CSC 4320/6320: Operating Systems

Chapter 05: CPU Scheduling - II

Spring 2025

Instructor: Dr. Md Mahfuzur Rahman
Department of Computer Science, GSU

Disclaimer

The slides to be used in this course have been created, modified, and adapted from multiple sources:

- *The slides are copyright Silberschatz, Galvin and Gagne, 2018. The slides are authorized for personal use, and for use in conjunction with a course for which Operating System Concepts is the prescribed text. Instructors are free to modify the slides to their taste, as long as the modified slides acknowledge the source and the fact that they have been modified.*

Outline

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- Multi-Processor Scheduling
- Real-Time CPU Scheduling
- Operating Systems Examples
- Algorithm Evaluation

Objectives

- Describe various CPU scheduling algorithms
- Assess CPU scheduling algorithms based on scheduling criteria
- Explain the issues related to multiprocessor and multicore scheduling
- Describe various real-time scheduling algorithms
- ~~Describe the scheduling algorithms used in the Windows, Linux, and Solaris operating systems~~
- ~~Apply modeling and simulations to evaluate CPU scheduling algorithms~~
- ~~Design a program that implements several different CPU scheduling algorithms~~

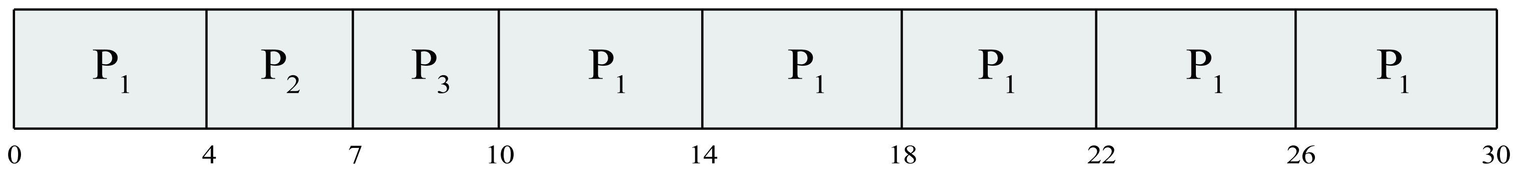
Review: Round Robin (RR)

- Each process gets a small unit of CPU time (**time quantum** *q*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are *n* processes in the ready queue and the time quantum is *q*, then each process gets 1/*n* of the CPU time in chunks of at most *q* time units at once. No process waits more than (*n*-1)*q* time units.
- Timer interrupts every quantum to schedule next process
- Performance
  - *q* large $\Rightarrow$ FIFO (FCFS)
  - *q* small $\Rightarrow$ RR (a large number of context switches.)
- Note that q must be large with respect to context switch, otherwise overhead is too high (as there are many context switches)

Example of RR with Time Quantum = 4

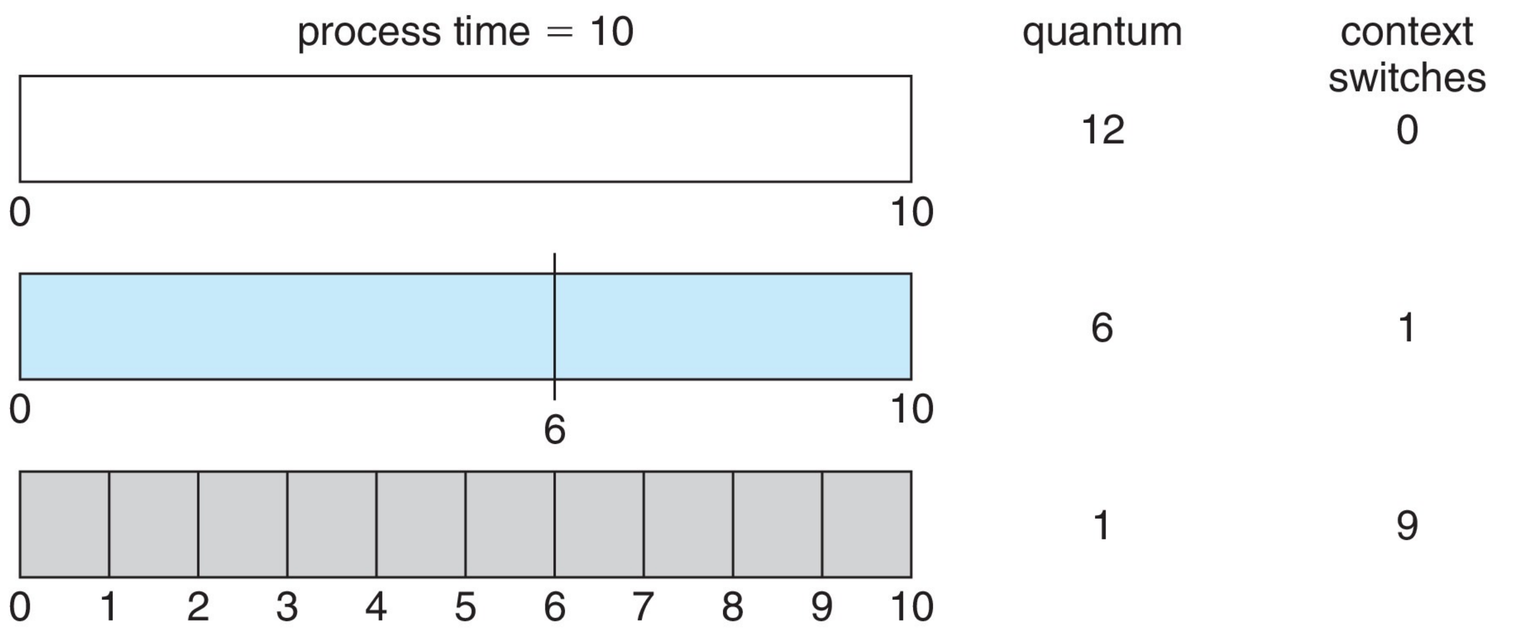
Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:



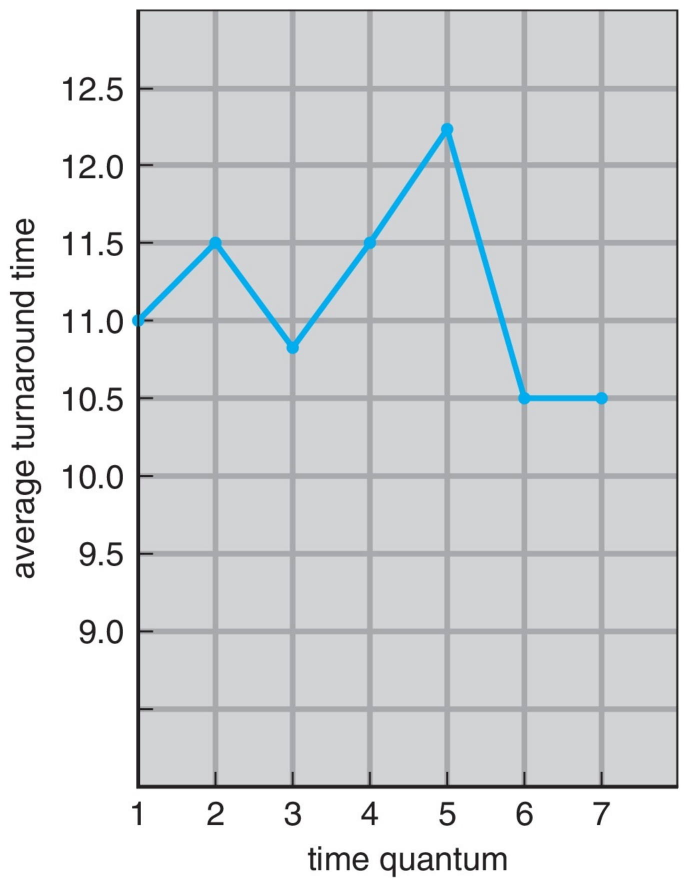
- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
  - q usually 10 milliseconds to 100 milliseconds,
  - Context switch < 10 microseconds

Time Quantum and Context Switch Time



a smaller time quantum increases context switches.

Turnaround Time Varies With The Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

A rule of thumb: 80% of CPU bursts should be shorter than q

- Turnaround time = Completion time - Arrival time (assuming all processes arrive at time 0)

Waiting Time=Turnaround Time−Burst Time

the average turnaround time can be improved if most processes finish their next CPU burst in a single time quantum.

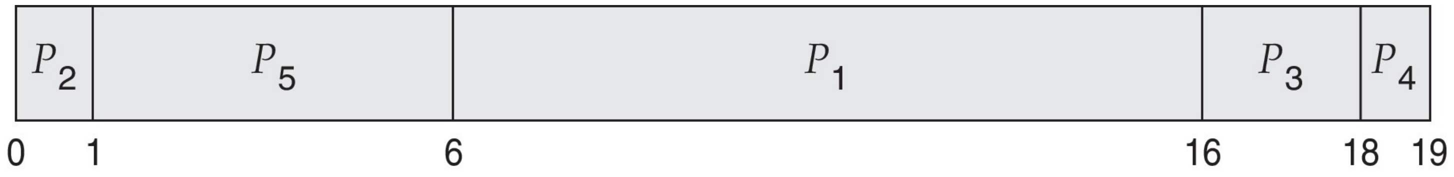
Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - Nonpreemptive
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ≡ **Starvation** – low priority processes may never execute
- Solution ≡ **Aging** – as time progresses increase the priority of the process

