Chapter: CPU/ Process Scheduling

Multilevel Queue

A **multilevel queue** scheduling algorithm partitions the ready queue into several separate queues.

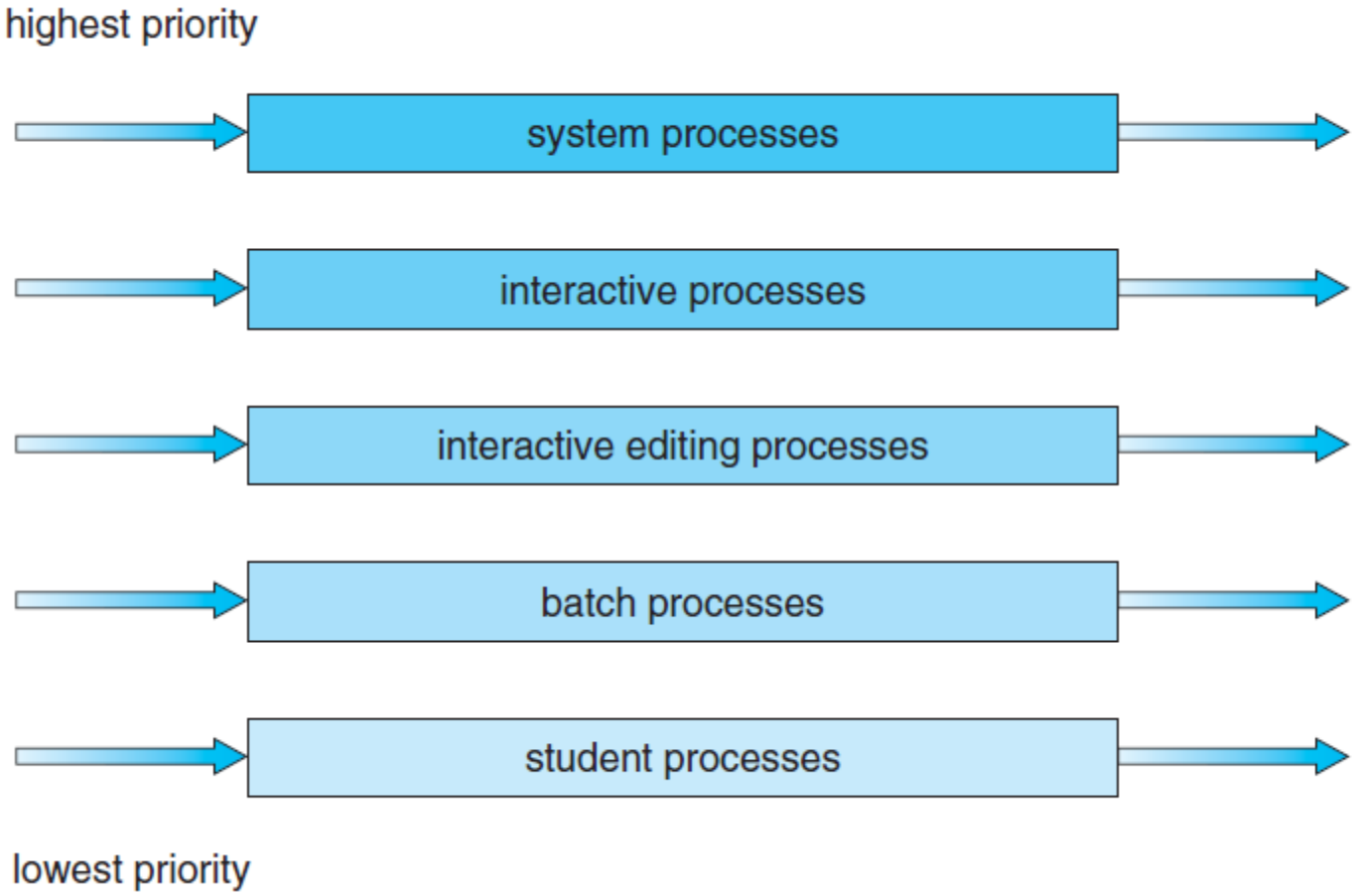
For Example: a multilevel queue scheduling algorithm with five queues, listed below in order of priority:

1. System processes
2. Interactive processes
3. Interactive editing processes
4. Batch processes
5. Student/ user processes

Multilevel Queue

- A process can move between various queues
- Multilevel Queue Scheduler defined by the following parameters:
  - No. of queues
  - Scheduling algorithms for each queue
  - Method used to determine when to upgrade / demote a process
  - Method used to determine which queue a process will enter and when that process needs service.

Multilevel Queue



Multilevel Queue

1. System Processes: These are programs that belong to OS (System Files)

2. Interactive Processes: Real Time Processes e.g. playing game online, listening to music online etc.

3. Batch Processes: Lots of processes are pooled and one process at a time is selected for execution. I/O by Punch Cards

4. Student/User Processes

Multilevel Queue

- As different type of processes are there so all cant be put into same queue and apply same scheduling algorithm.

