/* This note has been compiled from various sources and can be taken as reference only*/

Chapter 2: Process Management

Process

A process is defined as an entity which represents the basic unit of work to be implemented in the system i.e., a process is a program in execution. The execution of a process must progress in a sequential fashion. In general, a process will need certain resources such as the CPU time, memory, files, I/O devices and etc. to accomplish its task.

Operation on Process

- Process Creation
- Process Termination
- Run a process
- Change a process priority
- Get process information
- Set process information

Process Creation

It's a job of OS to create a process. There are four ways achieving it:

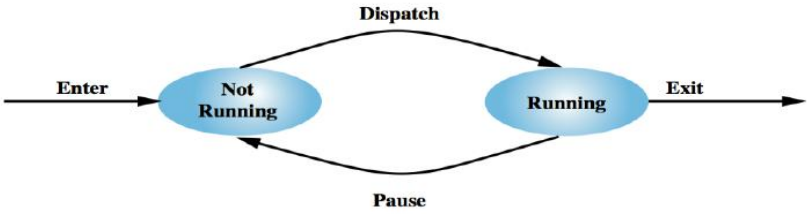
1. For a batch environment a process is created in response to submission of a job.
2. In Interactive environment, a process is created when a new user attempt to log on.
3. The OS can create process to perform functions on the behalf a user program.
4. A number of processes can be generated from the main process. For the purpose of modularity or to exploit parallelism a user can create numbers of process.

Process Termination

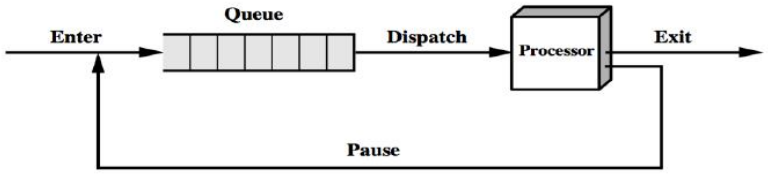
A process terminates when it finishes executing its last statement. Its resources are returned to the system, it is purged from any system lists or tables, and its process control block (PCB) is erased. The new process terminates the existing process, usually due to following reasons:

- **Normal Exit (Voluntary):** Most processes terminate because they have done their job.
- **Error Exit (Voluntary):** When process discovers a fatal error. For example, a user tries to compile a program that does not exist.
- **Program Error (Involuntary):** An error caused by process due to a bug in program for example, executing an illegal instruction, referring non-existing memory or dividing by zero.
- **Killed by another Process (Involuntary):** Process executes a system call telling the Operating Systems to terminate some other process. In UNIX, this call is kill.

Process State (Two state model)



(a) State transition diagram



(b) Queuing diagram

In this model, we consider two main states of the process. These two states are

State 1: Process is Running on CPU
State 2: Process is Not Running on CPU

New: First of all, when a new process is created, then it is in Not Running State. Suppose a new process P1 is created then P1 is in NOT Running State.

CPU: When CPU becomes free, Dispatcher gives control of the CPU to P1 that is in NOT Running state and waiting in a queue.

Dispatcher: Dispatcher is a program that gives control of the CPU to the process selected by the CPU scheduler. Suppose dispatcher allow P1 to execute on CPU.

Running: When dispatcher allows P1 to execute on CPU then P1 starts its execution. Here we can say that P1 is in running state.
Now, if any process with high priority wants to execute on CPU, Suppose P2 with high priority, then P1 should be paused or we can say that P1 will be in waiting state and P2 will be in running state. Now, when P2 terminates then P1 again allows the dispatcher to execute on CPU

Process State (Five state model)

