Operating System

Multilevel Queue Scheduling

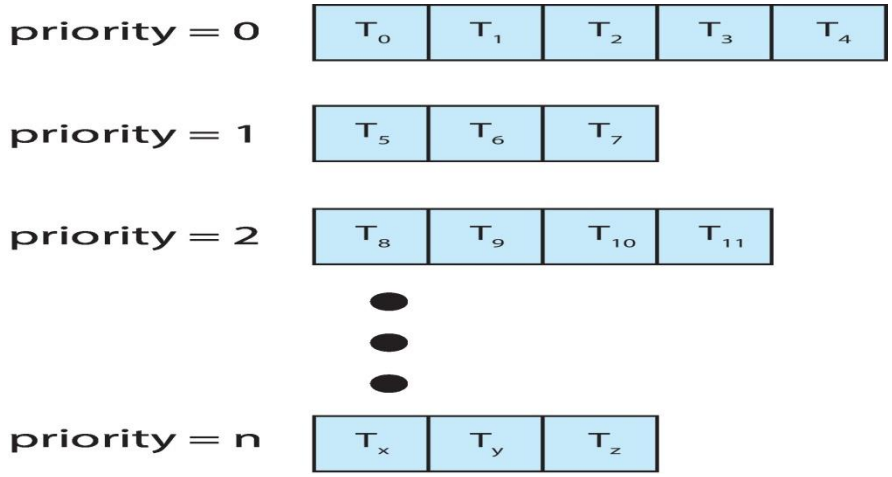
Class Presentations on Operating System

by Subhash Chand Agrawal

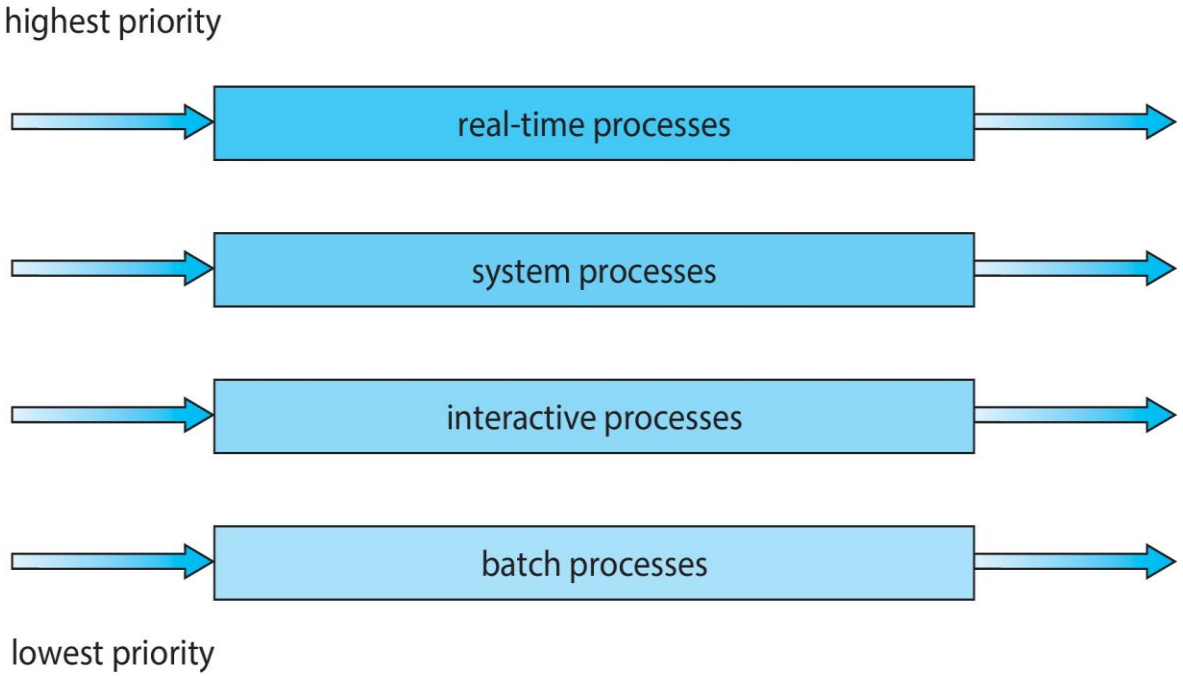
Multilevel Queue Scheduling

- Processes are easily classified into
  - Foregorund (Interactive)
  - Background(Batch)
- Different response-time requirements, and so might have different scheduling needs.
- In addition, foreground processes may have priority over background processes.
- A multilevel queue scheduling partitions the ready queue into 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.
- Each queue has its own scheduling algorithm.
- The foreground queue might be scheduled by Round Robin algorithm, while the background queue is scheduled by an FCFS algorithm.



