Process Scheduling

Categories of Scheduling Algorithms

Different environments need different scheduling algorithms

1. Batch Systems

- **Still widely used** in business and data processing.
- **Non-preemptive algorithms** (like FCFS, SJF) are preferred.

Why? Fewer process switches, which reduces overhead and is suitable for jobs that do not require user interaction.

2. Interactive Systems

- **Preemptive algorithms** are necessary (like Round Robin, Priority Scheduling).

Why? The system needs to switch between processes quickly to ensure responsive user interaction (e.g., multiple users or programs running at the same time).

3. Real-Time Systems

- Processes must **run quickly** and may **block** waiting for events.
- Require algorithms that guarantee deadlines and **predictable execution** (e.g., Rate Monotonic, Earliest Deadline First).

Scheduling Algorithm Goals

All Systems (General Scheduling Goals)

- **Fairness:** Every process should get a fair share of CPU time.
- **Policy Enforcement:** Ensure the scheduler follows the intended scheduling policy.
- **Balance:** Keep all system components (CPU, memory, I/O devices) as busy as possible.

1. Batch Systems:

- **Throughput:** Maximize the number of jobs completed per hour.
- **Turnaround Time:** Minimize the total time between job submission and completion.
- **CPU Utilization:** Keep the CPU busy all the time (avoid idle CPU).

2. Interactive Systems:

- **Response Time:** Respond to user requests as quickly as possible.
- **Proportionality:** Deliver results consistent with user expectations.

3. Real-Time Systems:

- **Meeting Deadlines:** Complete tasks within required time limits to avoid data loss or system failure.
- **Predictability:** Ensure consistent response times and quality (e.g., no jitter in multimedia applications).

Scheduling Criteria

- **Arrival time**: Time at which a process enters the ready queue or system.
- **Burst time**: Time required by a process to execute on the CPU.
- **Completion time**: Time at which a process completes its execution.
- **Turn Around Time**: Total time taken from arrival to completion. In simple words, it is the difference between the Completion time and the Arrival time.

Turn Around Time = Completion Time - Arrival Time

- **Waiting Time**: Time a process spends waiting in the ready queue. It is the difference between the Turn Around time and the Burst time of the process.

  Waiting Time = Turn around time - Burst Time

- **Response Time:** The time interval between a process's arrival in the ready queue and its first execution on the CPU.

  Response Time=Start Time-Arrival Time

- **Response Ratio :** Ratio used to prioritise processes based on waiting and burst time.

  Response Ratio= (Waiting Time + Burst time) / Burst time

- **CPU utilization :** It measures how effectively the CPU is being used

  CPU utilization= (CPU Busy time/ Total Time)*100

Example: Total execution time (all burst times) = 20 ms, CPU idle time = 5 ms then, Total time = 25 ms.

CPU utilization=(20/25)*100=80%

- **Throughput :** Number of completed jobs per time unit**.**

  Throughput= Number of process completed/ Total time

Example: 5 processes completed, Total time from start to end = 20 ms.

Throughput=5/20=0/25 process per ms.

Scheduling in Batch Systems

- First-come first-served
- Shortest job first
- Shortest Remaining Time Next

First-Come First-Serve(FCFS) Scheduling

- Non-preemptive: Once a process starts executing, it runs to completion.
- Queue Type: FIFO (First-In, First-Out)- like waiting in line at a bank, post office, or for a printer.
- Runnable processes are added to the end of the ready queue.

Advantages:

- Simple and easy to implement.
- Fair in the sense that it serves in the order of arrival.

Disadvantages:

- Long average waiting times, especially if:

✓ A long job arrives before short jobs ("convoy effect").

