CPU Scheduling

Introduction

- CPU scheduling in an operating system is the process of selecting which ready process should execute next on the CPU to ensure efficient utilization of the CPU and to improve system performance.

- It manages the allocation of the CPU among competing processes, deciding when, and for how long, each process gets to run.

- The primary goals of CPU scheduling are to maximize CPU utilization, minimize waiting and turnaround times for processes, and provide a responsive and fair system for all users and tasks.

Why is CPU Scheduling Necessary?

- In a multiprogrammed system, many processes are present in memory and are ready to execute. However, a CPU can only execute one process at a time. CPU scheduling is essential to:

- **Maximize CPU Utilization:**

  It keeps the CPU busy by assigning it to a process whenever a previous process finishes or goes into a waiting state (e.g., for I/O operations), preventing it from sitting idle.

- **Improve System Efficiency:**

  By keeping the CPU busy, the overall time wasted is minimized, leading to higher productivity.

- **Enhance User Experience:**

  Scheduling algorithms aim to provide quick response times for interactive processes and complete background tasks efficiently.

Key Concepts

- **Ready Queue:**
  - A queue of processes that are in the memory and are ready to be executed by the CPU.
- **Short-Term Scheduler (CPU Scheduler):**
  - The component of the operating system that makes the final decision of which process to execute next from the ready queue.
- **Types of Scheduling:**
  - **Non-Preemptive Scheduling:** Once a process starts executing, it continues until it completes or voluntarily releases the CPU.
  - **Preemptive Scheduling:** The operating system can interrupt a running process and allocate the CPU to another process, usually based on priority or time slicing.

CPU Scheduling Criteria

- **CPU Utilization:**
  - Aims to keep the CPU as busy as possible to avoid wasting cycles, ideally operating at 100%.
- **Throughput:**
  - Measures the total number of processes that are completed per unit of time; higher throughput indicates a more efficient system.
- **Turnaround Time:**
  - The total time taken for a process to execute, from its submission to its completion.
- **Waiting Time:**
  - The sum of the time a process spends waiting in the ready queue to acquire the CPU.
- **Response Time:**
  - The time from when a request is submitted to the system until the first response is produced, crucial for interactive systems.
- **Fairness:**
  - While not a distinct metric in the same way, fairness ensures that each process receives a fair share of the CPU time over a given period, preventing any single process from monopolizing the CPU.

Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that complete their execution per time unit
- **Turnaround time** – amount of time to execute a particular process
- **Waiting time** – amount of time a process has been waiting in the ready queue
- **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

Scheduling Algorithm Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

Arrival time (AT) – Arrival time is the time at which the process arrives in ready queue.

Burst time (BT) or CPU time of the process – Burst time is the unit of time in which a particular process completes its execution.

Completion time (CT) – Completion time is the time at which the process has been terminated.

Turn-around time (TAT) – The total time from arrival time to completion time is known as turn-around time. TAT can be written as,

Turn-around time (TAT) = Completion time (CT) – Arrival time (AT)

Or

TAT = Burst time (BT) + Waiting time (WT)

Waiting time (WT) – Waiting time is the time at which the process waits for its allocation while the previous process is in the CPU for execution. WT is written as,

Waiting time (WT) = Turn-around time (TAT) – Burst time (BT)

Response time (RT) – Response time is the time at which CPU has been allocated to a particular process first time. In case of non-preemptive scheduling, generally Waiting time and Response time is same.

Gantt chart – Gantt chart is a visualization which helps to scheduling and managing particular tasks in a project. It is used while solving scheduling problems, for a concept of how the processes are being allocated in different algorithms.

