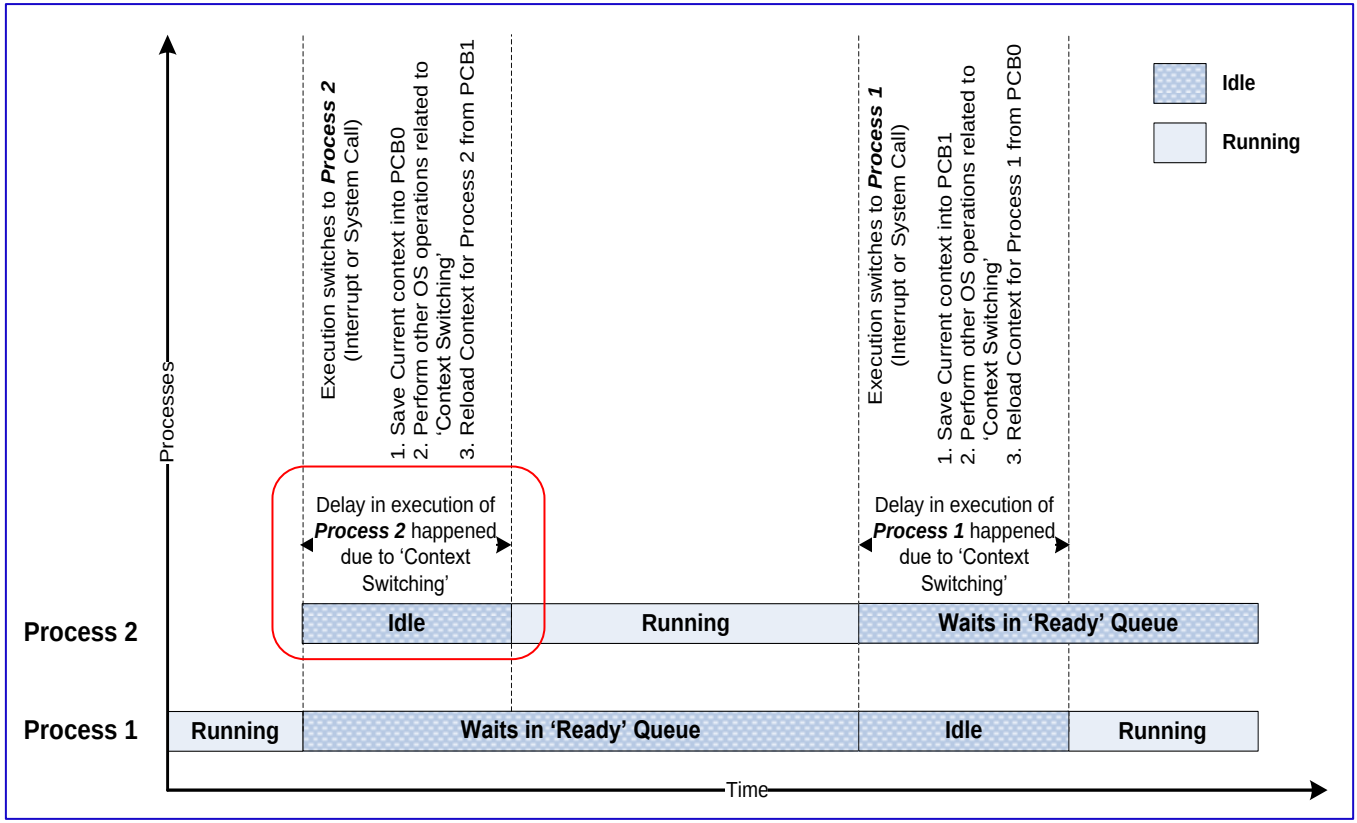
Multiprocessing & Multitasking

- *Multitasking* refers to the ability of an operating system to hold multiple processes in memory and switch the processor (CPU) from executing one process to another process
- *Multitasking* involves '*Context switching*', '*Context saving*' and '*Context retrieval*'
- ***Context switching*** refers to the switching of execution context from task to other
- When a task/process switching happens, the current context of execution should be saved to (***Context saving***) retrieve it at a later point of time when the CPU executes the process, which is interrupted currently due to execution switching
- During context switching, the context of the task to be executed is retrieved from the saved context list. This is known as ***Context retrieval***

CPU Switch From Process to Process



Multitasking – Context Switching



Types of Multitasking

Depending on how the task/process execution switching act is implemented, multitasking is classified into

- Co-operative Multitasking
- Preemptive Multitasking:
- Non-preemptive Multitasking

Co-operative Multitasking

- Co-operative multitasking is the most primitive form of multitasking in which a task/process gets a chance to execute only when the currently executing task/process ***voluntarily relinquishes*** the CPU.
- Any task/process can avail the CPU as much time as it wants.

- Since this type of implementation involves the mercy of the tasks each other for getting the CPU time for execution, it is known as ***co-operative multitasking.***
- If the currently executing task is non-cooperative, the other tasks may have to wait for a long time to get the CPU

Preemptive Multitasking

- Preemptive multitasking ensures that every task/process gets a chance to execute.
- When and how much time a process gets is dependent on the implementation of the preemptive scheduling.
- As the name indicates, in preemptive multitasking, the currently running task/process is **preempted** to give a chance to other tasks/process to execute.
- The preemption of task may be based on time slots or priority

Non-preemptive Multitasking

- The process/task, which is currently given the CPU time, is allowed to execute until it terminates (enters the 'Completed' state) or enters the 'Blocked/Wait' state, waiting for an I/O.
- The co-operative and non-preemptive multitasking differs in their behavior when they are in the 'Blocked/Wait' state.
- In co-operative multitasking, the currently executing process/task need not relinquish the CPU when it enters the 'Blocked/Wait' sate, waiting for an I/O, or a shared resource access or an event to occur whereas in non-preemptive multitasking the currently executing task relinquishes the CPU when it waits for an I/O.