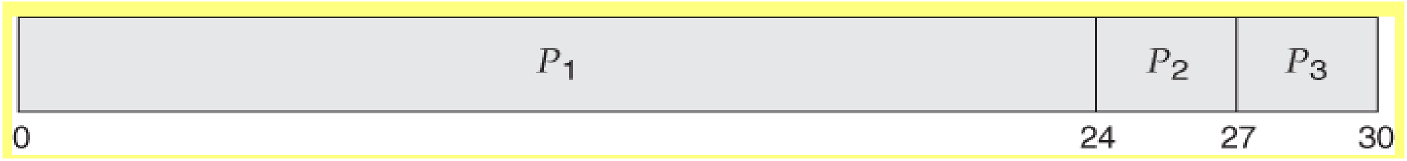
✓ The first process takes a long time, delaying others.

- Poor response time for short processes.

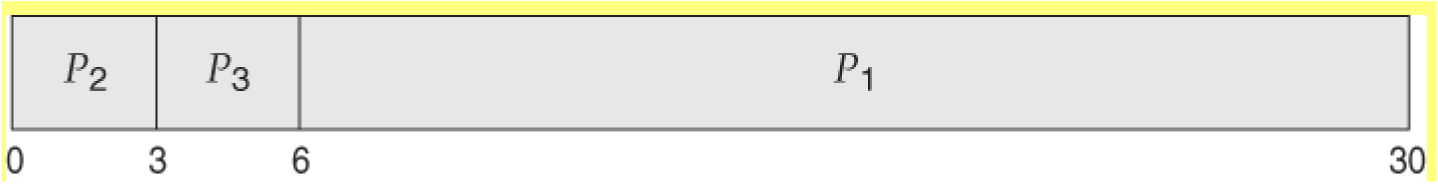
Example :1

Process	CPU Burst Time
P1	24
P2	3
P3	3

- In the first Gantt chart below, process P1 arrives first. Waiting time for *P1* = 0; *P2* = 24; *P3* = 27
- The average waiting time for the three processes is (0 + 24 + 27) / 3 = 17 ms.



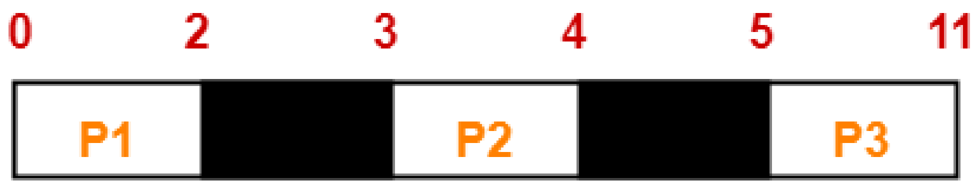
- Suppose that the processes arrive in the order *P2* , *P3* , *P1* ,Waiting time for *P1* = 6*;* *P2* = 0*;* *P3* = *3,*In the second Gantt chart below, the same three processes have an average wait time of (0 + 3 + 6) / 3 = 3 ms.



Example :2

Process Id	Arrival time	Burst time
P1	0	2
P2	3	1
P3	5	6

Gantt Chart: Here, black box represents the idle time of CPU.



Gantt Chart

Turn Around time = (Exit time/Completion time) – Arrival time

Waiting time = Turn Around time – (Burst time/CPU Execution time)

Process Id	Exit time	Turn Around time	Waiting time
P1	2	2 – 0 = 2	2 – 2 = 0
P2	4	4 – 3 = 1	1 – 1 = 0
P3	11	11- 5 = 6	6 – 6 = 0

Now,

Average Turn Around time = (2 + 1 + 6) / 3 = 9 / 3 = 3 unit

Average waiting time = (0 + 0 + 0) / 3 = 0 / 3 = 0 unit

Shortest-Job-First Scheduling (SJF)

- **Shortest Job First (SJF) / Shortest Job Next (SJN) / Shortest Process Next (SPN)**
- SJF is a CPU scheduling algorithm in which the **waiting process with the smallest execution (burst) time** is selected for the next execution.
- **Non-Preemptive SJF:** Once a process starts executing, it runs to completion. The scheduler always chooses the process with the **shortest next CPU burst** from the ready queue.
- **The shortest process executes first.**

Advantages:

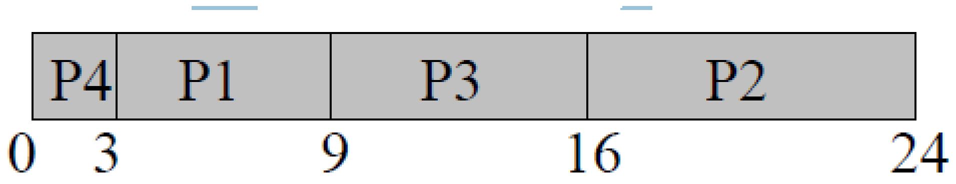
- Provides **minimum average waiting time** for a given set of processes.
- More efficient than FCFS for processes of varying lengths.

