EE382V: System-on-a-Chip (SoC) Design

Lecture 9 – Real-Time Scheduling

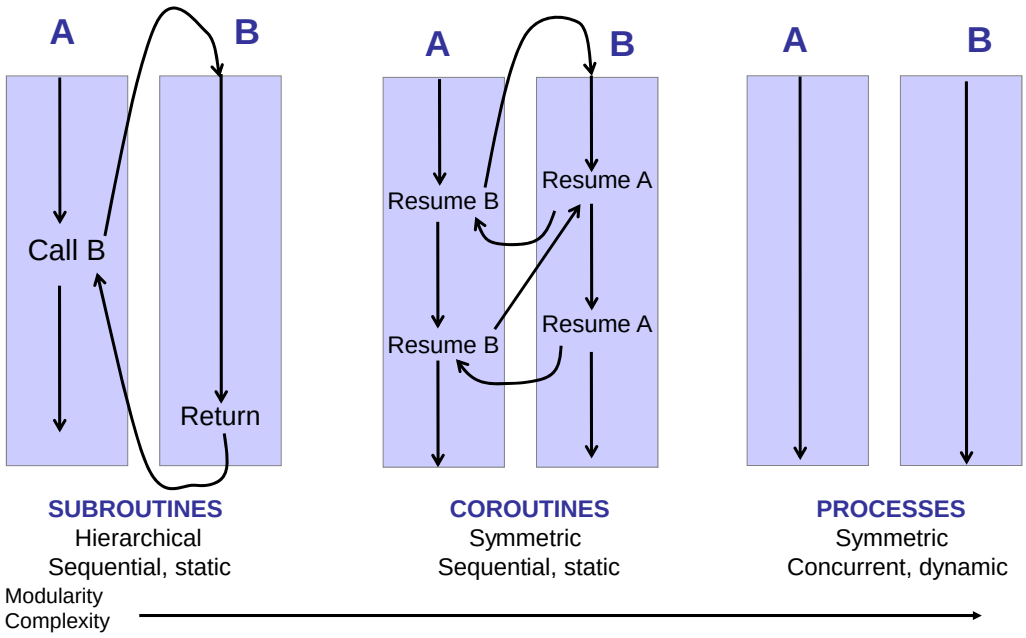
With sources from:
Prof. Margarida Jacome, UT Austin

Andreas Gerstlauer
Electrical and Computer Engineering
University of Texas at Austin
gerstl@ece.utexas.edu

Lecture 9: Outline

- **Rate-Monotonic Scheduling (RMS)**
  - Rate monotonic analysis
  - Scheduling algorithm
- **Earliest-Deadline First (EDF) scheduling**
  - EDF algorithm and analysis
- **Special cases**
  - Priority inversion
  - Context-switch times
  - Interrupts

Multiplexing Software Modules



Real-Time Scheduling

- **Task types**
  - Periodic
    - Set of tasks $\{ T_1, T_2, \ldots \}$
    - Period t_i
    - (Worst-case) execution time e_i
  - Aperiodic/sporadic
    - Arrival/release time a_i
- **Task dependencies**
  - Aperiodic tasks with precedence constraints
    - Dependencies, task graph
- **Preemptive vs. non-preemptive**
  - Task with higher priority can preempt lower priority one
- **Mono- and multi-processor scheduling**
  - Centralized RTOS, symmetric multi-processing (SMP)
  - Distributed RTOS, asymmetric multi-processing (AMP)

Periodic Task Scheduling

- **Scheduling Policies**
  - RMS – Rate Monotonic Scheduling
    - Task Priority = Rate = 1/Period
    - RMS is the optimal preemptive *fixed-priority* scheduling policy
  - EDF – Earliest Deadline First
    - Task Priority = Current Absolute Deadline
    - EDF is the optimal preemptive *dynamic-priority* scheduling policy
- **Scheduling assumptions**
  - Single processor
  - All tasks are periodic
  - Zero context-switch time
  - Worst-case task execution times are known
  - No data dependencies among tasks

