CHAPTER 1 PROCESS MANAGEMENT

1.1 Process Concept

- Process is a single instance of a program in execution.
- Process is a program in execution.
- There are many processes can be running the same program.
- Process contains various resources like memory space, disk, printers, scanners, etc. as per its requirements.

The **5** major **activities** of an operating system regarding process management are:

1. Creation and deletion of user and system processes.
2. Suspension and resumption of processes.
3. A mechanism for process synchronization. (combine)
4. A mechanism for process communication.
5. A mechanism for deadlock handling.

Main-Memory Management

Main-Memory is a large array of words or bytes. Each word or byte has its own address. Main memory is a repository of quickly accessible data shared by the CPU and I/O devices.

The major **3** activities of an operating system in regard to memory-management are:

1. Keep track of which part of memory are currently being used and by whom.
2. Decide which processes are loaded into memory when memory space becomes available.
3. Allocate and deallocate memory space as needed.

Operating System Components

There are 6 components as follows

1. **File Management:** A file is a collected of related information defined by its creator. Computer can store files on the disk (secondary storage), which provide long term storage.
   - The creation and deletion of files.
   - The creation and deletion of directions.
   - The support of primitives for manipulating files and directions.
   - The mapping of files onto secondary storage.
   - The backup of files on stable storage media.
2. **I/O System Management**: One of the purposes of an operating system is to hide the peculiarities of specific hardware devices from the user.
3. **Secondary-Storage Management** Generally speaking, systems have several levels of storage, including primary storage, secondary storage and cache storage. Instructions and data must be placed in primary storage or cache to be referenced by a running program.

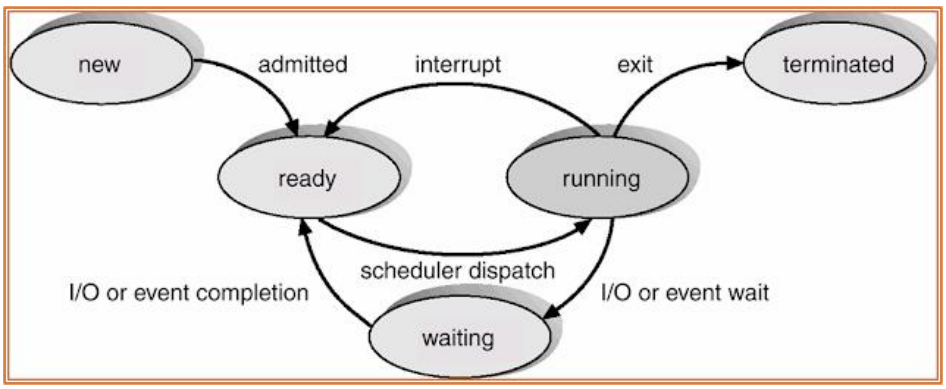
4. **Networking** A distributed system is a collection of processors that do not share memory, peripheral devices, or a clock. The processors communicate with one another through communication lines called network.
5. **Protection** System Protection refers to mechanism for controlling the access of programs, processes, or users to the resources defined by a computer system.
6. **Command Interpreter System** A command interpreter is an interface of the operating system with the user.

Operating Systems Services

1. **Program Execution** The system must be able to load a program into memory and to run it. The program must be able to end its execution, either normally or abnormally (indicating error).
2. **I/O Operations** A running program may require I/O. This I/O may involve a file or an I/O device.
3. **File System Manipulation(operations)** The output of a program may need to be written into new files or input taken from some files. The operating system provides this service.
4. **Error Detection** An error is one part of the system may cause malfunctioning of the complete system. To avoid such a situation the operating system constantly monitors the system for detecting the errors.

Process Life Cycle

The **life cycle** of a **process** can be described as follows:



Process State As a process executes, it changes state. The state of a process is defined in part by the current activity of that process. Each process may be in one of the following states:

- **New State**: The process being created.
- **Ready State**: A process is said to be ready if it is waiting to be assigned to a processor.
- **Running State**: A process is said to be running if it has the CPU, that is, process actually using the CPU at that particular instant.
- **Waiting State**: A process is said to be blocked if it is waiting for some event to happen such that as an I/O completion before it can proceed. Note that a process is unable to run until some external event happens.
- **Terminated state**: The process has finished execution.

Process State transition

From State	Event	To State
New	Accepted	Ready
Ready	Scheduled/Dispatch	Running
Running	Need I/O	Waiting
Running	Scheduler time-out	Ready
Running	Completion/Error/Killed	Terminated
waiting	I/O completed or wakeup event	ready

Process control block (PCB)

Process control block (PCB) is a data structure which is associated with any process and provides all the complete information about that process. Process control block is important in multiprogramming environment as it captures the information pertaining to the number of processes running simultaneously.

Role of process control block

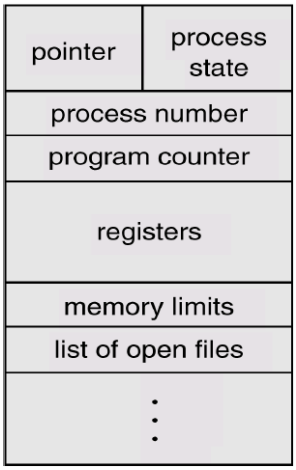
The role or work of process control block (PCB) in process management is that it can access or modified by most OS utilities including those are involved with memory, scheduling, and input / output resource access.

It can be said that the set of the process control blocks give the information of the current state of the operating system.

