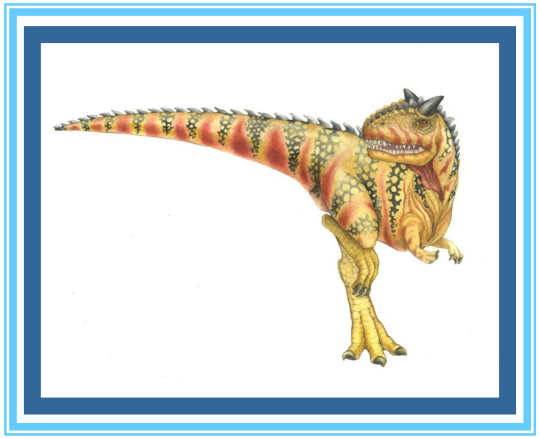
Chapter 5: CPU Scheduling
Part 2



Outline

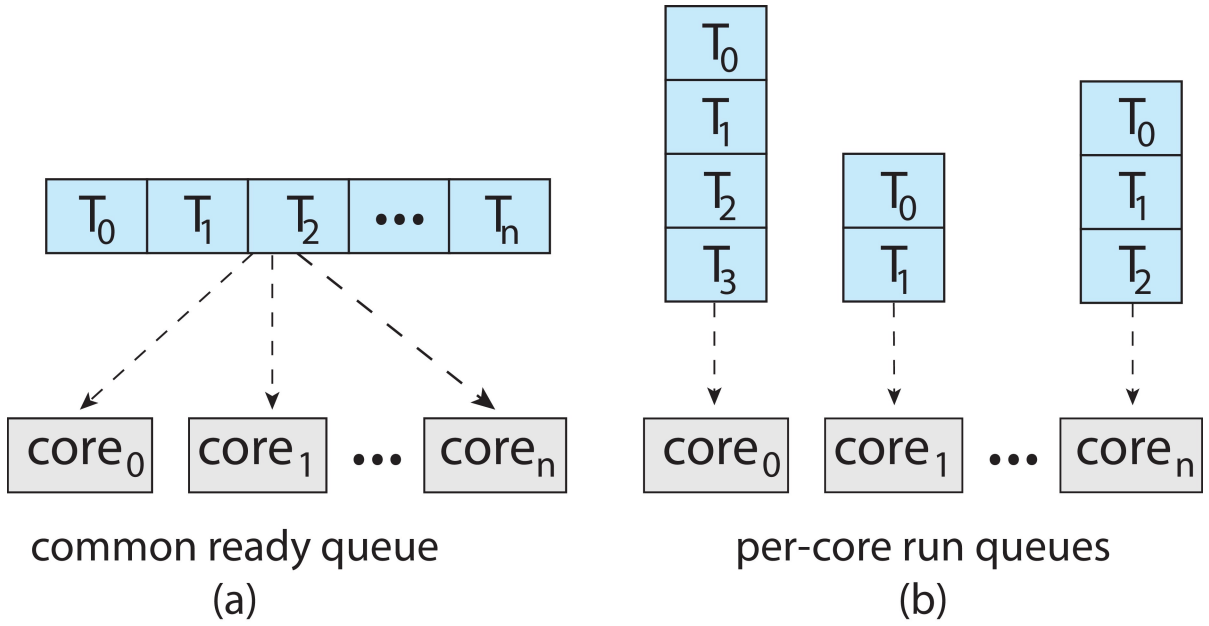
- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- **Multi-Processor Scheduling**
- **Real-Time CPU Scheduling**
- **Operating Systems Example**

