CSCI 4730/6730 OS
(Chap #5 CPU Scheduling – Part III)

In Kee Kim

Department of Computer Science

University of Georgia

Where are we?

- Scheduling for Single Processor (w/ Single Core)
  - FCFS (First-Come First-Served)
  - Shortest-Job-First (SJF)
    - Non Preemptive and Preemptive
  - Round Robin (RR)
  - Priority Scheduling
  - Multilevel Queue Scheduling
  - Multilevel Feedback Queue Scheduling

Priority Scheduling

- A priority number (Integer) is associated with each process
  - High priority = Smaller number or Larger number?
  - It depends on OS.
- Priority
  - Static – Priority is the same throughout the executions
  - Dynamic – Priority is changing

Priority Scheduling

- The CPU is allocated to the process with the highest priority, but multiple options…
  - What if there are processes with the same priority?
  - Preemptive vs. Non preemptive

- What is the relation of SJF/SRTF to priority scheduling?
  - Is SJF/SRTF a priority scheduling?
  - SJF/SRTF is ***priority scheduling*** where priority is the inverse of predicted next CPU burst time

Priority Scheduling

Proc	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

All processes arrive at 0
Assume that
small integer = high priority

Priority Scheduling

Proc	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

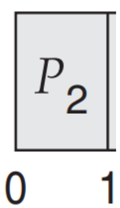
All processes arrive at 0

Which process should be scheduled first?

Priority Scheduling

Proc	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

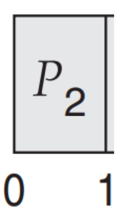
All processes arrive at 0



Priority Scheduling

Proc	Burst Time	Priority
P_1	10	3
~~P_2~~	~~1~~	~~1~~
P_3	2	4
P_4	1	5
P_5	5	2

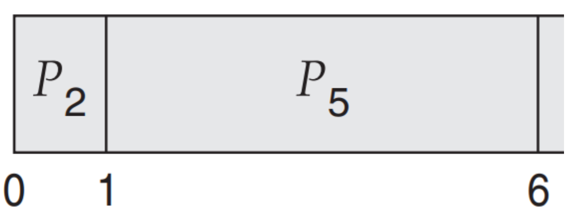
All processes arrive at 0



Priority Scheduling

Proc	Burst Time	Priority
P_1	10	3
~~P_2~~	~~1~~	~~1~~
P_3	2	4
P_4	1	5
P_5	5	2

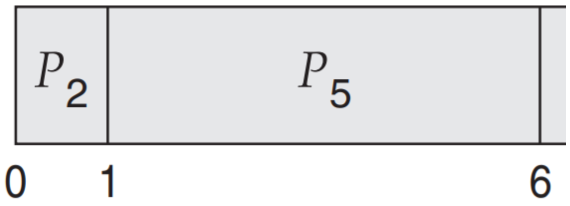
All processes arrive at 0



Priority Scheduling

Proc	Burst Time	Priority
P_1	10	3
~~P_2~~	~~1~~	~~1~~
P_3	2	4
P_4	1	5
~~P_5~~	~~5~~	~~2~~

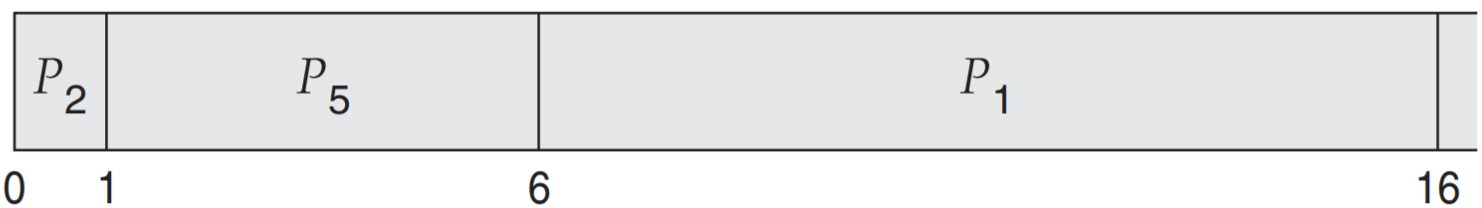
All processes arrive at 0



Priority Scheduling

Proc	Burst Time	Priority
P_1	10	3
~~P_2~~	~~1~~	~~1~~
P_3	2	4
P_4	1	5
~~P_5~~	~~5~~	~~2~~

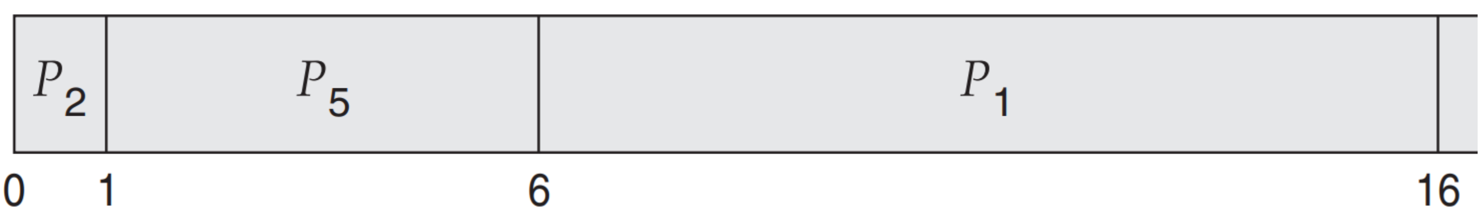
All processes arrive at 0



Priority Scheduling

Proc	Burst Time	Priority
~~P_1~~	~~10~~	~~3~~
~~P_2~~	~~1~~	~~1~~
P_3	2	4
P_4	1	5
~~P_5~~	~~5~~	~~2~~