CPU Scheduling Algorithms

1. First-Come, First-Served (FCFS)
2. Shortest Job First (SJF)
3. Shortest Remaining Time First (SRTF)
4. Round Robin (RR)
5. Priority Scheduling
6. Multilevel Queue Scheduling
7. Multilevel Feedback Queue Scheduling

First Come First Serve(FCFS)

In FCFS Scheduling,

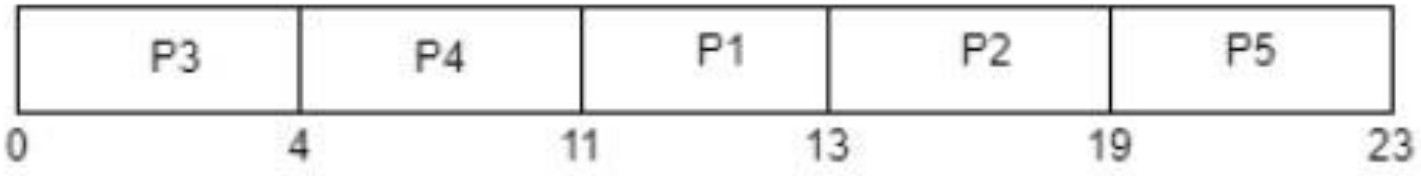
- The process which arrives first in the ready queue is firstly assigned the CPU.
- In case of a tie, process with smaller process id is executed first.
- It is always non-preemptive in nature.
- Jobs are always executed on a first-come, first-serve basis.
- It is easy to implement and use.
- This method is poor in performance, and the general wait time is quite high.

Problem 1

Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Response time (RT), Average Turn-around time and Average Waiting time.

Process ID	Arrival time	Burst time
P1	2	2
P2	5	6
P3	0	4
P4	0	7
P5	7	4

Solution
Gantt chart



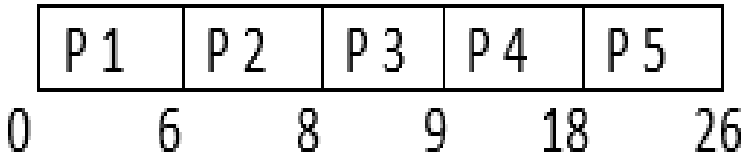
Process ID	Arrival time	Burst time	CT	TAT=CT-AT	WT=TAT-BT
P1	2	2	13	13-2= 11	11-2= 9
P2	5	6	19	19-5= 14	14-6= 8
P3	0	4	4	4-0= 4	4-4= 0
P4	0	7	11	11-0= 11	11-7= 4
P5	7	4	23	23-7= 16	16-4= 12

Examples

Process ID	Arrival Time	Burst Time
P 1	0	6
P 2	2	2
P 3	3	1
P 4	4	9
P 5	5	8

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
P 1	0	6	6	6	0
P 2	2	2	8	6	4
P 3	3	1	9	6	5
P4	4	9	18	14	5
P 5	5	8	26	21	13

Gantt Chart



Response Time (RT):

(ST – AT)

P1 -> 0-0 = 0

P2-> 6-2 = 4

P3-> 8-3 = 5

P4-> 9-4 = 5

P5-> 18-5 = 13

First- Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule is:

- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order:

$$P_2 , P_3 , P_1$$

- The Gantt chart for the schedule is:

- Waiting time for $P_1 = 6; P_2 = 0; P_3 = 3$
- Average waiting time: (6 + 0 + 3)/3 = 3
- Much better than previous case

The convoy effect in the FCFS (First-Come, First-Serve) scheduling algorithm occurs when a long-duration process occupies the CPU or other system resources, causing numerous shorter processes in the queue to wait behind it for an extended time.

This phenomenon leads to decreased overall system efficiency and increased average wait times for the shorter tasks.

Shortest-Job-First (SJF) Scheduling

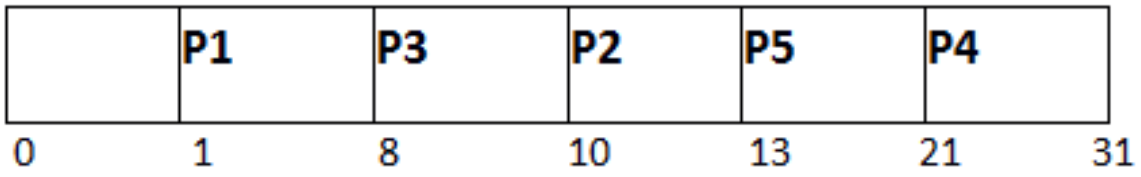
- Till now, we were scheduling the processes according to their arrival time (in FCFS scheduling). However, SJF scheduling algorithm, schedules the processes according to their burst time.
- In SJF scheduling, the process with the lowest burst time, among the list of available processes in the ready queue, is going to be scheduled next.
- However, it is very difficult to predict the burst time needed for a process hence this algorithm is very difficult to implement in the system.
- Advantages of SJF
  - Maximum throughput
  - Minimum average waiting and turnaround time
- Disadvantages of SJF
  - May suffer with the problem of starvation
  - It is not implementable because the exact Burst time for a process can't be known in advance.