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

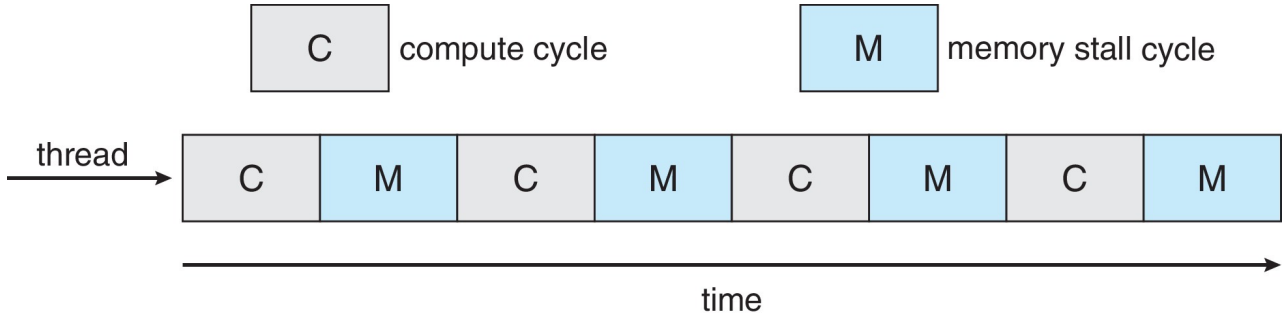
Multiple-Processor Scheduling

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



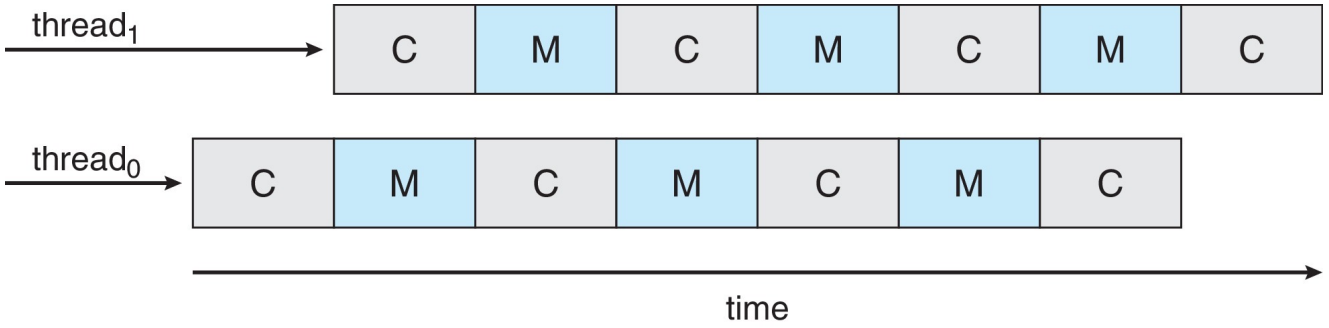
Multicore Processors

- Recent trend to place multiple processor cores on same physical chip
- SMP system is faster and consumes less power in this architecture
- Multiple threads per core also growing
  - Takes advantage of **memory stall** to make progress on another thread while memory retrieve happens
- Figure



Multithreaded Multicore System

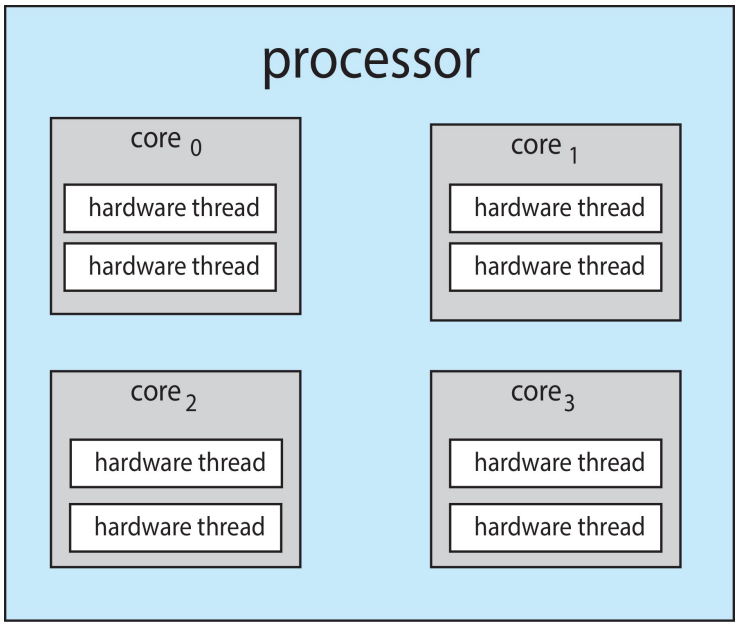
- Each core has > 1 hardware threads.
- If one thread has a memory stall, switch to another thread!
- Figure