Prioritization based upon process typevcv



Scheduling among the queues

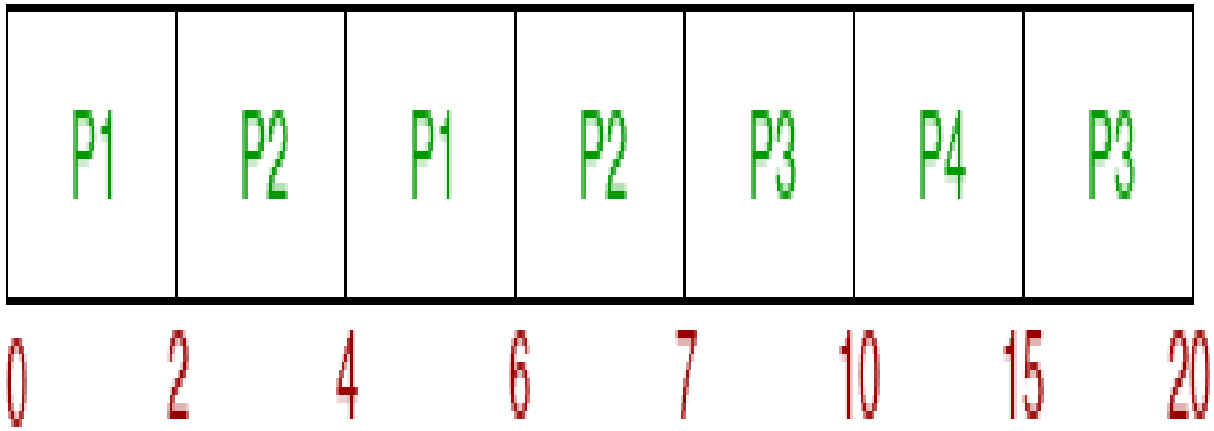
- Fixed priority preemptive scheduling method
- Time slicing

Example

- Priority of queue 1 is greater than queue 2. queue 1 uses Round Robin (Time Quantum = 2) and queue 2 uses FCFS.

Process	Arrival Time	CPU Burst Time	Queue Number
P1	0	4	1
P2	0	3	1
P3	0	8	2
P4	10	5	1

Solution

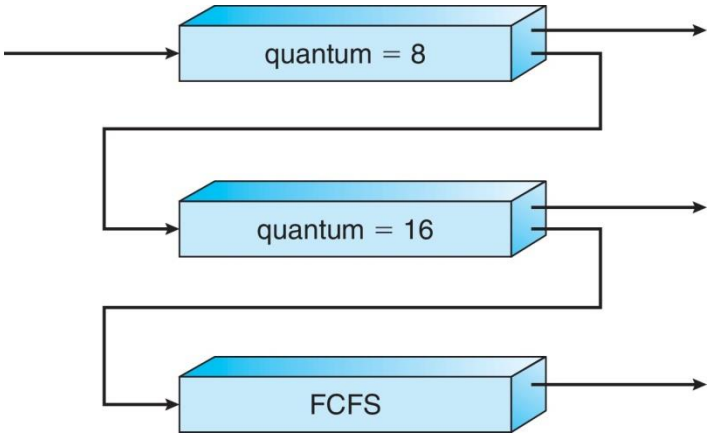


Multilevel feedback queue Scheduling

- 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.

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



Example

- Consider a system which has a CPU bound process, which requires the burst time of 40 seconds.The multilevel Feed Back Queue scheduling algorithm is used and the queue time quantum ‘2’ seconds and in each level it is incremented by ‘5’ seconds. Then how many times the process will be interrupted and on which queue the process will terminate the execution?