Disadvantages:

1. **Until high priority queue is not empty**, No process from lower priority queues will be selected.
2. Starvation for lower priority processes

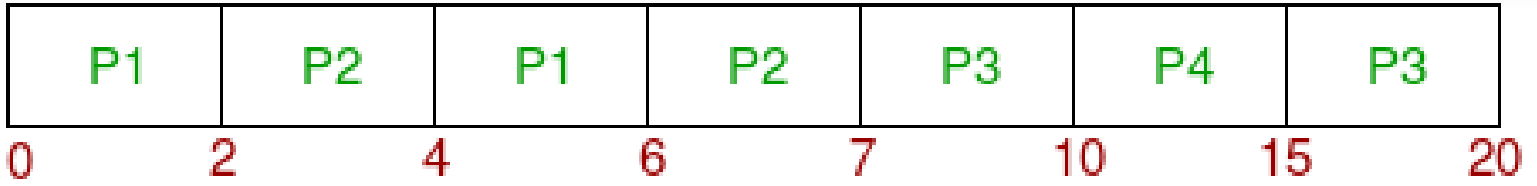
Advantage:

Can apply separate scheduling algorithm for each queue.

Practice: Multilevel Queue

Process	Arrival Time	Burst Time	Queue
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.


P1
P2
P1
P2
P3
P4
P3
0
2
4
6
7
10
15
20

Multilevel Feedback Queue

- Solution is: Multilevel Feedback Queue
- If a process is taking too long to execute.. Pre-empt it send it to low priority queue.
- Don’t allow a low priority process to wait for long.
- After some time move a least priority process to high priority queue → **Aging**

Multilevel Feedback Queue

✓Allows a process to move between queues.

✓The idea is to separate processes according to the characteristics of their CPU bursts.

✓If a process uses too much CPU time, it will be moved to a lower-priority queue.
✓This scheme leaves I/O-bound and interactive processes in the higher-priority queues.

✓In addition, 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.

Multilevel Feedback Queue

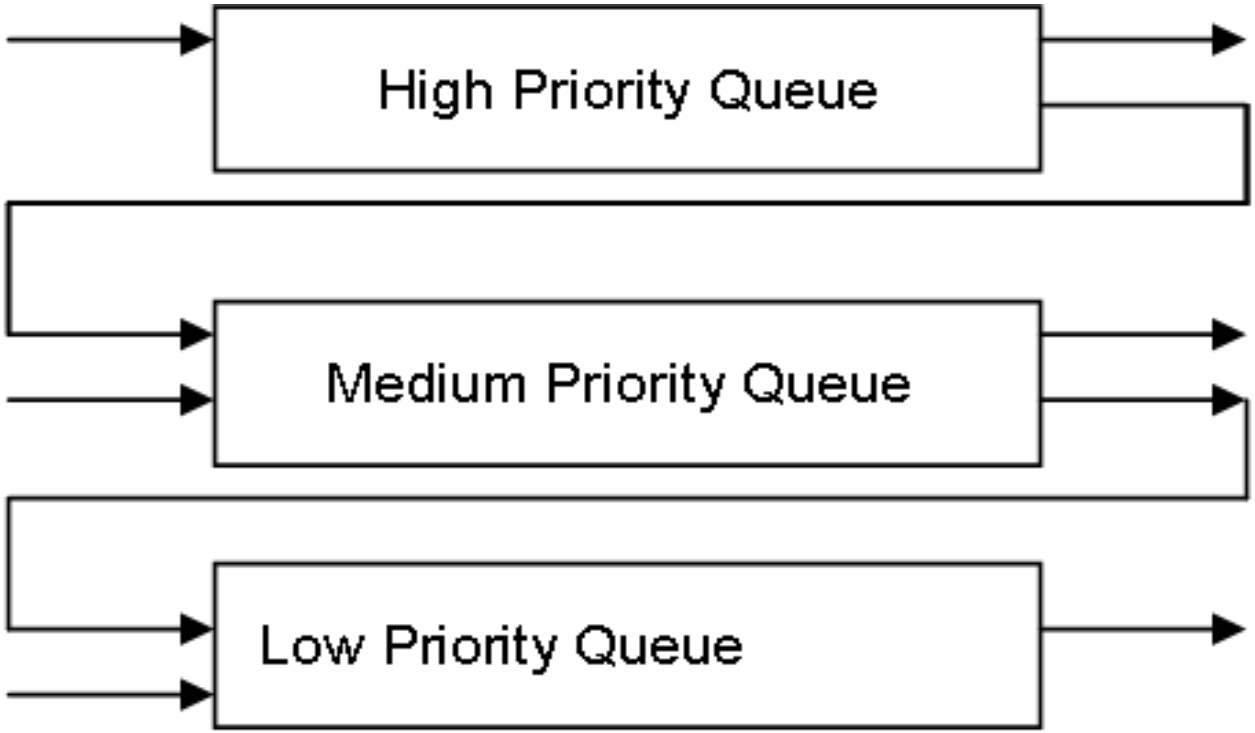


Figure – Multilevel Feedback Queue Scheduling

Multi-processor Scheduling

Concerns:

- If multiple CPUs are available, **load sharing** becomes possible.
  - Concentration is on systems in which the **processors are identical—homogeneous in terms of their functionality**.
  - Use any available processor to run any process in the queue

Approaches to Multiple-Processor Scheduling

- **1. Asymmetric multiprocessing**

All scheduling decisions, I/O processing, and other system activities **handled by a single processor—the master server**. The other processors execute only user code.