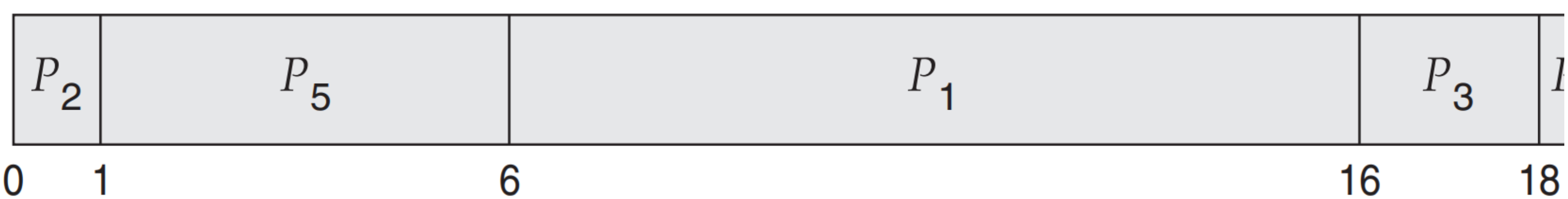
All processes arrive at 0



Priority Scheduling

Proc	Burst Time	Priority
~~P_1~~	~~10~~	~~3~~
~~P_2~~	~~1~~	~~1~~
P_3	2	4
P_4	1	5
~~P_5~~	~~5~~	~~2~~

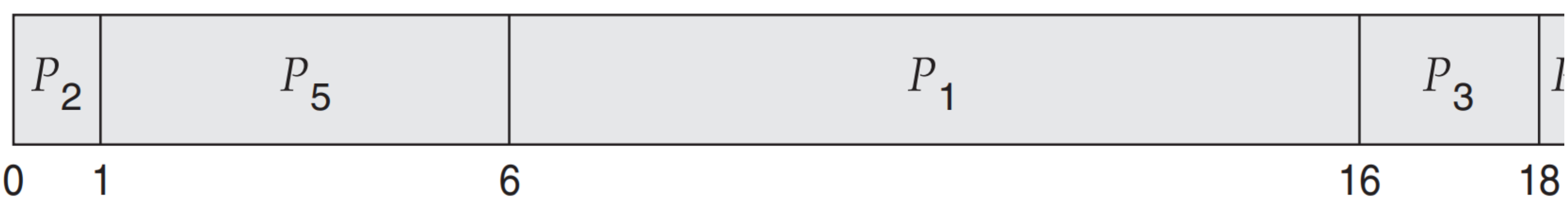
All processes arrive at 0



Priority Scheduling

Proc	Burst Time	Priority
~~P_1~~	~~10~~	~~3~~
~~P_2~~	~~1~~	~~1~~
~~P_3~~	~~2~~	~~4~~
P_4	1	5
~~P_5~~	~~5~~	~~2~~

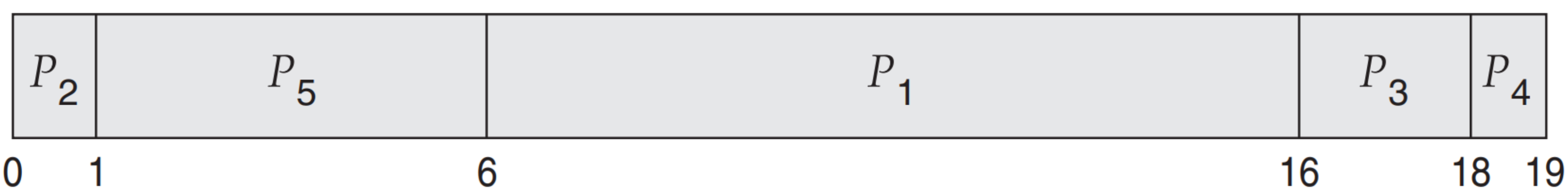
All processes arrive at 0



Priority Scheduling

Proc	Burst Time	Priority
~~P_1~~	~~10~~	~~3~~
~~P_2~~	~~1~~	~~1~~
~~P_3~~	~~2~~	~~4~~
P_4	1	5
~~P_5~~	~~5~~	~~2~~

All processes arrive at 0

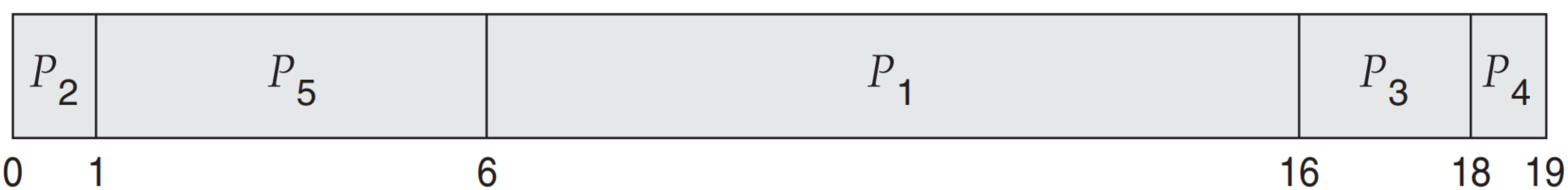


Non Preemptive

Priority Scheduling

Proc	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

All processes arrive at 0



Waiting Time?
Turnaround Time?