Example of SJF

There are five jobs named as P1, P2, P3, P4 and P5. Their arrival time and burst time are given in the table below.

PID	Arrival Time(AT)	Burst Time(BT)	Completion Time(CT)	Turn Around Time(TAT)	Waiting Time(WT)	Response Time(RT)
P1	1	7	8	7	0	
P2	3	3	13	10	7	
P3	6	2	10	4	2	
P4	7	10	31	24	14	
P5	9	8	21	12	4	

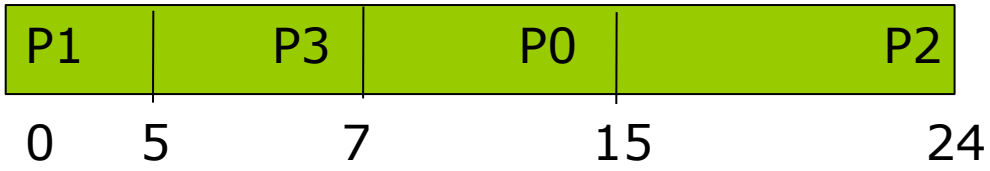
Gantt chart



PROCESS ID	ARRIVAL TIME	BURST TIME
P0	5	8
P1	0	5
P2	4	9
P3	1	2

Solution:

Let us use the Gantt Chart for the above-given problem.



PROCESS ID	ARRIVAL TIME	BURST TIME	COMPLETION TIME	TURN AROUND TIME	WAITING TIME
P0	5	8	21	16	8
P1	0	5	5	5	0
P2	4	9	16	12	3
P3	1	2	7	6	4

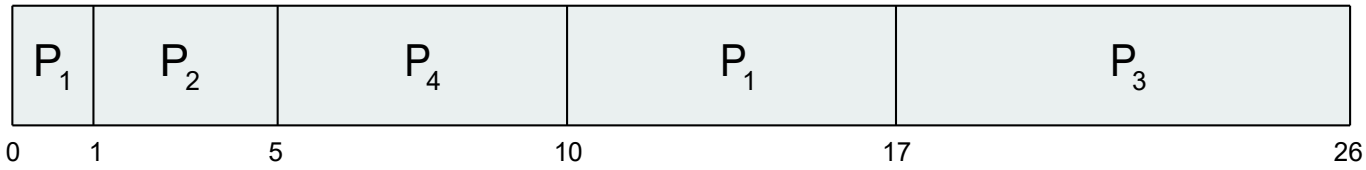
Shortest-Remaining-Time-First(SRTF)

This Algorithm is the **preemptive version** of **SJF scheduling**. In SRTF, the execution of the process can be stopped after certain amount of time. At the arrival of every process, the short term scheduler schedules the process with the least remaining burst time among the list of available processes and the running process.

- Now we add the concepts of varying arrival times and preemption to the analysis

Process	*Arrival* Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

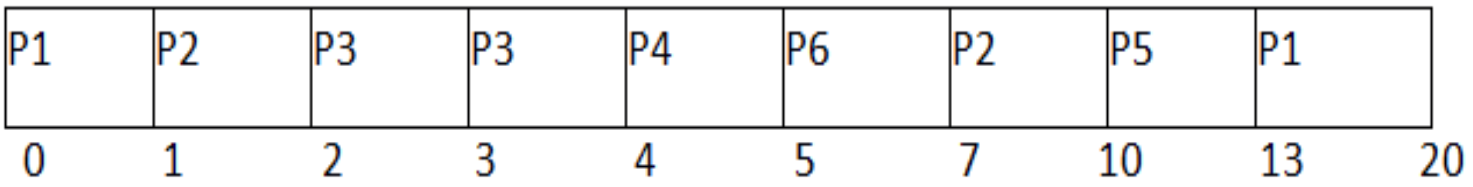
Gantt Chart



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

Example

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time	Response Time
P1	0	8	20	20	12	0
P2	1	4	10	9	5	1
P3	2	2	4	2	0	2
P4	3	1	5	2	1	4
P5	4	3	13	9	6	10
P6	5	2	7	2	0	5