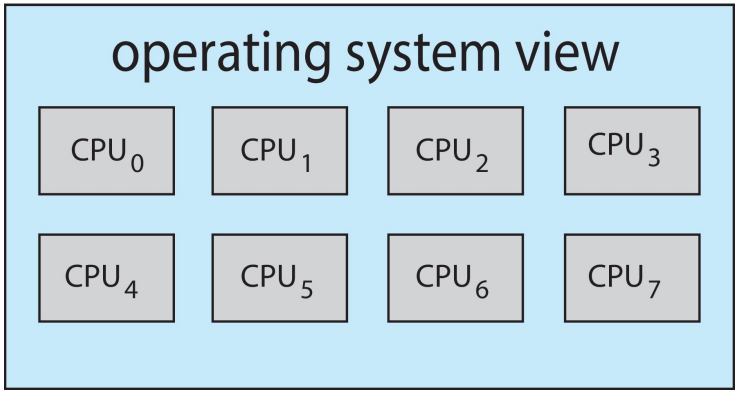


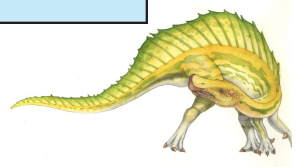
Multithreaded Multicore System

- **Chip-multithreading** (CMT) assigns each core multiple hardware threads. (Intel refers to this as **hyperthreading,** the i7—support two threads per core)

