SNS COLLEGE OF TECHNOLOGY

Coimbatore-35.

An Autonomous Institution

COURSE NAME : 23CST202– OPERATING SYSTEMS
II YEAR/ IV SEMESTER

UNIT-II Process Scheduling

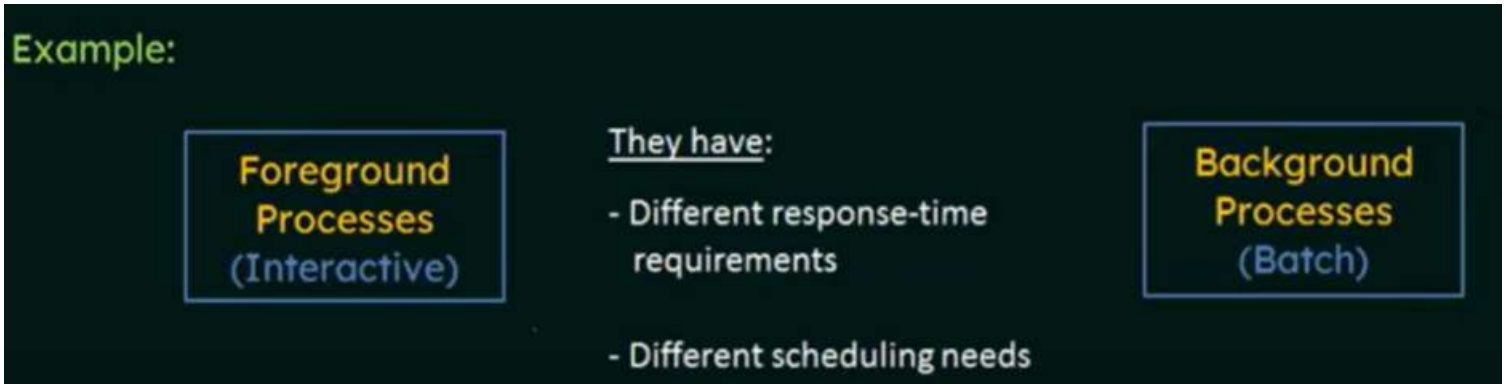
Topic:

- **Multilevel Queue Scheduling**
- **Multilevel Feedback Queue Scheduling**
- **Multiprocessor Scheduling**

Dr. B.Vinodhini
Associate Professor
Department of Computer Science and Engineering