➢ **RMS and EDF have both been extended to relax these**

Metrics

- **How do we evaluate a periodic scheduling policy**
  - Ability to satisfy all deadlines
  - CPU utilization
    - Percentage of time devoted to useful work
  - Scheduling overhead
    - Time required to make scheduling decision
- **Constraints**
  - Set of tasks *T* with period τ_i each
    - Response time r_i = finish time f_i – arrival time a_i
    - Deadline d_i: $r_i < d_i$, in periodic case often $d_i = \tau_i$
  - Minimize latency
    - Lateness $l_i = r_i - d_i$

Rate Monotonic Scheduling (RMS)

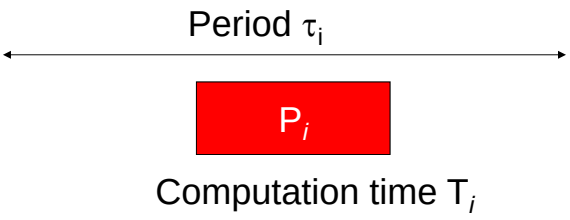
- **Model**
  - All process run on single CPU.
  - Zero context switch time.
  - No data dependencies between processes.
  - Process execution time is constant.
  - Deadline is at end of period.
  - Highest-priority ready process runs.

➢ **RMS [Liu and Layland, 73]**
  - Widely-used, analyzable scheduling policy.

➢ **Rate Monotonic Analysis (RMA)**
  - Theoretical analysis

Process Parameters

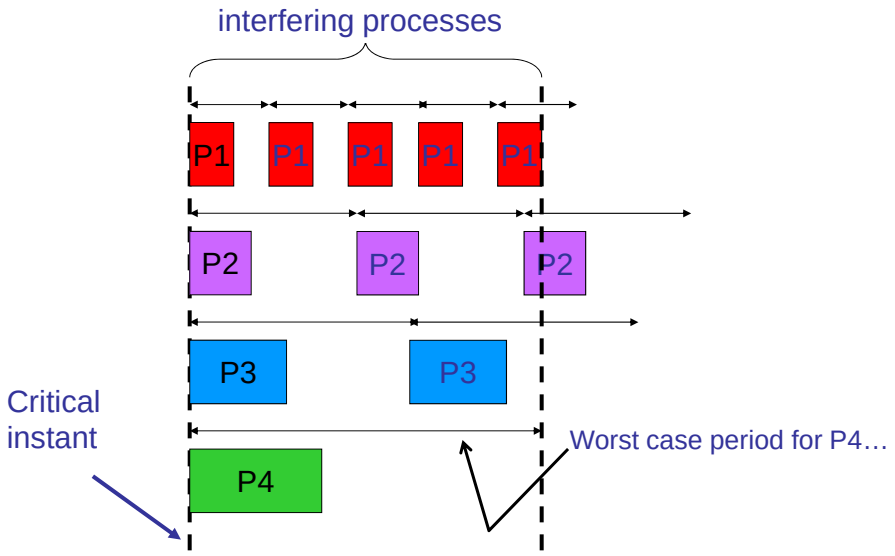
- **T_i is execution time of process *i***
- **Deadline τ_i is period of process *I***



➢ **Response time**
  - Time required to finish a process/task.

➢ **Critical instant**
  - Scheduling state that gives worst response time.
    - Occurs when all higher-priority processes are ready to execute.

Critical Instant



RMS Priorities

- **Optimal (fixed) priority assignment**
  - Shortest-period process gets highest priority
    - priority based preemption can be used…
  - Priority inversely proportional to period
  - Break ties arbitrarily

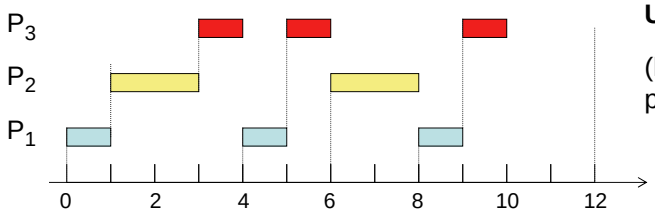
➢ **No fixed-priority scheme does better.**