Data structuring for processes is often done in terms of process control blocks.

For example, pointers to other process control blocks inside any process control block allows the creation of those queues of processes in various scheduling states.

The following are the various information that is contained by process control block:



Components of Process control block

The following are the various components that are associated with the process control block PCB:

1. Process ID:
In computer system there are various process running simultaneously and each process has its unique ID which is called **P**rocess **ID**entification (PID) or **U**nique **ID**entification(UID). This Id helps system in scheduling the processes. This Id is provided by the process control block.
In other words, it is an identification number that uniquely identifies the processes of computer system.

2. Process state:
The process state of any process can be new, ready, running, waiting and terminate. Process control block is used to define the process state of any process.
In other words, process control block refers the states of the processes.

3. Program counter:
Program counter is used to point to the address of the next instruction to be executed in any process. This is also managed by the process control block.

4. Register Information:
This information is comprising with the various registers, such as index and stack that are associated with the process. This information is also managed by the process control block.

5. Scheduling information:
Scheduling information is used to set the priority of different processes. This is extremely useful information which is set by the process control block. In computer system there were many processes running simultaneously and each process have its priority. The priority of primary feature of RAM is higher than other secondary features. Scheduling information is useful in managing any computer system.

6. Memory related information:
This section of the process control block comprises of page and segment tables. It also stores the data contained in base and limit registers.

7. Accounting information:
This section of process control block stores the details related to central processing unit (CPU) utilization and execution time of a process.

8. Status information related to input / output:
This section of process control block stores the details pertaining to resource utilization and file opened during the process execution.

The operating system maintains a table called **process table**, which stores the process control blocks related to all the processes.

1.2 Process Scheduling

The act of determining which process is in the **ready** state and should be moved to the **running** state is known as **Process Scheduling**.

The prime aim of the process scheduling system is to keep the CPU busy all the time and to deliver minimum response time for all programs. For achieving this, the **scheduler** must apply appropriate rules for swapping processes **IN** and **OUT** of CPU.

Scheduling has two general categories:

1. **Non-preemptive Scheduling** is a CPU scheduling technique the process takes the resource (CPU time) and holds it till the process gets terminated or is pushed to the waiting state. No process is interrupted until it is completed, and after that processor switches to another process.
    - Algorithms that are based on non-preemptive Scheduling are First Come, First Served (FCFS), and shortest Job first(SJF).
2. **Preemptive Scheduling** is a CPU scheduling technique that works by dividing time slots of CPU to a given process. The time slot given might be able to complete the whole process or might not be able to it. When the total time of the process is greater than CPU cycle, it is placed back into the ready queue and will execute in the next chance. This scheduling is used when the process switch to ready state.
    - Algorithms that are backed by preemptive Scheduling are round-robin (RR), priority, SRTF (shortest remaining time first).

Preemptive Vs Non-Preemptive Scheduling

Preemptive Scheduling	Non-Preemptive Scheduling
Resources are allocated according to the cycles for a limited time.	Resources are used and then held by the process until it gets terminated.
The process can be interrupted, even before the completion.	The process is not interrupted until its life cycle is complete.
Starvation may be caused, due to the insertion of priority process in the queue.	Starvation can occur when a process with large burst time occupies the system.
Maintaining queue and remaining time needs storage overhead.	No such overheads are required.

Key Differences Between Preemptive and Non-Preemptive Scheduling

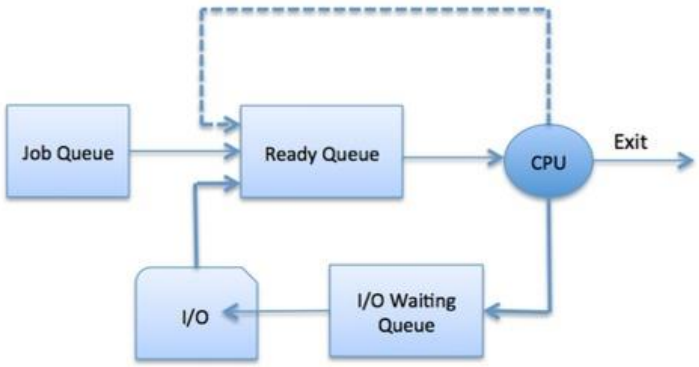
1. The basic difference between preemptive and non-preemptive scheduling is that in preemptive scheduling the CPU is allocated to the processes for the **limited** time. While in Non-preemptive scheduling, the CPU is allocated to the process till it **terminates** or switches to **waiting state**.
2. The executing process in preemptive scheduling is **interrupted** in the middle of execution whereas, the executing process in non-preemptive scheduling is **not interrupted** in the middle of execution.
3. Preemptive Scheduling has the **overhead** of switching the process from ready state to running state, vise-verse, and maintaining the ready queue. On the other hands, non-preemptive scheduling has **no overhead** of switching the process from running state to ready state.
4. In preemptive scheduling, if a process with high priority frequently arrives in the ready queue then the process with low priority have to wait for a long, and it may have to starve. On the other hands, in the non-preemptive scheduling, if CPU is allocated to the process with larger burst time then the processes with small burst time may have to starve.
5. Preemptive scheduling is quite **flexible** because the critical processes are allowed to access CPU as they arrive into the ready queue, no matter what process is executing currently. Non-preemptive scheduling is **rigid** as even if a critical process enters the ready queue the process running CPU is not disturbed.
6. The Preemptive Scheduling is cost associative as it has to maintain the integrity of shared data which is not the case with Non-preemptive Scheduling.