Solution

- Process P needs 40 Seconds for total execution.
  At Queue 1 it is executed for 2 seconds and then interrupted and shifted to queue 2.
  At Queue 2 it is executed for 7 seconds and then interrupted and shifted to queue 3.
  At Queue 3 it is executed for 12 seconds and then interrupted and shifted to queue 4.
  At Queue 4 it is executed for 17 seconds and then interrupted and shifted to queue 5.
  At Queue 5 it executes for 2 seconds and then it completes.
  Hence the process is interrupted 4 times and completes on queue 5.

Multi-Processor Scheduling

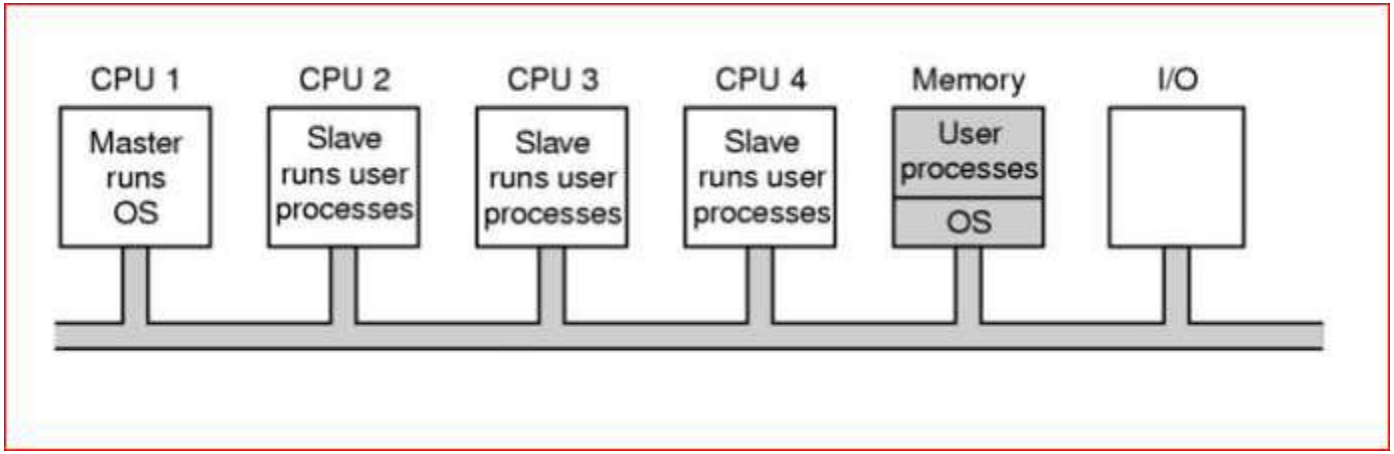
- In multiple-processor scheduling multiple CPU's are available.
- Load Sharing becomes possible, as distributed among these available processor.
- In general, the multiprocessor scheduling is complex as compared to single processor scheduling.

Multi-Processor scheduling

- Multi-Processor scheduling is of two types:
- # Asymmetric Scheduling
- # Symmetric Scheduling

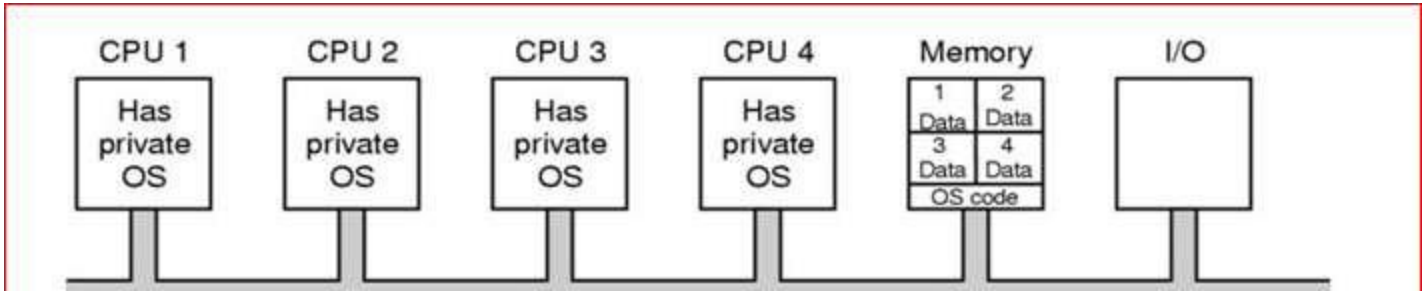
Approaches to Multiple-Processor Scheduling –

- One approach is when all the scheduling decisions, I/O processing, resource allocation etc. are handled by a single processor which is called the **Master Server** and the other processors executes only the **user code**. This entire scenario is called **Asymmetric Multiprocessing**.