➢ *RMS provides the highest worst case CPU utilization while ensuring that all processes meet their deadlines*

RMS Example 1

Process P_i	Execution Time T_i	Period t_i
P_1	1	**4**
P_2	2	**6**
P_3	3	**12**

Static priority: P1 >> P2 >> P3



Unrolled schedule

(least common multiple of process periods)

RMS CPU Utilization

- **Utilization for *n* processes is**

  $\sum_i T_i / \tau_i$

- **Schedulability analysis**

  $\sum_i T_i / \tau_i \leq n(2^{1/n} - 1)$

- **As number of tasks approaches infinity, the *worst case* maximum utilization approaches 69%**
  - Yet, is not uncommon to find total utilizations around .90 or more (.69 is worst case behavior of algorithm)
  - Achievable utilization is strongly dependent upon the relative values of the periods of the tasks comprising the task set…

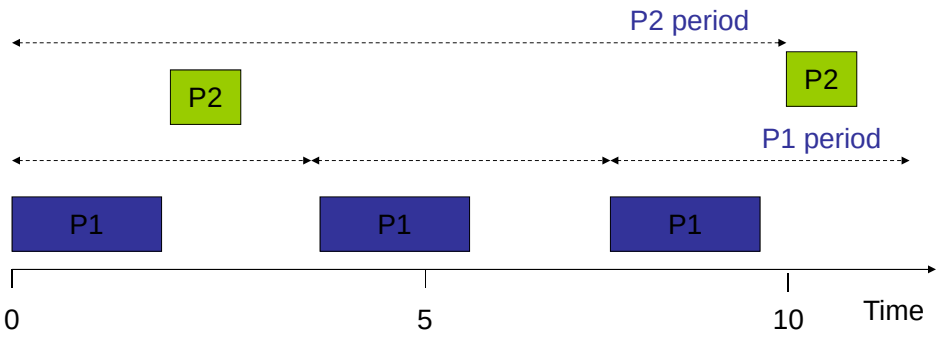
RMS Example 2

Process P_i	Execution Time T_i	Period t_i
P_1	1	**4**
P_2	6	**8**

Is this task set schedulable?? If yes, give the CPU utilization.

RMS CPU Utilization (cont'd)

- **RMS cannot asymptotically *guarantee* use of 100% of CPU, even with zero context switch overhead.**
  - Must keep idle cycles available to handle worst-case scenario.
- **However, RMS guarantees all processes will always meet their deadlines.**



RMS Implementation

- **Statically fixed priority assignment**
  - Inversely proportional to period

➢ **Efficient implementation**
  - Scan processes
  - Choose highest-priority active process

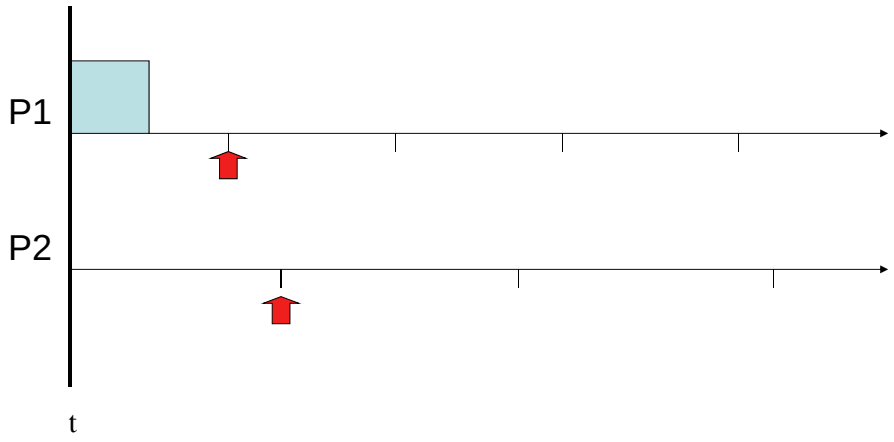
Earliest-Deadline-First (EDF) Scheduling