Example of Priority Scheduling

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

- Priority scheduling Gantt Chart



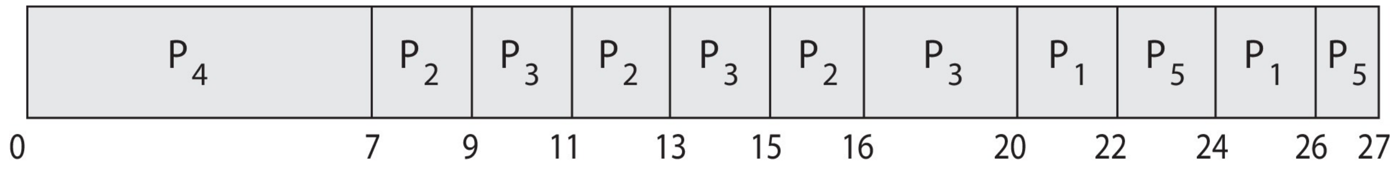
- Average waiting time = 8.2

Priority Scheduling w/ Round-Robin

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

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

- Gantt Chart with time quantum = 2

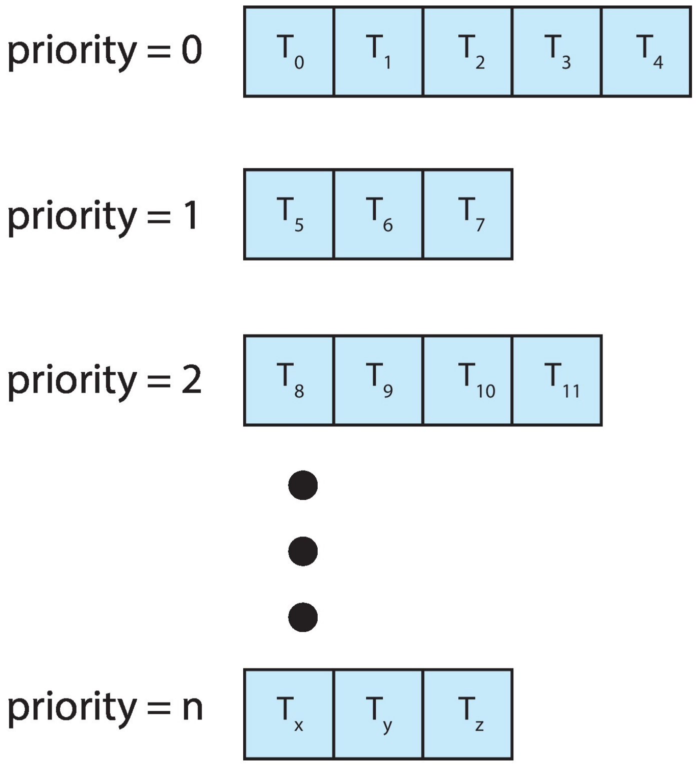


Multilevel Queue

- The ready queue consists of multiple queues
- Multilevel queue scheduler defined by the following parameters:
  - Number of queues
  - Scheduling algorithms for each queue
  - Method used to determine which queue a process will enter when that process needs service
  - Scheduling among the queues

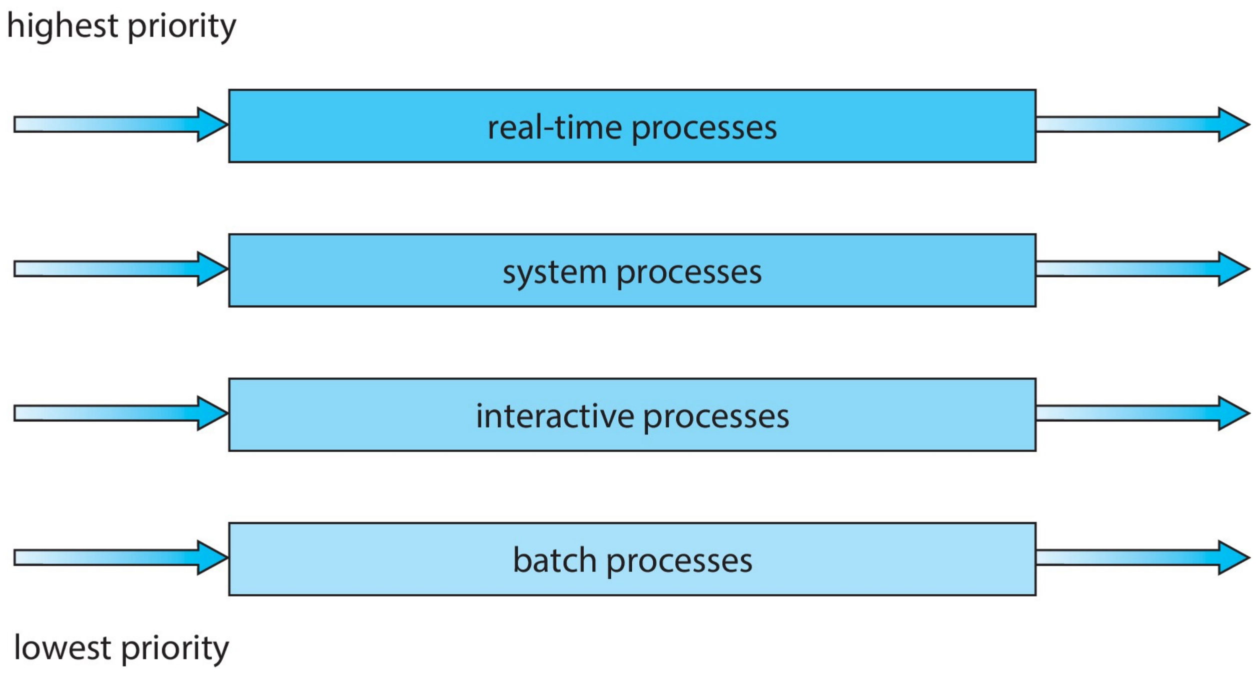
Multilevel Queue

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



Multilevel Queue

- Can be used for prioritization based upon process type



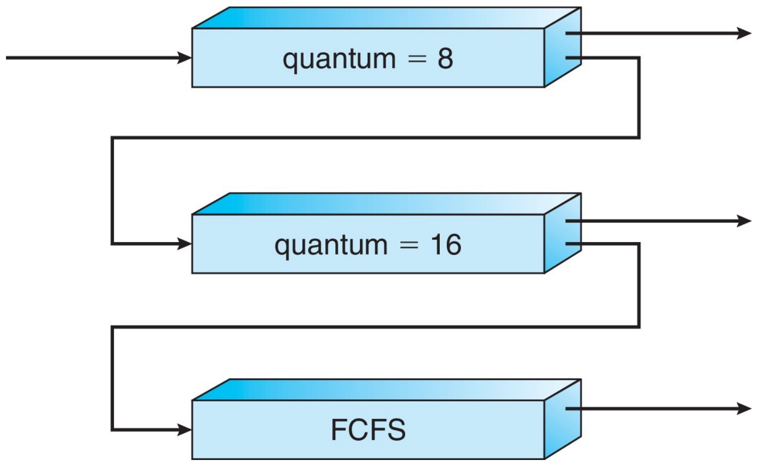
- Each queue has absolute priority over lower-priority queues.
- No process in the batch queue, for example, could run unless the upper queues were all empty.
- known as fixed priority preemptive scheduling.