Task Scheduling

Task Scheduling

- Mechanism in place to share the CPU among the different tasks and to decide which process/task is to be executed at a given point of time
- Determining **which task/process is to be executed** at a given point of time is known as task/process scheduling
- Task scheduling forms the basis of multitasking
- **Scheduling policies forms** the guidelines for determining which **task is to be executed when**
- The scheduling policies are implemented in an **algorithm** and **it is run by the kernel as a service**
- The kernel service/application, which implements the scheduling algorithm, is known as '***Scheduler***'
- The task scheduling policy can be
  - *pre-emptive*
  - *non-preemptive*
  - *co-operative*

Task Scheduling

- Depending on the scheduling policy the task/process scheduling decision may take place when a process switches its state to
  - *'Ready'* state from *'Running'* state
  - *'Blocked/Wait'* state from *'Running'* state
  - *'Ready'* state from *'Blocked/Wait'* state
  - *'Completed'* state

Task Scheduling- Scheduler Selection

The selection of a scheduling criteria/algorithm should consider

- **CPU Utilization:**
  - The scheduling algorithm should always make the CPU utilization high.
  - CPU utilization is a direct measure of how much percentage of the CPU is being utilized.
- **Throughput:**
  - Gives an indication of the number of processes executed per unit of time.
  - The throughput for a good scheduler should always be higher.

Task Scheduling- Scheduler Selection

The selection of a scheduling criteria/algorithm should consider

- **Turnaround Time (TAT):**
  - It is the amount of time taken by a process for **completing its execution**.
  - It includes the time spent by the process for waiting for the main memory, time spent in the ready queue, time spent on completing the I/O operations, and the time spent in execution.
  - The ***turnaround time should be a minimum*** for a good scheduling algorithm.

Task Scheduling- Scheduler Selection

The selection of a scheduling criteria/algorithm should consider

- **Waiting Time:**
  - It is the amount of time spent by a process in the '*Ready*' queue **waiting to get the CPU** time for execution.
  - The ***waiting time should be minimal*** for a good scheduling algorithm.

Task Scheduling- Scheduler Selection

The selection of a scheduling criteria/algorithm should consider

- **Response Time:**
  - It is the time elapsed between the submission of a process and the first response.
  - For a good scheduling algorithm, the response time should be as least as possible.

To summarize, a good scheduling algorithm has high CPU utilization, minimum Turn Around Time (TAT), maximum throughput and least response time.