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

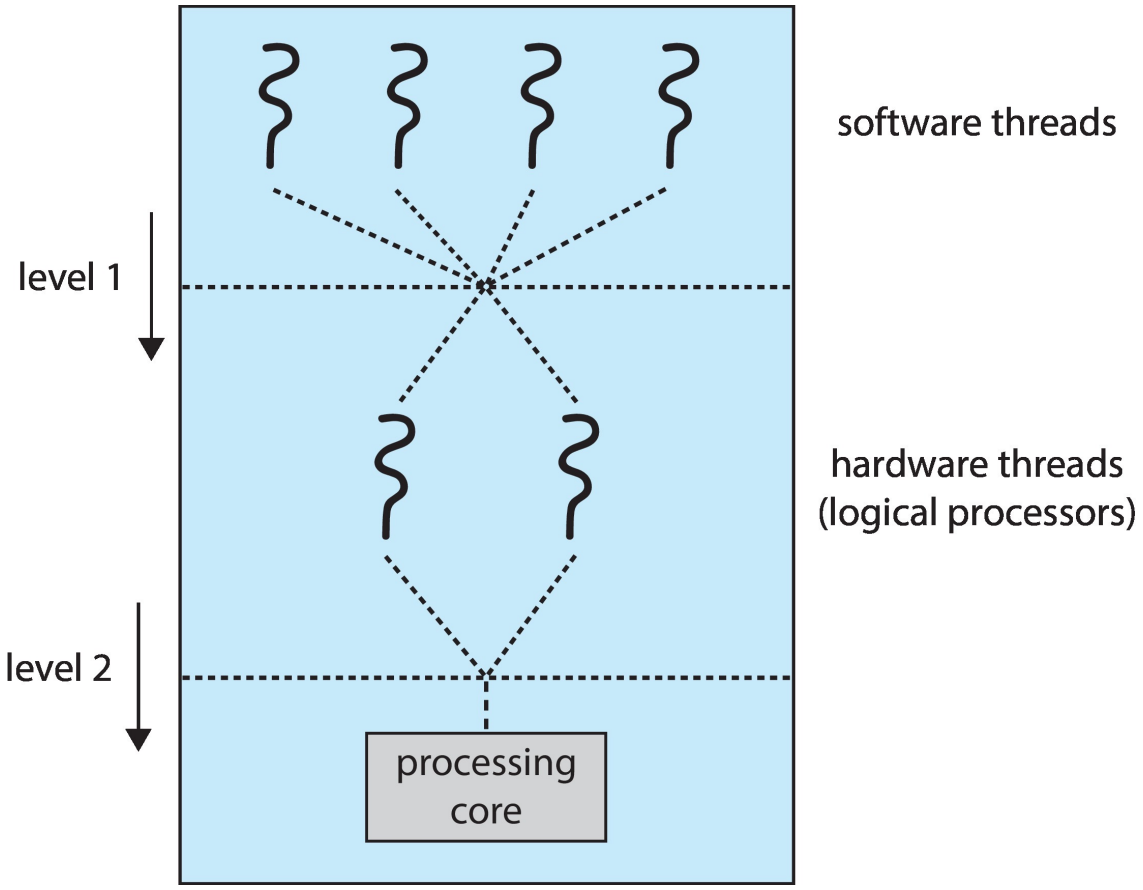






Multithreaded Multicore System

- 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

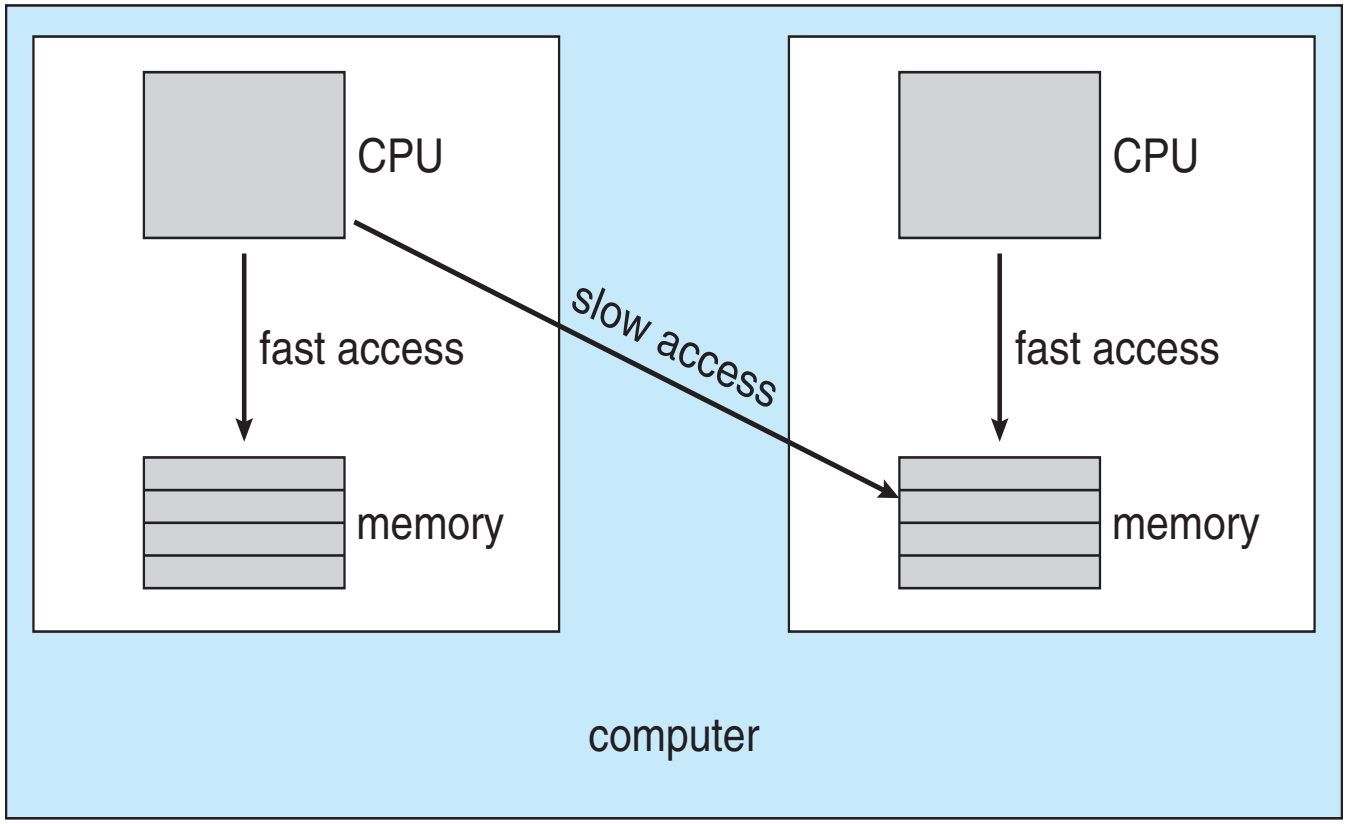
- If SMP, need to keep all CPUs loaded for efficiency
- **Load balancing** attempts to keep workload evenly distributed
- Approaches to load balancing:
  - **Push migration** – **periodic task** checks load on each processor, and if found pushes task from overloaded CPU to other CPUs
  - **Pull migration** – **idle processors** pulls waiting task from busy processor