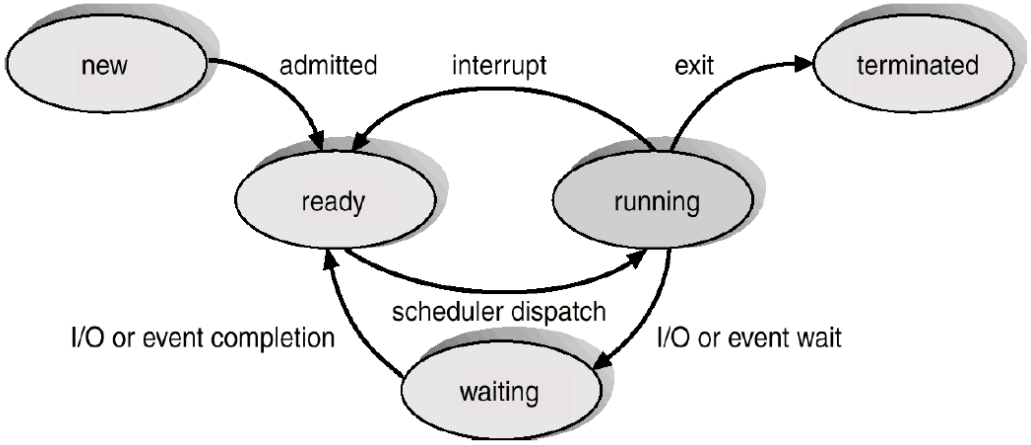


Fig: Five States Process Model

1. New: A process that has just been created but has not yet been admitted to the pool of executable processes by the OS.
2. Ready: Process that is prepared to execute when given the opportunity. That is, they are not waiting on anything except the CPU availability.
3. Running: the process that is currently being executed.
4. Blocked: A process that cannot execute until some event occurs, such as the completion of an I/O operation.
5. Exit: A process that has been released from the pool of executable processes by the OS, either because it is halted or because it is aborted for some reason A process that has been released by OS either after normal termination or after abnormal termination (error).

Process Control Block (PCB)

A process control block or PCB is a data structure (a table) that holds information about a process. Every process or program that runs needs a PCB. When a user requests to run a particular program, the operating system constructs a process control block for that program. It is also known as Task Control Block (TCB). The process control block typically contains:

- An ID number that identifies the process
- Pointers to the locations in the program and its data where processing last occurred
- Register contents
- States of various flags and switches
- Pointers to the upper and lower bounds of the memory required for the process
- A list of files opened by the process
- The priority of the process
- The status of all I/O devices needed by the process

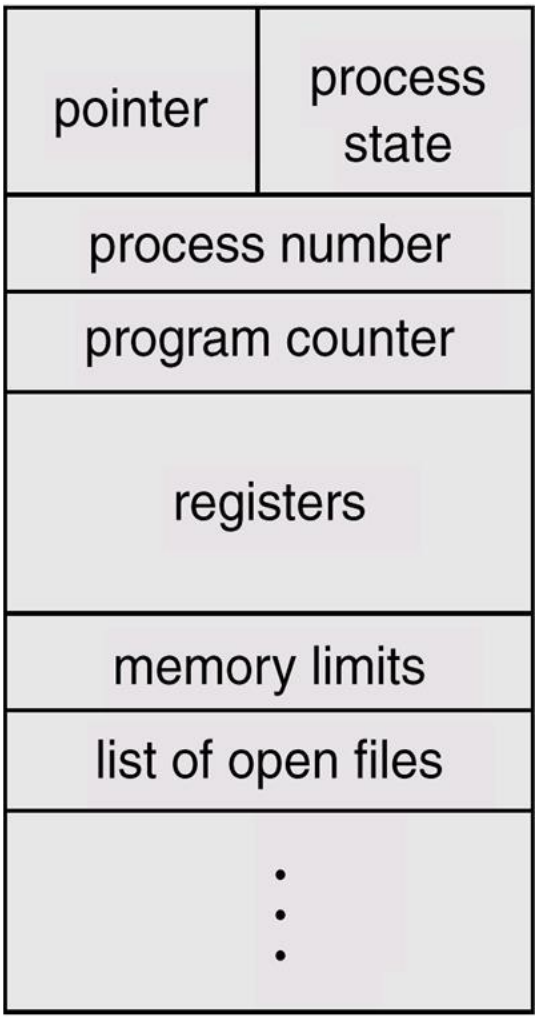


Fig Process Control Block

The information stored in the Process Control Block in given below

 Process State: The state may be new, ready, running, and waiting, halted, and so on.

 Program Counter: the counter indicates the address of the next instruction to be executed for this process.

 CPU register: The registers vary in number and type, depending on the computer architecture. They include accumulator, index registers, stack pointers, and general-purpose registers. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward.

 CPU Scheduling information: This information includes a process priority, pointers to scheduling queues, and other scheduling parameters.

 Memory management information: this information includes the value of the base and limit registers, the page table, or the segment tables, depending on the memory system used by the OS.

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

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

Difference Between a Program and process – H.W. given

Program	Process
Consists of set of instructions in programming language	It is a sequence of instruction execution
It is a static object existing in a file form	It is a dynamic object (i.e. program in execution)
Program is loaded into secondary storage device	Process is loaded into main memory
The time span is unlimited	Time span is limited
It is a passive entity	It is an active entity

Scheduler

Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run.

Types of Schedulers

- Long term scheduler
- Mid - term scheduler
- Short term scheduler

1. Long Term Scheduler

It selects the process that are to be placed in ready queue. The long term scheduler basically decides the priority in which processes must be placed in main memory. Processes of long term scheduler are placed in the ready state because in this state the process is ready to execute waiting for calls of execution from CPU which takes time that's why this is known as long term scheduler.