Priority Scheduling

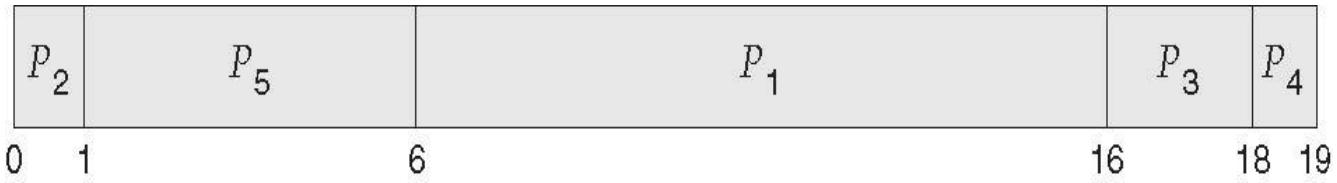
- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - Non-preemptive
- Problem ≡ **Starvation** – low priority processes may never execute
  - In priority scheduling, starvation occurs when low-priority processes are indefinitely blocked from CPU execution due to continuous arrival of higher-priority processes.
- Solution ≡ **Aging** – as time progresses increase the priority of the process
  - Aging is a technique that prevents starvation by gradually increasing the priority of processes that have waited in the ready queue for a long time, ensuring they eventually gain CPU access. For example, a background process with initially low priority might have its priority increased over time, allowing it to run after many user applications have finished.

Priority Scheduling

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

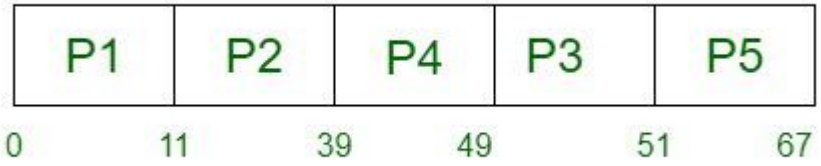
Non-Preemptive

- Priority scheduling Gantt Chart



- Average waiting time = 8.2 msec

Process	Arrival Time	Burst Time	Priority
P1	0	11	2
P2	5	28	0
P3	12	2	3
P4	2	10	1
P5	9	16	4



Preemptive Priority Scheduling Algorithm

Process	Arrival Time	Priority	Burst Time
P1	0 ms	3	3 ms
P2	1 ms	2	4 ms
P3	2 ms	4	6 ms
P4	3 ms	6	4 ms
P5	5 ms	10	2 ms

Process	A.T	Priority	B.T	C.T	T.A.T	W.T	R.T
P1	0	3	3	7	7	4	0
P2	1	2(H)	4	5	4	0	0
P3	2	4	6	13	11	5	5
P4	3	6	4	17	14	10	10
P5	5	10(L)	2	19	14	12	12

Total Turn Around Time = 7 + 4 + 11 + 14 + 14 = 50 ms
Average Turn Around Time = (Total Turn Around Time)/(no. of processes) = 50/5 = 10.00 ms

Total Waiting Time = 4 + 0 + 5 + 10 + 12 = 31 ms
Average Waiting Time = (Total Waiting Time)/(no. of processes) = 31/5 = 6.20 ms

Total Response Time = 0 + 0 + 5 + 10 + 12 = 27 ms
Average Response Time = (Total Response Time)/(no. of processes) = 27/5 = 5.40 ms

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

If the CPU scheduling policy is priority preemptive, calculate the average waiting time and average turn around time. (Higher number represents higher priority)

Solution-

Gantt Chart-

Gantt Chart

Id	CT	Turn Around time	Waiting time
P1	15	15 – 0 = 15	15 – 4 = 11
P2	12	12 – 1 = 11	11 – 3 = 8
P3	3	3 – 2 = 1	1 – 1 = 0
P4	8	8 – 3 = 5	5 – 5 = 0
P5	10	10 – 4 = 6	6 – 2 = 4

AVG(Turn Around time) = (15 + 11 + 1 + 5 + 6) / 5 = 38 / 5 = 7.6 unit
AVG(waiting time) = (11 + 8 + 0 + 0 + 4) / 5 = 23 / 5 = 4.6 unit

Q. Consider the following set of processes with their CPU burst time, arrival time given in milliseconds and priority. Draw GANTT charts for each execution using SRTF, RR(quantum=2) and preemptive priority scheduling. Compute average response time, average waiting time and average turnaround time for each of the algorithms.