Preemptive Priority Scheduling

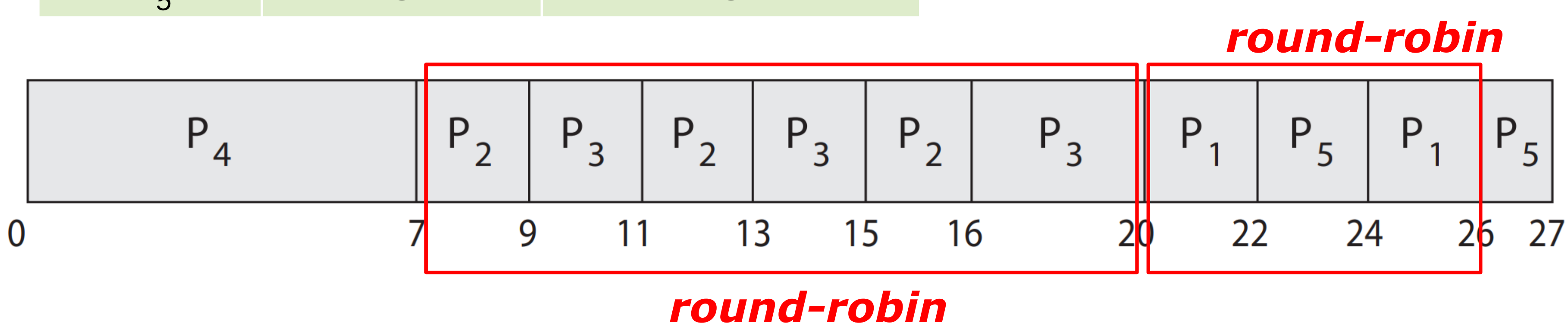
Proc	Arrival Time	Burst Time	Priority
P_1	0	8	3
P_2	1	2	4
P_3	3	4	4
P_4	4	1	5
P_5	5	6	2
P_6	6	5	6
P_7	10	1	1

Priority Scheduling + Round Robin

- Run the process with the highest priority. Processes with the same priority run round-robin

Proc	Burst Time	Priority
P_1	4	3
P_2	5	2
P_3	8	2
P_4	7	1
P_5	3	3

All processes arrive at 0
***q* = 2**



Priority Scheduling Problem

- Can have "Indefinite Blocking" or ***"Starvation"***
  - Low priority processes are in ready queue.
  - What if steady stream of high priority processes are coming to the system.
  - High priority processes are continuously occupying the CPU.
- Two approaches
  - Aging – Dynamic priority

Aging

- Very simple –OS gradually increases the priority of (low priority) processes
- The processes will eventually get CPU

Priority Scheduling

- Pros and Cons

Pros	Cons
• Having *priority* – in real-world, different processes have different significance	• Starvation • Determining the priority is difficult

Priority Scheduling

- Pros and Cons

Pros	Cons
• Having ***priority*** – in real-world, different processes *may* have different significance	• Starvation • Determining the priority is difficult

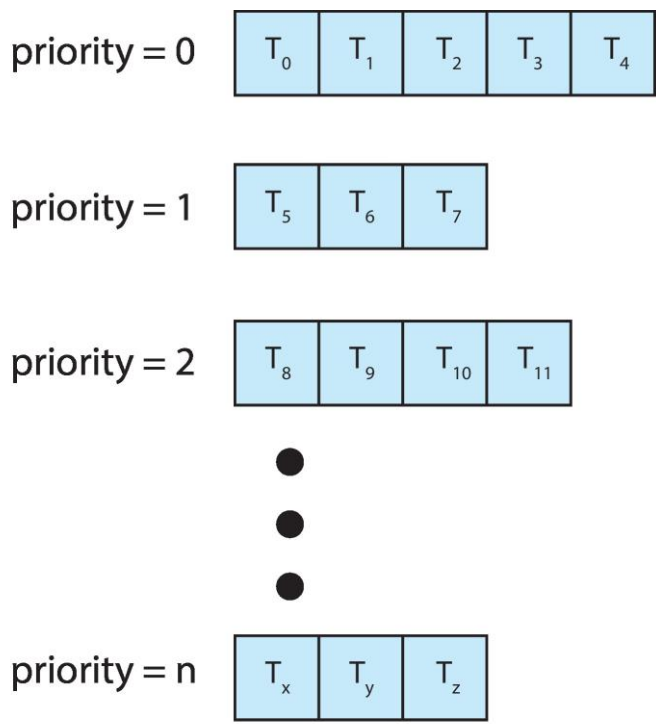
Priority Scheduling

- Pros and Cons

Pros	Cons
• Having ***priority*** – in real-world, different processes *may* have different significance	• Starvation • Determining the priority is difficult

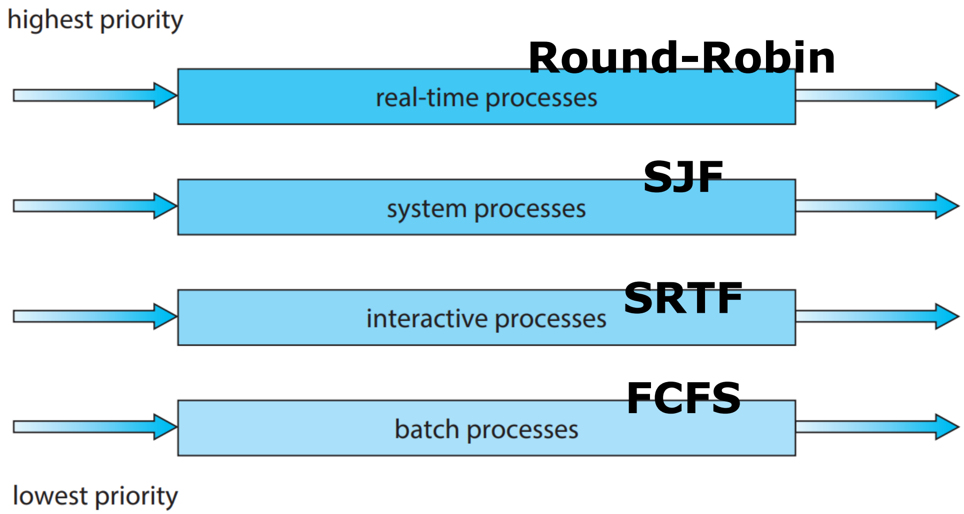
Multilevel Queue Scheduling

- With priority scheduling, have separate queues for each priority.
- Schedule the process in the highest-priority queue!