- ***Dynamic* priority scheduling scheme.**
  - Process closest to its deadline has highest priority
  - Requires recalculating processes at every timer interrupt

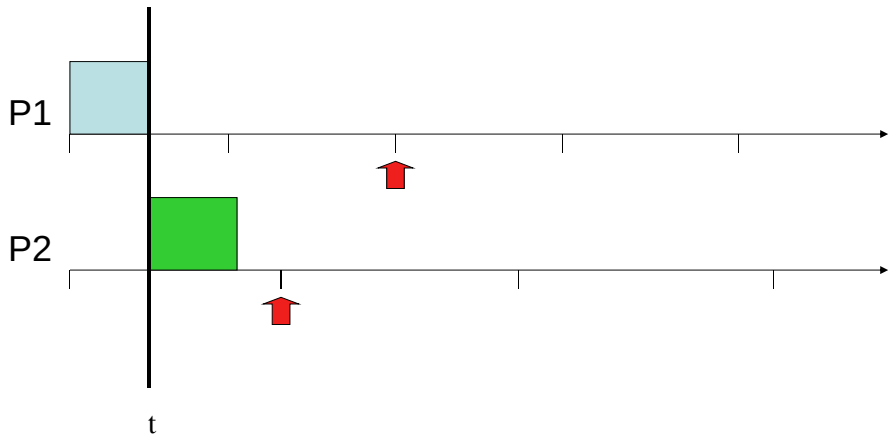
- **EDF analysis**
  - EDF can use 100% of CPU for worst case
  - ➢ Optimal for periodic scheduling

- **EDF implementation**
  - On each timer interrupt:
    - Compute time to deadline
    - Choose process closest to deadline
  - Generally considered too expensive to use in practice, unless the task count is small
    - Does not work in an OS with only fixed priorities!