Disadvantages:

- **Difficult to predict the length** of the next CPU request.
- Can lead to **starvation** (long jobs might never get CPU if short jobs keep arriving).

Example :1

Process	CPU burst time
P1	6
P2	8
P3	7
P4	3



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

Example : 2

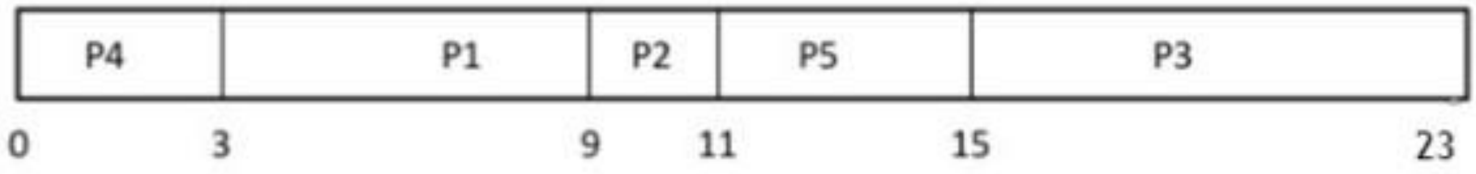
Process	Burst time/Execution time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

- ✓ **Completion time:** p1=9,p2=11,p3=23,p4=3, p5=15
- ✓ **Turn Around Time = (Completion Time - Arrival Time)**
- P1=9-2=7,p2=11-5=6,p3=23-1=22,p4=3-0=3,p5=15-4=11

Average Turn around time =7+6+22+3+11/5=49/5

- ✓ **Waiting Time = (Turn around time - Burst Time)**
- P4= 3-3=0 P1= 7-6=1 P2= 6-2=4 P5= 11-4=7 P3= 22-8=14

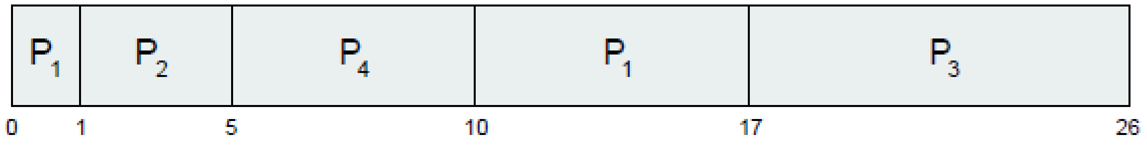
Average Waiting Time= 0+1+4+7+14/5 = 26/5 = 5.2



Shortest remaining time next

- A **preemptive** version of shortest job first is **shortest remaining time next**.
- A process with shortest burst time begins execution.
- If a process with even a shorter burst time arrives, the current process is removed or preempted from execution, and the shorter job is allocated CPU cycle.

Example :1



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

Average Waiting Time = ((10-1) +(1-1) + (17-2) +(5-3))/4 = 26/4= 6.5 ms

Scheduling in Interactive Systems

- Round-Robin Scheduling
- Priority Scheduling
- Multiple Queues
- Shortest Process Next
- Guaranteed Scheduling
- Fair-Share Scheduling
- Lottery Scheduling

Round-Robin Scheduling

- One of the Most widely used, oldest, fairest, and easiest algorithms is **round robin**.
- Round robin is a preemptive algorithm
- The CPU is shifted to the next process after fixed interval time, which is called time quantum/time slice.
- Process runs until it blocks or time quantum exceeded
- If the process has blocked or finished before the quantum has elapsed, the CPU switching is done when the process blocks, of course.