Multilevel Queue Scheduling

- Prioritization based upon process type



1. Different Qs with different priority (“Priority Scheduling”)
2. Different Qs can have different scheduling policy
3. **No process migration to a different Q**
4. Processes in a specific Q can be executed when higher priority Qs are empty.
5. “Preemptive” scheduling

Multilevel Queue Scheduling

- Pros and Cons

Pros	Cons
• Flexibility with multiple queues	• Starvation – “aging” doesn’t work

Multilevel Queue Scheduling

- Pros and Cons

Pros	Cons
• Flexibility with multiple queues	• Starvation – "aging" doesn't work

Multilevel Queue Scheduling

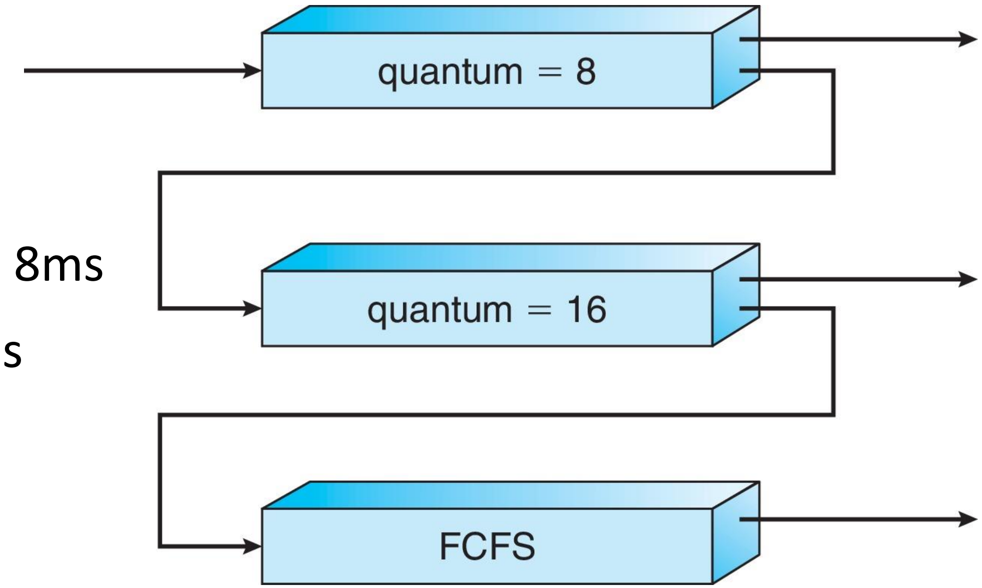
- Pros and Cons

Pros	Cons
• Flexibility with multiple queues	• Starvation – “aging” doesn’t work

MFQS

- Multilevel Feedback Queue Scheduling
- A process can move between the various queues
  - Aging can be implemented using MFQS
  - **“No Starvation”**
- MFQS defined by the following parameters:
  - Number of Qs
  - Scheduling algorithms for each Queue
  - Method used to determine when to upgrade a process
  - Method used to determine when to demote a process
  - Method used to determine which Queue a process will enter when that process needs service

MFQS



- Three queues:
  - Q_0 – RR with time quantum 8ms
  - Q_1 – RR time quantum 16ms
  - Q_2 – FCFS
- Scheduling
  - A new process enters queue Q_0 which is served in RR
    - When it gains CPU, the process receives 8 ms
    - 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

Multi-Processor Scheduling

- Previous algorithms are for Single Processor (Core)
- For Multi-Processor Systems
  - Algorithms designed for single core/processor are ***sub-optimal***
  - CPU scheduling is more complex when multiple CPUs are available

Multi-Processor Scheduling

- Multiprocessor systems
  - Multicore CPUs
  - Multithreaded Cores
  - NUMA Systems
  - Heterogeneous Multi-Processing (HMP)

Homogeneous Processors/Cores

Multiple-Processor Scheduling

- Asymmetric Multi-Processing (AMP)
  - A “central scheduler” running on a designated core
  - Centralized (or Monolithic) Scheduler (in Distributed Systems)
- Symmetric Multi-Processing (SMP)
  - Each processor is self-scheduling
- Most real-world systems (including large-scale distributed systems) use a hybrid model of AMP/SMP.

Multicore Processors

- In MP
  - Each core maintains its architectural state
  - Each core is considered as a separate logical CPU
- Leveraging memory stall → multiple HW threads
  - When a processor accesses memory, it spends a significant amount of time waiting for the data to become available.