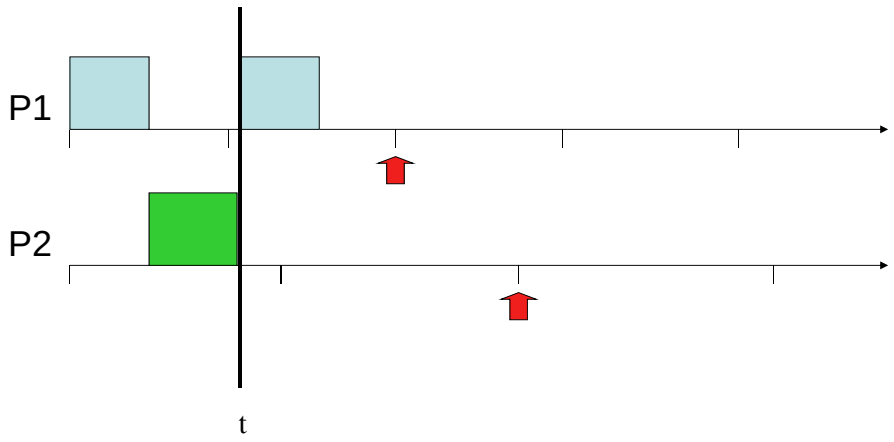
EDF Example



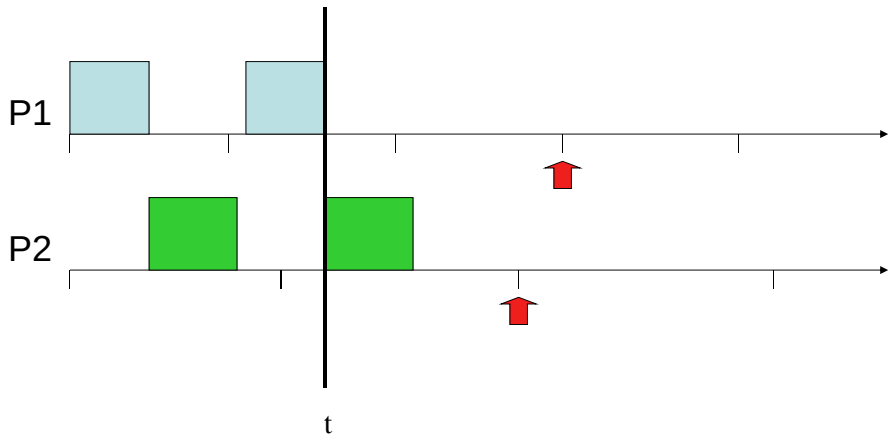
EDF Example



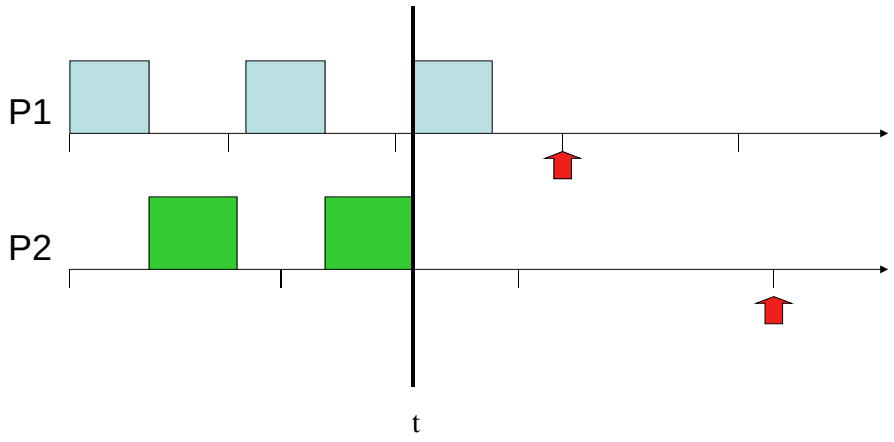
EDF Example



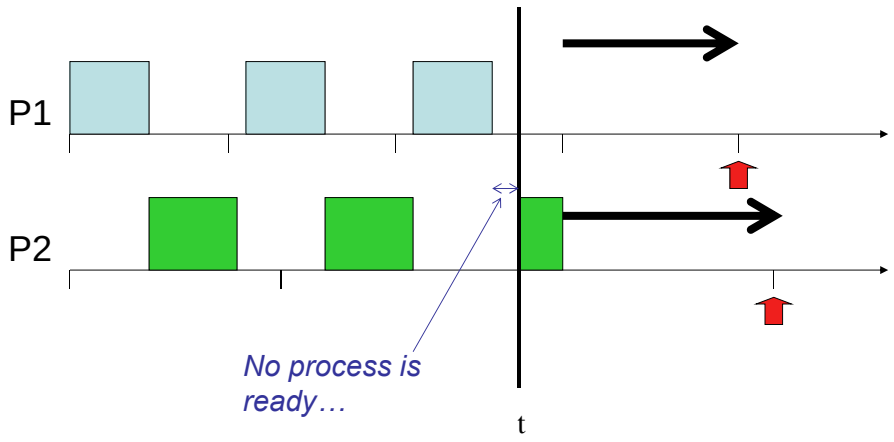
EDF Example



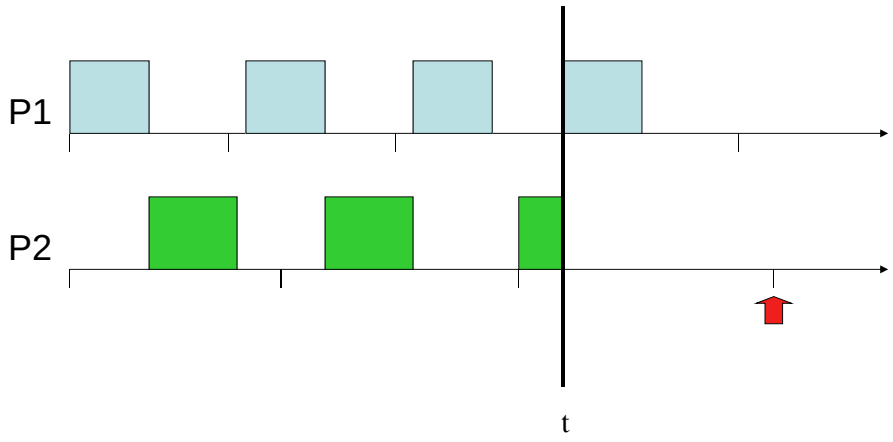
EDF Example



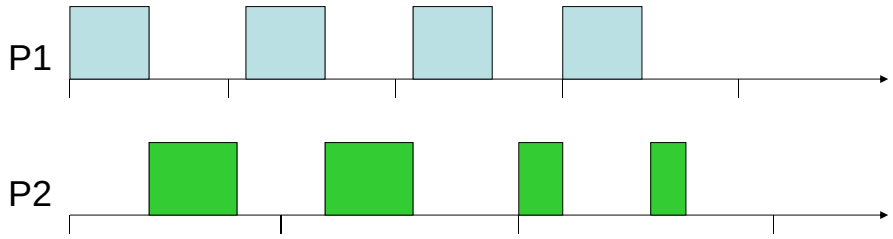
EDF Example



EDF Example



EDF Example



Real-Time Scheduling Algorithms (1)

- **Periodic, independent tasks**
  - Schedulability only (preemptive, static or dynamic)
    - Rate Monotonic Scheduling (RMS) is optimal fixed priority scheme
      - » Does not achieve 100% CPU utilization for guaranteed schedulability
    - Earliest Deadline First (EDF) is optimal dynamic priority scheme
      - » 100% utilization, but runtime support/overhead for dynamic priorities
- **Aperiodic, independent tasks (task set)**
  - Simultaneous (at system start) arrival times
    - Earliest Due Date (EDD) minimizes max. lateness (non-preemptive)
  - Arbitrary arrival times (statically know or dynamic)
    - Earliest Deadline First (EDF) minimizes max. lateness (preemptive)
    - Without preemption optimality only possible if arrival times known
- **Aperiodic, dependent tasks (task graph)**
  - Simultaneous (at system start) arrival times
    - Latest Deadline First (LDF) minimizes max. lateness (non-preempt.)
  - Arbitrary arrival times (statically know or dynamic)
    - Modified EDF* w/ successor-adjusted deadlines

Real-Time Scheduling Algorithms (2)