2. Mid – Term Scheduler

It places the blocked and suspended processes in the secondary memory of a computer system. The task of moving from main memory to secondary memory is called **swapping out**. The task of moving back a swapped-out process from secondary memory to main memory is known as **swapping in**. The swapping of processes is performed to ensure the best utilization of main memory.

3. Short Term Scheduler

It decides the priority in which processes is in the ready queue are allocated the central processing unit (CPU) time for their execution. The short term scheduler is also referred as central processing unit (CPU) scheduler.

Dispatcher

The dispatcher is the module that gives control of the CPU to the process selected by the short-time scheduler (selects from among the processes that are ready to execute).

The function involves:

- Context Switching
- Switching to user mode
- Restart the execution of process

Context Switching

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. Using this technique, a context switcher enables multiple processes to share a single CPU. Context switching is an essential part of a multitasking operating system features. When the scheduler switches the CPU from executing one process to execute another, the state from the current running process is stored into the process control block. After this, the state for the process to run next is loaded from its own PCB and used to set the PC, registers, etc. At that point, the second process can start executing.

The speed of context switching depends on:

- The speed of memory
- The no. of registers that must be copied
- The existence of some special instruction

Scheduling Criteria

Scheduling criteria is also called as scheduling methodology. Key to multiprogramming is scheduling. Different CPU scheduling algorithm have different properties .The criteria used for comparing these algorithms include the following:

- **CPU Utilization:**

Keep the CPU as busy as possible. It range from 0 to 100%. In practice, it range from 40 to 90%.

- **Throughput:** Throughput is the rate at which processes are completed per unit of time.

- **Turnaround time:**

This is the how long a process takes to execute a process. It is calculated as the time gap between the submission of a process and its completion.

- **Waiting time:**

Waiting time is the sum of the time periods spent in waiting in the ready queue.

- **Response time:**

Response time is the time it takes to start responding from submission time. It is calculated as the amount of time it takes from when a request was submitted until the first response is produced.

- **Fairness:** Each process should have a fair share of CPU.

Types of Process Scheduling

Process scheduling can be categorized based on the timing and nature of CPU allocation:

1. Preemptive Scheduling

- **Definition**: The OS can interrupt and suspend a currently running process to assign the CPU to another process.
- **Use Case**: Suitable for time-sharing and real-time systems where responsiveness is critical.

2. Non-Preemptive Scheduling

- **Definition**: Once a process starts executing, it runs to completion or until it voluntarily yields control (e.g., waiting for I/O).
- **Use Case**: Simpler to implement; used in batch systems where processes are executed without interruption.

Difference between Preemptive and Non Preemptive Scheduling

PREEMPTIVE SCHEDULING	NON PREEMPTIVE SCHEDULING
The resources are allocated to a process for a limited time.	Once resources are allocated to a process, the process holds it till it completes its burst time or switches to waiting state.
Process can be interrupted in between.	Process cannot be interrupted till it terminates or switches to waiting state.
If a high priority process frequently arrives in the ready queue, low priority process may starve.	If a process with long burst time is running CPU, then another process with less CPU burst time may starve.
Preemptive scheduling has overheads of scheduling the processes.	Non-preemptive scheduling does not have overheads.
Preemptive scheduling is flexible.	Non-preemptive scheduling is rigid.
Preemptive scheduling is cost associated.	Non-preemptive scheduling is not cost associative.

Some Common Process Scheduling Algorithm

1. First Come First Serve (FCFS)

- **Definition**: A non-preemptive scheduling algorithm where processes are executed in the order they arrive in the ready queue.
- **Advantages**:
  - Simple and easy to implement.
  - Fair in terms of process arrival time.
- **Disadvantages**:
  - Can lead to the "convoy effect," where short processes wait behind long ones.
  - Poor average waiting time.

---

2. Shortest Job First (SJF)

- **Definition**: A non-preemptive scheduling algorithm that selects the process with the smallest execution time.
- **Advantages**:
  - Minimizes average waiting time.
  - Efficient for batch processing.
- **Disadvantages**:
  - Requires accurate prediction of burst times.
  - Can cause starvation for longer processes.

---

3. Shortest Remaining Time First (SRTF)

- **Definition**: A preemptive version of SJF where the process with the smallest remaining execution time is selected next.
- **Advantages**:
  - Minimizes average waiting time.
  - Efficient for short processes.
- **Disadvantages**:
  - Requires precise knowledge of remaining burst times.
  - Longer processes may suffer from starvation.

---

4. Round Robin (RR)

- **Definition**: A preemptive algorithm where each process is assigned a fixed time quantum in a cyclic order.
- **Advantages**:
  - Fair allocation of CPU time.
  - Suitable for time-sharing systems.
- **Disadvantages**:
  - Performance depends on the choice of time quantum.
  - High context switching overhead if the time quantum is too small.

