**CPU Scheduling Algorithms**

**Introduction**

* CPU scheduling concerns how the CPU time is allocated among the competing processes or threads.
* Maximum CPU utilization obtained with multiprogramming.
* CPU-I/O Burst Cycle: Process execution consists of a cycle of CPU execution and I/O wait.
* CPU Scheduling function: Selects from among processes in memory that are ready to execute, and allocates the CPU to one of them.

**Scheduling Criteria**

* **CPU utilization**: keep the CPU as busy as possible.
* **Throughput**: number 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).

**Optimization Criteria**

1. Max CPU utilization.
2. Max throughput.
3. Min turnaround time.
4. Min waiting time.
5. Min response time.

**CPU Scheduling function:**

* **Dispatcher module**: gives control of the CPU to the process selected by the short-time scheduler, this involves :
  + Switching context.
  + Switching to user mode.
  + Jumping to the proper location in the user program to restart that program.
* **Dispatch latency:** time it takes for the dispatcher to stop one process and start another running.

**Types of CPU Scheduling Algorithms**

* **Non-Preemptive Scheduling:**
  + First Come First Served Scheduling (FCFS).
  + Shortest Job Next Scheduling (SJN).
  + Priority Scheduling.
* **Preemptive Scheduling:**
  + Shortest Remaining Time (SRTF).
  + Round Robin Scheduling (RR).
* Other (**Hybrid**):
  + Multilevel Queue Scheduling.
  + Multilevel Feedback Queue Scheduling.

**Preemptive vs. Non-preemptive Scheduling**

* The selection of a new process to run occurs whenever a process must release the CPU. This can happen if:

1. The process terminates.

2. The process switches from a running state to a waiting state (e.g. I/O wait).

3. The process is forced to switch from the running state to the ready state.

* **In non-preemptive scheduling:** the process releases the CPU only when it needs to (case 1 and 2 above).
* **In preemptive scheduling:** the process releases the CPU when it needs to (case 1 and 2 above) or when it is forced by the operating system (case 3).

**Basic Scheduling Algorithm (FCFS):**

* Non-preemptive.
* Ready queue is a FIFO queue.
* The arriving Jobs are placed at the end of queue.
* Dispatcher selects first job in queue and this job runs to completion of CPU burst.
* Advantages: simple, low overhead.
* Disadvantages: inappropriate for interactive systems, large fluctuations in average turnaround time are possible.

**FCFS Example:**

|  |  |
| --- | --- |
| Job | Burst Time |
| A | **15** |
| B | **2** |
| C | **1** |

**Time Line**

|  |  |  |
| --- | --- | --- |
| A | B | C |

**0 15 17 18**

|  |  |  |
| --- | --- | --- |
| Job | Wait Time | Turnaround Time |
| A | **0** | **15** |
| B | **15** | **17** |
| C | **17** | **18** |

Average Waiting Time = (0+15+17)/3 = 10.666

Average Turnaround Time = (15+17+18)/3 =16.666

**SJN - Shortest Job Next**

* Ready queue treated as a priority queue based on Smallest CPU-time requirement.
* Arriving jobs inserted at proper position in queue.
* Dispatcher selects shortest job (1st in queue) and runs to completion.
* Also known SJF (Shortest Job First).
* Its easiest to implement in batch environment where the estimated CPU time required to run the job's given by each user at the start of each job, it doesn’t in interactive system because users don’t estimate the CPU time required to run their jobs.
* **Two schemes:**

1. **Non preemptive** – once CPU given to the process it cannot be preempted until completes its CPU burst.
2. **Preemptive** – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is known as the Shortest-Remaining-Time-First (SRTF).

**SJN Example:**

|  |  |
| --- | --- |
| Job | Burst Time |
| A | 15 |
| B | 2 |
| C | 1 |

**Time Line**

|  |  |  |
| --- | --- | --- |
| C | B | A |

0 1 3 18

|  |  |  |
| --- | --- | --- |
| Job | Wait Time | Turnaround Time |
| A | 3 | 18 |
| B | 1 | 3 |
| C | 0 | 1 |

Average Waiting Time = (3 +1+0)/3 = 1.333

Average Turnaround Time = (18+3+1)/3 = 7.333

**Priority Scheduling**

* It's a non-preemptive algorithm and one of the most common scheduling algorithms in batch system.
* It allows the programs with the highest priority to be processed first and they aren't interrupted until their CPU cycle are completed or a natural wait occurs, if two or more jobs with equal priority are present in the READY queue, the processor is allocated to the one that arrived FCFS with priority.
* Priority can be assigned by a system administrator using characteristics extrinsic to the jobs or it can also be determined by the processor manager based on characteristics intrinsic to the job such as:

1. **Memory Requirements:** Jobs requiring large amount of memory could be allocated lower priorities than those requesting small amount of memory.
2. **Number and Type of Peripheral Devices:** Jobs requiring many peripheral devices would be allocated lower priorities than those requesting fewer devices.
3. **CPU Time:** Jobs having along CPU cycle or estimated runtime would be given lower priorities than those having a brief estimated runtime.
4. **Amount of Time Already Spent in the System:** This is a total amount of elapsed time since the job was accepted for processing.