Multiple-Processor Scheduling – Processor Affinity

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

NUMA and CPU Scheduling

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



Test your understanding

1. How many threads may run in parallel on a single processing core with two hardware threads?
   a) 1
   b) 2
   c) 4

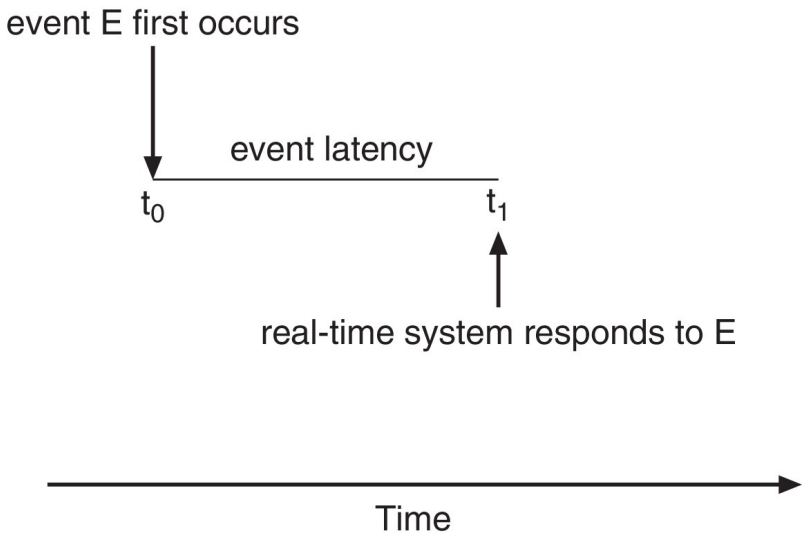
2. ____ means that a process has an affinity for the processor on which it is currently running.

Real-Time CPU Scheduling

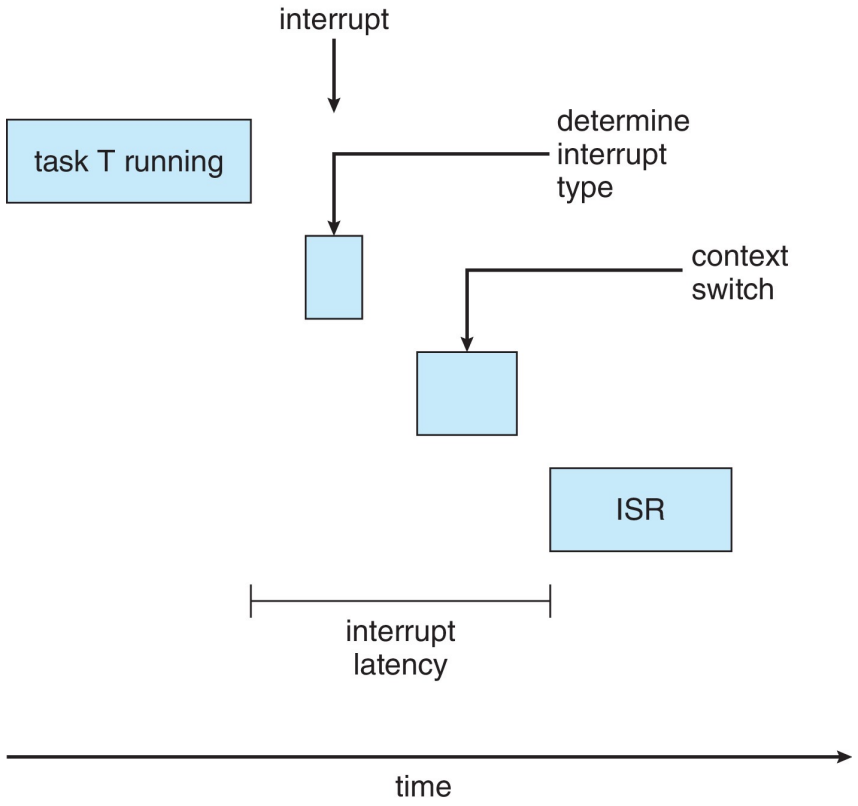
- Can present obvious challenges
- **Soft real-time systems** – Critical real-time tasks have the highest priority, but no guarantee as to when tasks will be scheduled
- **Hard real-time systems – task must be serviced by its deadline**