5. Highest Response Ratio Next (HRRN)

- **Definition**: HRRN is a non-preemptive scheduling algorithm that selects the next process to execute based on the highest response ratio. The response ratio is calculated as:

$$\text{Response Ratio} = \frac{\text{Waiting Time} + \text{Burst Time}}{\text{Burst Time}}$$

  This approach prioritizes processes that have been waiting longer and those with shorter burst times, effectively reducing starvation issues present in algorithms like Shortest Job First (SJF).

- **Advantages**:
  - Reduces the chance of starvation by increasing the priority of waiting processes over time.
  - Balances between short and long processes, leading to improved overall system performance.
- **Disadvantages**:
  - Requires accurate estimation of burst times, which may not always be feasible.
  - Being non-preemptive, it may not be suitable for time-sharing systems where responsiveness is critical.

6. Completely Fair Scheduling

- will be studied after the concept of Tree in DSA

Numerical for all algorithms

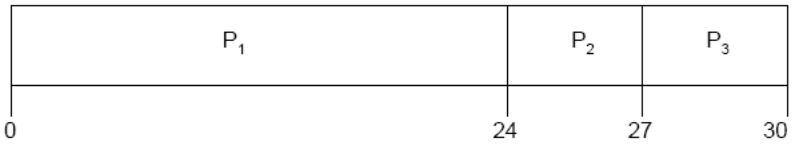
1. FCFS

Example

Q1. Consider the following set of processes that arrives at time 0, with the length of the CPU burst given in milliseconds:

Process	Burst Time
P1	24
P2	3
P3	3

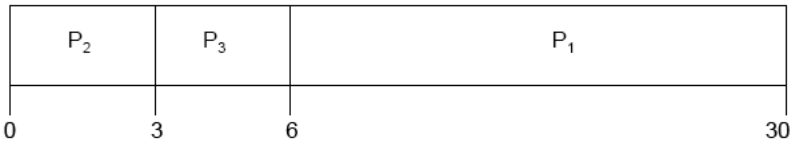
(a) Suppose that the processes arrive in the order: *P1, P2,* and *P3*
The Gantt chart for the schedule is:



Waiting time for *P1* = 0; *P2* = 24; *P3* = 27

- ✓ Average waiting time: (0 + 24 + 27)/3 = 17 ms
- ✓ Turnaround time for *P1* = 24; *P2* = 27; *P3* = 30
- ✓ Average Turnaround time: (24 + 27 + 30)/3 = 27 ms

(b) Suppose that the processes arrive in the order: *P2, P3, and P1*
The Gantt chart for the schedule is:



- ✓ Waiting time for *P1* = 6*; P2* = 0*; P3* = 3
- ✓ Average waiting time: (6 + 0 + 3)/3 = 3 milliseconds
- ✓ Turnaround time for *P1* = 30; *P2* = 3; *P3* = 6
- ✓ Average Turnaround time: (30 + 3 + 6)/3 = 13 ms

Q2. Consider the following set of processes that arrives at time 0, with the length of the CPU burst given in milliseconds:

Process	Arrival Time	Burst Time
P_1	0	10
P_2	1	6
P_3	3	2
P4	5	4

Find:
1. Average Turnaround Time (ATAT) (*ans* 14.25)
2. Average Waiting Time (AWT) (*ans* 8.75)
3. Waited TAT (WTAT) (*ans* 1.295)
4. Average WTAT (AWTAT) (*ans* 14.25)

Solutions

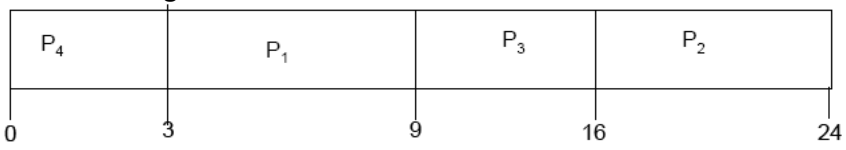
1. Average Turnaround Time (ATAT) = (10+15+15+17)/4 = 14.25
2. Average Waiting Time (AWT) = (0+9+13+13) /4 = 8.75
3. Waited TAT (WTAT) $WTAT_i = TAT_i / RT_i$ where i = 1 to n
= (10+15+15+17)/ (0+10+16+18)
= 1.295
4. Average WTAT (AWTAT) = WTAT/n = (10+15+15+17)/4 = 14.25

2. SJF

Q1. Consider the following set of processes that arrives at time 0, with the length of the CPU burst given in milliseconds:

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

SJF scheduling chart



The waiting time for process P1= 3, P2 = 16, P3 = 9 and P4 = 0 milliseconds, Thus