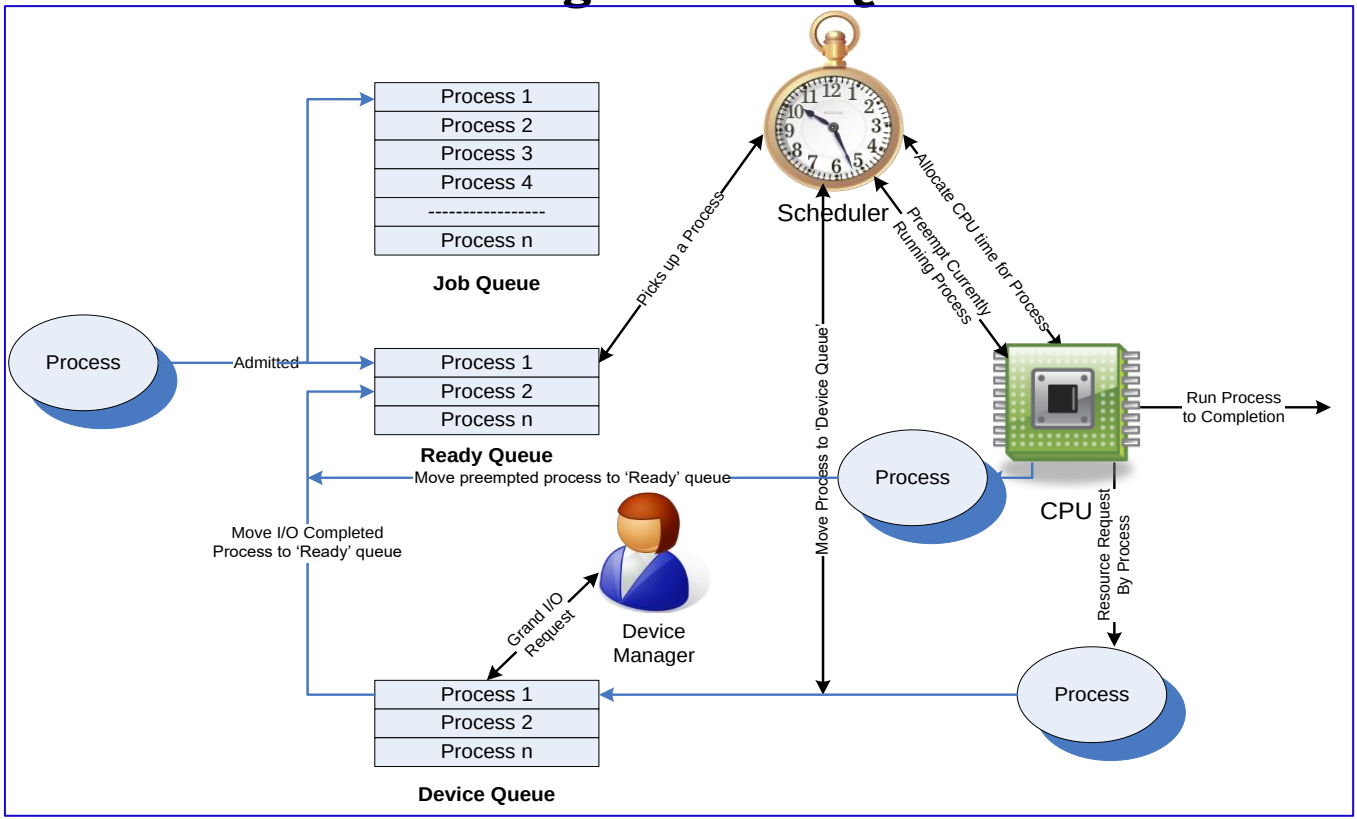
Task Scheduling - Queues

The various queues maintained by OS in association with CPU scheduling are

- **Job Queue:** Job queue contains all the processes in the system
- **Ready Queue:** Contains all the processes, which are ready for execution and waiting for CPU to get their turn for execution. The Ready queue is empty when there is no process ready for running.
- **Device Queue:** Contains the set of processes, which are waiting for an I/O device

Task Scheduling

Task transition through various Queues



Non-preemptive scheduling Algorithms

Non-preemptive scheduling Algorithms

1. First Come First Served (FCFS)/First In First Out (FIFO) Scheduling
2. Last Come First Served (LCFS)/Last In First Out (LIFO) Scheduling
3. Shortest Job First (SJF)
4. Priority based Scheduling

Non-preemptive scheduling – First Come First Served (FCFS)/First In First Out (FIFO) Scheduling

- Allocates CPU time to the processes based on the order in which they enters the '*Ready*' queue
- The first entered process is serviced first
- It is same as any real world application where queue systems are used; E.g. Ticketing

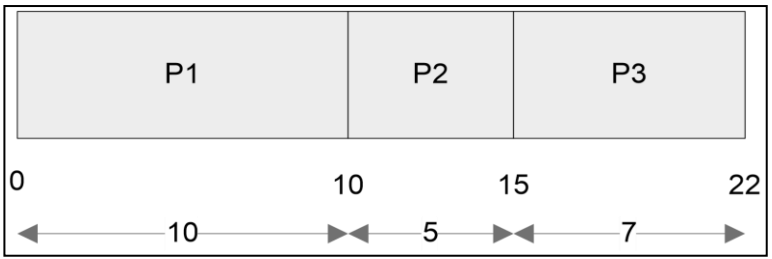
Non-preemptive scheduling – First Come First Served (FCFS)/ First In First Out (FIFO) Scheduling

Drawbacks:

- Favors monopoly of process.
- A process, which does not contain any I/O operation, continues its execution until it finishes its task
- In general, **FCFS** favors CPU bound processes and I/O bound processes may have to wait until the completion of CPU bound process, if the currently executing process is a CPU bound process. This leads to **poor device utilization**.
- The average waiting time is not minimal for FCFS scheduling algorithm

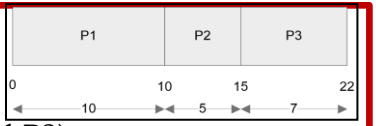
Example 1 (FCFS / FIFO)

- Three processes with process IDs P1, P2, P3 with estimated completion time 10, 5, 7 milliseconds respectively enters the ready queue together in the order P1, P2, P3.
- Calculate the **Waiting time, Average waiting time, Turn Around Time (TAT) for each process** and **Average Turn Around Time** (Assuming there is no I/O waiting for the processes).
- The sequence of execution of the processes by the CPU is represented as



Assuming the CPU is readily available at the time of arrival of P1, P1 starts executing without any waiting in the 'Ready' queue. Hence the waiting time for P1 is zero.

Example 1 (FCFS / FIFO)



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.

The 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 milliseconds

Average Turn Around Time = Average Waiting time + Average execution time
= 8.33 + 7.33
= 15.66 milliseconds

Example 2 (FCFS / FIFO)

- Three processes with process IDs P1, P2, P3 with estimated completion time 10, 5, 7 milliseconds respectively enters the ready queue together in the order **P2, P1, P3**.
- Calculate the **Waiting time, Average waiting time, Turn Around Time (TAT) for each process** and **Average Turn Around Time** (Assuming there is no I/O waiting for the processes).

Assuming the CPU is readily available at the time of arrival of P2, P2 starts executing without any waiting in the 'Ready' queue. Hence the waiting time for 2s zero.

Example 2 (FCFS / FIFO)

Waiting Time for P2 = 0 ms (P2 starts executing first)
Waiting Time for P1 = 5 ms (P1 starts executing after completing P2)
Waiting Time for P3 = 15 ms (P3 starts executing after completing P2 and P1)
Average waiting time = (Waiting time for all processes) / No. of Processes
= (Waiting time for (P2+P1+P3)) / 3
= (0+5+15)/3 = 20/3 = 6.66 milliseconds
Turn Around Time (TAT) for P2 = 5 ms (Time spent in Ready Queue + Execution Time)
Turn Around Time (TAT) for P1 = 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 (P2+P1+P3)) / 3
= (5+15+22)/3 = 42/3
= 14 milliseconds
Average Turn Around Time (TAT) is the sum of average waiting time and average execution time.
The average Execution time = (Execution time for all processes)/No. of processes
= (Execution time for (P2+P1+P3))/3
= (5+10+7)/3 = 22/3 = 7.33 milliseconds
Average Turn Around Time = Average Waiting time + Average execution time
= 6.66 + 7.33
= 14 milliseconds

Non-preemptive scheduling – Last Come First Served (LCFS)/ Last In First Out (LIFO) Scheduling