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.



A new process is initially put in the **Ready queue**. It waits in the ready queue until it is selected for execution (or dispatched). Once the process is assigned to the CPU and is executing, one of the following several events can occur:

- The process could issue an I/O request, and then be placed in the **I/O queue**.
- The process could create a new subprocess and wait for its termination.
- The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.

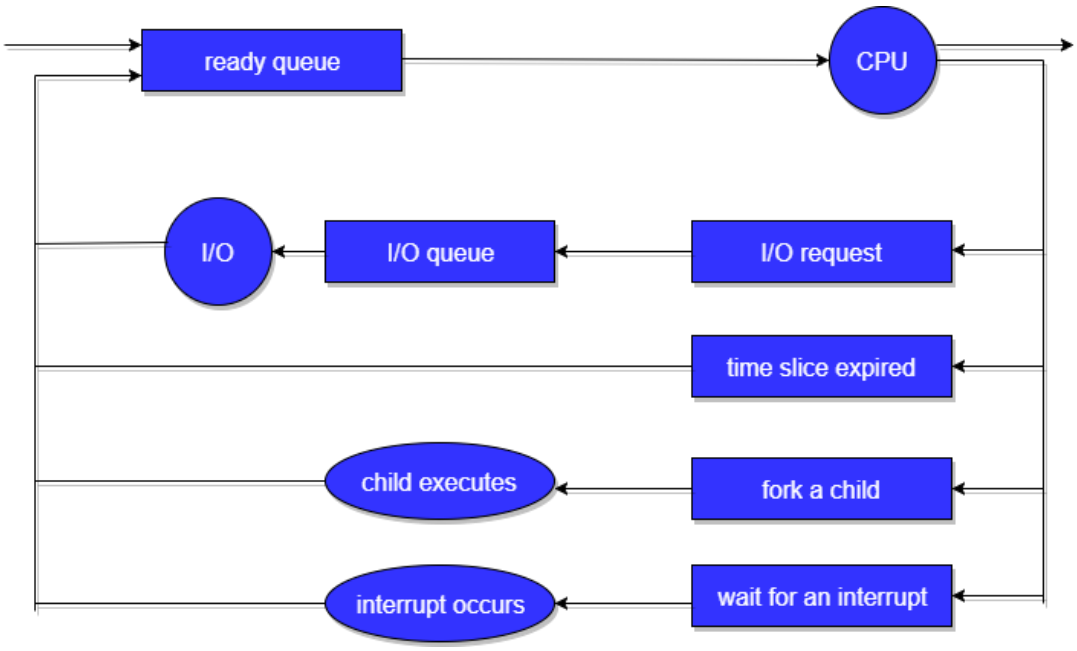
- **Scheduling Queuing Diagram Explanation:**

- All processes, upon entering into the system, are stored in the **Job Queue**.
- Processes in the Ready state are placed in the **Ready Queue**.
- Processes waiting for a device to become available are placed in **Device Queues**. There are unique device queues available for each I/O device.

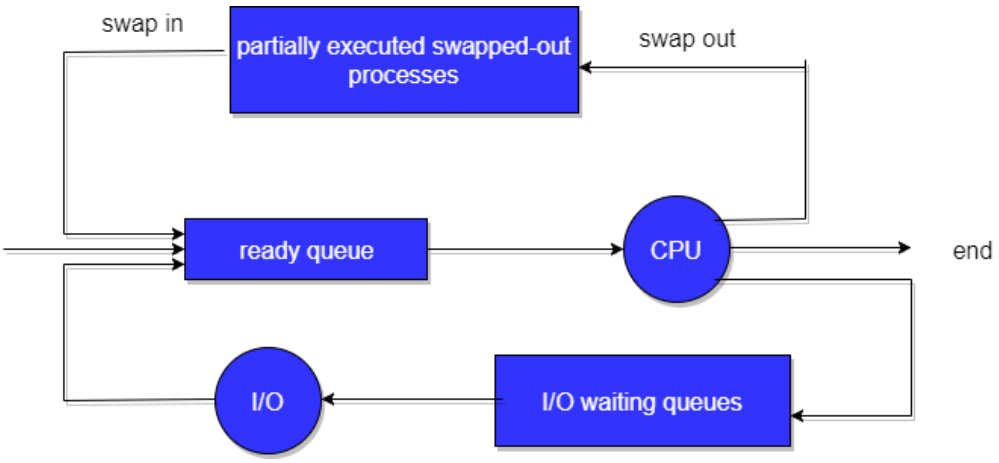
A new process is initially put in the **Ready queue**. It waits in the ready queue until it is selected for execution(or dispatched). Once the process is assigned to the CPU and is executing, one of the following several events can occur:

- The process could issue an I/O request, and then be placed in the **I/O queue**.
- The process could create a new subprocess and wait for its termination.
- The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.

In the first two cases, the process eventually switches from the waiting state to the ready state, and is then put back in the ready queue. A process continues this cycle until it terminates, at which time it is removed from all queues and has its PCB and resources deallocated.