Multilevel Feedback Queue

- A process can move between the various queues.
- 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
  - Method used to determine when to demote a process
  - Method used to determine which queue a process will enter when that process needs service
- Aging can be implemented using multilevel feedback queue

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 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



Questions

1. ____ scheduling is approximated by predicting the next CPU burst with an exponential average of the measured lengths of previous CPU bursts.

A) Multilevel queue

B) RR

C) FCFS

[✓ D) SJF]

The ____ scheduling algorithm is designed especially for time-sharing systems.

A) SJF

B) FCFS

[✓ C) RR]

D) Multilevel queue

Questions

3. Which of the following is true of multilevel queue scheduling?

A) Processes can move between queues.

B) Each queue has its own scheduling algorithm. ✓

C) A queue cannot have absolute priority over lower-priority queues.

D) It is the most general CPU-scheduling algorithm.

Thread Scheduling

- We learned threads based on the distinction between user-level and kernel-level threads
- On most modern operating systems, it is kernel-level threads—not processes—that are being scheduled by the operating system.
- User-level threads are managed by a thread library, and the kernel is unaware of them.
- Many-to-one and many-to-many models, thread library schedules user-level threads to run on LWP (lightweight process – appears as a virtual processor)
  - Known as **process-contention scope** (**PCS**) since scheduling competition is within the process
  - Typically done via priority set by programmer
- Kernel thread scheduled onto available CPU is **system-contention scope** (**SCS**) – competition among all threads in system

Pthread Scheduling

- API allows specifying either PCS or SCS during thread creation
  - PTHREAD_SCOPE_PROCESS schedules threads using PCS scheduling
  - PTHREAD_SCOPE_SYSTEM schedules threads using SCS scheduling
- Can be limited by OS – Linux and macOS only allow PTHREAD_SCOPE_SYSTEM

Question

1. _____ involves the decision of which kernel thread to schedule onto which CPU.

A) Process-contention scope

B) System-contention scope ✓

C) Dispatcher

D) Round-robin scheduling

Multiple-Processor Scheduling

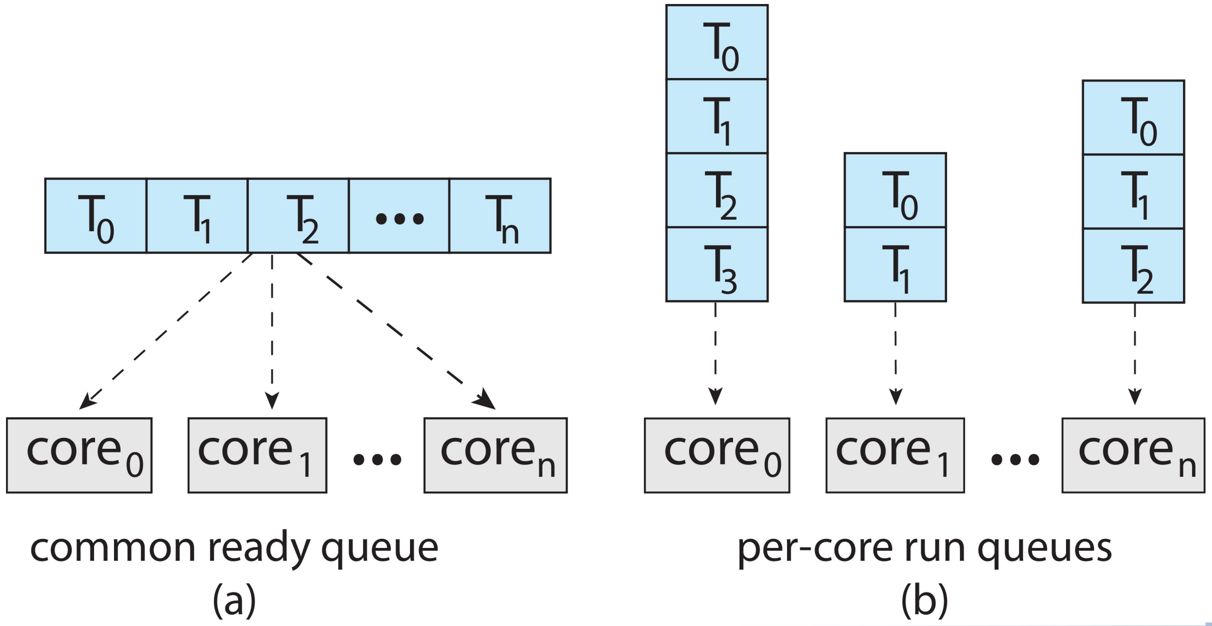
- CPU scheduling more complex when multiple CPUs are available
- Multiprocess may be any one of the following architectures:
  - Multicore CPUs
  - Multithreaded cores
  - NUMA systems
  - Heterogeneous multiprocessing

In the first three examples, the processors are identical—homogeneous—in terms of their functionality.

Multiple-Processor Scheduling

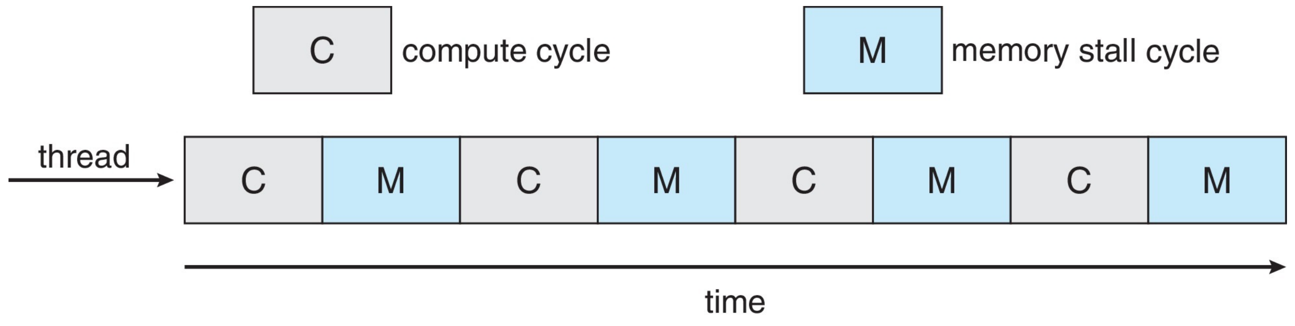
- Symmetric multiprocessing (SMP) is where each processor is self scheduling.
- All threads may be in a common ready queue (a)
- Each processor may have its own private queue of threads (b)