- Allocates CPU time to the processes based on the order in which they are entered in the '*Ready*' queue
- The last entered process is serviced first
- LCFS scheduling is also known as Last In First Out (LIFO) where the process, which is put last into the '*Ready*' queue, is serviced first

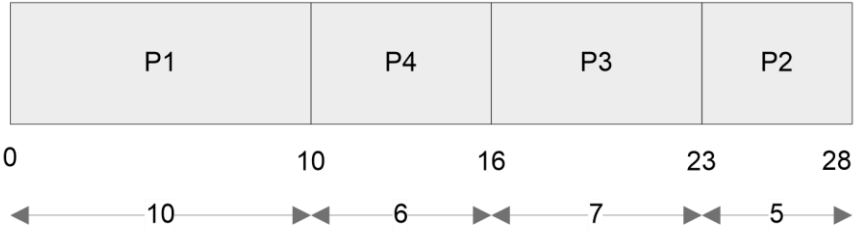
Non-preemptive scheduling – Last Come First Served (LCFS)/ Last In First Out (LIFO) Scheduling

Drawbacks:

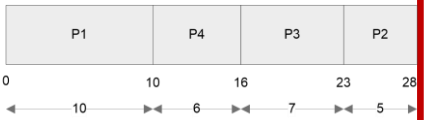
- Favors monopoly of process.
- A process, which does not contain any I/O operation, continues its execution until it finishes its task
- In general, LCFS favors CPU bound processes and I/O bound processes may have to wait until the completion of CPU bound process, if the currently executing process is a CPU bound process. This leads to poor device utilization.
- The average waiting time is not minimal for LCFS scheduling algorithm

Example 1 (LCFS / LIFO)

- Three processes with process IDs P1, P2, P3 with estimated completion time 10, 5, 7 milliseconds respectively enters the ready queue together in the order P1, P2, P3 (Assume only P1 is present in the '*Ready*' queue when the scheduler picks up it and P2, P3 entered '*Ready*' queue after that).
- Now a new process P4 with estimated completion time 6ms enters the 'Ready' queue after 5ms of scheduling P1. Calculate the waiting time and Turn Around Time (TAT) for each process and the Average waiting time and Turn Around Time (Assuming there is no I/O waiting for the processes).
- Assume all the processes contain only CPU operation and no I/O operations are involved.
- Initially there is only P1 available in the Ready queue and the scheduling sequence will be P1, P3, P2. P4 enters the queue during the execution of P1 and becomes the last process entered the '*Ready*' queue. Now the order of execution changes to P1, P4, P3, and P2 as given below.



Example 1 (LCFS / LIFO)



The waiting time for all the processes are 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 5ms 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

Non-preemptive scheduling – Shortest Job First (SJF) Scheduling

- Allocates CPU time to the processes based on the execution completion time for tasks
- The average waiting time for a given set of processes is minimal in SJF scheduling
- Optimal compared to other non-preemptive scheduling like FCFS

Non-preemptive scheduling – Shortest Job First (SJF) Scheduling

Drawbacks:

- A process whose estimated **execution completion time is high** may not get a chance to execute **if more and more processes with least estimated execution time enters** the '*Ready*' queue before the process with longest estimated execution time starts its execution
- May lead to the '*Starvation*' of processes with high estimated completion time
- Difficult to know in advance the next shortest process in the '*Ready*' queue for scheduling since new processes with different estimated execution time keep entering the '*Ready*' queue at any point of time.

Example 1 – Shortest Job First (SJF) Scheduling

- Three processes with process IDs P1, P2, P3 with estimated completion time 10, 5, 7 milliseconds respectively enters the ready queue together in the order P1, P2, P3 (Assume P1, P2, P3 is present in the '*Ready*' queue when the scheduler picks up it).
- Calculate the waiting time and Turn Around Time (TAT) for each process and the Average waiting time and Turn Around Time (Assuming there is no I/O waiting for the processes). In SJF Algorithm.
- Sequence of execution: P2, P3, P1 (shortest estimated completion)

Animation


Shortest Job First Scheduling
Process
Burst Time
Timer
P1
P2
P3
5
3
2
0

Example 1 – Shortest Job First (SJF) Scheduling

The waiting time for all the processes are given as

Waiting Time for P2 = 0 ms (P2 starts executing first)
Waiting Time for P3 = 5 ms (P3 starts executing after completing P2)
Waiting Time for P1 = 12ms (P1 starts executing after completing P2 and P3)

Average waiting time = (Waiting time for all processes) / No. of Processes
= (Waiting time for (P2+P3+P1)) / 3
= (0 + 5 + 12)/3 = 17/3
= 5.66 milliseconds

Turn Around Time (TAT) for P2 = 5 ms (Time spent in Ready Queue + Execution Time)
Turn Around Time (TAT) for P3 = 12 ms (Time spent in Ready Queue + Execution Time
Turn Around Time (TAT) for P1 = 22 ms (Time spent in Ready Queue + Execution Time)

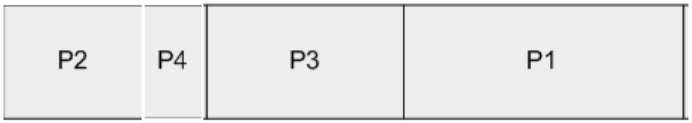
Average Turn Around Time = (Turn Around Time for all processes) / No. of Processes
= (Turn Around Time for (P2+P3+P1)) / 3
= (5+12+22)/3 = 39/3
= 13 milliseconds

Average waiting time and turn around time is much improved with SJF compare to FCFS