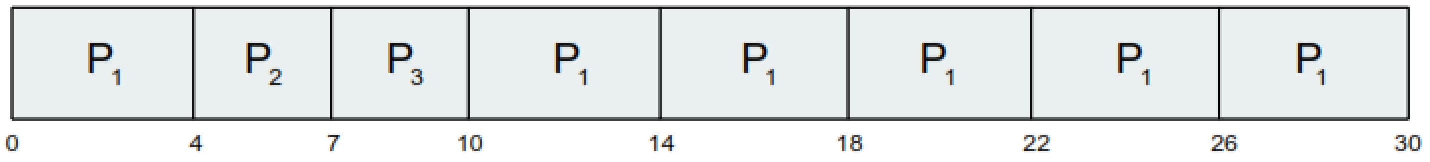
Example :1

Process	Burst time
P1	24
P2	3
P3	3

Time Slice=4.

Ready Queue: P1,P2,P3,P1

The Gantt chart assuming all processes arrive at time 0 is:



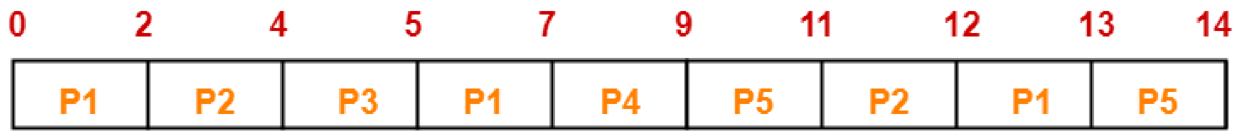
Average Waiting Time={(10-4)+(4-0)+(7-0)}/3=17/3=5.66 ms.

Example :2

If the CPU scheduling policy is Round Robin with time quantum = 2 unit, calculate the average waiting time and average turn around time.

Process	Arrival time	Burst time
P1	0	5
P2	1	3
P3	2	1
P4	3	2
P5	4	3

Ready Queue: P1, P2,P3,P1, P4, P5,P2, P1,P5



Gantt Chart

Process	**Turn Around time**	**Waiting time**
P1	13 – 0 = 13	13 – 5 = 8
P2	12 – 1 = 11	11 – 3 = 8
P3	5 – 2 = 3	3 – 1 = 2
P4	9 – 3 = 6	6 – 2 = 4
P5	14 – 4 = 10	10 – 3 = 7

Now,

Average Turn Around time = (13 + 11 + 3 + 6 + 10) / 5 = 43 / 5 = 8.6 unit

Average waiting time = (8 + 8 + 2 + 4 + 7) / 5 = 29 / 5 = 5.8 unit

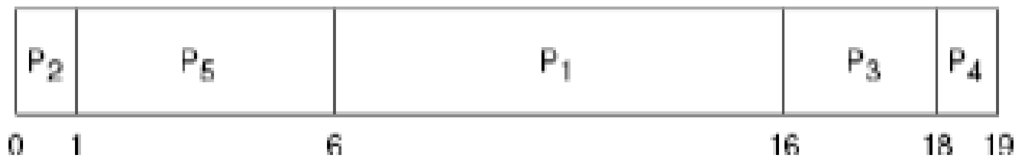
Priority Scheduling

- Jobs are run based on their priority, Always run the job with the highest priority
- Priorities can be **externally defined** (e.g., by user) or based on some **process-specific metrics** (e.g., their expected CPU burst)
- Can be preemptive
- Can be no preemptive
- Priorities can be **static**(i.e. they don’t change) or **dynamic** (they may change during execution)
- Problem =**Starvation** – low priority processes may never execute
- Solution = **Aging** – as time progresses increase the priority of the process

Example :1 If the CPU scheduling policy is priority non-preemptive, calculate the average waiting time and average turn around time. (lower number represents higher priority)