Assume each processor has one single-core



Multicore Processors

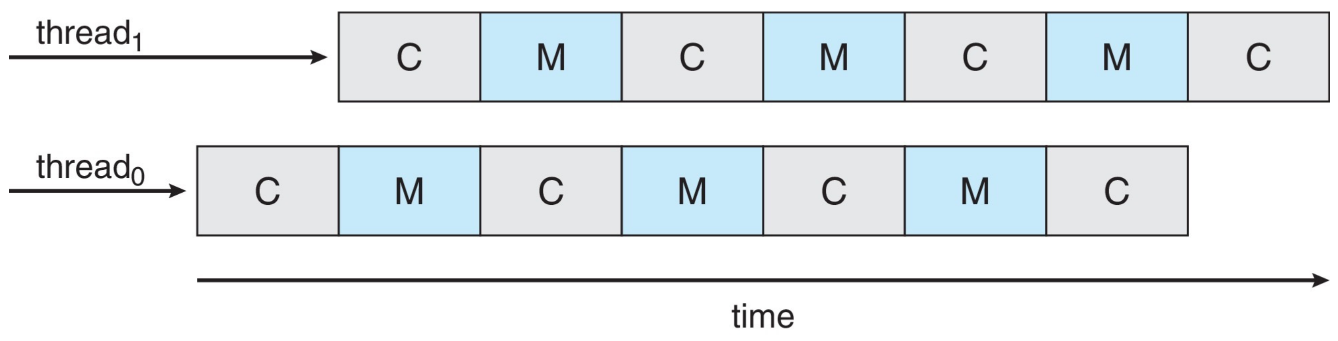
- Recent trend to place multiple processor cores on same physical chip (multicore processor)
- Faster and consumes less power
- Multiple threads per core also growing
  - Takes advantage of memory stall to make progress on another thread while memory retrieve happens
- Figure



Memory Stall: When a processor accesses memory, it spends a significant amount of time waiting for the data to become available, because of speed gap, cache miss, etc.

Multithreaded Multicore System

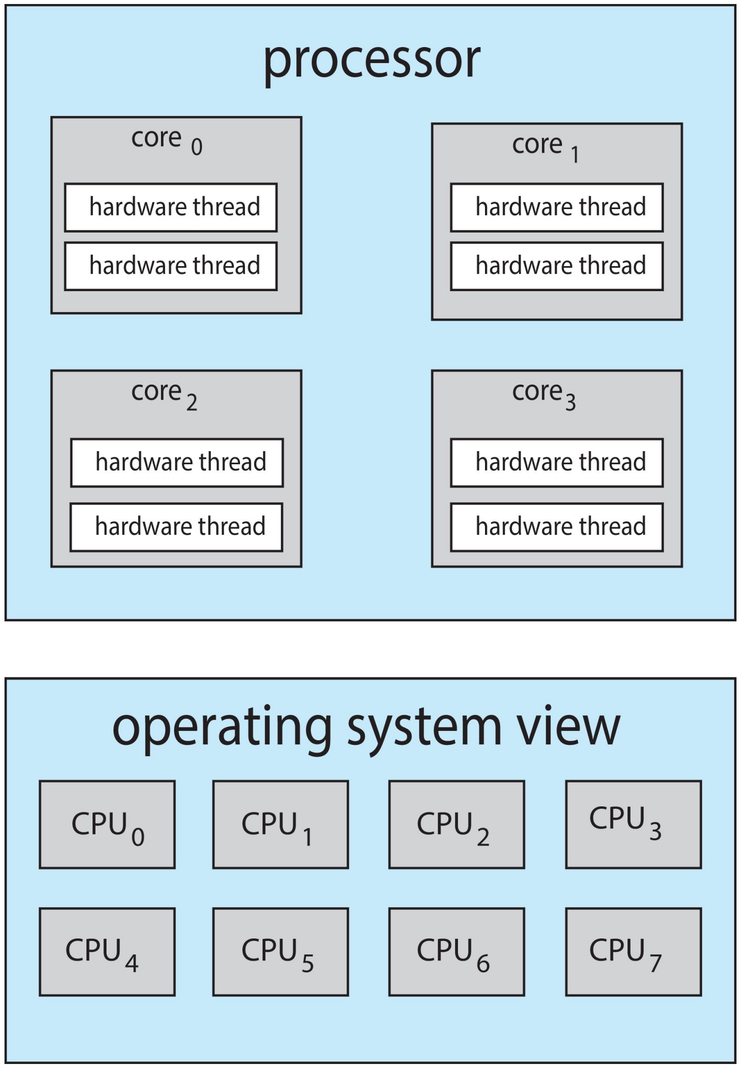
- Each core has > 1 hardware threads.
- If one thread has a memory stall, switch to another thread!
- Figure



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.