[Addition of Medium-term scheduling to the queueing diagram]



Types of Schedulers

There are three types of schedulers available:

1. Long Term Scheduler
2. Short Term Scheduler
3. Medium Term Scheduler

1. Long Term Scheduler (Job scheduler-First Level Scheduler)

- Long term scheduler runs less frequently.
- It can be found in Batch Operating System.
- Long Term Schedulers decide which program must get into the job queue.
- From the job queue, the Job Processor, selects processes and loads them into the memory for execution.
- **Primary aim of the Job Scheduler is to maintain a good degree of Multiprogramming.**
- An optimal degree of Multiprogramming means the average rate of process creation is equal to the average departure rate of processes from the execution memory.
- 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.

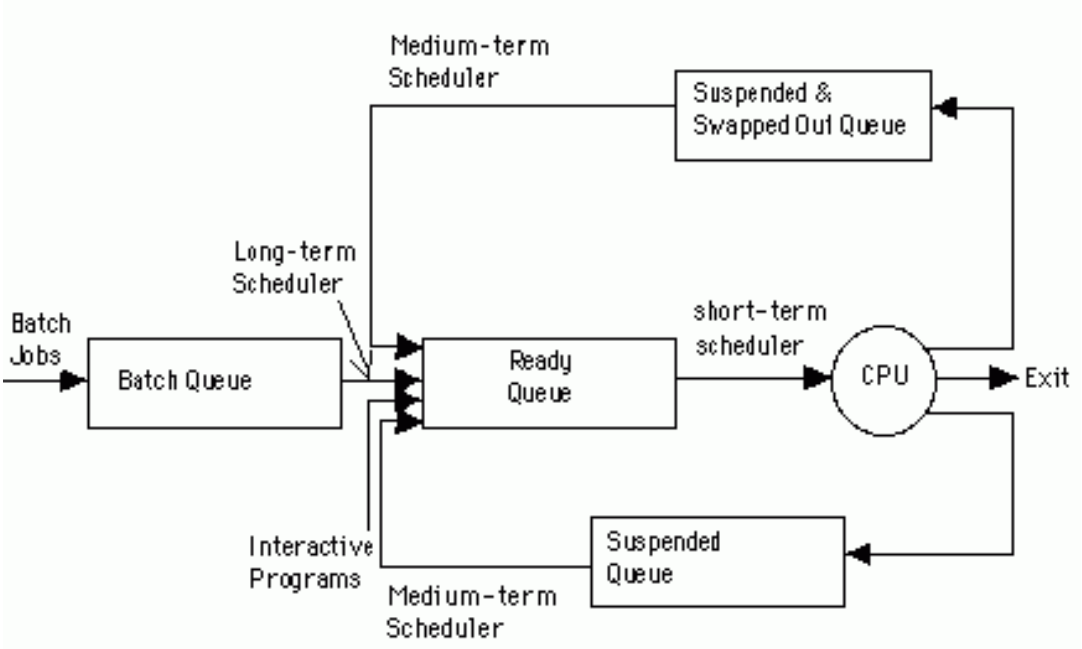
2. Short Term Scheduler (CPU scheduler- Third Level Scheduler)

- It is also called as **CPU scheduler**.
- It can be found always in all modern operating System
- 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.
- **The primary aim of this scheduler is to enhance CPU performance and increase process execution rate.**

3. Medium Term Scheduler (second level Scheduler)

- Medium-term scheduling is a part of **swapping**.
- It can be found in operating system where there is a support for '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.
- **The primary aim of this scheduler is to make efficient use of primary memory.**

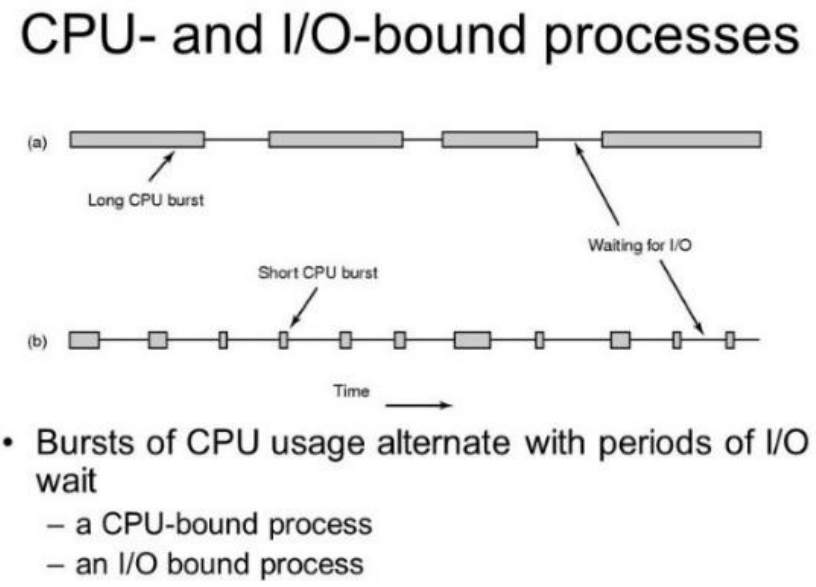


- **Explanation:-**
  Initially process is maintained in the Batch queue.
- Then, job of the Long-term scheduler is to fetch the process from batch queue to ready queue (Main Memory).
- Then, short-term scheduler selects process from the ready queue and assigns them to CPU. When process completes their task, then, process will end or terminate. During that period, some processes may ask for I/O device so kept in I/O waiting queue.
- After that I/O completion, they are moved back to ready queue.
- Now, consider process is executing and due to time slice process is suspended so it will be swap out from CPU and it will be kept in suspended & swapped out queue.
- Once swap out is done, it will be swap in ready queue. Thus scheduling is done by mid-term scheduler and this completes job done in time-sharing system.