Example 2 – Shortest Job First (SJF) Scheduling

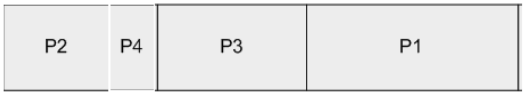
- Calculate the waiting time and Turn Around Time (TAT) for each process and the Average waiting time and Turn Around Time (Assuming there is no I/O waiting for the processes) for **the previous example** if a new process **P4 with estimated completion time 2ms** enters the 'Ready' queue **after 2ms of execution of P2**.
- *(Assuming there is no I/O waiting for the processes).*
- *P1, P2, P3 with estimated completion time 10, 5, 7 milliseconds respectively*

- Initially P2 available in executing and P4 enters the queue during the execution of P2 and after completing the P2, based on the SJF algorithm, execution of P4. Now the order of execution changes to P2, P4, P3, and P1 as given below.



Example 2 – Shortest Job First (SJF) Scheduling

The waiting time for all the processes are given as



Waiting Time for P2 = 0 ms (P2 starts executing first)
Waiting Time for P4 = 3 ms (P4 starts executing after completing P2.
But P4 arrived after 2ms of execution of P2.
Hence its waiting time = Execution start time – Arrival Time = 5-2 = 3)
Waiting Time for P3 = 7 ms (P3 starts executing after completing P2 and P4)
Waiting Time for P1 = 14 ms (P1 starts executing after completing P2, P4 and P3)
Average waiting time = (Waiting time for all processes) / No. of Processes
= (Waiting time for (P2+P4+P3+P1)) / 4
= (0 + 3 + 7 + 14)/4 = 24/4
= 6 milliseconds

Turn Around Time (TAT) for P2 = 5 ms (Time spent in Ready Queue + Execution Time)
Turn Around Time (TAT) for P4 = 5 ms (Time spent in Ready Queue + Execution Time
= (Execution Start Time – Arrival Time) + Estimated Execution Time
= (5-2) + 2 = 3 + 2
Turn Around Time (TAT) for P3 = 14 ms (Time spent in Ready Queue + Execution Time)
Turn Around Time (TAT) for P1 = 24 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 (P2+P4+P3+P1)) / 4
= (5+5+14+24)/4 = 48/4
= 12 milliseconds

Average waiting time for a given set of process is minimal in SJF compare to FCFS

Non-preemptive scheduling – Priority based Scheduling

- A **priority**, which is unique or same is associated with each task
- The priority of a task is expressed in different ways, like a **priority number, the time required to complete** the execution etc.
- In number based priority assignment the priority is a number ranging from 0 to the maximum priority supported by the OS. The maximum level of priority is OS dependent.
- Windows CE supports 256 levels of priority (0 to 255 priority numbers, with 0 being the highest priority)
- The priority is assigned to the task on creating it. It can also be changed dynamically (If the Operating System supports this feature)
- **The non-preemptive priority based scheduler sorts the 'Ready' queue based on the priority and picks the process with the highest level of priority for execution**

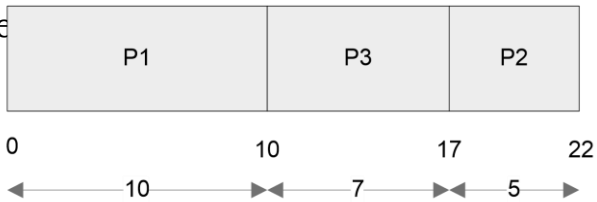
Non-preemptive scheduling – Priority based Scheduling

Drawbacks:

- ✓ Similar to SJF scheduling algorithm, non-preemptive priority based algorithm also possess the drawback of '*Starvation*' where a process whose priority is low may not get a chance to execute if more and more processes with higher priorities enter the '*Ready*' queue before the process with lower priority starts its execution.
- ✓ '*Starvation*' can be effectively tackled in priority based non-preemptive scheduling by dynamically raising the priority of the low priority task/process which is under starvation (waiting in the ready queue for a longer time for getting the CPU time)
- ✓ The technique of gradually raising the priority of processes which are waiting in the 'Ready' queue as time progresses, for preventing '*Starvation*', is known as '*Aging*'.

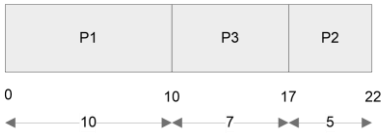
Example 1 – Priority based Scheduling

- Three processes with process IDs P1, P2, P3 with estimated completion time 10, 5, 7 milliseconds and priorities 0, 3, 2 (0- highest priority, 3 lowest priority) respectively enters the ready queue together.
- Calculate the waiting time and Turn Around Time (TAT) for each process and the Average waiting time and Turn Around Time (Assuming there is no I/O waiting for the processes) in priority based scheduling algorithm.
- The scheduler sorts the '*Ready*' queue based on the priority and schedules the process with the highest priority (P1 with priority number 0) first and the next high priority process (P3 with priority number 2) as second and so on. The order in which the processes are scheduled for exe



Example 1 – Priority based Scheduling

The waiting time for all the processes are given as



Waiting Time for P1 = 0 ms (P1 starts executing first)
Waiting Time for P3 = 10 ms (P3 starts executing after completing P1)
Waiting Time for P2 = 17 ms (P2 starts executing after completing P1 and P3)

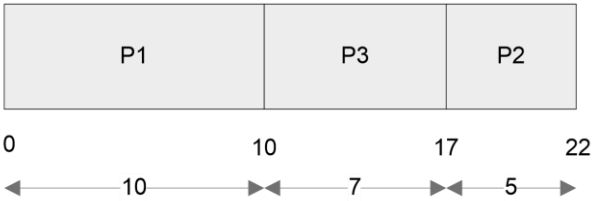
Average waiting time = (Waiting time for all processes) / No. of Processes
= (Waiting time for (P1+P3+P2)) / 3
= (0+10+17)/3 = 27/3
= 9 milliseconds