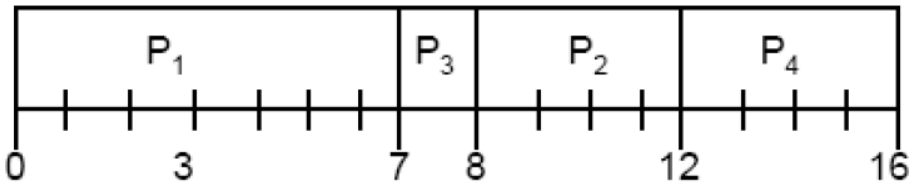
✓ Average waiting time = (3 + 16 + 9 + 0) / 4 = 7 milliseconds

A preemptive SJF algorithm will preempt the currently executing, where as a non-preemptive SJF algorithm will allow the currently running process to finish its CPU burst. Preemptive SJF scheduling is also known **Shortest-Remaining-time (SRT) First Scheduling.**

Example: Non-Preemptive SJF

Process	Arrival time	Burst time
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4

• SJF (non-preemptive) Gant Chart



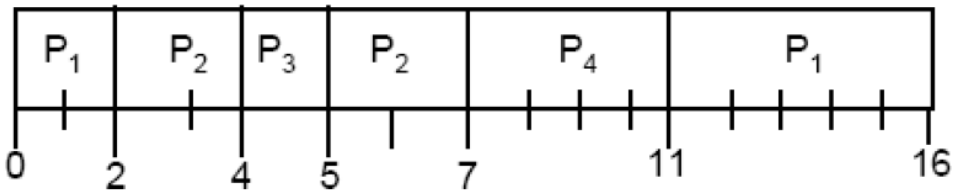
✓ Average waiting time

= (P1 + P2 + P3 + P4) / 4
= [(0 – 0) + (8 – 2) + (7 – 4) + (12 – 5)] / 4
= (0 + 6 + 3 + 7) / 4

Example: Preemptive SJF

Process	Arrival time	Burst time
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4

• SJF (preemptive) (Shortest Remaining Time First, SRTF) Gantt chart



• Average waiting time

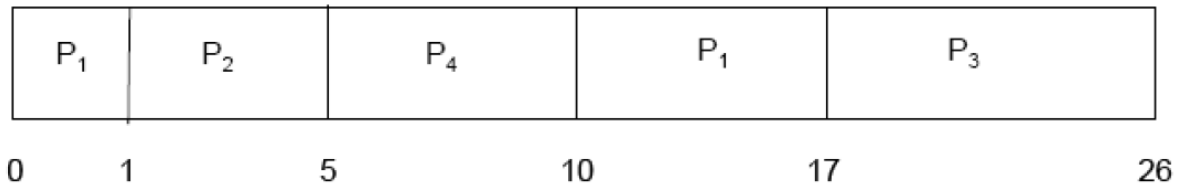
= [(11 – 2) + (5 – 4) + (4 – 4) + (7 – 5)] / 4
= (9 + 1 + 0 +2) / 4
= 3 milliseconds

3. SRTN

It is a preemptive version of SJF algorithm where the remaining processing time is considered for assigning CPU to the next process

Process	Arrival time	Burst time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

✓ *Preemptive* SJF Gantt Chart



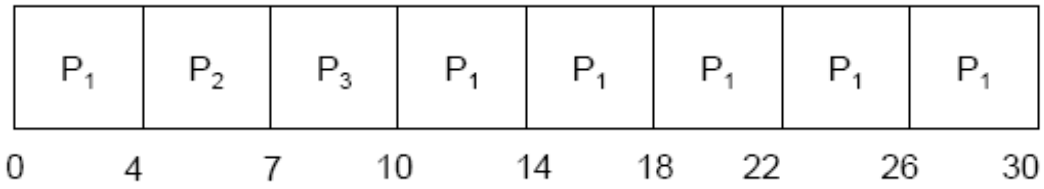
✓ Average waiting time = [(10-1) +(1-1) +(17-2) +5-3)]/4 = 26/4 = 6.5 milliseconds

Round Robin (RR)

Q1. Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds.
Example 1: Quantum time = 4

Process	Burst Time
P1	24
P2	3
P3	3

The Gantt chart is:



The process P1 gets the first 4 milliseconds. Since it requires another 20 milliseconds, it is preempted after the first-time quantum, and the CPU is given to the next process in the queue, process P2. Process P2 does not need 4 milliseconds, so it quits before its time quantum expires. The CPU is then given to the next process, process P3. Once each process has received 1-time quantum time, the CPU is returned to the process for an additional time quantum.