Process	CPU burst time	Arrival time	Priority
P1	3	0	1
P2	2	1	0
P3	4	3	2
P4	5	4	0
P5	3	5	1

Round Robin (RR)

Round Robin is a CPU scheduling algorithm where each process is cyclically assigned a fixed time slot.

1. Round Robin CPU Algorithm generally focuses on Time Sharing technique.

2. The period of time for which a process or job is allowed to run in a pre-emptive method is called time **quantum**.

3. Each process or job present in the ready queue is assigned the CPU for that time quantum, if the execution of the process is completed during that time then the process will **terminate** else the process will go back to the **Ready Queue** and wait for its next turn to complete the execution.

Advantages of Round Robin CPU Scheduling Algorithm

1. There is fairness since every process gets an equal share of the CPU.

2. The newly created process is added to the end of the ready queue.

3. A round-robin scheduler generally employs time-sharing, giving each job a time slot or quantum.

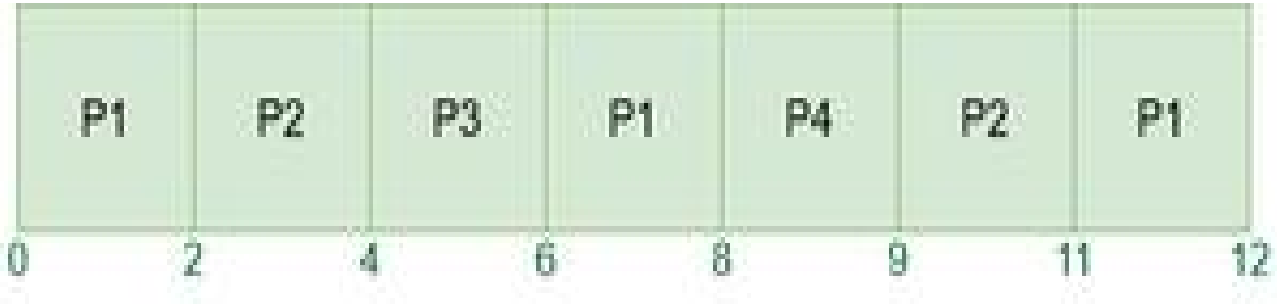
4. While performing a round-robin scheduling, a particular time quantum is allotted to different jobs.

5. Each process get a chance to reschedule after a particular quantum time in this scheduling.

Example-1:

Consider the following table of arrival time and burst time for four processes **P1, P2, P3, and P4** and given **Time Quantum = 2**

Process	Arrival Time	Burst Time
P1	0 ms	5 ms
P2	1 ms	4 ms
P3	2 ms	2 ms
P4	4 ms	1 ms



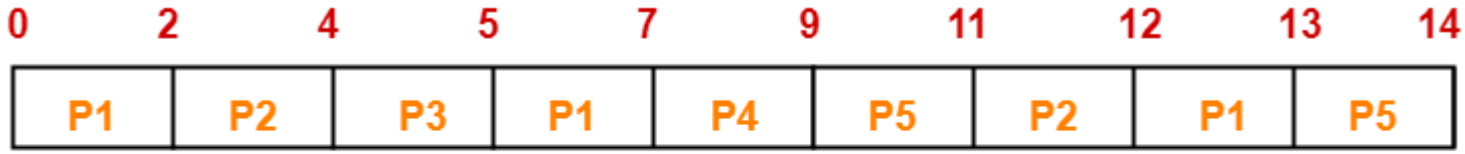
Processes	AT	BT	CT	TAT	WT
P1	0	5	12	12-0 = 12	12-5 = 7
P2	1	4	11	11-1 = 10	10-4 = 6
P3	2	2	6	6-2 = 4	4-2 = 2
P4	4	1	9	9-4 = 5	5-1 = 4