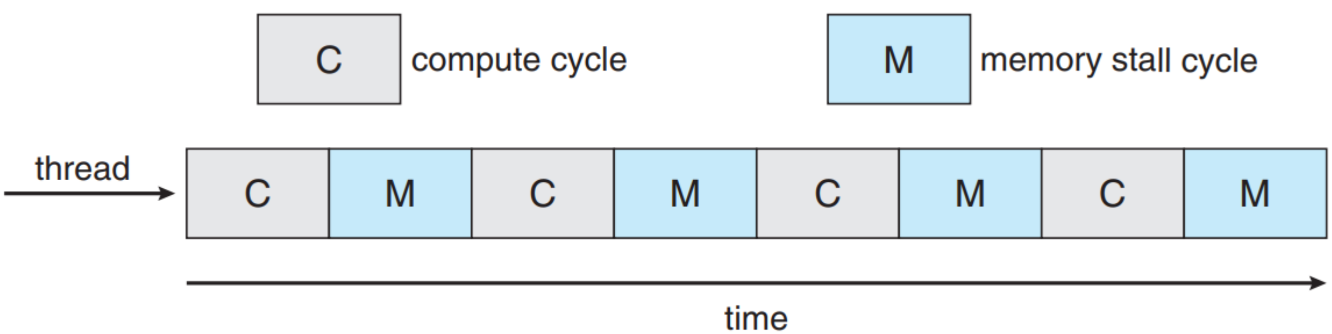


Figure 5.12 Memory stall.

Multicore Processors

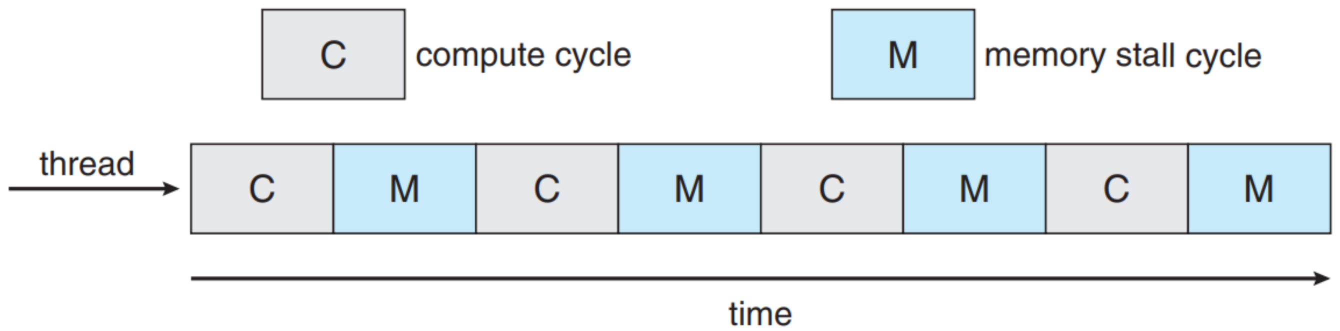


Figure 5.12 Memory stall.

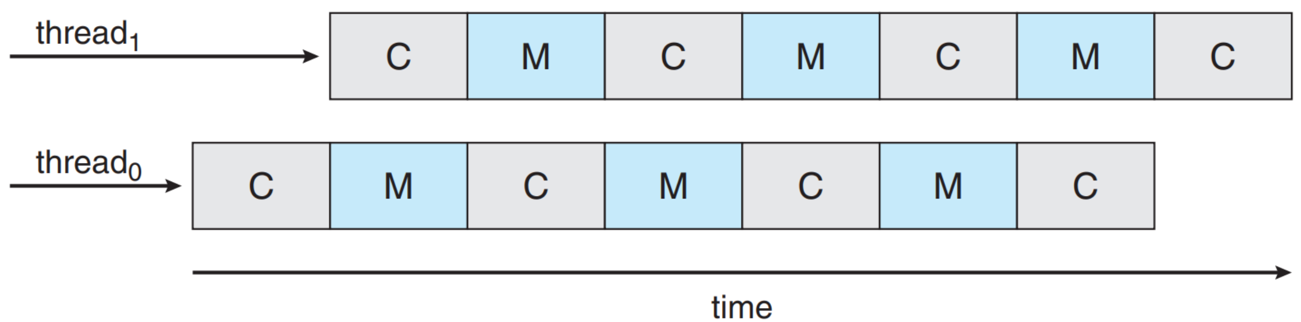
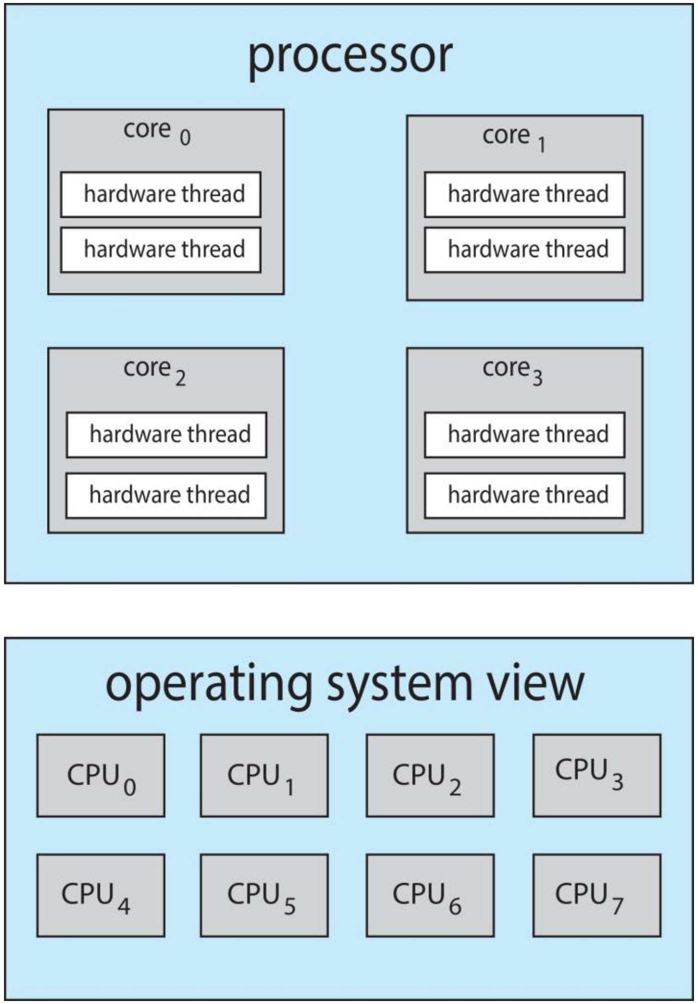


Figure 5.13 Multithreaded multicore system.