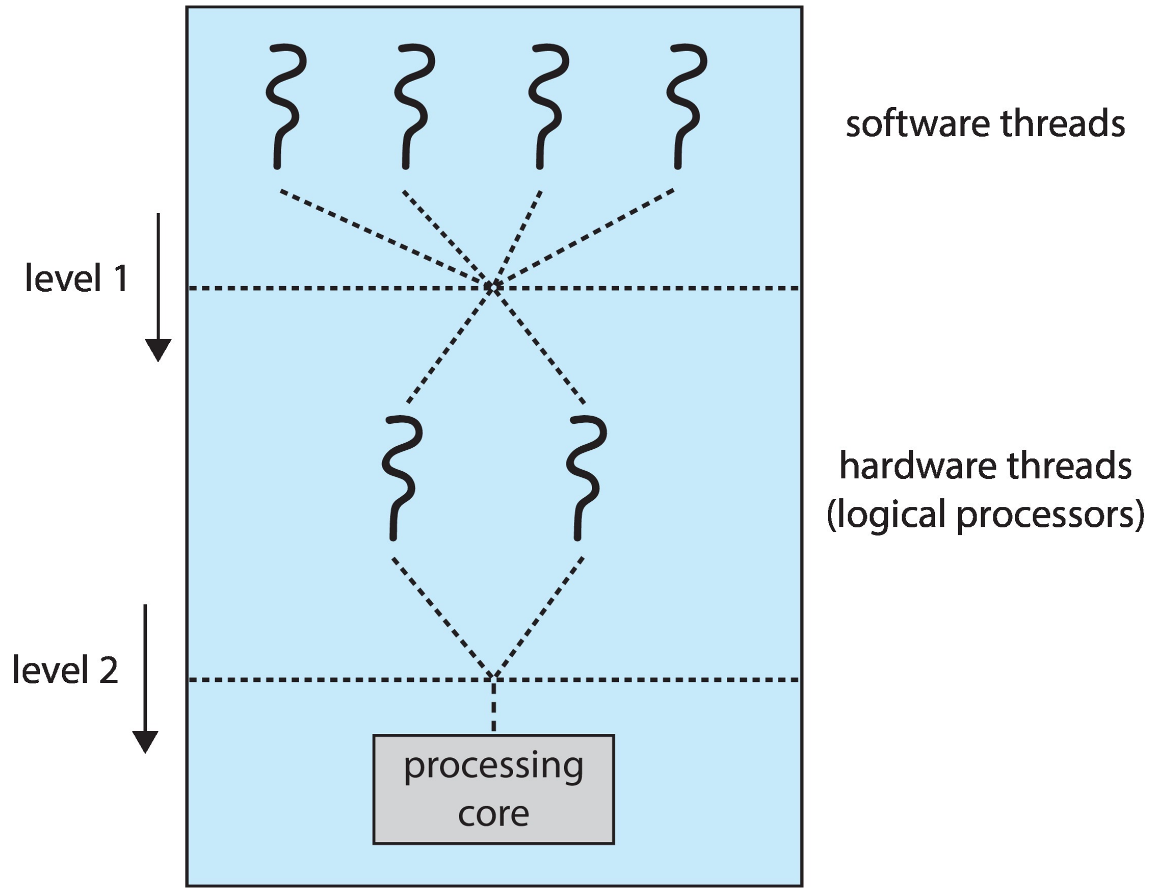


Multithreaded Multicore System

- 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.



Multiple-Processor Scheduling – Load Balancing

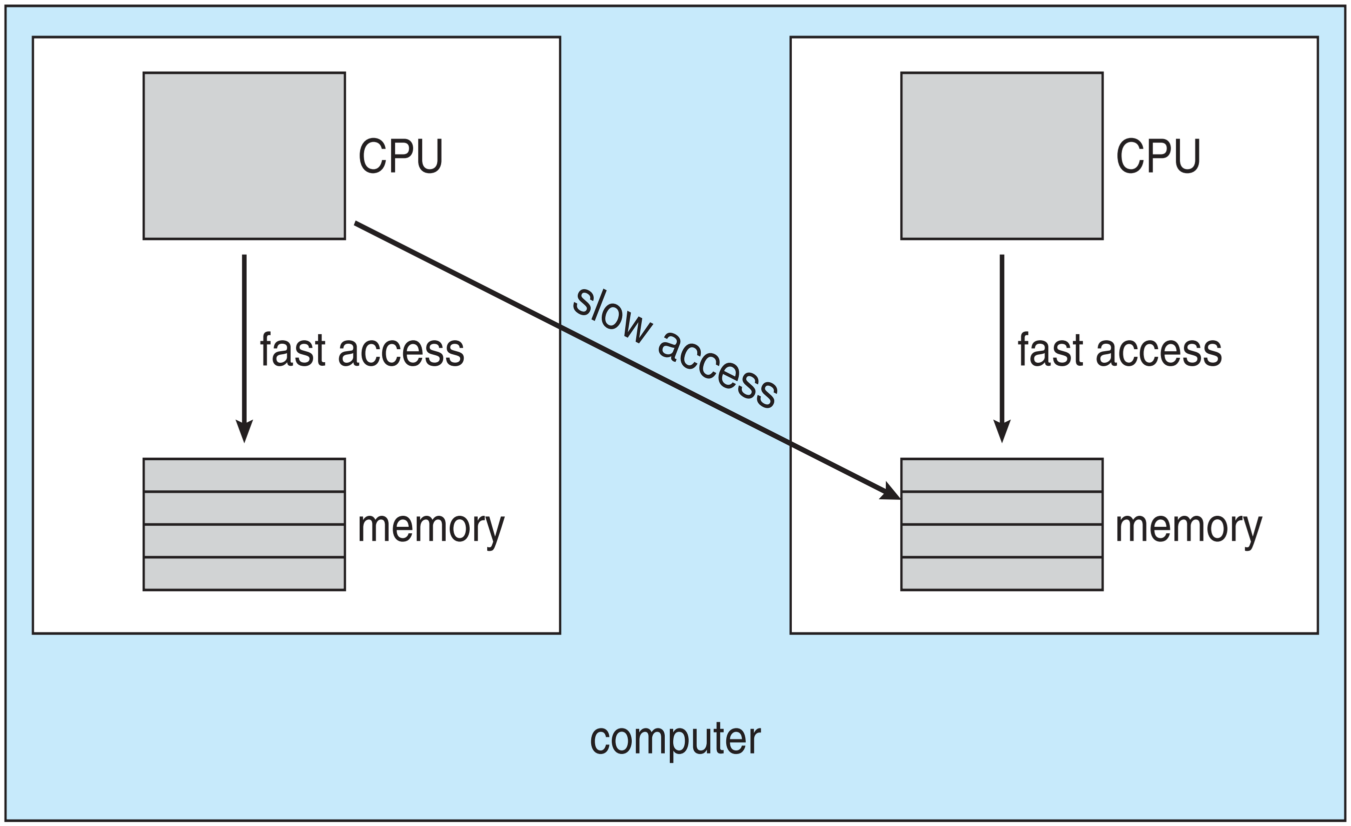
- If SMP, need to keep all CPUs loaded for efficiency
- **Load balancing** attempts to keep workload evenly distributed
- **Push migration** –a specific task periodically checks the load on each processor and—if it finds an imbalance---pushes task from overloaded CPU to other CPUs
- **Pull migration** – idle processors pulls waiting task from busy processor
- **Push and pull migration** need not be mutually exclusive and are, in fact, often implemented in parallel on load-balancing systems