Average Turn around time *= (12 + 10 + 4 + 5)/4 = 31/4 = 7.7*

Average waiting time *= (7 + 6 + 2 + 4)/4 = 19/4 = 4.7*

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

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



Gantt Chart

Let us consider a system that has four processes which have arrived at the same time in the order P1, P2, P3 and P4. The burst time in milliseconds of each process is given by the following table –

Process	CPU Burst Times in ms
P1	8
P2	10
P3	6
P4	4

GANTT Chart with time quantum of 2ms

Multilevel Queue (MLQ)

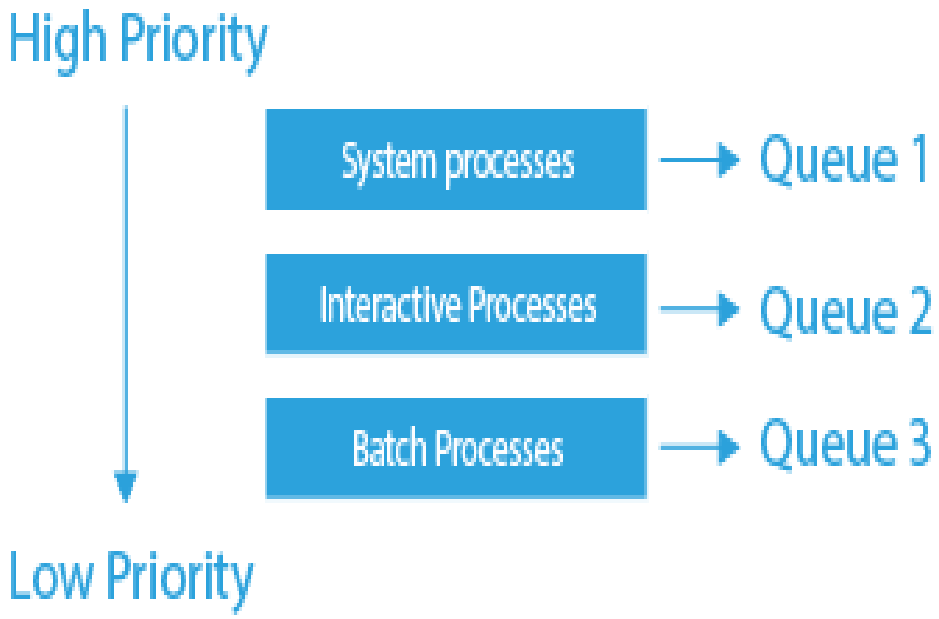
A method of organizing the tasks or processes that a computer must perform is multilevel queue scheduling.

The computer system divides tasks or processes into different queues based on their priority in this method.

A task's priority can be determined by factors such as memory capacity, process priority, or type.

There are two main types of processes in the computer system:

interactive processes and background processes. Interactive processes need to be done quickly because they are being used by a person, while background processes can wait because they are not as important.