Turn Around Time (TAT) for P1 = 10 ms (Time spent in Ready Queue + Execution Time)
Turn Around Time (TAT) for P3 = 17 ms (-Do-)
Turn Around Time (TAT) for P2 = 22 ms (-Do-)

Average Turn Around Time = (Turn Around Time for all processes) / No. of Processes
= (Turn Around Time for (P1+P3+P2)) / 3
= (10+17+22)/3 = 49/3
= 16.33 milliseconds

Example 2 – Priority based Scheduling

- Calculate the waiting time and Turn Around Time (TAT) for each process and the Average waiting time and Turn Around Time (Assuming there is no I/O waiting for the processes) for **the previous example** if a new process **P4 with estimated completion time 6ms and priority 1** enters the 'Ready' queue **after 5ms of execution of P1**.
- *(Assuming there is no I/O waiting for the processes).*
- *P1, P2, P3 with estimated completion time 10, 5, 7 milliseconds respectively*
- Initially P1 available in executing(priority) and P4 enters the queue during the execution of P4 (priority 1) and after completing the P2, based on the algorithm, execution of P4. Now the order of execution changes to P1, P4, P3, and P2.



Example 2 – Priority based Scheduling

The waiting time for all the processes are 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 (prio -1) arrived after 5ms 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

Average waiting time fro a given set of process is minimal in SJF compare to FCFS

Preemptive Scheduling Algorithms

Preemptive scheduling Algorithms

- Employed in systems, which implements **preemptive multitasking model**
- **Every** task in the '*Ready*' queue gets **a chance to execute.**
- **When and how often each process gets a chance ???**
- The scheduler can preempt (stop temporarily) the currently executing task/process and select another task from the '*Ready*' queue for execution
- **When to pre-empt a task and which task is to be picked up for execution after preempting ???**
- The act of moving a '*Running*' process/task into the '*Ready*' queue by the scheduler, without the processes requesting for it is known as '*Preemption*'
- **Preemption approaches :**
  1. Time-based preemption
  2. Priority-based preemption

Preemptive scheduling Algorithms

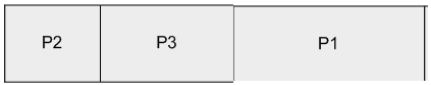
- Preemptive SJF / SRT Scheduling
- Round Robin (RR) Scheduling
- Priority based Scheduling

Preemptive SJF / SRT Scheduling

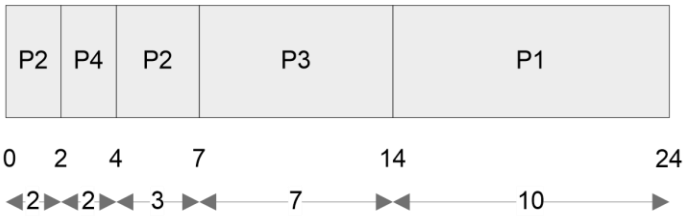
- The ***non preemptive SJF*** scheduling algorithm sorts the 'Ready' queue only after the current process completes execution or enters wait state. **(SRT-Shortest Remaining Time)**
- ***Preemptive SJF*** scheduling **algorithm sorts the 'Ready'** queue **when a new process enters** the 'Ready' queue and checks whether the execution time of the new process is shorter than the remaining of the total estimated execution time of the currently executing process
- If the execution time of the new process is less, the currently executing process is preempted (stopped) and the new process is scheduled for execution
- Always compares the execution completion time (ie the remaining execution time for the new process) of a new process entered the 'Ready' queue with the remaining time for completion of the currently executing process and schedules the process with shortest remaining time for execution

Example 1 – Preemptive SJF / SRT Scheduling

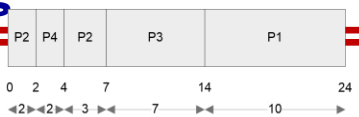
- Three processes with process IDs P1, P2, P3 with estimated completion time 10, 5, 7 milliseconds respectively enters the ready queue together.
- A new process P4 with estimated completion time 2ms enters the 'Ready' queue after 2ms. Assume all the processes contain only CPU operation and no I/O operations are involved.
- At the beginning, there are only three processes (P1, P2 and P3) and P2 gets resource



- Now process P4 with estimated execution completion time 2ms enters the '*Ready*' queue after 2ms of start of execution of P2. The processes are re-scheduled for execution in the following order



Example 1 – Preemptive SJF Scheduling



The waiting time for all the processes are given as

Waiting Time for P2 = 0 ms + (4 -2) ms = 2ms (P2 starts executing first and is interrupted by P4 and has to wait till the completion of P4 to get the next CPU slot)

Waiting Time for P4 = 0 ms (P4 starts executing by preempting P2 since the execution time for completion of P4 (2ms) is less than that of the Remaining time for execution completion of P2 (Here it is 3ms))

Waiting Time for P3 = 7 ms (P3 starts executing after completing P4 and P2)

Waiting Time for P1 = 14 ms (P1 starts executing after completing P4, P2 and P3)

Average waiting time = (Waiting time for all the processes) / No. of Processes
= (Waiting time for (P4+P2+P3+P1)) / 4
= (0 + 2 + 7 + 14)/4 = 23/4
= 5.75 milliseconds

Turn Around Time (TAT) for P2 = 7 ms (Time spent in Ready Queue + Execution Time)