- Only one processor accesses the system data structures, reducing the need for data sharing.

Approaches to Multiple-Processor Scheduling

- **2. Symmetric multiprocessing (SMP)**

•Each processor is self-scheduling.

•All processes may be in a **common ready queue**, or each processor may have its **own private queue** of ready processes.

•Scheduling proceeds by having the scheduler for each processor examine the ready queue and select a process to execute.

Virtually all modern operating systems support SMP, including Windows, Linux, and Mac OS X

Issues concerning SMP systems

- **1. Processor Affinity**

(a process has an affinity for the processor on which it is currently running.)

- Consider what happens to cache memory when a process has been running on a specific processor?

The data most recently accessed by the process populate the cache for the processor. As a result, successive memory accesses by the process are often satisfied in cache memory.

Issues concerning SMP systems

- **1. Processor Affinity**

Because of the **high cost of invalidating and repopulating caches**, most SMP systems try to **avoid migration of processes from one processor to another and instead attempt to keep a process running on the same processor**.

This is known as **processor affinity—that is, a process has an affinity for the processor on which it is currently running.**

Issues concerning SMP systems

Forms of Processor Affinity

1. **Soft affinity**

When an operating system has a policy of attempting to keep a process running on the same processor—but not guaranteeing that it will do so—we have a situation known as soft affinity.

2. Hard affinity

Operating system will attempt to **keep a process on a single processor**, it is possible for a process to migrate between processors.

Issues concerning SMP systems

2. Load Balancing

On SMP systems, it is important to keep the workload balanced among all processors to fully utilize the benefits of having more than one processor.

Need of Load Balancing

One or more processors may sit idle while other processors have high workloads, along with lists of processes awaiting the CPU.

Issues concerning SMP systems

2. Load Balancing

Load balancing is necessary only on systems **where each processor has its own private queue** of eligible processes to execute.

On systems with a **common run queue, load balancing is often unnecessary,** because once a processor becomes idle, it immediately extracts a run able process from the common run queue.

In most operating systems that support SMP, each processor have a private queue of eligible processes.

Issues concerning SMP systems

2. Load Balancing

Two approaches to load balancing: **push migration and pull migration.**

1. Push migration: a specific task periodically checks the load on each processor and—if it finds an imbalance—evenly distributes the load by **moving (or pushing) processes from overloaded to idle or less-busy processors**.

2. Pull migration: occurs when an idle processor **pulls a waiting task from a busy processor.**

Issues concerning SMP systems

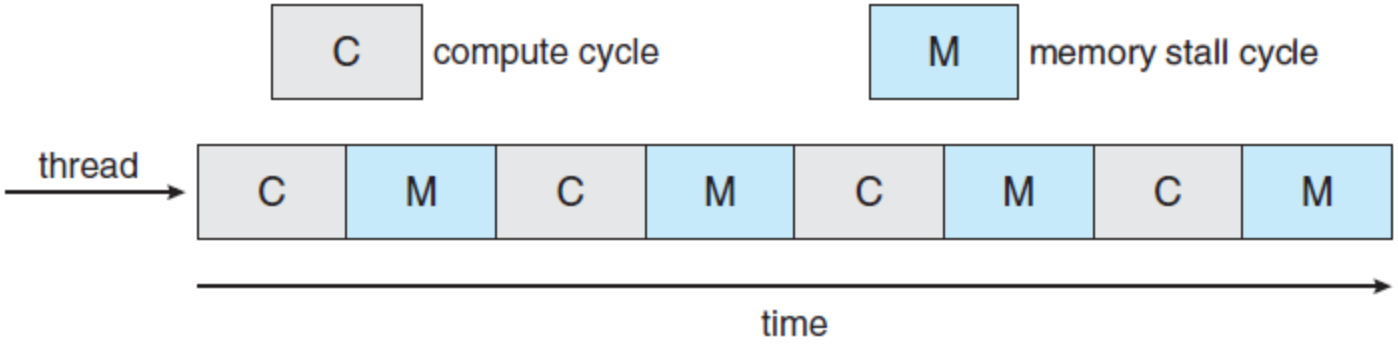
3. Multicore Processors

- A **multi-core processor** is a single computing component with two or more independent processing units called cores, which read and execute program instructions.
- A processor, or more commonly a CPU, is an individual processing device. It may contain multiple cores.
- **A core is a bank of registers and dedicated cache**
- Core is a structure that performs all of a processor's tasks, but is not an entire processor.

Issues concerning SMP systems

Memory Stall

Memory stall cycles **Number of cycles during which** processor is. stalled waiting for a memory access.



the processor can spend up to 50 percent of its time waiting for data to become available from memory.