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 :** period of time from the arrival of an interrupt at the CPU to the start of the routine that services the interrupt.
  2. **Dispatch latency** – time for schedule to take current process off CPU and switch to another

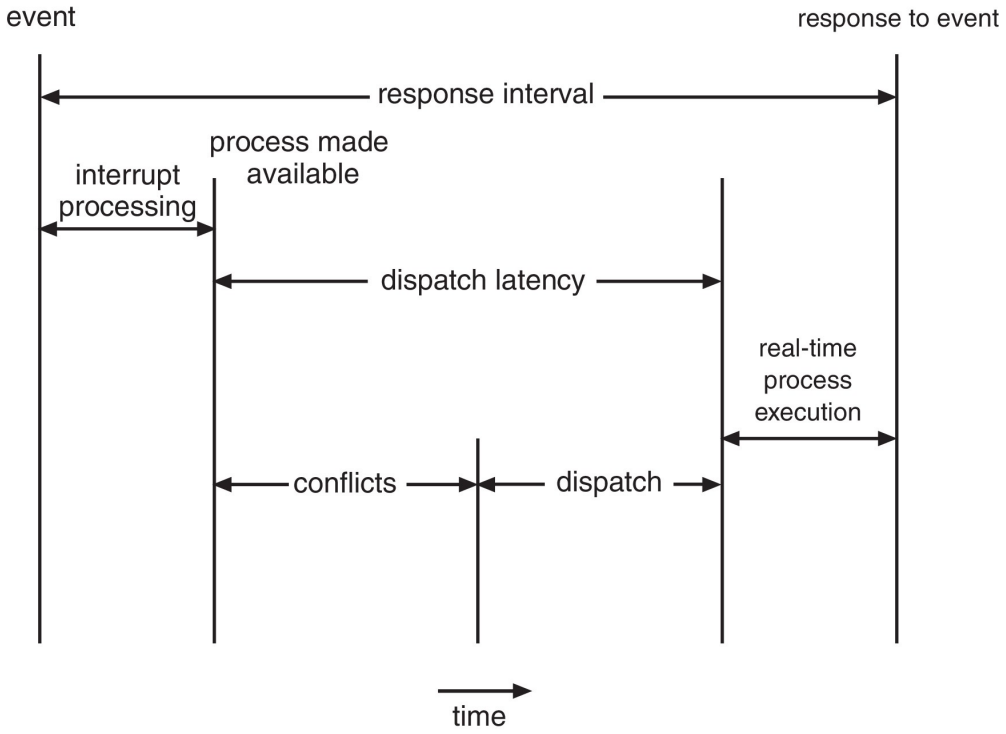


Interrupt Latency



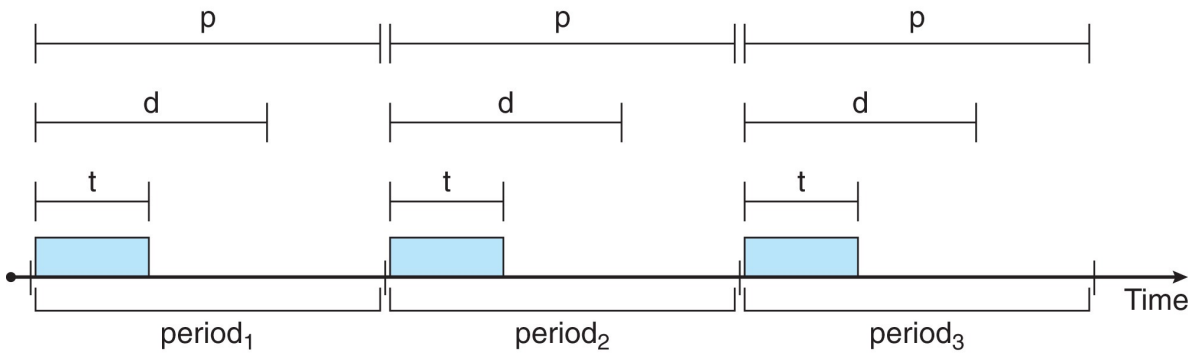
Dispatch Latency

- 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



Priority-based Scheduling

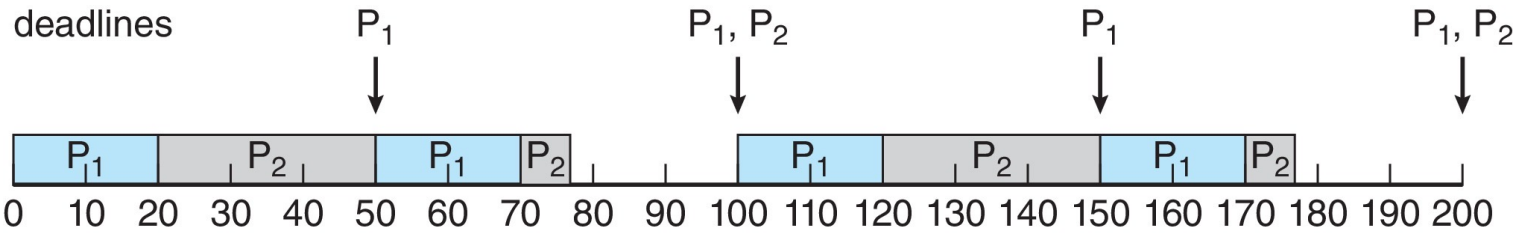
- For real-time scheduling, scheduler must support preemptive, priority-based scheduling
  - But only guarantees soft real-time
- For hard real-time must also provide ability to meet deadlines
