MEI, MSI, MI
DI-FCUL, Portugal

Sistemas Embebidos e de Tempo-Real

Scheduling

Prof. António Casimiro

Summary

- Scheduling
  - Least Laxity First scheduling
  - Deadline Monotonic scheduling
  - Dealing with the priority inversion problem
    - Priority inheritance protocols
  - Dealing with event overloading problems
    - Mode changes

Least Laxity First (LLF)

- Assumptions:
  - Tasks are independent and periodic
  - Priorities are dynamically updated
  - Dynamic scheduling, like EDF
- Algorithm:
  - Tasks are ordered according to laxity: $T_{lax}(t) = (t_{dead} - t) - (T_{WCET} - T_{ET}(t))$
  - Lower laxity tasks are ordered first
  - Laxity remains constant while a task is executing
  - Laxity of the ready tasks decreases with each clock "tick"
- Schedulability:
  - Optimal for dynamic scheduling class, like EDF
- Problem:
  - **ill effect** or **thrashing**: continuous switching of tasks with similar laxity

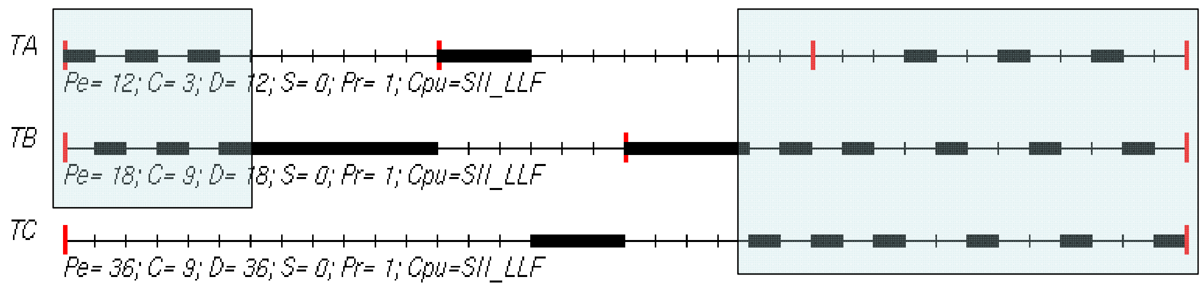
Least Laxity First

LLF Example	Periodic Tasks	Period T_i	Deadline D_i	Execution Time - C_i	Priority	Utilization	Utilization Limit	Schedule
	A	12	12	3	Dynamic	0,250		
	B	18	18	9	Dynamic	0,500		
	C	36	36	9	Dynamic	0,250		
Number of tasks	3					1,000	1,000	✓

Clock (Max. granularity)	3
Period (Max. granularity)	6
Minimum Period	12
Base Period	36

Least Laxity First

Example: Thrashing in LLF



Deadline Monotonic (DM)

- Assumptions (almost the same as in rate monotonic):
  - Activities independent and periodic
  - Bounded and known worst-case execution times (ci)
  - Negligible context switch time
  - **Deadline di ≤ Ti**
- Algorithm:
  - The task with the smaller relative deadline is assigned the highest priority
  - Tasks with higher priority preempt tasks with lower priority
- Schedulability:
  - Like the RM sufficient test, but using deadline instead of period:

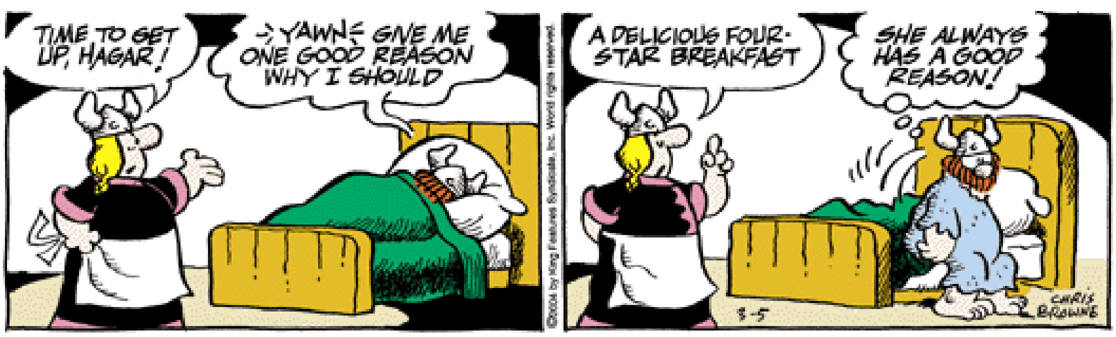
$$U = \sum_{i=1}^{n} c_i / D_i \leq n \cdot (2^{1/n} - 1)$$