Process	Burst Time (ms)	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

- Priority scheduling Gantt Chart assuming all arrive at time 0



Turn Around time = Exit time/completion time – Arrival time

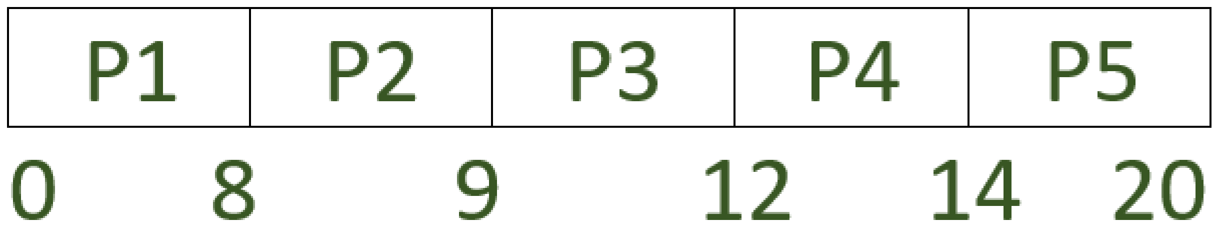
Waiting time = Turn Around time – Burst time/Execution time

Average waiting time = (0+1+6+16+18)/5 = 8.2 msec

Example: 2

PROCESS	ARRIVAL TIME	BURST TIME	PRIORITY
P1	0	8	3
P2	1	1	1
P3	2	3	2
P4	3	2	3
P5	4	6	4

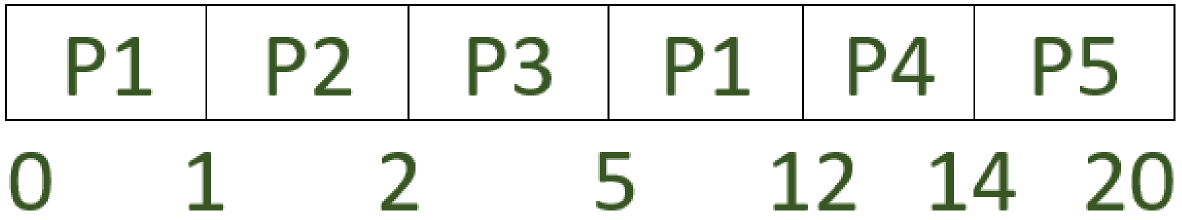
Priority Non-preemptive scheduling :



Average waiting time (AWT)= ((0-0) + (8-1) + (9-2) + (12-3) + (14-4)) / 5 = 33 / 5 = 6.6

Average turnaround time (TAT)= ((8-0) + (9-1) + (12-2) + (14-3) + (20-4)) / 5= 53 / 5 = 10.6

Priority Preemptive scheduling :



Average waiting time (AWT)= ((5-1) + (1-1) + (2-2) + (12-3) + (14-4)) / 5 = 23/5 = 4.6

Average turnaround time (TAT)= ((12-0) + (2-1) + (5-2) + (14-3) + (20-4)) / 5 = 43 / 5 = 8.5