- Processes have new characteristics: **periodic** ones require CPU at constant intervals
  - Has processing time *t*, deadline *d*, period *p*
  - $0 \leq t \leq d \leq p$
  - **Rate** of periodic task is $1/p$



Rate Monotonic Scheduling

- The ***rate-monotonic*** scheduling algorithm schedules periodic tasks using a static priority policy with preemption.
- A priority is assigned based on the inverse of its period
- Shorter periods = higher priority;
- Longer periods = lower priority
- P_1 is assigned a higher priority than P_2.

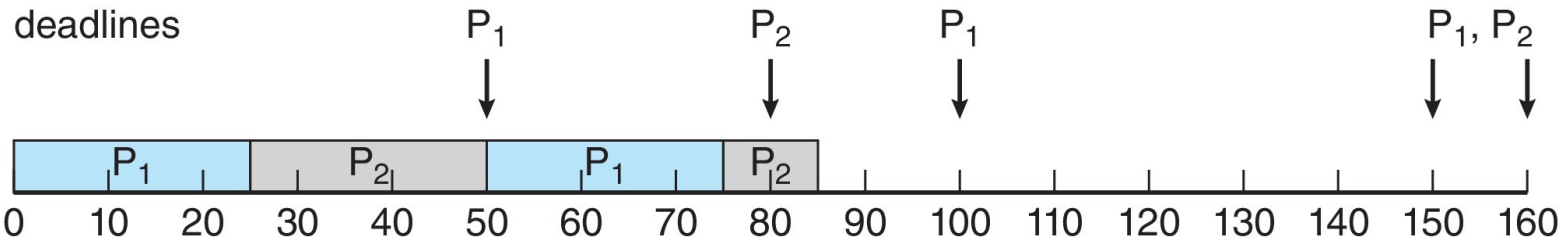
Period p1 = 50, p2 = 100
Burst/processing time t1 = 20, t2 = 35
the deadline for each process requires that it complete its CPU burst by the start of its next period