First-Come-First-Served (FCFS)

- Schedules activities by release ordering, without preemption
- Does not make use of other scheduling parameter but the order in the ready queue

Illustrating the relevance of correctly scheduled tasks

Dirk Browne HAGAR, O TERRÍVEL



Scheduling approaches
Summary

- Fundamental approaches:
  - Cyclic executive
    - No notion of thread
    - Predetermined order of execution, in periodic cycle
  - Fixed priority scheduling
    - Threads have fixed, static, pre-computed priorities
  - Dynamic priority scheduling
    - Priorities are computed during the execution

Scheduling approaches
Summary

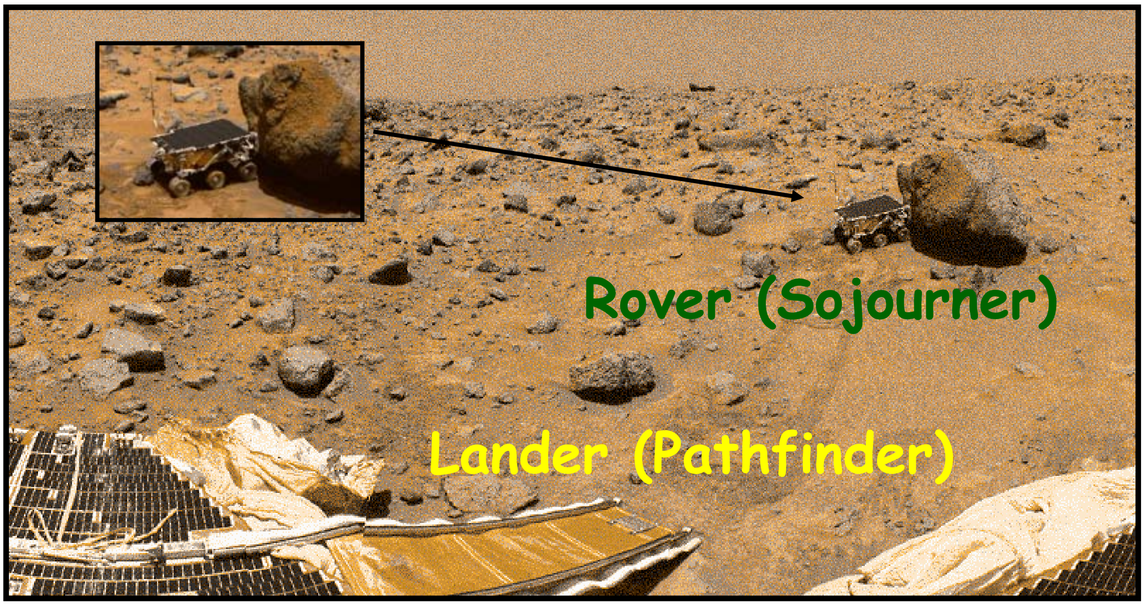
- Priority assignment
  - With FPS
    - Rate Monotonic: based on period
    - Deadline Monotonic: based on relative deadline
  - With dynamic priority scheduling
    - Earliest Deadline First: based on (absolute, calculated at given moment) deadline
    - Least Laxity First: based on (absolute, calculated at given moment) laxity

Sistemas Embebidos e de Tempo-Real
Copyright © 2009-16 A. Casimiro, J. Rufino DI – FCUL, Lisboa, Portugal.