- **Periodic/sporadic, dependent tasks**
  - NP-complete in general
    - Use of heuristics
    - Split into periodic, independent and aperiodic, dependent subgraphs
- ➢ **Scheduling anomalies through dependencies (blocking)**
  - Deadlocks
  - Priority inversions

Scheduling Anomalies

- **"What really happened on Mars?" [WindRiver97]**

The New York Times
Monday, February 16, 2009

Archives

WORLD | U.S. | N.Y. / REGION | BUSINESS | TECHNOLOGY | SCIENCE | HEALTH | SPORTS

Mars Craft Again Halts Transmission

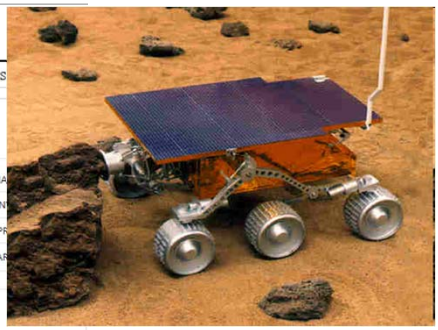
Published: July 15, 1997

The computer aboard the Mars Pathfinder overloaded and reset itself early today for the second time in just over three days, interrupting the transmission of a full-color panoramic scene.

The mishap delayed chemical analysis of a tubby rock named Yogi, but no information was lost, and controllers will be able to resume transmission where it was left off, officials said.

Mary Beth Murrill, a spokeswoman for NASA's Jet Propulsion Laboratory, said transmission of the panoramic shot took "a lot of processing power." She likened the data overload to what happens with a personal computer "when we ask it to do too many things at once."

The project manager, Brian Muirhead, said that to prevent a recurrence, controllers ... schedule activities one after another, instead of at the same time. It was the second t... the Pathfinder's computer had reset itself while trying to carry out several activities a... once.