Comparison between Schedulers

Sr. No.	Long Term	Short Term	Medium Term
1	It is job scheduler	It is CPU Scheduler	It is swapping
2	Speed is less than short term scheduler	Speed is very fast	Speed is in between both
3	It controls degree of multiprogramming	Less control over degree of multiprogramming	Reduce the degree of multiprogramming.
4	Absent or minimal in time sharing system.	Minimal in time sharing system.	Time sharing system use medium term scheduler.
5	It select processes from pool and load them into memory for execution.	It select from among the processes that are ready to execute.	Process can be reintroduced into memory and its execution can be continued.
6	Process state is (New to Ready)	Process state is (Ready to Running)	-
7	Select a good process, mix of I/O bound and CPU bound.	Select a new process for a CPU quite frequently.	-

PROCESS BEHAVIOUR: CPU Bound and I/O Bound



- **CPU Bound :** tend to have long CPU bursts for example: matrix multiplication
  - **Process spent most of its time using processor**
  - **Very long CPU burst**
  - **Example: compiler, simulator, scientific applications**
  - **CUP Bound** means the rate at which process progresses is limited by the speed of the CPU.
  - A task that performs calculations on a small set of numbers, for example multiplying small matrices, is likely to be CPU bound.
  - A program is CPU bound if it would go faster if the CPU were faster.
- **I/O Bound :** tend to have short CPU bursts for example: netscape
  - **Process spent most of its time using I/O**
  - **Very short CPU Burst**
  - **Example: word processor, database application**
  - **I/O Bound** means the rate at which a process progresses is limited by the speed of the I/O subsystem.
  - A task that processes data from disk, for example, counting the number of lines in a file is likely to be I/O bound.
  - A program is I/O bound if it would go faster if the I/O subsystem was faster.
- **Memory bound** means the rate at which a process progresses is limited by the amount memory available and the speed of that memory access.
  - A task that processes large amounts of in memory data, for example multiplying large matrices, is likely to be Memory Bound.
- **Cache bound** means the rate at which a process progress is limited by the amount and speed of the cache available.
  - A task that simply processes more data than fits in the cache will be cache bound.

I/O Bound would be slower than Memory Bound would be slower than Cache Bound would be slower than CPU Bound.

Context Switch in Operating System Processes

- A context switch occurs when a computer's CPU switches from one process or thread to a different process or thread.
- Context switching allows for one CPU to handle numerous processes or threads without the need for additional processors.
- A context switch is the mechanism to store and restore the state or context of a CPU in Process Control block so that a process execution can be resumed from the same point at a later time.
- Any operating system that allows for multitasking relies heavily on the use of context switching to allow different processes to run at the same time.

Typically, there are three situations that a context switch is necessary, as shown below.

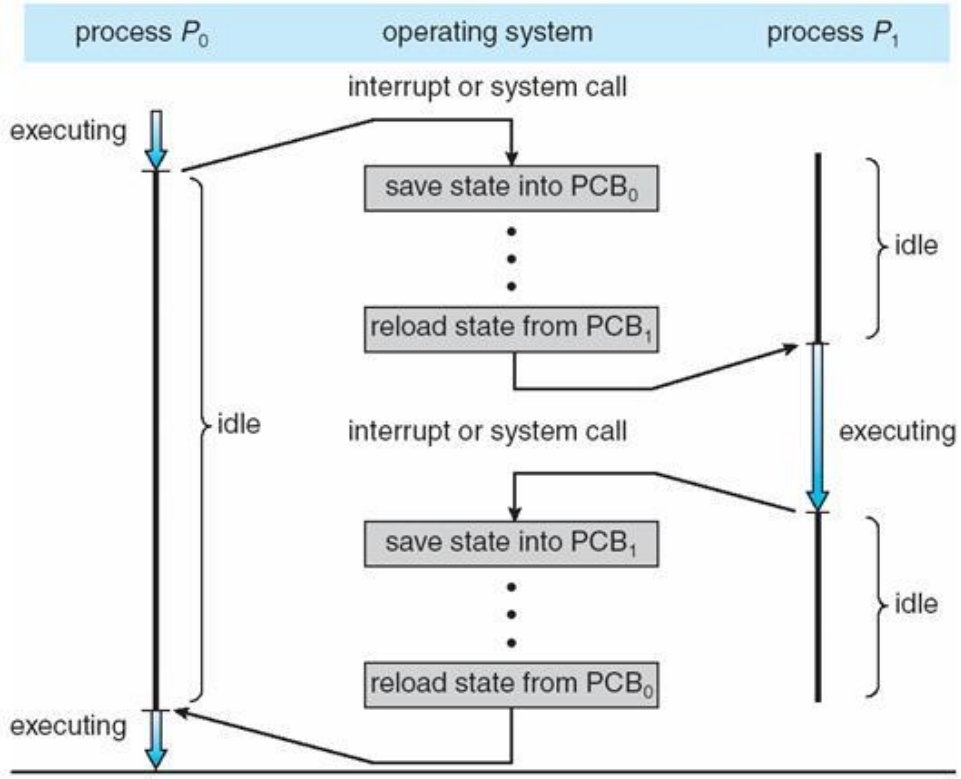
- **Multitasking** – When the CPU needs to switch processes in and out of memory, so that more than one process can be running.
- **Kernel/User Switch** – When switching between user mode to kernel mode, it may be used (but isn't always necessary).
- **Interrupts** – When the CPU is interrupted to return data from a disk read.