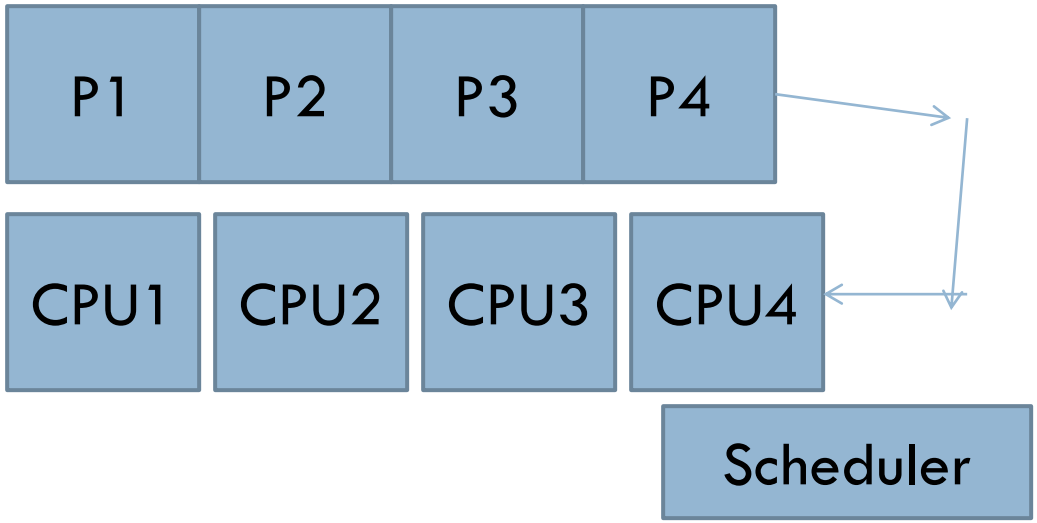
Symmetric Multiprocessing

- A second approach uses Symmetric Multiprocessing where each processor is self scheduling.
- All processes may be in a common ready queue or each processor may have its own private queue for ready processes.
- The scheduling proceeds further by having the scheduler for each processor examine the ready queue and select a process to execute.



ASMP

- In ASMP, there is a dedicated CPU to run the scheduler.
- This scheduler decides which process will execute on which processor.