Turn Around Time (TAT) for P4 = 2 ms
(Time spent in Ready Queue + Execution Time = (Execution Start Time – Arrival Time) + Estimated Execution Time = (2-2) + 2)

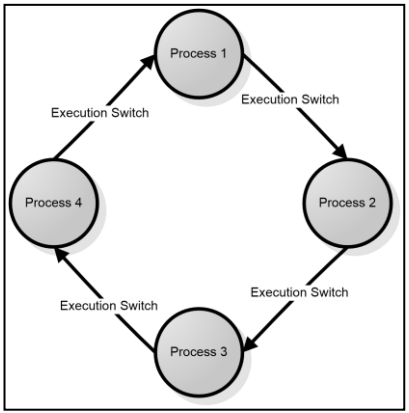
Turn Around Time (TAT) for P3 = 14 ms (Time spent in Ready Queue + Execution Time)

Turn Around Time (TAT) for P1 = 24 ms (Time spent in Ready Queue + Execution Time)

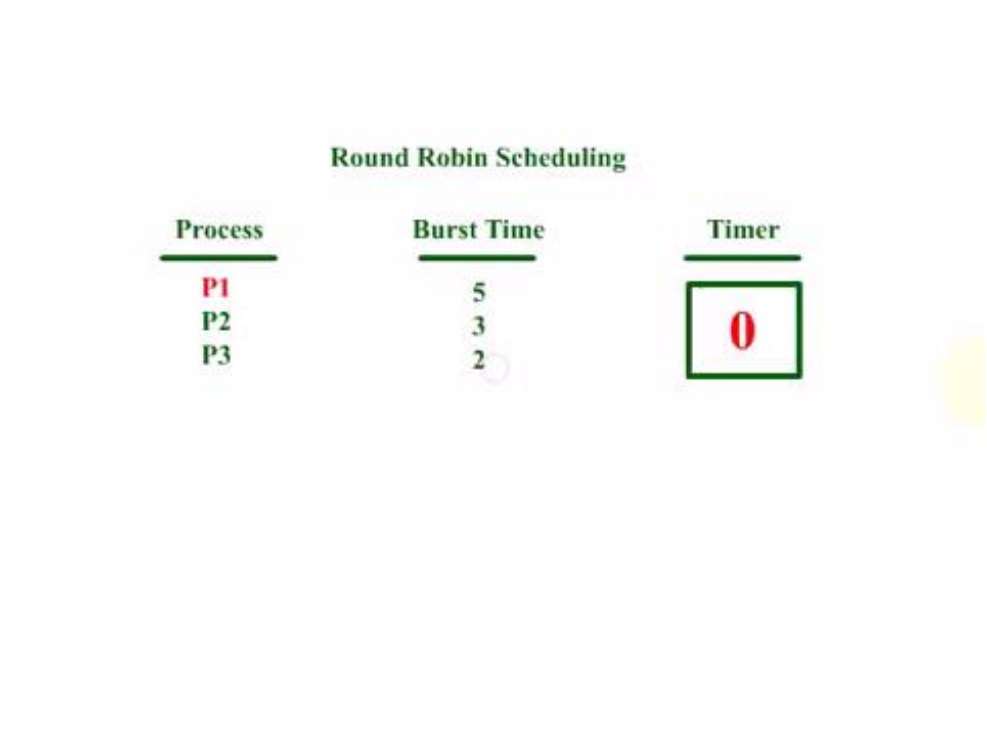
Average Turn Around Time = (Turn Around Time for all the processes) / No. of Processes
= (Turn Around Time for (P2+P4+P3+P1)) / 4
= (7+2+14+24)/4 = 47/4
= 11.75 milliseconds

Preemptive scheduling – Round Robin (RR) Scheduling

- Each process in the 'Ready' queue is executed for a **pre-defined time slot**.
- The execution starts with picking up the first process in the 'Ready' queue. It is executed for a pre-defined time
- When the pre-defined time elapses or the process completes (before the pre-defined time slice), **the next process in the 'Ready' queue** is selected for execution.
- This is repeated for all the processes in the 'Ready' queue
- Once each process in the 'Ready' queue is executed for the pre-defined time period, the scheduler comes back and picks the first process in the 'Ready' queue again for execution
- **Round Robin scheduling is similar to the FCFS scheduling and the only difference is that a time slice based preemption is added** to switch the execution between the processes in the 'Ready' queue



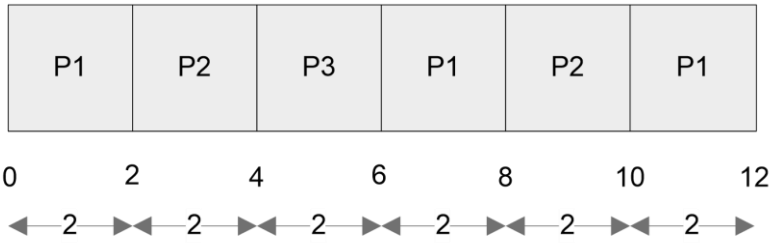
Animation


Round Robin Scheduling
Process
Burst Time
Timer
P1
P2
P3
5
3
2
0

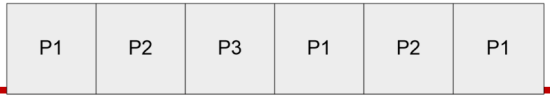
Example 1 – Round Robin (RR) Scheduling

- Three processes with process IDs P1, P2, P3 with estimated completion time 6, 4, 2 milliseconds respectively, enters the ready queue together in the order P1, P2, P3. Calculate the waiting time and Turn Around Time (TAT) for each process and the Average waiting time and Turn Around Time (Assuming there is no I/O waiting for the processes) in **RR** algorithm **with Time slice= 2ms**.