Multithreaded Multicore System

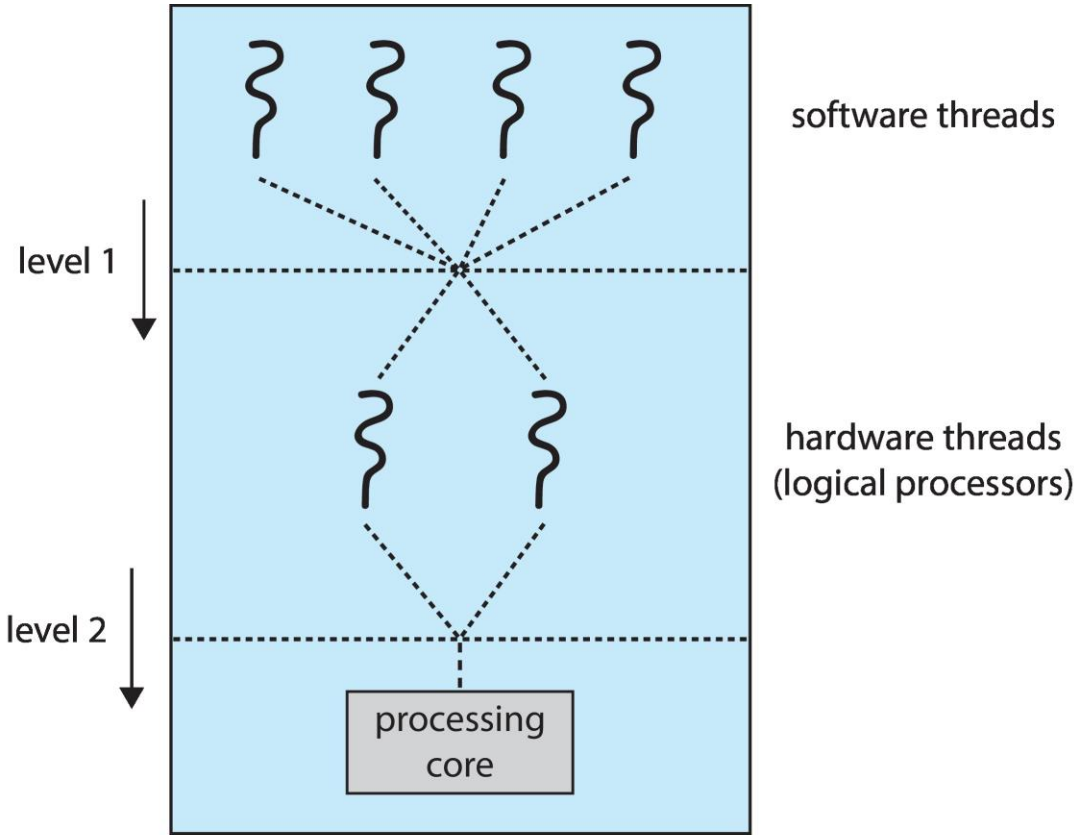
- **Chip-multithreading** (CMT) assigns each core multiple hardware threads. (Intel refers to this as **hyperthreading**.)

- On a quad-core system with 2 hardware threads per core, the operating system sees 8 logical processors.



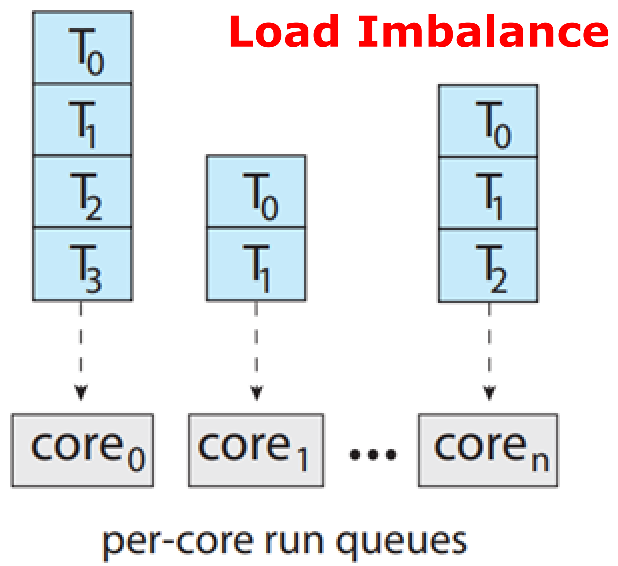
Multithreaded Multicore System

- Two levels of scheduling:
  1. The operating system deciding which software thread to run on a logical CPU
  2. How each core decides which hardware thread to run on the physical core.



Load Balancing

- If SMP, need to keep all CPUs loaded for efficiency
  - Each processor has own Q.