* A major problem with priority scheduling is indefinite blocking or starvation.
* A solution to the problem of indefinite blockage of the low-priority process is aging. Aging is a technique of gradually increasing the priority of processes that wait in the system for a long period of time.

**Example of Priority Algorithm**

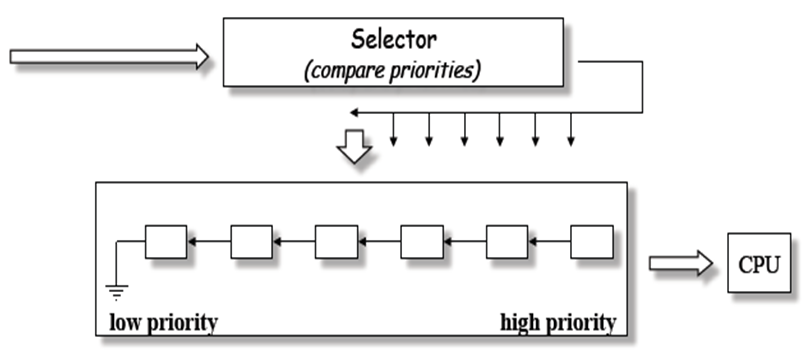
****

Figure 1: Priority Algorithm

|  |  |  |
| --- | --- | --- |
| Process | Burst Time | Priority |
| P1 | **4** | **2** |
| P2 | **5** | **4** |
| P3 | **3** | **3** |
| P4 | **7** | **1** |
| P5 | **1** | **5** |

**TimeLine of CPU or Gantt chart:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P5 | P2 | P3 | P1 | P4 |

**0 1 6 9 13 20**

**Compute Wait Time and Turnaround Time:**

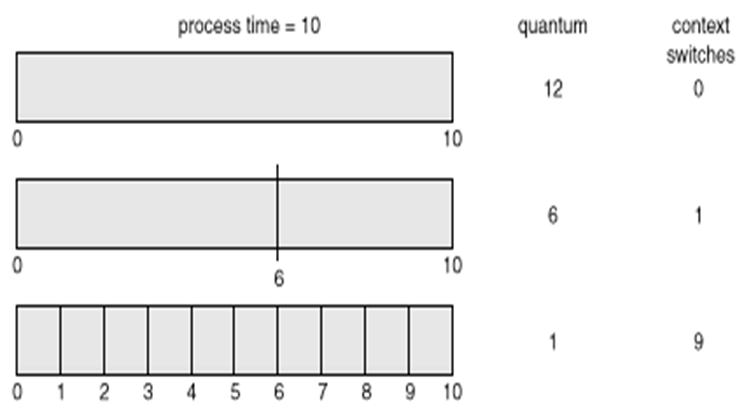
|  |  |  |
| --- | --- | --- |
| Process | Wait Time | Turnaround Time |
| P1 | **9** | **13** |
| P2 | **1** | **6** |
| P3 | **6** | **9** |
| P4 | **13** | **20** |
| P5 | **0** | **1** |

Average Wait Time = (9+1+6+13+0)/5 = 5.8

**Round Robin (RR)**

* It is a preemptive version of FCFS.
* Each process gets a small unit of CPU time (time quantum), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
* If there are n processes in the ready queue and the time quantum is q, then each process gets 1/n of the CPU time in chunks of at most q time units at once. No process waits more than (n-1) q time units.
* Performance:
  + *q large -> FIFO*
  + *q small -> q must be enlarge with respect to context switch, otherwise* overhead is too high.

**Time Quantum and Context Switch Time:**

****

**Example of RR:**

Example of RR with Time Quantum = 20.

|  |  |
| --- | --- |
| Process | Burst Time |
| P1 | 53 |
| P2 | 17 |
| P3 | 68 |
| P4 | 24 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P4 | P1 | P3 | P4 | P1 | P3 | P3 |

**0 20 37 57 77 97 117 121 134 154 162**

**Multi-level Queue**

* Ready queue is partitioned into separate queues:
* Foreground (interactive).
* Background (batch).
* 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 among its processes; i.e., 80% to foreground in RR, 20% to background in FCFS.
* In a multilevel queue scheduling processes are permanently assigned to one queues based on some **property of the process**, such as:
* Memory size.
* Process type (as in the next slide).
* Algorithm chooses the process from the occupied queue that has the highest priority, and run that process either Preemptive or Non-preemptive.
* Each queue has its own scheduling algorithm or policy.
* If a queue has absolute priority over lower-priority queue then no process in the queue could run unless the queues for the highest-priority processes were all empty.