The steps in a full process switch are:

1. **Save the context** of the processor, including program counter and other registers.
2. **Update the process control block** of the process that is currently in the Running state. This includes changing the state of the process to one of the other states (Ready; Blocked; Ready/Suspend; or Exit). Other relevant fields must also be updated, including the reason for leaving the Running state and accounting information.
3. **Move** the **process control block** of this process to the **appropriate queue** (Ready; Blocked on Event *i ; Ready/Suspend).*
4. **Select another process** for execution.
5. Update the process control block of the process selected. This includes changing the state of this process to Running.
6. Update memory management data structures. This may be required, depending on how address translation is managed.
7. **Restore the context** of the processor to that which existed at the time the selected process was last switched out of the Running state, by loading in the previous values of the program counter and other registers.

Example:

Above figure explain the context switch between two processes process-0 and process-1.



Initially process-0 executes on CPU and process-1 is in idle (Waiting) state. After some time duration, process-0 release CPU. Now there is a turn of process-1. In other words there is a need of context switch.

At this point, context of process-0 is stored in its PCB0 also context of process-1 is loaded from its PCB1. CPU is allocated to process-1 and it start its execution.

Now, when process-1 release CPU. Context of process-1 in stored in its PCB! And context of process-0 loaded from its PCB0. CPU is allocated to process-0 and it continues its execution from the point when it left it off. Information regarding from where to start execution is stored in program Counter field in PCB.

1.3 Scheduling Criteria (Performance Criteria)

Following are the scheduling criteria:

1. CPU Utilization:

- It is an average fraction of time when CPU is busy.
- It ranges from 0 to 100 percent. In a real system. it should be between 40 to 90 percent.
- CPU should remain as busy as possible, or CPU utilization should remain as high as possible.

2. Throughput:

- Number of processes completed per time unit is called throughput.
- For long processes, this rate may be 1 process per hour: for short transactions, throughput might be 10 processes per second.

3. Turn-around Time:

- Time required to complete execution of a process is called turn-around time.
- It specifies that how long it takes to complete a process execution.
- Turn-around time = Process finish time — Process arrival time.

4. Waiting Time:

- It is total time duration spent by a process waiting in 'Ready' queue.
- Waiting time = Turn-Around Time — Burst Time.
- Scheduling algorithm affects the time that a process spends waiting in the ‘Ready’ state.

5. Response Time:

- It is a time- between issuing a command/request and getting output/result.
- It is more important in interactive systems. User requests should be served as quickly as possible. Here. user requests are given more priority compared to background system processes.

For better performance of' the system. CPU utilization and throughput should be maximized: while turnaround time. waiting time. and response time should be minimized.

1.4 Scheduling Algorithms

To decide which process to execute first and which process to execute last to achieve maximum CPU utilization, computer scientists have defined some algorithms, they are:

1. First Come First Serve(FCFS) Scheduling.
2. Shortest-Job-First(SJF) Scheduling.
3. Shortest Remaining Time Next (SRTN).
4. Round Robin(RR) Scheduling.
5. Priority based Non-Preemptive Scheduling.
6. Priority based Preemptive Scheduling.
7. Multilevel Queue Scheduling.

Non – Preemptive Scheduling Algorithms

1. First Come, First Served (FCFS)

Selection Criteria:

The process that requests first is served first. It means that processes are served in the exact order as they come.

Decision Mode:

Non-preemptive: Once a process is selected, it runs until it blocks for an I/O or some event, or it terminates.

Implementation:

This strategy can be easily implemented by using FIFO queue. FIFO means First In First Out. When first process enters the system, it starts its execution.

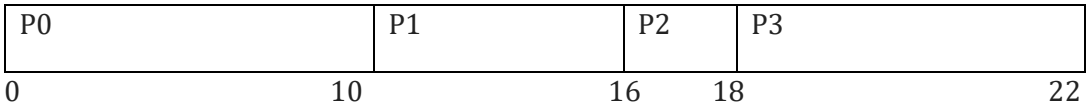
All other processes are appended in a queue. When CPU becomes free, a process from the first position in a queue is selected to run.

Example:

Consider the following set of four processes. Their arrival time and time required to complete the execution are given in following table. Consider all time values in milliseconds. Calculate Average Turn Around Time and Average Waiting Time.

Process	Arrival Time(T0)	Time Required for Completion
P0	0	10
P1	1	6
P2	3	2
P3	5	4

Gantt Chart:



Statistics:

Process	Arrival Time	Burst Time	Finish Time	Turn Around Time(TAT) (Finish Time – Arrival Time)	Waiting Time (TAT – Burst Time)
P0	0	10	10	10	0
P1	1	6	16	15	9
P2	3	2	18	15	13
P3	5	4	22	17	13

Average Turn Around Time : (10+15+15+17)/4 = 57/4 = 14.25 ms

Average Waiting Time : (0+9+13+13)/4 = 35/4 = 8.75 ms

2. Shortest Job First (SJF) / Shortest Process Next (SPN)

Selection criteria:

The process, that requires shortest time to complete execution. is served first.

Decision Mode:

Non-preemptive: Once a process is selected, it runs until for a I/O or some event. or it terminates.

Implementation:

This strategy can be implemented by using sorted FIFO queue. All in a queue are sorted in ascending order based on their required CPU bursts. When CPU becomes free, a process from the first position in a queue is selected to run.