Average time: = (P1 + P2 + P3) / 3
= [(10 – 4) + 4 + 7] / 3
= 17 / 3
= 5.66 milliseconds

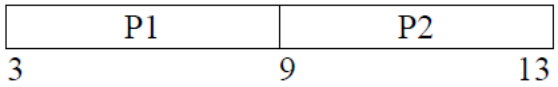
Highest Response Ration Next (HRRN)

Example:
Q1. Consider the Processes with following Arrival time, Burst Time and priorities

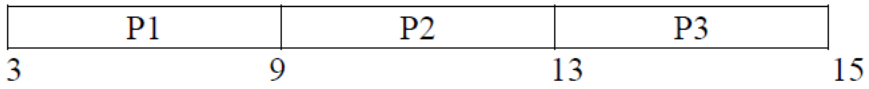
Process	**Arrival time**	**Burst time**
P1	0	3
P2	2	6
P3	4	4
P4	6	5
P5	8	2

- At time 0 only process p1 is available, so p1 is considered for execution

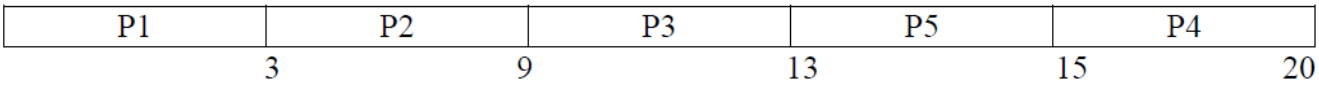
- At time 3 only process p2 is available, so p2 is considered for execution
- Since at time 9 all process has arrived so we have to calculate Response Ratio for these process



Response Ratio for P3 = (5 + 4)/4 = 2.25.
Response Ratio for P4 = (3 + 5)/5 = 1.6
Response Ratio for P5 = (1+2)/2 = 1.5



Response Ratio for P4= (7+ 5)/5= 2.4
Response Ratio for P5 = (5+2)/2 = 3.5



Calculate Average Waiting Time and Average Turn Around Time.

1. **Completely Fair Scheduler (CFS)**
   - ⇨ **Later after Concept of Tree-> Remind me**
2. **Multilevel Queues**

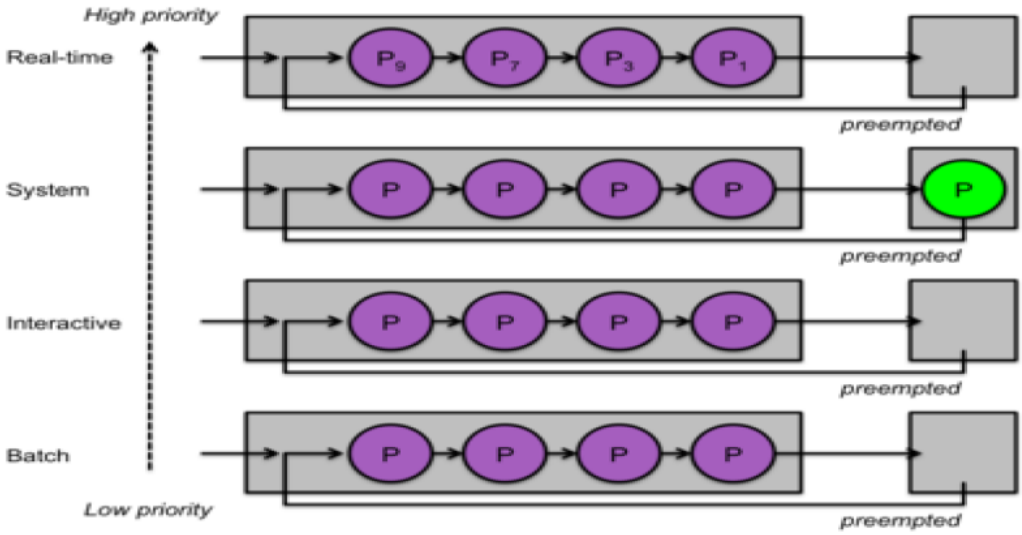
- Multi-level queue scheduling ***was created for situation in which processes are easily classified into different groups***. In multilevel queue Processes are divided into different queue based on their type. ***Process is permanently assigned to one queue, generally based on some property of process*** i.e., system process, interactive, batch system, end user process, memory size, process priority and process type. ***Each queue has its own scheduling algorithm***. For example, interactive process may use round robin scheduling method, while batch job uses the FCFS method.

In addition, there must be scheduling among the queue and is generally implemented as fixed priority preemptive scheduling. Foreground process may have higher priority over the background process.

Features of Multilevel Queue (MLQ) CPU Scheduling

- **Multiple Queues:** In MLQ scheduling, processes are divided into multiple queues based on their priority, with each queue having a different priority level. Higher-priority processes are placed in queues with higher priority levels, while lower-priority processes are placed in queues with lower priority levels.
- **Priorities Assigned:** Priorities are assigned to processes based on their type, characteristics, and importance. For example, interactive processes like user input/output may have a higher priority than batch processes like file backups.