- The scheduler sorts the '*Ready*' queue based on the FCFS policy and picks up the first process P1 from the 'Ready' queue and executes it for the time slice 2ms.
- When the time slice is expired, P1 is preempted and P2 is scheduled for execution. The Time slice expires after 2ms of execution of P2. Now P2 is preempted and P3 is picked up for execution. P3 completes its execution within the time slice and the scheduler picks P1 again for execution for the next time slice. This procedure is repeated till all the processes are serviced. The order in which the processes are scheduled for execution is represented as



Example 1 – RR Scheduling



The waiting time for all the processes are given as

Waiting Time for P1 = 0 + (6-2) + (10-8) = 0+4+2= **6ms** (P1 starts executing first and waits for two time slices to get execution back and again 1 time slice for getting CPU time)

Waiting Time for P2 = (2-0) + (8-4) = 2+4 = **6ms** (P2 starts executing after P1 executes for 1 time slice and waits for two time slices to get the CPU time)

Waiting Time for P3 = (4 -0) = **4ms** (P3 starts executing after completing the first time slices for P1 and P2 and completes its execution in a single time slice.)

Average waiting time = (Waiting time for all the processes) / No. of Processes
= (Waiting time for (P1+P2+P3)) / 3
= (6+6+4)/3 = 16/3
= **5.33** milliseconds

Turn Around Time (TAT) for P1 = 12 ms (Time spent in Ready Queue + Execution Time)
Turn Around Time (TAT) for P2 = 10 ms (-Do-)
Turn Around Time (TAT) for P3 = 6 ms (-Do-)

Average Turn Around Time = (Turn Around Time for all the processes) / No. of Processes
= (Turn Around Time for (P1+P2+P3)) / 3
= (12+10+6)/3 = 28/3
= 9.33 milliseconds

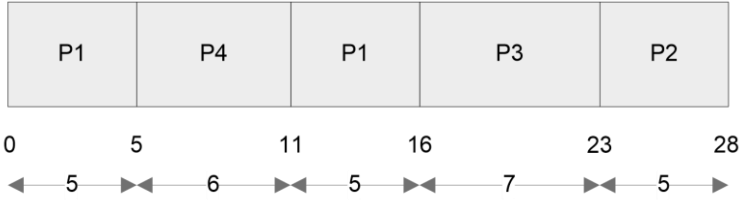
Demo
RT51 Tiny
Round Robin (RR) Scheduling

Preemptive scheduling – Priority based Scheduling

- Same as that of the ***non-preemptive priority*** based scheduling except for the switching of execution between tasks
- ***Preemptive priority*** based scheduling, any high priority process entering the 'Ready' queue is **immediately scheduled for execution**
- In the ***non-preemptive*** scheduling any high priority process entering the 'Ready' queue is scheduled **only after the currently executing process completes** its execution or only when it voluntarily releases the CPU
- The priority of a task/process in preemptive priority based scheduling is indicated in the same way as that of the mechanisms adopted for non-preemptive multitasking

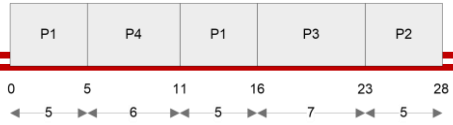
Example – 1 Priority based Scheduling

- Three processes with process IDs P1, P2, P3 with estimated completion time 10, 5, 7 milliseconds and priorities 1, 3, 2 (0-highest priority, 3 lowest priority) respectively enters the ready queue together.
- A new process P4 with estimated completion time 6ms and priority 0 enters the 'Ready' queue after 5ms of start of execution of P1. Assume all the processes contain only CPU operation and no I/O operations are involved.



- At the beginning, there are only three processes (P1, P2 and P3) available in the '*Ready*' queue and the scheduler picks up the process with the highest priority (In this example P1 with priority 1) for scheduling.
- Now process P4 with estimated execution completion time 6ms and priority 0 enters the '*Ready*' queue after 5ms of start of execution of P1.

Example – 1 Priority based Scheduling



The waiting time for all the processes are given as

Waiting Time for P1 = 0 + (11-5) = 0+6 =**6 ms** (P1 starts executing first and gets preempted by P4 after 5ms and again gets the CPU time after completion of P4)

Waiting Time for P4 = **0 ms** (P4 starts executing immediately on entering the 'Ready' queue, by preempting P1)

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 the processes) / No. of Processes

= (Waiting time for (P1+P4+P3+P2)) / 4

= (6 + 0 + 16 + 23)/4 = 45/4

= **11.25 milliseconds**

Turn Around Time (TAT) for P1 = **16 ms** (Time spent in Ready Queue + Execution Time)

Turn Around Time (TAT) for P4 = **6ms** (Time spent in Ready Queue + Execution Time = (Execution Start Time – Arrival Time) + Estimated Execution Time = (5-5) + 6 = 0 + 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 the processes) / No. of Processes

= (Turn Around Time for (P2+P4+P3+P1)) / 4

= (16+6+23+28)/4 = 73/4

= **18.25 milliseconds**

Summary

- Priority based preemptive scheduling **gives real time attention** to high priority tasks
- Priority based preemptive scheduling is adopted in systems which demands '**Real Time**' behavior
- Most of the RTOSs implements the preemptive priority based scheduling algorithm for process/task scheduling
- Preemptive priority based scheduling also possess the same drawback of non-preemptive priority based scheduling – '***Starvation***'
- This can be eliminated by the '***Aging***' technique (Temporarily boosting the priority of a task which is 'starving' for a long time

Thanks