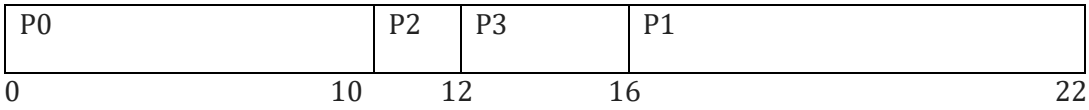
Example:

Consider the following set of four processes. There arrival time and time required to complete the execution are given in following table. Consider time values in milliseconds. Calculate Average Turn Around Time and Average Waiting Time.

Process	Arrival Time(T0)	Time Required for Completion
P0	0	10
P1	1	6
P2	3	2

P3	5	4

Gantt Chart:



Statistics:

Process	Arrival Time	Burst Time	Finish Time	Turn Around Time(TAT) (Finish Time – Arrival Time)	Waiting Time (TAT – Burst Time)
P0	0	10	10	10	0
P1	1	6	22	21	15
P2	3	2	12	9	7
P3	5	4	16	11	7

Average Turn Around Time : (10+21+9+11)/4 = 51/4 =12.75 ms

Average Waiting Time : (0+15+7+7)/4 = 29/4 = 7.25 ms

3. Shortest Remaining Time Next (SRTN)

Selection criteria:

The process, whose remaining run time is shortest, is served first. This a preemptive version of SJF scheduling.

Decision Mode:

Preemptive: When a new process arrives, its total time is compared to the current process' remaining time. If the new job needs less time to finish than the current process, the current process is suspended and the new job started.

Implementation: This strategy can also be implemented by using sorted FIFO queue. All processes in a queue are sorted in ascending order based on their remaining run time. When CPU becomes free, a process from the first position in a queue is selected to run.

Example:

Consider the following set of four processes. Their arrival time and time required to complete the execution are given in following table. Consider all time values in milliseconds.

Process	Arrival Time(T0)	Time Required for Completion
P0	0	10

P1	1	6
P2	3	2
P3	5	4

Gantt Chart:

P0	P1	P2	P1	P3	P0
0	1	3	5	9	13 22

Initially only process PO is present and (it is allowed to run. But, when PI comes, it has shortest remaining run time. So, PO is preempted and PI is allowed to run. Whenever new process comes or current process blocks, such type of decision is taken. This procedure is repeated till all processes complete their execution.

Statistics:

Process	Arrival Time	Burst Time	Finish Time	Turn Around Time(TAT) (Finish Time – Arrival Time)	Waiting Time (TAT – Burst Time)
P0	0	10	22	22	12
P1	1	6	9	8	2
P2	3	2	5	2	0
P3	5	4	13	8	4

Average Turn Around Time : (22+8+2+8)/4 = 40/4 =10.0 ms

Average Waiting Time : (12+2+0+4)/4 = 18/4 = 4.5 ms

4. Round Robin (RR) Scheduling

Selection criteria:
The process that requests first is served first. It means that processes served in the exact order as they come. The selection. criteria are same as of FCFS scheduling.

Decision Mode:
Preemptive: Each selected process is assigned a time interval, called time interval called time quantum or time slice. Process is allowed to run only for this time interval.

Here, one of two things is possible: First, process needs CPU burst less than time quantum. In this case, process will voluntarily release CPU and will be moved to the end of the queue. Second, process needs CPU burst longer than time quantum. In this case, process will be running at the end time quantum. Now, it will be preempted and moved to the end of the queue CPU will be allocated to another process.

Here, length of time quantum is critical to determine.

Implementation:
This strategy can be implemented by using circular FIFO queue. When new process comes, or process releases CPU, or process is preempted, it is moved to the end of the queue. When CPU becomes free. a process from the first position in a queue is selected to run.

Example: Consider the following set of four processes. Their arrival time and time required to complete the execution are given in following table. Consider all time values in milliseconds with Quantum time of 2 Milliseconds. Calculate **Average Turn Around Time** and **Average Waiting Time**

Process	Arrival Time(T0)	Time Required for Completion
P1	0	5
P2	1	4
P3	2	2
P4	4	1

Solution:
For Round Robin Scheduling Algorithm, it is compulsory to maintain the ready queue to ease the context switching and preemption of the process

Ready Queue:

P1	P2	P3	P1	P4	P2	P1

Gantt Chart:

P1	P2	P3	P1	P4	P2	P1

0 2 4 6 8 9 11 12

Statistics:

Process	Arrival Time(T0)	Time Required for Completion	Completion Time	Turn Around Time [CT - AT]	Waiting Time [TAT -BT]
P1	0	5	12	12	7
P2	1	4	11	10	6
P3	2	2	6	4	2
P4	4	1	9	5	4

Average Turn Around Time $= (12+10+4+5)/4 = 7.75$ ms

Average Waiting Time = $(7+6+2+4)/4 = 4.75$ ms

5. Priority based Non-Preemptive Scheduling

Selection criteria:
The process, that has highest priority, is served first.

Decision Mode:
Non-preemptive: Once a process is selected, it runs until it blocks for I/O or some event, or it terminates.

Implementation:
This strategy can be implemented by using sorted FIFO queue. All process in a queue is sorted based on their priority with highest priority process front end. When CPU becomes free, a process from the first position in queue is selected to run.