Multiple-Processor Scheduling – Processor Affinity

- When a thread has been running on one processor, the cache contents of that processor stores the memory accesses by that thread.
- We refer to this as a thread having affinity for a processor (i.e., “processor affinity”)
- Load balancing may affect processor affinity as a thread may be moved from one processor to another to balance loads, yet that thread loses the contents of what it had in the cache of the processor it was moved off of.
- **Soft affinity** – the operating system 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 (via system call).

NUMA and CPU Scheduling

If the operating system is **NUMA-aware**, it will assign memory closes to the CPU the thread is running on.



Questions

1. _____ allows a thread to run on only one processor.

A) Processor affinity

✓

B) Processor set

C) NUMA

D) Load balancing

2. The two general approaches to load balancing are _____ and _____.

A) soft affinity, hard affinity

B) coarse grained, fine grained

C) soft real-time, hard real-time

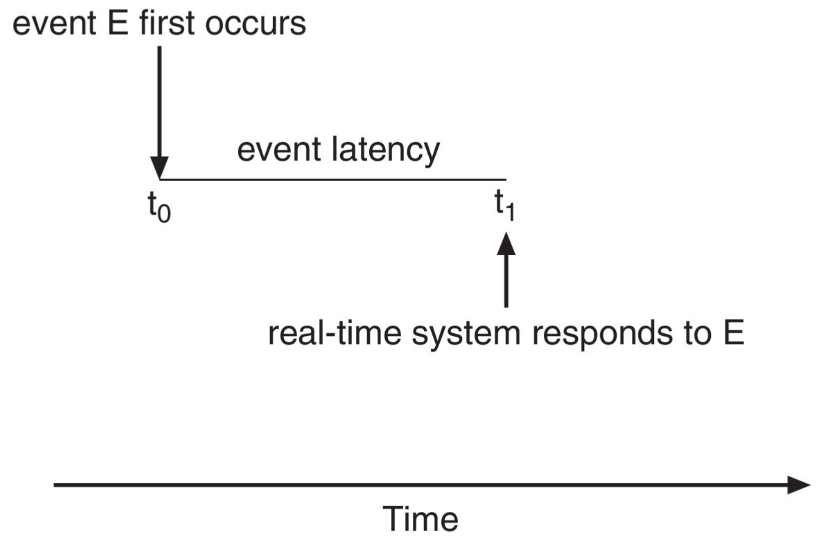
[✓ D) push migration, pull migration]

Real-Time CPU Scheduling

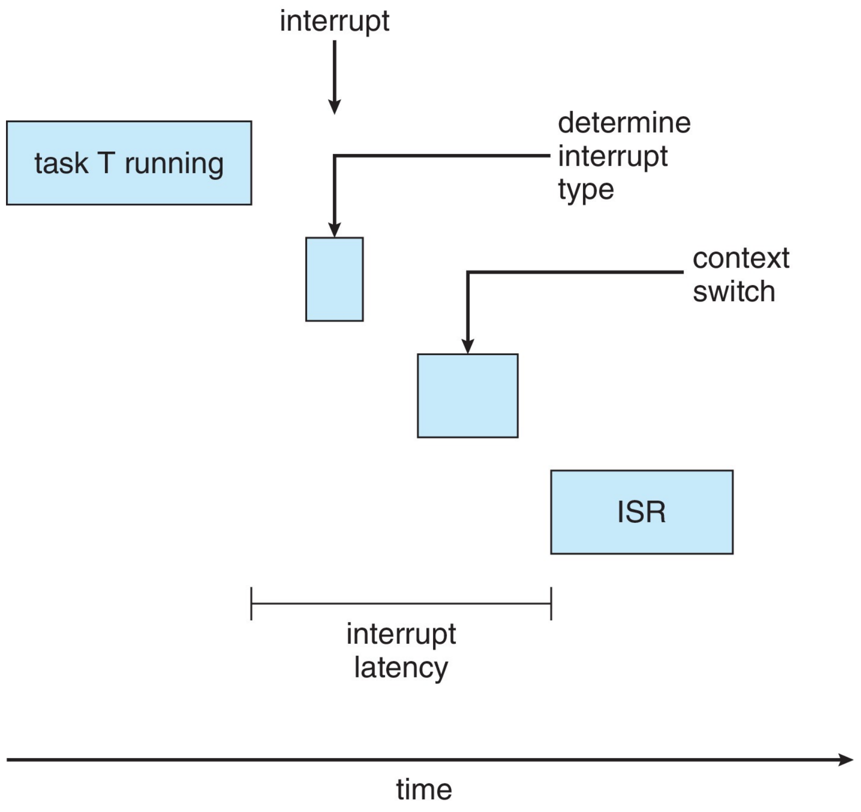
- Can present obvious challenges
- Real-time systems are common in embedded systems, telecommunications, industrial automation, and robotics.
- **Soft real-time systems** – Critical real-time tasks have the highest priority, but no guarantee as to when tasks will be scheduled
- **Hard real-time systems** – task must be serviced by its deadline