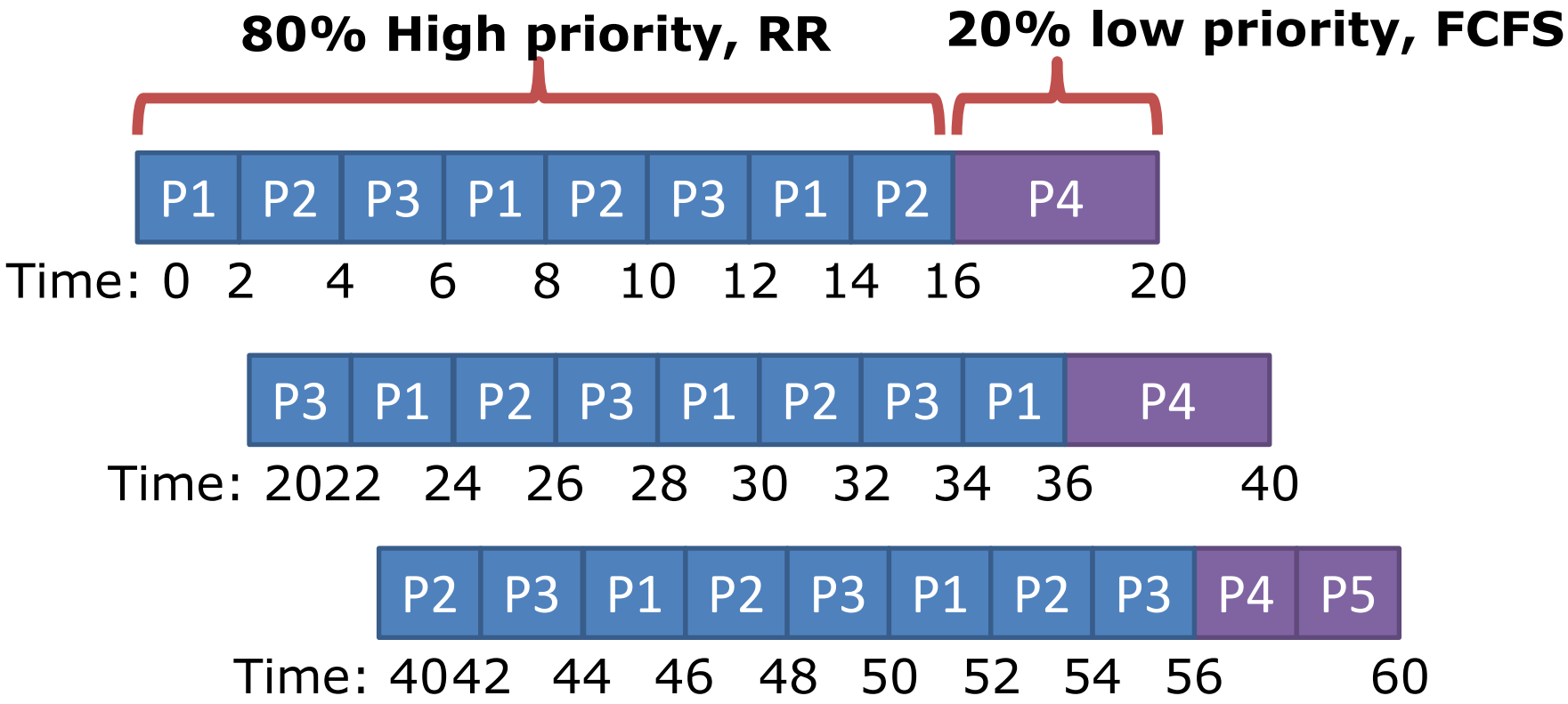
System Process
The OS has its own process to run, known as the System Process.
Interactive Process
The System Process, a process that the OS has to run, is known as such.
Batch Process
An operating system function called batch processing gathers applications and data into a batch before processing begins.

Multilevel Queue (MLQ)

- Key idea: divide the ready queue in two
  1. High priority queue for interactive processes
     - RR scheduling
  2. Low priority queue for CPU bound processes
     - FCFS scheduling
- Simple, static configuration
  - Each process is assigned a priority on startup
  - Each queue is given a fixed amount of CPU time
    - 80% to processes in the high priority queue
    - 20% to processes in the low priority queue

MLQ Example

Process	Arrival Time	Priority
P1	0	1
P2	0	1
P3	0	1
P4	0	2
P5	1	2



Problems with MLQ

- Assumes you can classify processes into high and low priority
  - How could you actually do this at run time?
  - What of a processes' behavior changes over time?
    - i.e. CPU bound portion, followed by interactive portion
- Highly biased use of CPU time
  - Potentially too much time dedicated to interactive processes
  - Convoy problems for low priority tasks