Load Balancing

- **Load balancing**
  - Attempts to keep workload evenly distributed
- **Push migration**
  - Periodic task checks load on each processor, and if found pushes task from overloaded CPU to other CPUs
- **Pull migration**
  - Idle processors pulls waiting task from busy processor

Multiple-Processor Scheduling – Processor Affinity

- Regarding process/thread placements on cores
- When thread ***A*** has been running on processor ***P***, the cache contents of processor ***P*** stores the memory accesses by thread ***A***.
- **Processor Affinity**
  - Thread ***A*** has affinity for Processor ***P***.

Multiple-Processor Scheduling – Processor Affinity

- **Processor Affinity**
  - Thread ***A*** has affinity for Processor ***P***.
- How about load balancing?
  - Should OS move Thread ***A*** to another processor when load imbalance happens?
- Load balancing can be a problem
  - A thread may be moved from one processor to another to balance loads
  - Loss of cache content

Processor Affinity

- **Soft affinity** – OS attempts to keep a thread running on the same processor, but no guarantees.
- **Hard affinity** – allows a process to specify a set of processors it may run on.

NUMA and CPU Scheduling

- If OS is ***NUMA-aware***, it will assign memory closes to the CPU the thread is running on.
  - **Supporting Data Locality!**

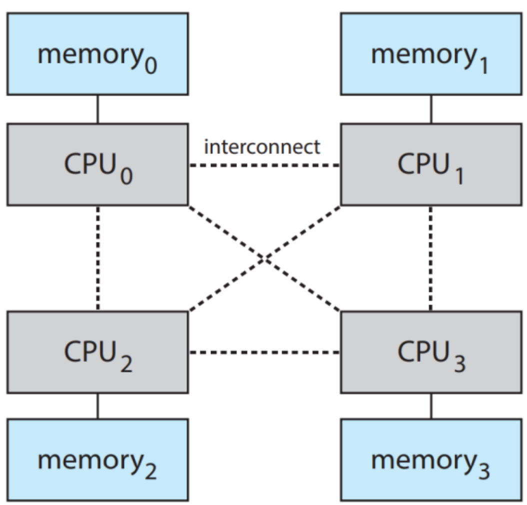
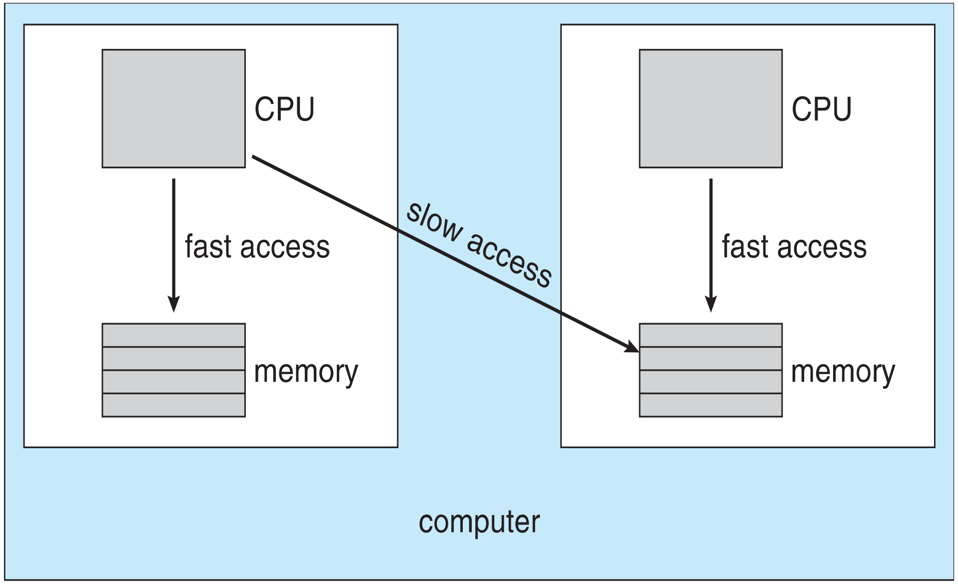


Figure 1.10 NUMA multiprocessing architecture.



Heterogeneous Multiprocessing

- **ARM Big.LITTLE**, HMP (Heterogeneous Multi-Processing)

Cortex A57/A53 MP Core
big.LITTLE CPU chip

- Apple A-core series, Samsung Exynos, Nvidia Tegra 3

Heterogeneous Multiprocessing

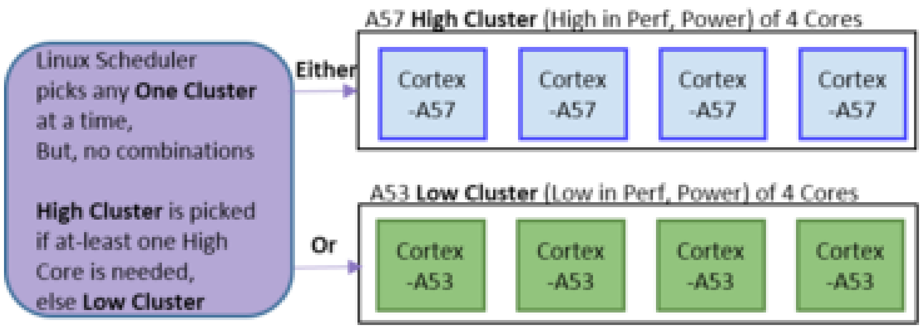
- Idea is very simple!
  - Chip has both high-end (high-power consumption) and low-end (energy efficient) cores
  - For performance – use all cores or high-end cores
  - For energy efficiency – use low-end cores
- But, Scheduling becomes complex (or interesting)
  - Foreground w/ high priority apps – BIG cores
  - Background w/ low priority apps – LITTLE cores