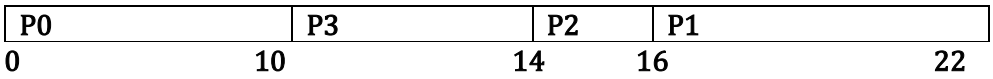
Example:
Consider the following set of four processes. Their arrival time, total time required to complete the execution and priorities are given in following table.

Consider all time values in milliseconds and small value for priority means higher priority of a process.

Process	Arrival Time(T0)	Time Required for Completion	Priority
P0	0	10	5
P1	1	6	4
P2	3	2	2
P3	5	4	1

Here, process priorities are in this order: P3 > P2 > P1 > P0.

Gantt Chart:



Statistics :

Process	Arrival Time	Burst Time	Finish Time	Turn Around Time(TAT) (Finish Time – Arrival Time)	Waiting Time (TAT – Burst Time)
P0	0	10	10	10	0
P1	1	6	22	21	15
P2	3	2	16	13	11
P3	5	4	14	9	5

Average Turn Around Time : (10+21+13+9)/4 = 53/4 =13.25 ms

Average Waiting Time : (0+15+11+5)/4 = 31/4 = 7.75 ms

6. Priority Based Preemptive Scheduling

Selection criteria:

The process, that has highest priority, is served first.

Decision Mode:

Preemptive: When a new process arrivesy its priority is compared to the current process' priority. If the new job has higher priority than the current process, the current process is suspended and the new job started.

Implementation:

This strategy can be implemented by using sorted FIFO queue. All processes in a queue are sorted based on their priority with highest priority process at front end. When CPU becomes free, a process from the first position in a queue is selected to run.

Example:

Process	Arrival Time(T0)	Time Required for Completion	Priority
P0	0	10	5
P1	1	6	4
P2	3	2	2
P3	5	4	1

Here, process priorities are in this order: P3 > P2 > P1 > P0.

P0	P1	P2	P3	P1	P0
0	1	3	5	9	13 22

Initially only process PO is present, and it is allowed to run. But, when P1 comes, it has higher priority. So, PO is preempted, and P1 is allowed to run. This procedure repeated till all processes complete their execution.

Statistics:

Process	Arrival Time	Burst Time	Finish Time	Turn Around Time(TAT) (Finish Time – Arrival Time)	Waiting Time (TAT – Burst Time)
P0	0	10	22	22	12

P1	1	6	13	12	6
P2	3	2	5	2	0
P3	5	4	9	4	0

Average Turn Around Time : (22+12+2+4)/4 = 40/4 =10.0 ms

Average Waiting Time : (12+6+0+0)/4 = 18/4 = 4.5 ms

7. Multilevel Queue Scheduling (MLQ)

It is possible to classify various processes into different groups. Such groups can be constructed, based on criteria such as similar priorities. required response-time. Each such group will "require different scheduling needs.

For example, a simple classification can be made between foreground (or interactive) processes and background (or batch) processes. Foreground processes will have higher priorities compared to background processes. Foreground processes will need smaller response-time compared to background processes. In other words, these both groups have different scheduling needs, and require different scheduling algorithms.

A multilevel queue scheduling algorithm partitions the ready queue into several separate queues. Each queue represents single group. Processes are assigned to related queues based on classification criteria. Each queue may contain its own scheduling algorithm.

For example, separate queues can be used for foreground processes and background processes. The foreground queue may use Round Robin scheduling algorithm, while the background queue may use FCFS algorithm. So, processes in each group are scheduled using different scheduling algorithms.

Along with this, scheduling must also be done between queues. This can be done using Priority-based scheduling algorithm, having foreground queue higher priority comparatively. In this case, background processes can only be executed if the foreground queue is empty. And this may result in starvation for background processes.

Another possibility is to assign fixed time slice to each queue. In this case, each queue gets a certain portion of the CPU time. This time duration can then be used to schedule processes in that queue. In our example, foreground queue can be given of the CPU time for RR scheduling among its processes, where as background queue can be given 20% of the CPU time for FCFS scheduling among its processes.

The following below figure depicts five different queues representing five different process-groups as given below:

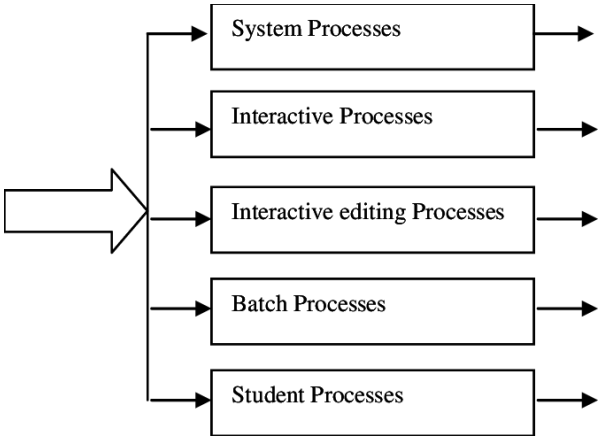
- System Processes.
- Interactive Processes.
- Interactive Editing Processes.

- Batch Processes.
- Student Processes.

This figure also depicts that System Processes possess highest priority, while Student Processes possess lowest priority.

If these queues are scheduled using Priority-based scheduling algorithm, then, Batch processes can only be executed if three other higher queues are empty. As an alternative, each queue can be assigned a fixed amount of CPU time, during which, processes from that queue will be scheduled.

Highest Priority



Lowest Priority