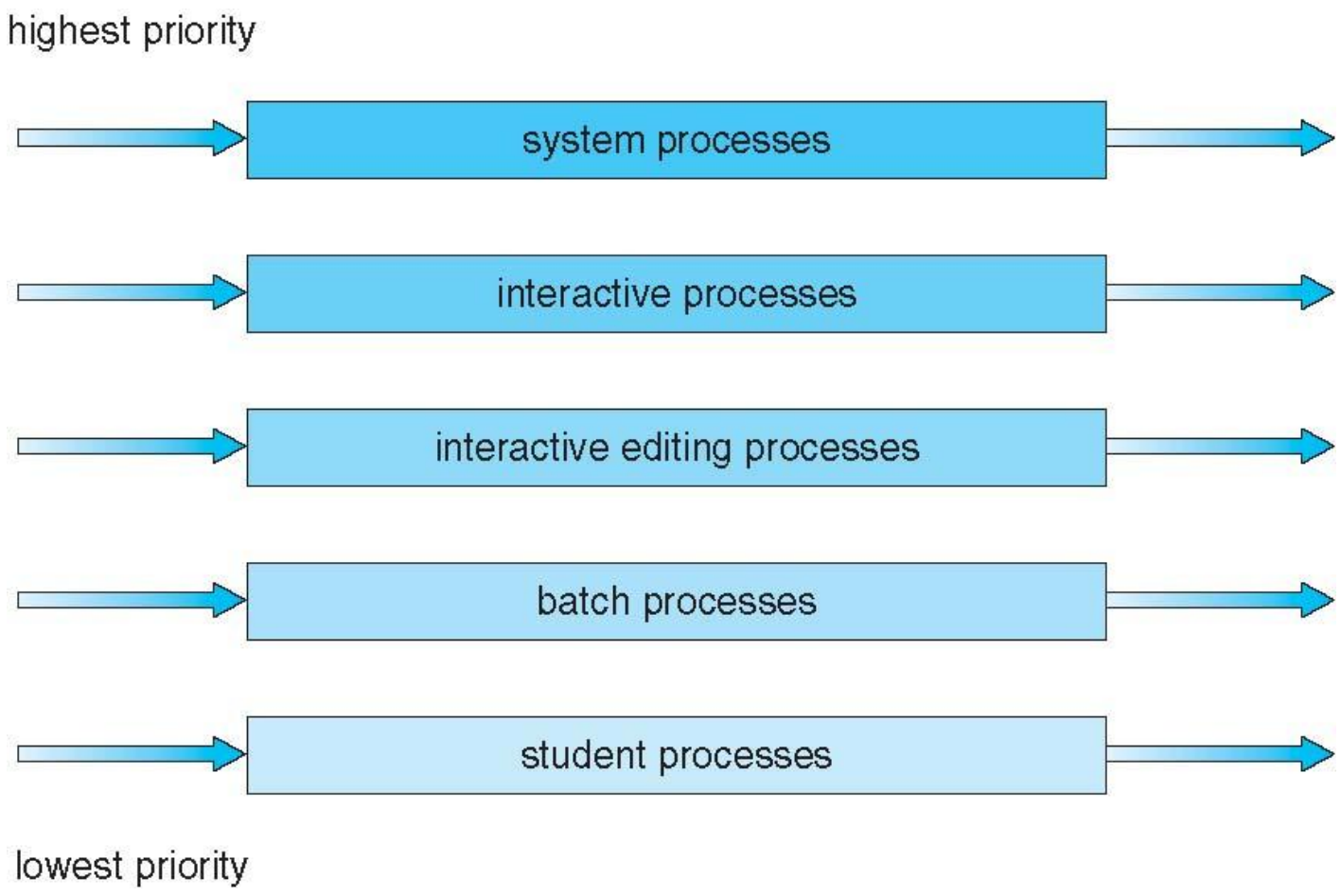
Multilevel Feedback Queue (MLFQ)

- Goals
  - Minimize response time and turnaround time
  - Dynamically adjust process priorities over time
    - No assumptions or prior knowledge about burst times or process behavior
- High level design: generalized MLQ
  - Several priority queues
  - Move processes between queue based on observed behavior (i.e. their history)

Multilevel Queue

- Ready queue is partitioned into separate queues, eg:
  - **foreground** (interactive)
  - **background** (batch)
- Process permanently in a given queue
- Each queue has its own scheduling algorithm:
  - foreground – RR
  - background – FCFS
- Scheduling must be done between the queues:
  - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR
  - 20% to background in FCFS

Multilevel Queue Scheduling



Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

Example of Multilevel Feedback Queue

- Three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS
- Scheduling
  - A new job enters queue Q_0 which is served FCFS
    - When it gains CPU, job receives 8 milliseconds
    - If it does not finish in 8 milliseconds, job is moved to queue Q_1
  - At Q_1 job is again served FCFS and receives 16 additional milliseconds
    - If it still does not complete, it is preempted and moved to queue Q_2