Heterogeneous Multiprocessing

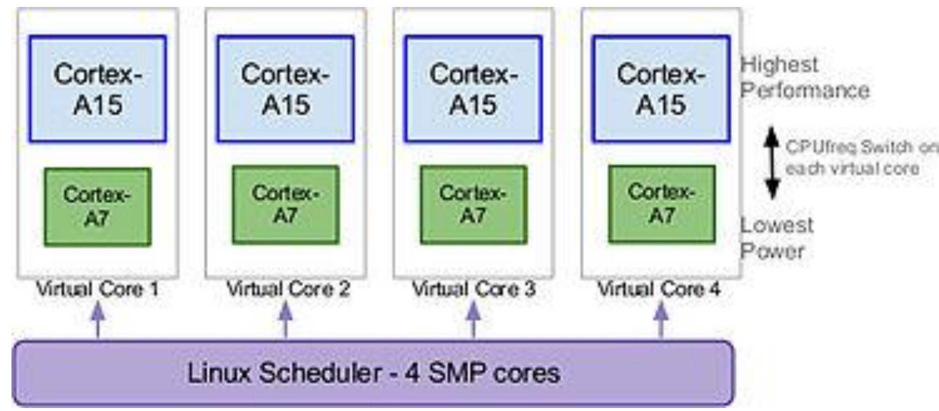
- Three Process/Thread Execution Models
  - Clustered Switching
  - CPU Migration (In-Kernel Switcher)
  - Global Task Scheduling (Heterogeneous Multiprocessing)

Heterogeneous Multiprocessing

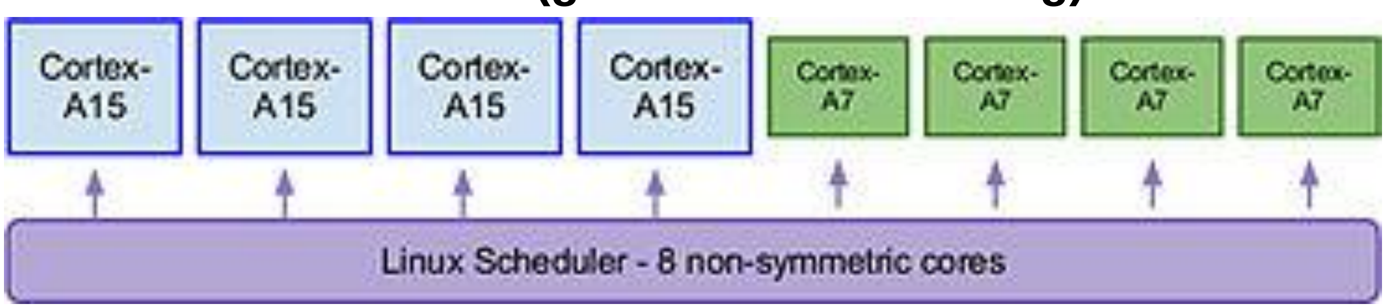
Clustered switching



CPU migration



HMP (global task scheduling)



Real-Time CPU Scheduling

- Rate Monotonic Scheduling
- Earliest Deadline First Scheduling (EDF)

What is Real-Time?

- Does it mean really fast??
- Computation **“with a deadline”**
- Soft real-time vs. Hard real-time

Soft real-time vs Hard real-time

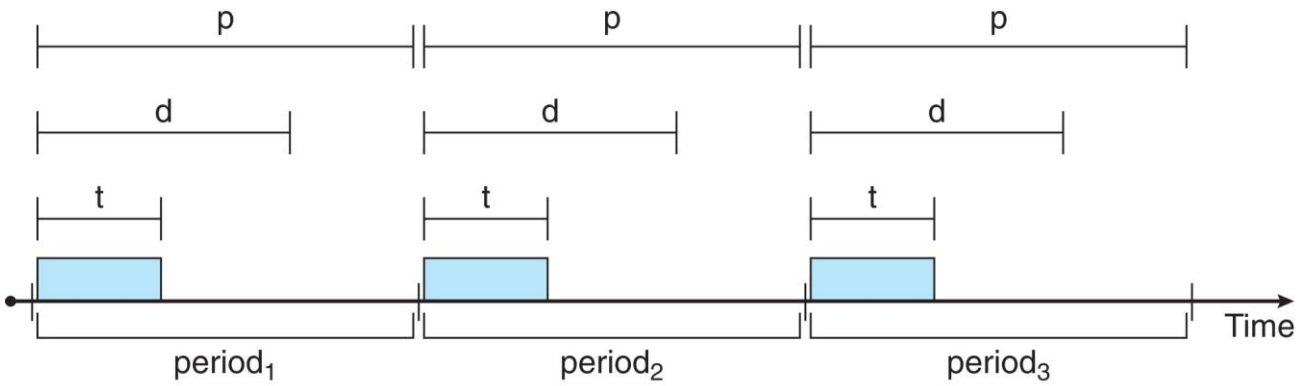
- Hard real-time
  - Missing job deadline can result in system failure
  - e.g., nuclear reactor, medical applications, military applications.
- Soft real-time
  - Can miss some deadlines
  - Missing deadlines can result in the degradation of the system's QoS

Priority-based Scheduling

- "Priority-based" Scheduling must be supported by RTS (Real-Time System)
  - Deadline!
- General "Priority-based" Scheduling
  - More important processes have higher priorities than those deemed less important.
  - Preemption

Priority-based Scheduling

- Characteristics in real-time tasks
  - **Periodic** – processes require CPU at constant intervals (periods)
  - $0 \leq t \leq d \leq p$
    - t: processing time, d: deadline, p: period
  - **Rate** of periodic task is $1/p$



Priority-based Scheduling

- Admission Control
  - Admit or not