Multilevel Queue

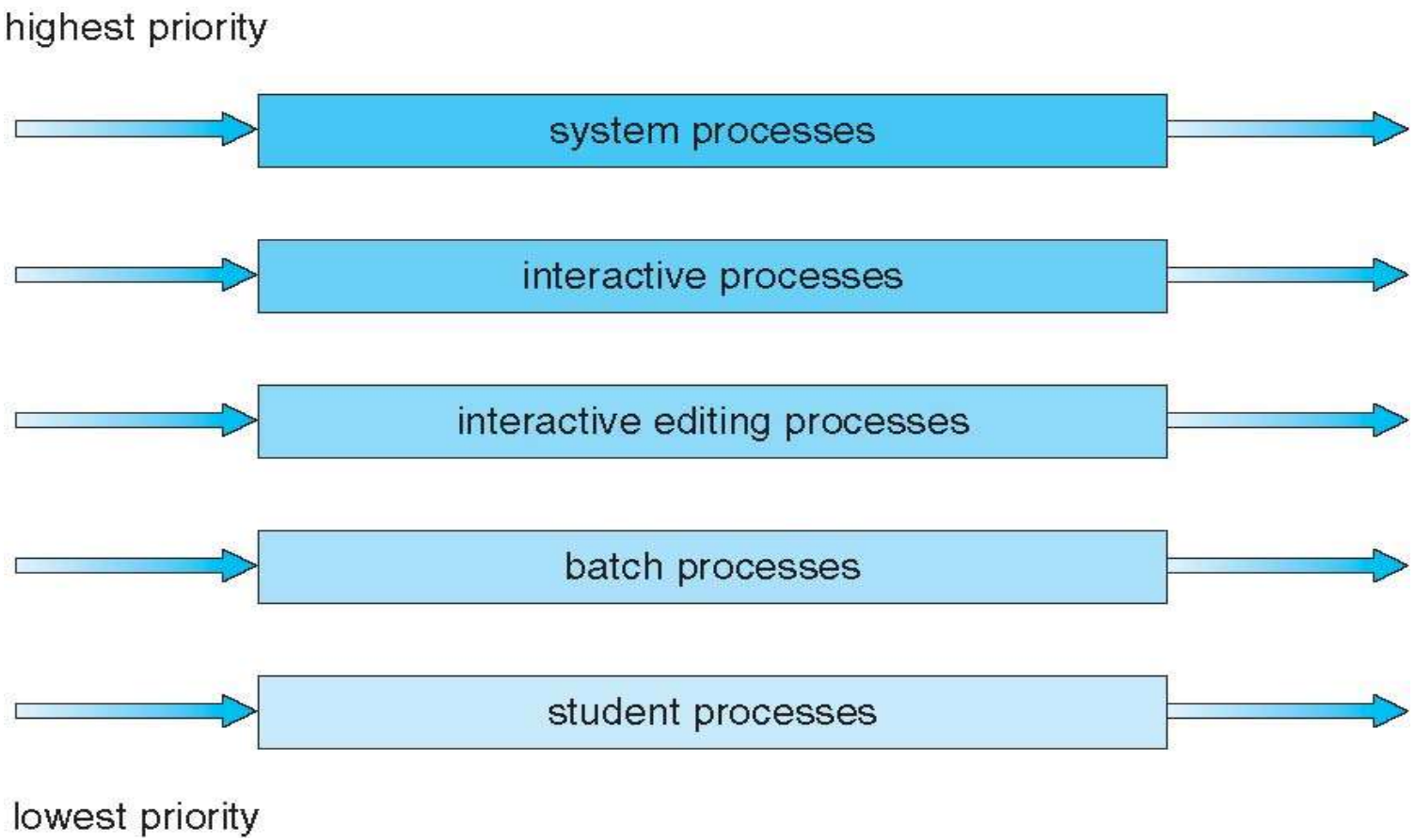
- Another class of scheduling algorithms has been created for situations in which processes are easily classified into different groups.
  - **foreground (interactive)- may have priority over back ground processes**
  - **background (batch)**



- A multilevel queue scheduling algorithm partitions the ready queue into several 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:
  - foreground – RR
  - background – FCFS

Multilevel Queue

- 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 Feedback Queue

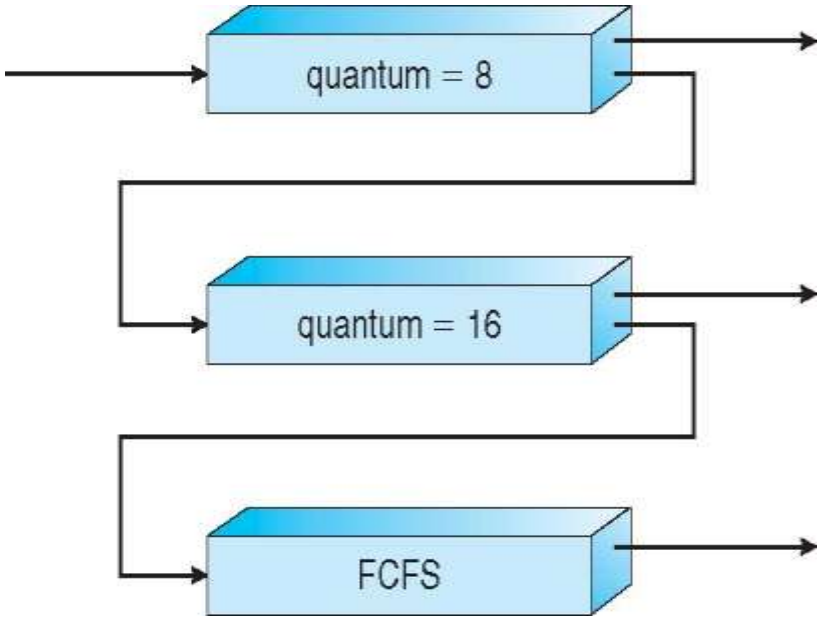
- The multilevel feedback –queue scheduling algorithm allows a process to move between queues.
- The idea is to separate processes according to the characteristics of their CPU bursts.
- If a process uses too much CPU time, it will be moved to a lower-priority queue.
- This scheme leaves I/O bound and interactive processes in the highest priority queues
- 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