Courtesy NASA/JPL-Caltech

➢ **Priority inversion**

Priority Inversion

- **Low-priority process keeps high-priority process from running.**
  - Improper use of system resources can cause scheduling problems
    - Low-priority process grabs I/O device.
    - High-priority device needs I/O device, but can't get it until low-priority process is done.
  - ➢ Can cause deadlock

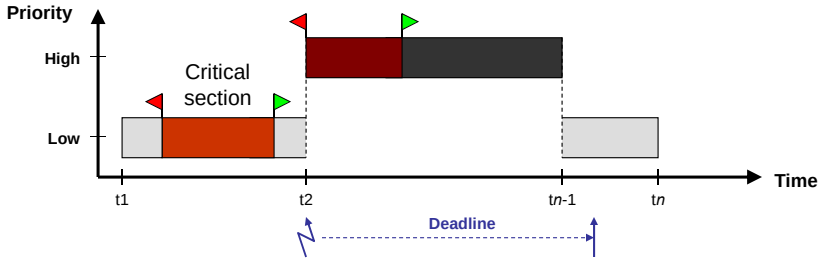
➢ **Give priorities to system resources**

➢ **Have process inherit the priority of a resource that it requests**

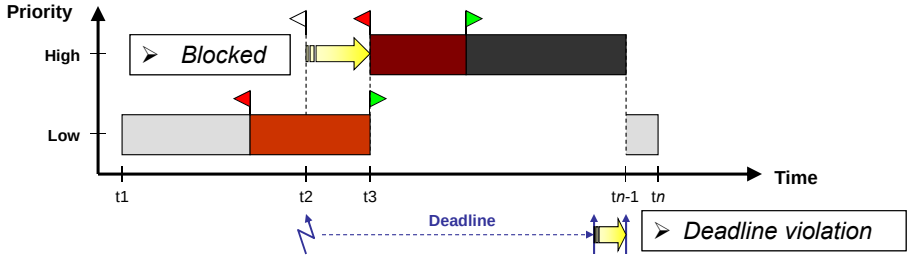
- ➢ Low-priority process inherits priority of device if higher

Priority-Based Scheduling

- **Normal operation**

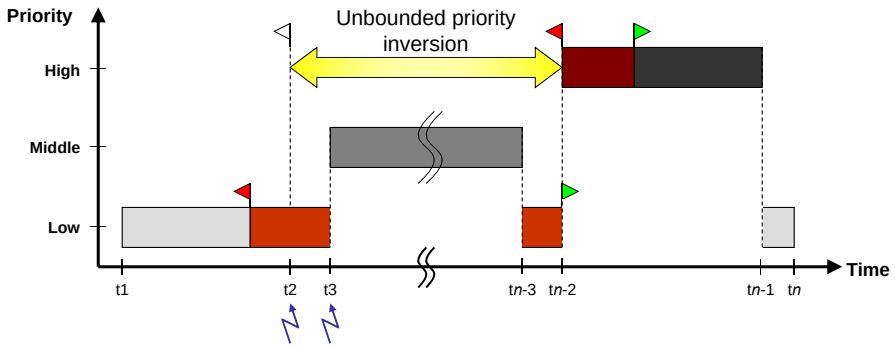


- **Priority inversion**



Priority Inversion

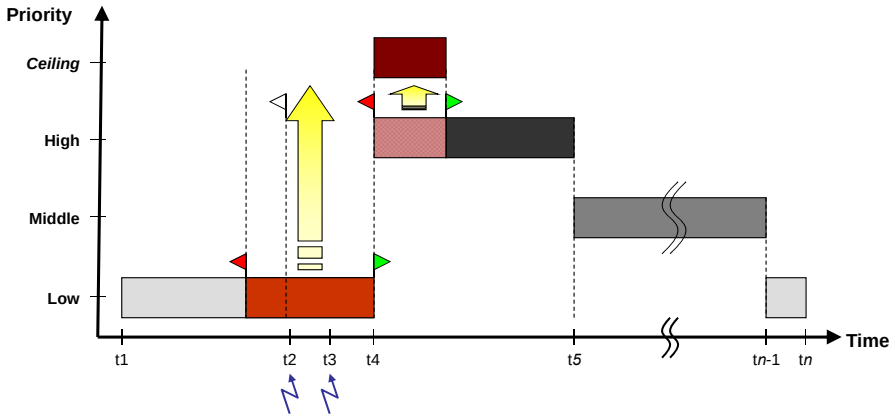
- **Low-priority process blocking a high-priority one**
  - Starvation of high priority processes