- **Preemption:** Preemption is allowed in MLQ scheduling, which means a higher priority process can preempt a lower priority process, and the CPU is allocated to the higher priority process. This helps ensure that high-priority processes are executed in a timely manner.
- **Scheduling Algorithm:** Different scheduling algorithms can be used for each queue, depending on the requirements of the processes in that queue. For example, Round Robin scheduling may be used for interactive processes, while First Come First Serve scheduling may be used for batch processes.
- **Feedback Mechanism:** A feedback mechanism can be implemented to adjust the priority of a process based on its behavior over time. For example, if an interactive process has been waiting in a lower-priority queue for a long time, its priority may be increased to ensure it is executed in a timely manner.
- **Efficient Allocation of CPU Time:** MLQ scheduling ensures that processes with higher priority levels are executed in a timely manner, while still allowing lower priority processes to execute when the CPU is idle.
- **Fairness:** MLQ scheduling provides a fair allocation of CPU time to different types of processes, based on their priority and requirements.
- **Customizable:** MLQ scheduling can be customized to meet the specific requirements of different types of processes.

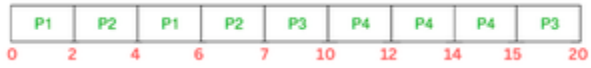


Example Problem:
Consider the below table of four processes under Multilevel queue scheduling. Queue number denotes the queue of the process.

Process	Arrival Time	CPU Burst Time	Queue Number
P1	0	4	1
P2	0	3	1
P3	0	8	2
P4	10	5	1

Priority of queue 1 is greater than queue 2. queue 1 uses Round Robin (Time Quantum = 2) and queue 2 uses FCFS.

Below is the **Gantt chart** of the problem:



Gantt Chart

Working

- At starting, both queues have process so process in queue 1 (P1, P2) runs first (because of higher priority) in the round-robin fashion and completes after 7 units
- Then process in queue 2 (P3) starts running (as there is no process in queue 1) but while it is running P4 comes in queue 1 and interrupts P3 and start running for 5 seconds and
- After its completion P3 takes the CPU and completes its execution.

What is a Thread?

A thread is a single sequence stream within in a process. Because threads have some of the properties of processes, they are sometimes called lightweight processes. In many respects, threads are popular way to improve application through parallelism. The CPU switches rapidly back and forth among the threads giving illusion that the threads are running in parallel. Like a traditional process i.e., process with one thread, a thread can be in any of several states (Running, Blocked, Ready or terminated). A thread consists of a program counter (PC), a register set, and a stack space. Threads are not independent of one other like process. Threads shares address space, program code, global variables, OS resources with other thread.

A **thread** is the smallest unit of CPU execution. It exists within a **process**, which is a running program. A process can have **multiple threads** (also called **lightweight processes**) sharing the same memory space.

Processes Vs Threads

As we mentioned earlier that in many respect threads operate in the same way as that of processes. Some of the similarities and differences are:

Similarities

 Like processes threads share CPU and only one thread active (running) at a time.
 Like processes, threads within processes execute sequentially.
 Like processes, thread can create children.
 And like process, if one thread is blocked, another thread can run.

Difference Between Process and Thread

Feature	Process	Thread
Memory	Separate memory	Shared memory within a process
Overhead	High (context switching)	Low (lightweight context switch)
Communication	IPC (Inter-Process Communication)	Easier via shared memory
Failure	One process fails → isolated	One thread fails → may affect others