- 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 to higher priority queue
  - method used to determine when to demote a process to lower priority queue
  - 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



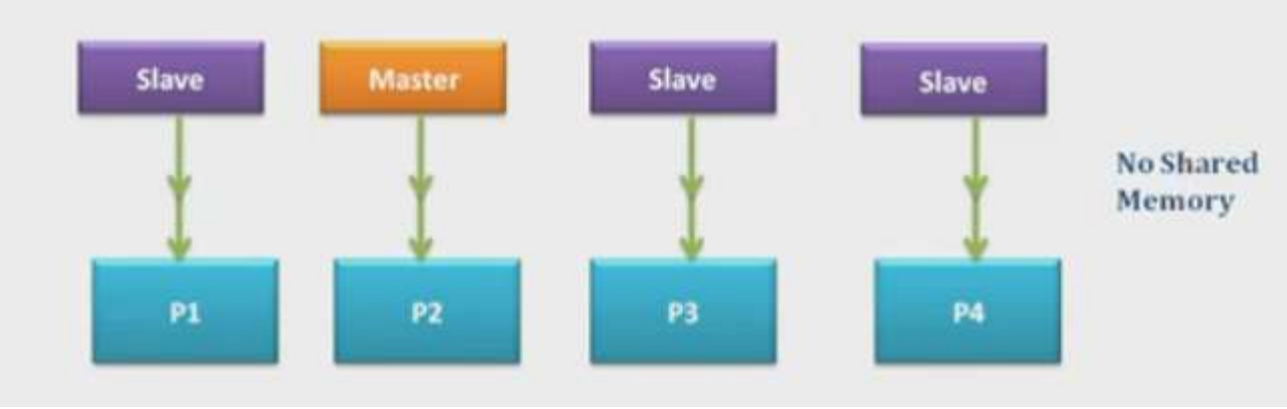
- Scheduling
  - A new job enters queue Q_0 which is served FCFS
    - When it gains CPU, job receives 8 milliseconds
    - If it does not finish in 8 milliseconds, job is moved to queue Q_1
  - At Q_1 job is again served FCFS and receives 16 additional milliseconds
    - If it still does not complete,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

Multiple-Processor Scheduling

- CPU scheduling becomes more complex when multiple CPUs are available
- processors are identical- **Homogeneous processors** in terms of their functionality; we can then use any available processor to run any process in the queue.

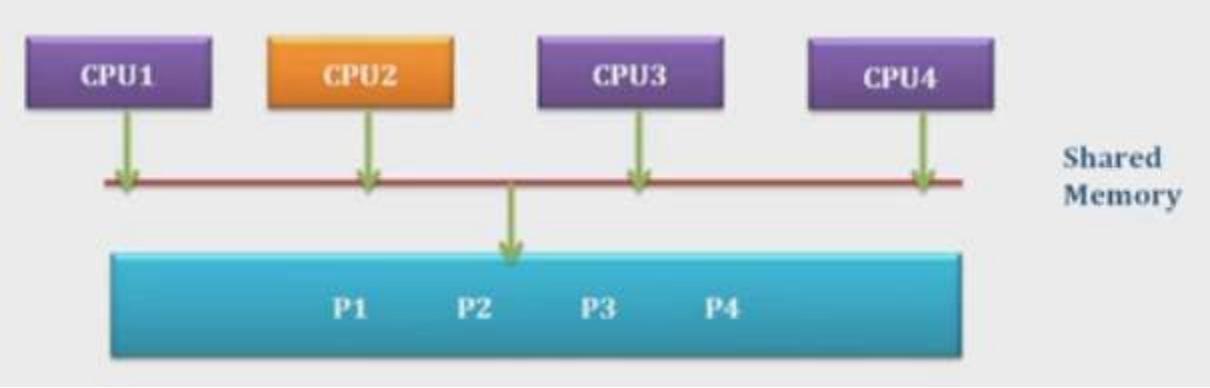
Approaches to Multiple-Processor Scheduling