Missed Deadlines with Rate Monotonic Scheduling

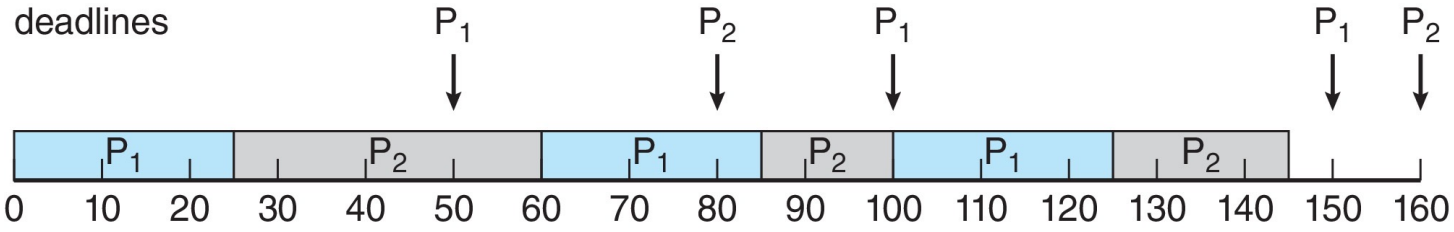
Assume:
p1= 50 and t1 = 25
p2= 80 and t2= 35
the deadline for each process requires that it complete its CPU burst by the start of its next period

- Process P_2 misses finishing its deadline at time 80
- Figure



Earliest Deadline First Scheduling (EDF)

- Priorities are assigned according to deadlines:
  - The earlier the deadline, the higher the priority
  - The later the deadline, the lower the priority
- Under the EDF policy, when a process becomes runnable, it must announce its deadline requirements to the system. Priorities may have to be adjusted to reflect the deadline of the newly runnable process
- Figure



Assume:
p1= 50 and t1 = 25
p2= 80 and t2= 35
the deadline for each process requires that it complete its CPU burst by the start of its next period

Windows Scheduling

- Windows uses priority-based preemptive scheduling
- Highest-priority thread runs next
- **Dispatcher** is scheduler
- Thread runs until (1) blocks, (2) uses time slice, (3) preempted by higher-priority thread
- Real-time threads can preempt non-real-time
- 32-level priority scheme
- **Variable class** is 1-15, **real-time class** is 16-31
- Priority 0 is memory-management thread
- Queue for each priority
- If no run-able thread, runs **idle thread**

Windows Priority Classes

- Win32 API identifies several priority classes to which a process can belong
  - REALTIME_PRIORITY_CLASS, HIGH_PRIORITY_CLASS, ABOVE_NORMAL_PRIORITY_CLASS,NORMAL_PRIORITY_CLASS, BELOW_NORMAL_PRIORITY_CLASS, IDLE_PRIORITY_CLASS
  - All are variable except REALTIME
- A thread within a given priority class has a relative priority
  - TIME_CRITICAL, HIGHEST, ABOVE_NORMAL, NORMAL, BELOW_NORMAL, LOWEST, IDLE
- Priority class and relative priority combine to give numeric priority
- Base priority is NORMAL within the class
- If quantum expires, priority lowered, but never below base

Windows Priorities

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

Windows Priority Classes (Cont.)

- If wait occurs, priority boosted depending on what was waited for
- Foreground window given 3x priority boost
- Windows 7 added **user-mode scheduling** (**UMS**)
  - Applications create and manage threads independent of kernel
  - For large number of threads, much more efficient
  - UMS schedulers come from programming language libraries like C++ **Concurrent Runtime** (ConcRT) framework

End of Chapter 5