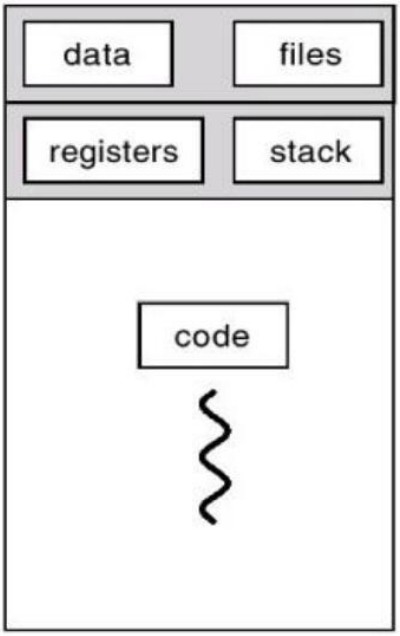


Fig Process with single thread

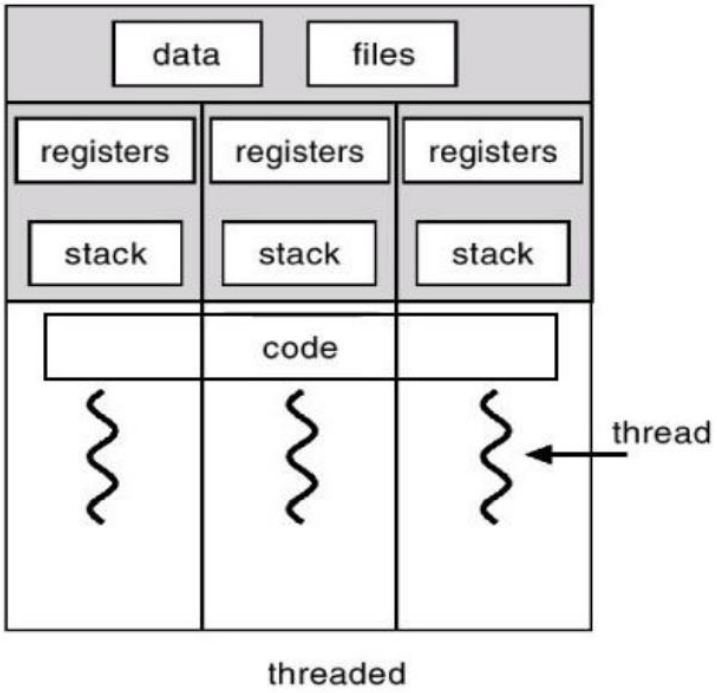


Fig: Process With multiple thread

Why Threads?

 Process with multiple threads makes a great server (e.g., print server)
 Increase responsiveness, i.e. with multiple threads in a process, if one threads blocks then other can still continue executing
 Sharing of common data reduce requirement of inter-process communication
 Proper utilization of multiprocessor by increasing concurrency
 Threads are cheap to create and use very little resources
 Context switching is fast (only have to save/reload PC, Stack, SP, Registers)

Thread Scheduling

What is Thread Scheduling?

Thread scheduling is the process of deciding which thread in a process (or across processes) gets to use the CPU next.

Threads can be **user-level** (managed by the user-space library) or **kernel-level** (managed by the OS). Scheduling varies based on this.

Thread scheduling is the OS's method of determining **which thread should run next** on the CPU(s). It's critical when:

- Multiple threads are ready to run
- There are limited CPU cores
- Threads are competing for CPU time

Efficient thread scheduling ensures:

- **Fairness** (no thread starves)
- **Responsiveness** (important in user interfaces)
- **High CPU utilization**
- **Load balancing** on multicore systems

Thread Scheduling Levels

Normally threads are scheduled on deadline and periods. Thread scheduling can happen at two levels:

1. User-Level Thread (ULT) Scheduling

- Managed by a thread library (not the OS)
- Fast context switch (no kernel mode switch)
- The OS is unaware of these threads

Drawback: If one ULT blocks (e.g., for I/O), all threads in that process may block

Example Libraries: POSIX pthread (if implemented at user level), green threads in Java

2. Kernel-Level Thread (KLT) Scheduling

- Managed by the OS kernel
- True parallelism on multicore systems
- OS decides which thread to run next

Example OSs: Windows, Linux, macOS — all support kernel threads

Thread Scheduling vs Process Scheduling

Feature	Thread Scheduling	Process Scheduling
Context Switch Cost	Low (threads share resources)	High (separate memory and resources)
Execution Context	Within a process	Independent
Use Case	Fine-grained parallelism	Coarse-grained multitasking

Types of Scheduling Approaches

☑ **Preemptive Scheduling**

- Threads are forcibly suspended after a **time slice** (quantum)
- Improves responsiveness
- OS uses a timer interrupt

Example: Most general-purpose OSs (e.g., Linux, Windows)

🛑 **Non-Preemptive Scheduling**

- A thread voluntarily gives up the CPU (yield or finish)
- Simpler, but may cause thread starvation

Example: Some real-time or embedded systems

Thread Scheduling Algorithms (with Examples)

1. First-Come, First-Served (FCFS)

- Simple queue structure
- Non-preemptive
- Poor responsiveness for long threads

T1 → T2 → T3 → T4

---

2. Round Robin (RR)

- Each thread gets a fixed **time quantum**
- Preemptive
- Fair, good for time-sharing systems

T1 (10ms) → T2 (10ms) → T3 (10ms) → T1 (10ms) ...

---

3. Priority Scheduling

- Threads assigned priority values
- High-priority threads run first
- Can be preemptive or non-preemptive

⚠️ **Problem:** Starvation — low-priority threads may never run
☑ **Solution:** Aging — increase priority over time

---

4. Multilevel Queue Scheduling

- Threads are divided into queues based on type or priority:
  - Interactive (high)
  - Batch (low)
- Each queue has its own scheduling algorithm

Queue 1 (High priority, RR)
Queue 2 (Medium, FCFS)
Queue 3 (Low, FCFS)

---

5. Multilevel Feedback Queue (MLFQ)

- Dynamic version of multilevel queue
- Threads can **move between queues** based on behavior
- Ideal for general-purpose OSs