- **Asymmetric multiprocessing** – When all the scheduling decisions and I/O processing are handled by a single processor which is called the master server and the other processors executes only the user code.
- This is simple and reduces the need for data sharing.
- only one processor accesses the system data structures, alleviating the need for data sharing



Multiple-Processor Scheduling

- **Symmetric multiprocessing (SMP)** – each processor is self-scheduling, all processes in common ready queue, or each has its own private queue of ready processes
- The scheduling proceeds further by having the scheduler for each processor examine the ready queue and selects a process to execute.



Multiple-Processor Scheduling

- **Processor affinity** –System tries to avoid migration of processes from one processor to another and try to keep a process running on the same processor.
- Two Types of affinity

- **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 this situation is called soft affinity.
- **hard affinity-** allowing a process to specify that it is not *to migrate to other processors*

Multiple-Processor Scheduling – 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. Otherwise, one or more processors may sit idle while other processors have high workloads, along with lists of processes awaiting the CPU.
- **Load balancing** attempts to keep workload evenly distributed
- There are two general approaches to load balancing: push migration and pull migration.
- **Push migration** – periodic task checks load on each processor, and if found pushes task from overloaded CPU to other CPUs
- **Pull migration** –It occurs idle processors pulls waiting task from busy processor

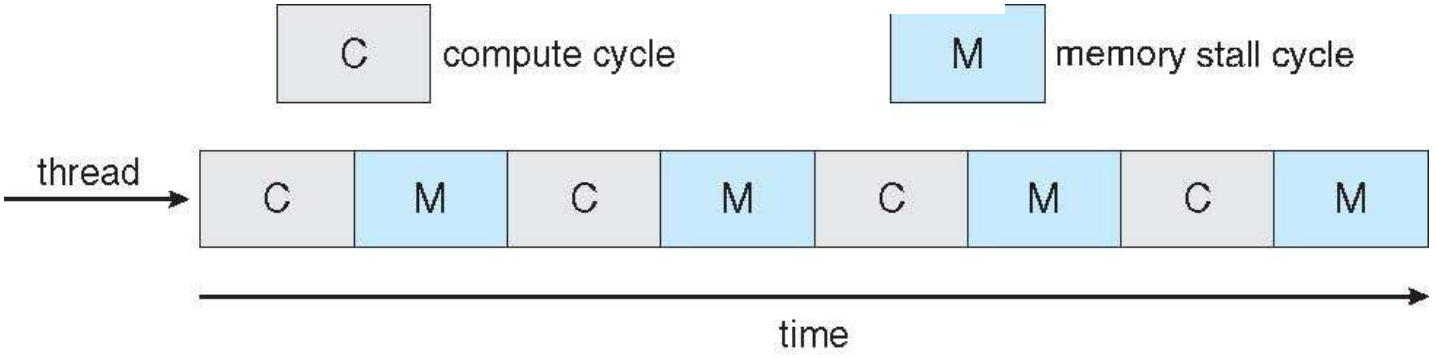
Multicore Processors

- A recent trend in computer hardware has been *to place multiple processor cores on the same physical chip*, resulting in Multicore processor. Each core has a register set to maintain its architectural state and appears to the operating system to be a separate physical processor.
- Faster and consumes less power
- Multiple threads per core also growing
  - Takes advantage of memory stall to make progress on another thread while memory retrieve happens