- **Avoid preemption in critical sections [Sha90]**
  - Interrupt masking
  - Priority Ceiling Protocol (PCP)
  - Priority Inheritance Protocol (PIP)

Priority Ceiling Protocol (PCP)

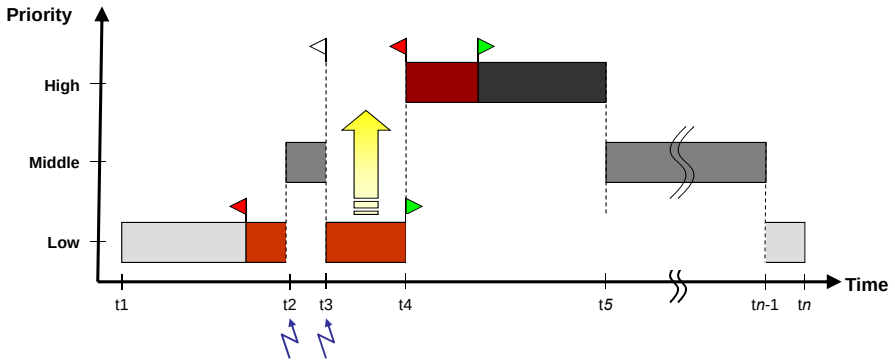
- **Elevate priorities in critical sections**
  - Assign priority ceilings to semaphore/mutex



- **Change task priority on semaphore/mutex access**
  - Also avoid potential deadlocks
  - Potential overhead & blocking of unrelated processes

Priority Inheritance Protocol (PIP)

- **Dynamically elevate priorities only when needed**
  - Raise priorities to level of requesting task



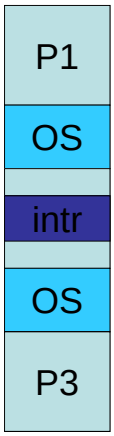
- **Change priority on request by higher-priority task**
  - Potential for deadlocks remains
  - Potentially multiple priority changes per critical section

Performance Evaluation

- **Context switch time**
  - Non-zero context switch time can push limits of a tight schedule
  - Hard to calculate effects
    - Depends on order of context switches
  - In practice, OS context switch overhead is small
- **May want to test**
  - Context switch time assumptions on real platform
  - Scheduling policy

What about interrupts?

- **Interrupt overhead**
  - Interrupts take time away from processes
  - Other event processing may be masked during interrupt service routine (ISR)
  - Perform minimum work possible in the interrupt handler
- **Device processing structure**
  - Interrupt service routine (ISR) performs minimal I/O.
    - Get register values, put register values
  - Interrupt service process/thread performs most of device function.



Caches

- **Processes can cause additional caching problems.**
  - Even if individual processes are well-behaved, processes may interfere with each other
  - Worst-case execution time with *bad cache behavior* is usually much worse than execution time with good cache behavior

➢ **Perform schedulability analysis without caches**
  - Take any online performance gains as "free lunch"

Lecture 9: Summary

- **Scheduling**
  - Dynamic, preemptive & priority-based scheduling
    - Real-time operating system (RTOS)
  - Periodic or aperiodic tasks
    - Earliest-deadline-first (EDF)
  - Independent tasks
    - Heuristics or partitioning first in case of dependencies

- **What if your set of processes is unschedulable?**
  - Change deadlines in requirements.
  - Reduce execution times of processes.
  - Get a faster CPU
  - ➢ Get an Accelerator!