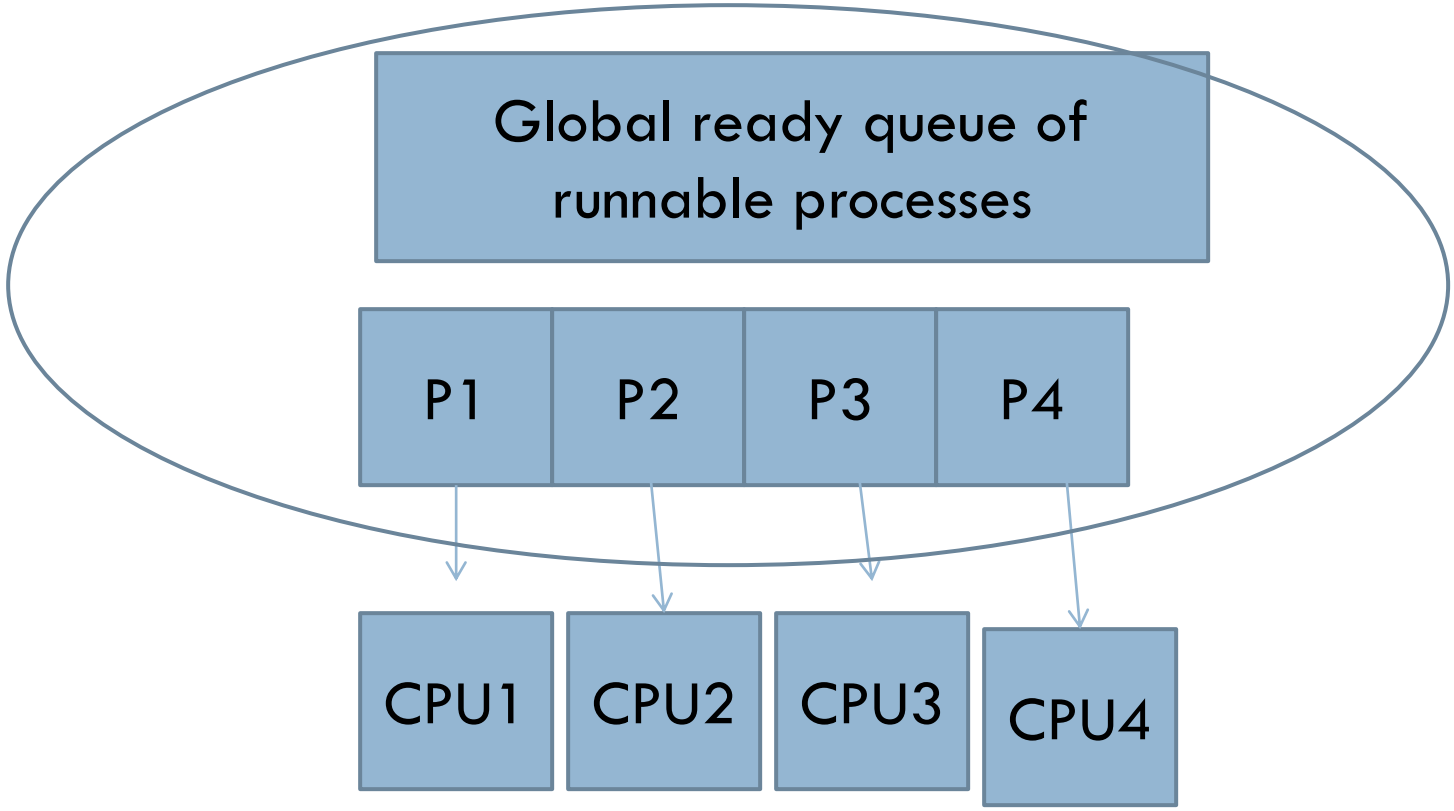


ASMP

- **Advantage**
- Simple to implement.
  - Scheduler have local queue in which processes are ready to run. It would use some mechanism to decide which process will execute on which processor.
- **Disadvantages**
- Performance Degradation
  - All CPUs are waiting for the scheduler CPU for which process has to execute next.

SMP

- The processes requiring CPU execution are maintained in the Global Ready queue.
- Each processor can do the process scheduling on its own and select a process from the global ready queue for the execution


Global ready queue of runnable processes
P1
P2
P3
P4
CPU1
CPU2
CPU3
CPU4

Problems

- **Locking system**
- **Contention for the global queue**

Processor Affinity

- If the process migrates to another processor, the contents of the cache memory must be invalidated for the first processor and the cache for the second processor must be repopulated.
- Because of the high cost of invalidating and repopulating caches, most of the SMP(symmetric multiprocessing) systems try to avoid migration of processes from one processor to another and try to keep a process running on the same processor. This is known as **PROCESSOR AFFINITY.**

- There are two types of processor affinity:
- # Soft Affinity
- # Hard Affinity

Soft Affinity

- When an operating system has a policy of attempting to keep a process running on the same processor but not guaranteeing it will do so, this situation is called soft affinity.

Hard Affinity

- Some systems such as Linux also provide some system calls that support Hard Affinity which allows a process to migrate between processors.

Load Balancing

- Load Balancing is the **phenomena which keeps** the **workload evenly distributed across all processors in an** SMP
- There are two general approaches to load balancing :
- # Push Migration
- # Pull Migration system.

Load Balancing : Push Migration

- In push migration a task routinely checks the load on each processor and if it finds an imbalance then it evenly distributes load on each processors by moving the processes from overloaded to idle or less busy processors.

Load Balancing : Pull Migration

- Pull Migration occurs when an idle processor pulls a waiting task from a busy processor for its execution.