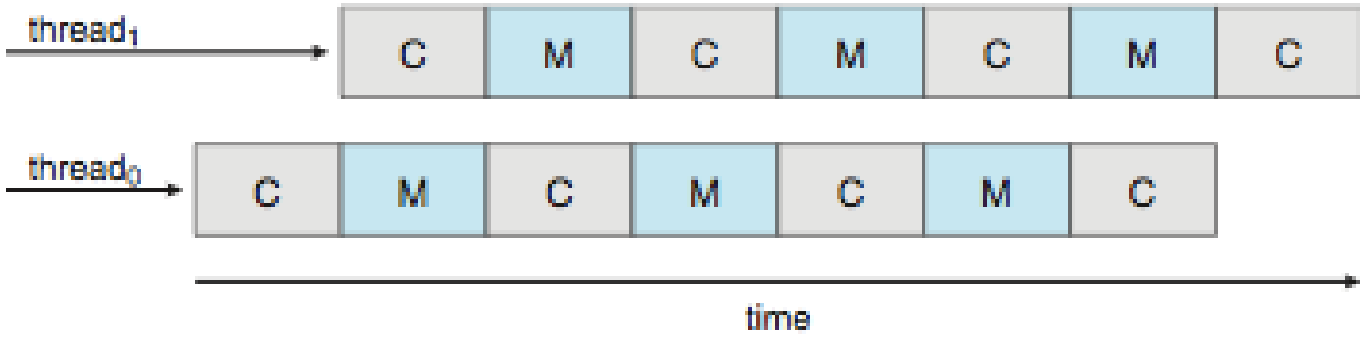
Multithreaded Multicore System

Researchers have discovered that when a processor accesses memory, it spends a significant amount of time waiting for the data to become available. This situation, known as a Memory stall.

Figure Memory stall.

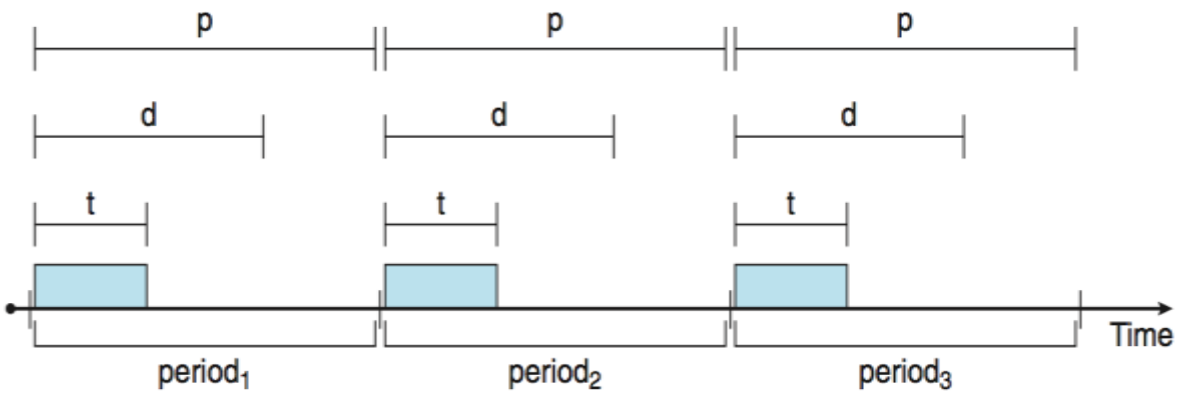


To remedy this situation, many recent hardware designs have implemented multithreaded processor cores in which two (or more) hardware threads are assigned to each core. That way, if one thread stalls while waiting for memory, the core can switch to another thread. Two ways-Coarse grind,Fine grind



Real time CPU Scheduling

- Hard real-time systems have stricter requirements. A task must be serviced by its deadline; service after the deadline has expired is the same as no service at all.
- Processes have new characteristics: **periodic** ones require CPU at constant intervals
  - Has processing time *t*, deadline *d,* period *p*
  - $0 \leq t \leq d \leq p$
  - **Rate** of periodic task is $1/p$



Real time CPU Scheduling

A process may have to announce its deadline requirements to the scheduler. Then, using a technique known as an admission-control algorithm, the scheduler either admits the process, guaranteeing that the process will complete on time, or rejects the request as impossible.

we explore scheduling algorithms that address the deadline requirements of hard real-time systems

.

- Rate-Monotonic Scheduling
- Earliest-Deadline-First Scheduling
- Proportional Share Scheduling
- Pthread Scheduling

Algorithm Evaluation

- How to select CPU-scheduling algorithm for an OS?
- Determine criteria, then evaluate algorithms
- **Deterministic modeling**
  - Type of **analytic evaluation**
  - Takes a particular predetermined workload and defines the performance of each algorithm for that workload
- Consider 5 processes arriving at time 0:

Process	Burst Time
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

Algorithm Evaluation