Real-Time CPU Scheduling

- Event latency – the amount of time that elapses from when an event occurs to when it is serviced.
- Two types of latencies affect performance
  1. **Interrupt latency** – time from arrival of interrupt to start of routine that services interrupt
  2. **Dispatch latency** – time for scheduler dispatcher to take current process off CPU and switch to another

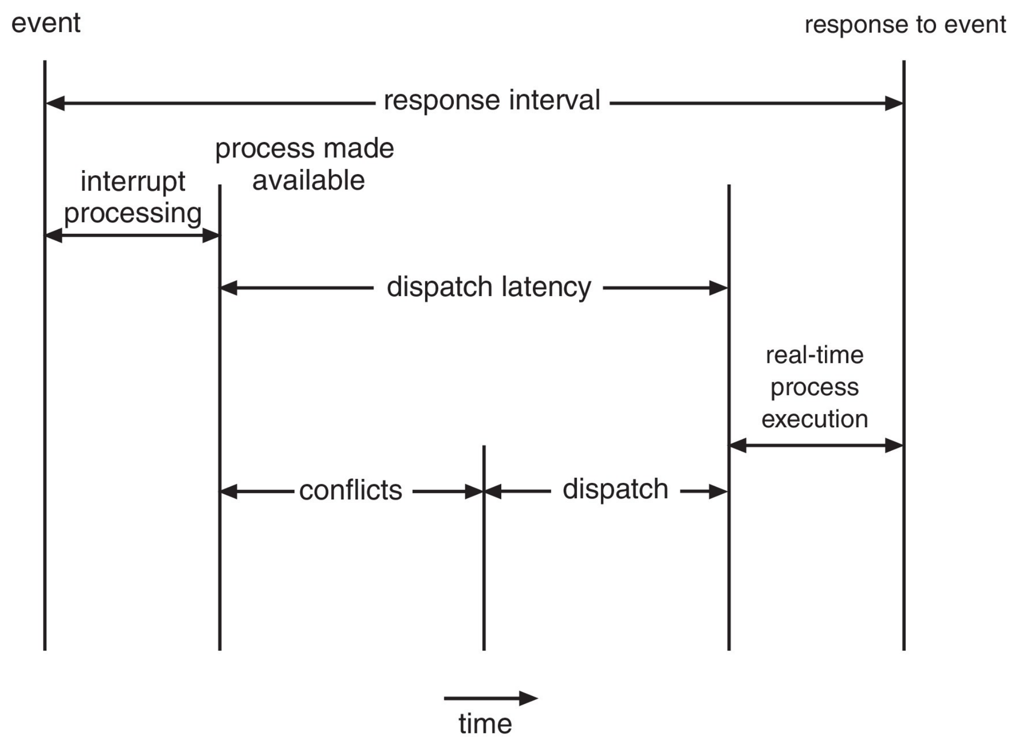


Interrupt Latency



Dispatch Latency

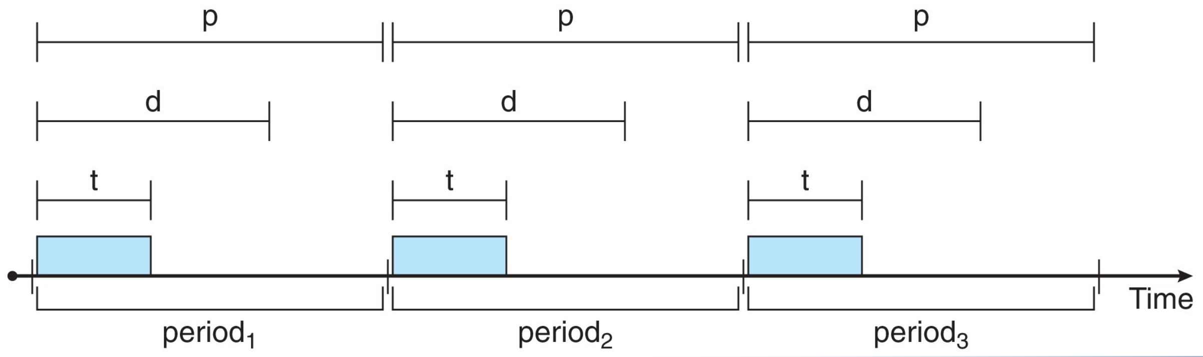
- Conflict phase of dispatch latency:
  1. Preemption of any process running in kernel mode
  2. Release by low-priority process of resources needed by high-priority processes



Note: "process made available" refers to the point when the operating system finishes processing the interrupt, and the real-time process (or task) that should respond to the event is ready to be executed.

Priority-based Scheduling

- For real-time scheduling, scheduler must support preemptive, priority-based scheduling
  - But only guarantees soft real-time
- For hard real-time must also provide ability to meet deadlines
- 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*


p
p
p
d
d
d
t
t
t
Time
period_1
period_2
period_3