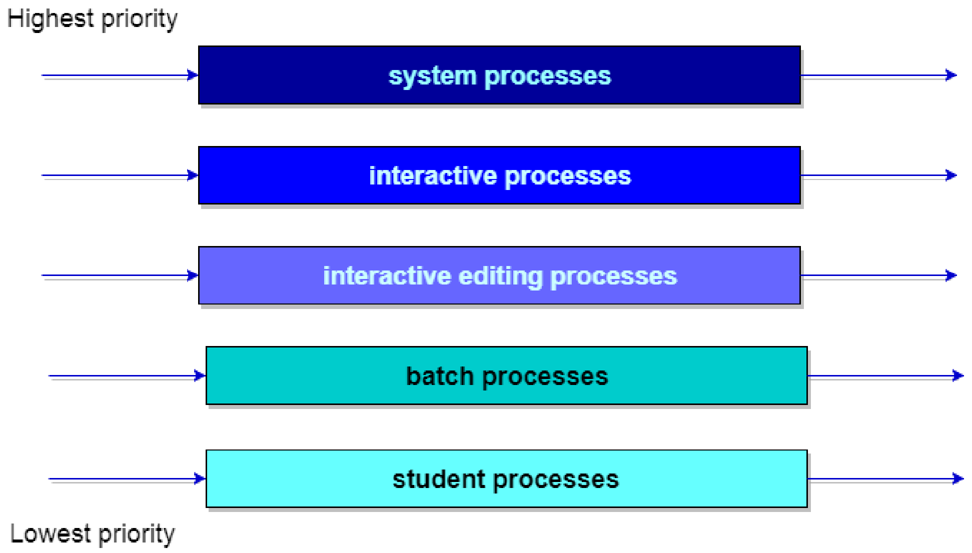
Multi-level queue

- A multi-level queue scheduling algorithm partitions the ready queue into several separate queues.
- The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type

For Example:

- Separate queues might be used for foreground and background processes.
- The foreground queue might be scheduled by Round Robin algorithm, while the background queue is scheduled by an FCFS algorithm.

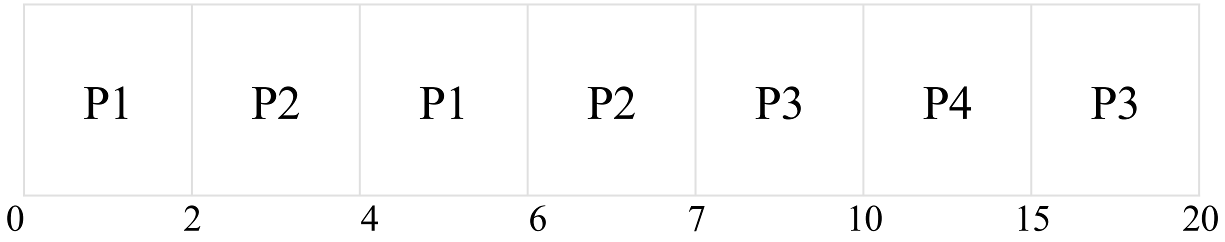
- Let us consider an example of a multilevel queue-scheduling algorithm with five queues:
  - System Processes
  - Interactive Processes
  - Interactive Editing Processes
  - Batch Processes
  - Student Processes



Example 1: Consider below table of four processes under Multilevel queue scheduling. Queue number denotes the queue of the process. Priority of queue 1 is greater than queue 2. queue 1 uses Round Robin (Time Quantum = 2) and queue 2 uses FCFS.

Process	Burst time	Arrival time	Queue number
P1	4	0	1
P2	3	0	1
P3	8	0	2
P4	5	10	1

Solution: **Gantt chart** of the problem



Example 2: Consider below table of four processes under Multilevel queue scheduling. Queue number denotes the queue of the process. Queue1 is having higher priority and queue1 is using the FCFS approach and queue2 is using the round-robin approach(time quantum = 2ms).

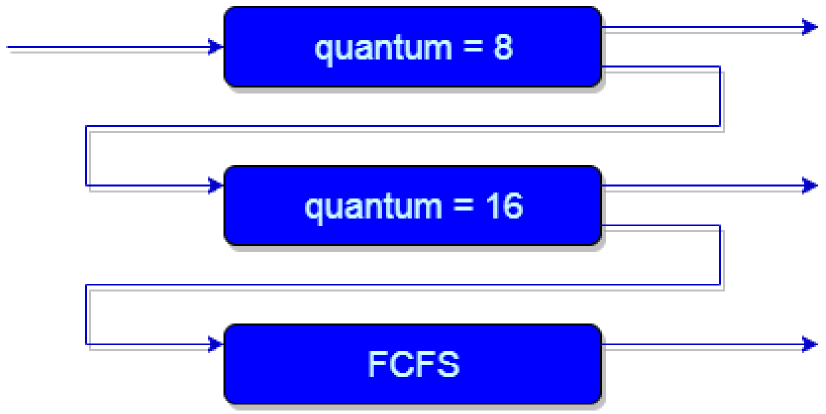
Process	Arrival time	Burst time	Queue
P1	0 ms	5 ms	1
P2	0 ms	3 ms	2
P3	0 ms	8 ms	2
P4	0 ms	6 ms	1

Gantt Chart

P1	P4	P2	P3	P2	P3
0 5	5 11	11 13	13 15	15 16	16 22

Multilevel Feedback Queue

- Multilevel feedback queue scheduling, allows a process to move between queues.
- The idea is to separate processes with different CPU-burst characteristics.
- If a process uses too much CPU time, it will be moved to a lower-priority queue.
- Similarly, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue.



