Multilevel Feedback Queue Scheduling

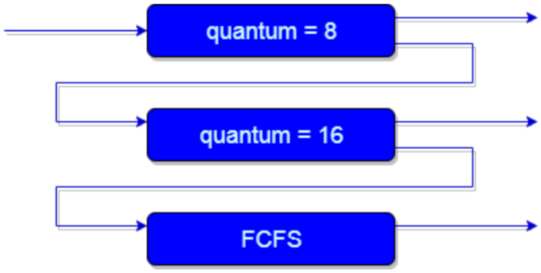
In a multilevel queue-scheduling algorithm, processes are permanently assigned to a queue on entry to the system. Processes do not move between queues. This setup has the advantage of low scheduling overhead, but the disadvantage of being inflexible.

Multilevel feedback queue scheduling, however, 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. This form of aging prevents starvation.

In general, a 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.

The definition of a multilevel feedback queue scheduler makes it the most general CPU-scheduling algorithm. It can be configured to match a specific system under design. Unfortunately, it also requires some means of selecting values for all the parameters to define the best scheduler. Although a multilevel feedback queue is the **most general scheme**, it is also the **most complex**.



An example of a multilevel feedback queue can be seen in the above figure.

Explanation:

First of all, Suppose that queues 1 and 2 follow round robin with time quantum 8 and 16 respectively and queue 3 follows FCFS. One of the implementations of Multilevel Feedback Queue Scheduling is as follows:

1. If any process starts executing then firstly it enters queue 1.
2. In queue 1, the process executes for 8 unit and if it completes in these 8 units or it gives CPU for I/O operation in these 8 units unit than the priority of this process does not change, and if for some reasons it again comes in the ready queue than it again starts its execution in the Queue 1.
3. If a process that is in queue 1 does not complete in 8 units then its priority gets reduced and it gets shifted to queue 2.
4. Above points 2 and 3 are also true for processes in queue 2 but the time quantum is 16 units. Generally, if any process does not complete in a given time quantum then it gets shifted to the lower priority queue.
5. After that in the last queue, all processes are scheduled in an FCFS manner.
6. It is important to note that a process that is in a lower priority queue can only execute only when the higher priority queues are empty.
7. Any running process in the lower priority queue can be interrupted by a process arriving in the higher priority queue.

Also, the above implementation may differ for the example in which the last queue will follow **Round-robin Scheduling.**

In the above Implementation, there is a problem and that is; Any process that is in the lower priority queue has to suffer starvation due to some short processes that are taking all the CPU time.

And the solution to this problem is : There is a solution that is to boost the priority of all the process after regular intervals then place all the processes in the highest priority queue.

The need for Multilevel Feedback Queue Scheduling(MFQS)

Following are some points to understand the need for such complex scheduling:

- This scheduling is more flexible than Multilevel queue scheduling.
- This algorithm helps in reducing the response time.
- In order to optimize the turnaround time, the SJF algorithm is needed which basically requires the running time of processes in order to schedule them. As we know that the running time of processes is not known in advance. Also

processes in order to schedule them. As we know that the running time of processes is not known in advance. Also, this scheduling mainly runs a process for a time quantum and after that, it can change the priority of the process if the process is long. Thus this scheduling algorithm mainly learns from the past behavior of the processes and then it can predict the future behavior of the processes. In this way, MFQS tries to run a shorter process first which in return leads to optimize the turnaround time.

Advantages of MFQS

- This is a flexible Scheduling Algorithm
- This scheduling algorithm allows different processes to move between different queues.
- In this algorithm, A process that waits too long in a lower priority queue may be moved to a higher priority queue which helps in preventing starvation.

Disadvantages of MFQS

- This algorithm is too complex.
- As processes are moving around different queues which leads to the production of more CPU overheads.
- In order to select the best scheduler this algorithm requires some other means to select the values

← Prev

Next →

studytonight.com

About Us
Testimonials
Authors
Collaborate
Privacy Policy
Terms
Contact Us
Suggest

Learn Coding (for beginners)
Tutorial Library
Interview Tests
Curious
Practice Coding
Educators Program

Coding Courses
Learn HTML
Learn CSS

Resources
C Language
C++/STL
Java
DBMS
Python
PHP
Android
Game Development
Data Structure & Alog.
Operating System
Computer Network
Computer Architecture
More...

Interview Tests
Java Interview Tests
Python Interview Tests
DBMS Interview Tests
Linux Interview Tests
Aptitude Tests
GATE 2022 Tests
More...