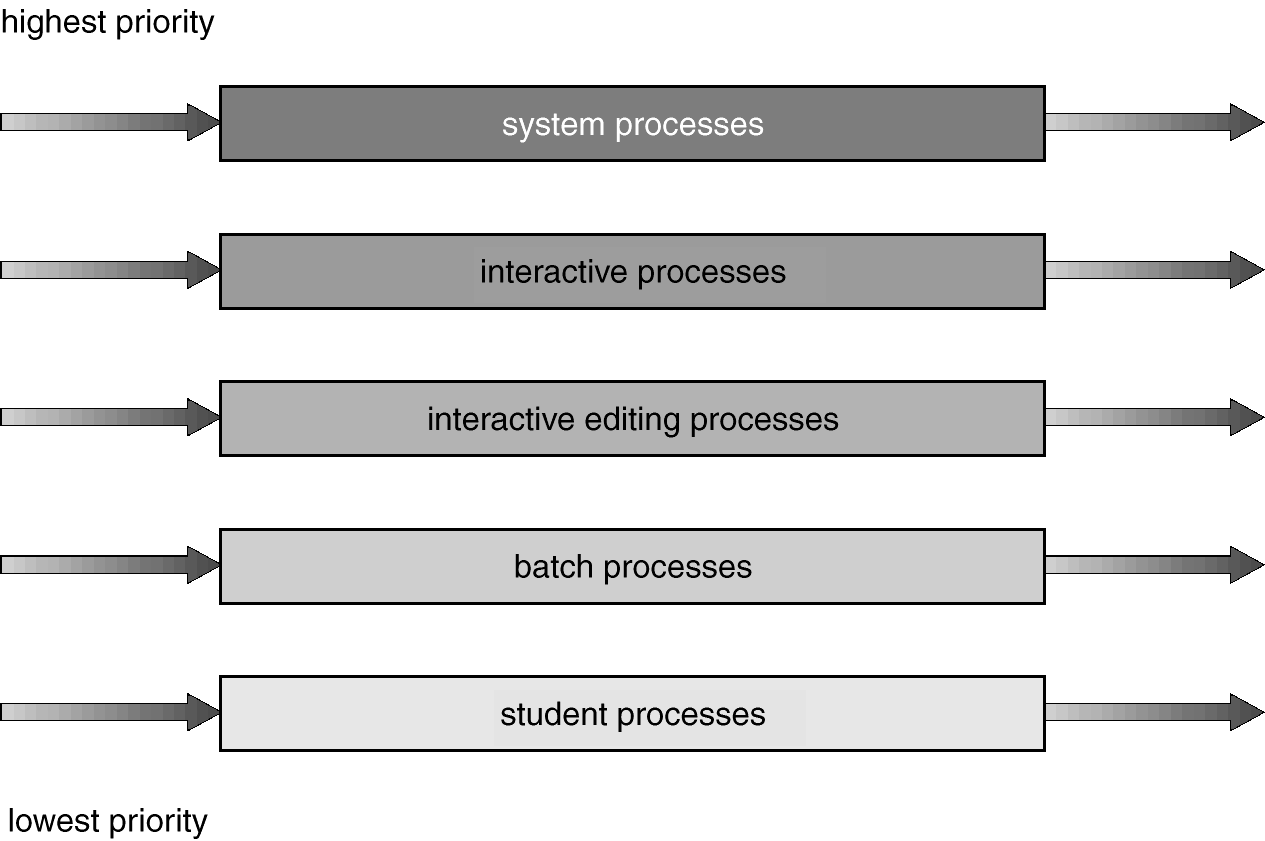


Figure 2: Multi-level Queue Algorithm

**Multi-level Feedback Queue**

* Multilevel feedback queue-scheduling algorithm allows a process to move between queues.
* It uses many ready queues and associates a different priority with each queue.
* The Algorithm chooses to process with highest priority from the occupied queue and run that process either preemptively or none preemptively.
* If the process uses too much CPU time it will moved to a lower-priority queue. Similarly, a process that waits too long in the lower-priority queue may be moved to a higher-priority queue.
* Note: this form of aging prevents starvation.

**Example of Multilevel Feedback Queue**

* Three queues:
* *Q0 – time quantum 8 milliseconds.*
* *Q1 – time quantum 16 milliseconds.*
* *Q2 – FCFS.*
* Scheduling: A new job enters queue *Q0 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 *Q1.*  At *Q1 job is again served FCFS and receives 16 additional* milliseconds. If it still does not complete, it is preempted and moved to queue *Q2.*

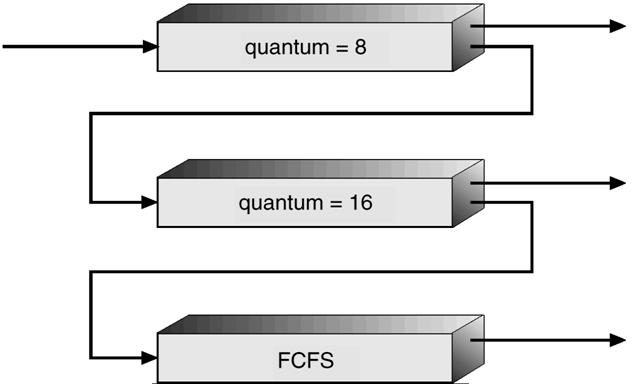


Figure 3: Multilevel Feedback Queues