- Multilevel feedback queue scheduler is defined by the following parameters:
  - The number of queues.
  - The scheduling algorithm for each queue.
  - The method used to determine when to upgrade a process to a higher-priority queue.
  - The method used to determine when to demote a process to a lower-priority queue.
  - The method used to determine which queue a process will enter when that process needs service.

Guaranteed Scheduling

- Make real promises to the users about performance.
- If there are n users logged in while you are working, you will receive about 1 /n of the CPU power.
- Similarly, on a single-user system with n processes running, all things being equal, each one should get 1 /n of the CPU cycles.
- To make good on this promise, the system must keep track of how much CPU each process has had since its creation.
- CPU Time entitled= (Time Since Creation)/n
- Then compute the ratio of Actual CPU time consumed to the CPU time entitled.

- A ratio of 0.5 means that a process has only had half of what it should have had, and a ratio of 2.0 means that a process has had twice as much as it was entitled to.
- The algorithm is then to run the process with the lowest ratio until its ratio has moved above its closest competitor.

Fair-Share Scheduling

- Users are assigned some fraction of the CPU
  - Scheduler takes into account who owns a process before scheduling it
- E.g., two users each with 50% CPU share
  - User 1 has 4 processes: A, B, C, D
  - User 2 has 2 processes: E, F
- If round-robin scheduling is used, a possible scheduling sequence that meets all the constraints is this one:

A E B E C E D E A E B E C E D E ...

- On the other hand, if user 1 is entitled to twice as much CPU time as user 2, we might get

A B E C D E A B E C D E ...

Numerous other possibilities exist, of course, and can be exploited, depending on what the notion of fairness is.

Lottery Scheduling [Waldspurger and Weihl 1994]

- Jobs receive lottery tickets for various resources
  - E.g., CPU time
- At each scheduling decision, one ticket is chosen at random and the job holding that ticket wins
- If there are 100 tickets outstanding, and one process holds 20 of them, it will have a 20% chance of winning each lottery. In the long run, it will get about 20% of the CPU.
- Lottery scheduling can be used to solve problems that are difficult to handle with other methods.
- One example is a video server in which several processes are feeding video streams to their clients, but at different frame rates.
- Suppose that the processes need frames at 10, 20, and 25 frames/sec. By allocating these processes 10, 20, and 25 tickets, respectively, they will automatically divide the CPU in approximately the correct proportion, that is, 10 : 20 : 25.

Scheduling in Real-Time Systems

- A **real-time scheduling System** is composed of the **scheduler**, clock and the processing hardware elements.
- In a **real-time system**, a process or task has Schedulability ; tasks are accepted by a **real-time system** and completed as specified by the task deadline depending on the characteristic of the **scheduling** algorithm.
- Real-time systems are generally categorized as **hard real time**, meaning there are absolute deadlines that must be met—or else!— and **soft real time**, meaning that missing an occasional deadline is undesirable, but nevertheless tolerable.
- The events that a real-time system may have to respond to can be further categorized as **periodic** (meaning they occur at regular intervals) or **aperiodic** (meaning they occur unpredictably).

- For example, if there are *m* periodic events and event *i* occurs with period *Pi* and requires *Ci* sec of CPU time to handle each event, then the load can be handled only if A real-time system that meets this criterion is said to be **schedulable**.

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

- A process that fails to meet this test cannot be scheduled because the total amount of CPU time the processes want collectively is more than the CPU can deliver.