Scheduling approaches
Summary

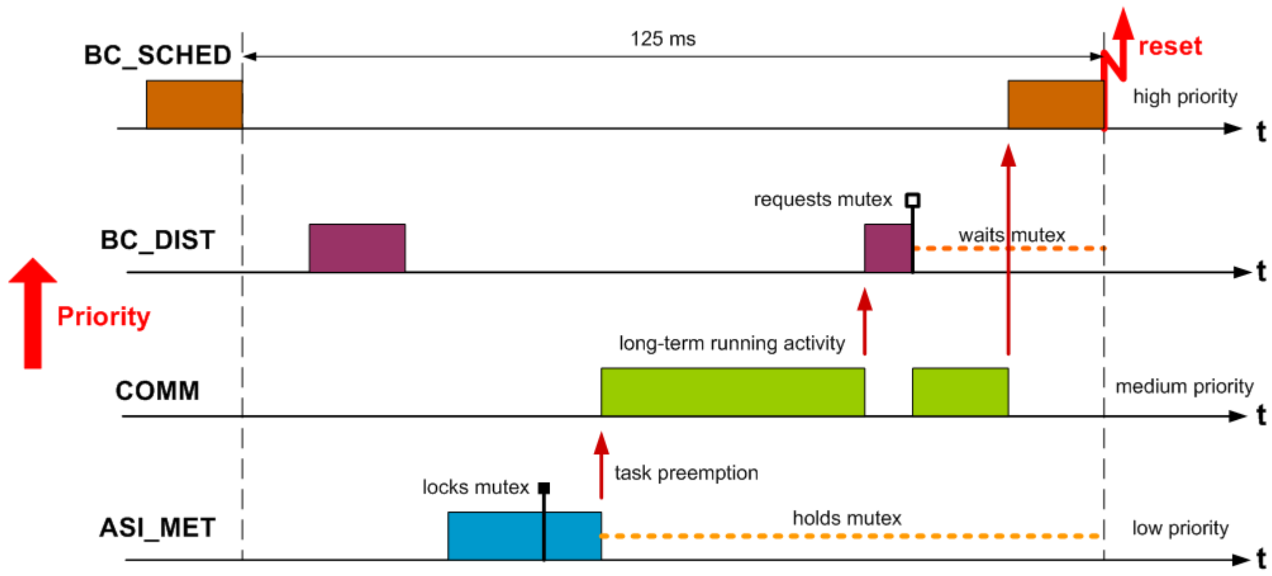
- Fixed Priority Scheduling
  - Easy implementation (scheduling attribute is static)
  - Easy handling of processes without need for deadlines
  - Easy handling of arbitrary parameters of importance
  - More predictable behavior
  - Not optimal (may not find feasible schedule with U <1)
- Dynamic priority Scheduling
  - Optimal utilization of resources (feasible with U<1)
  - More difficult to implement

Scheduling: practical problems
Case study - Mars Pathfinder, June 1997



Mars Pathfinder

Task execution scenario



The problem

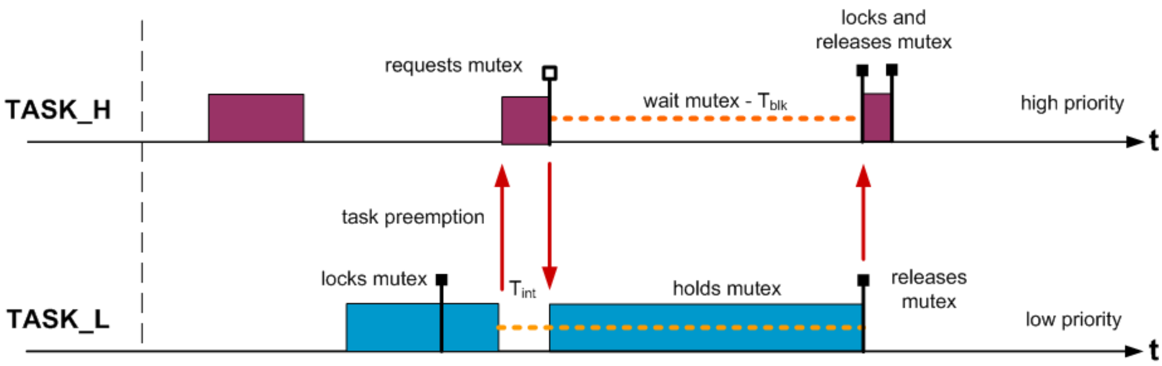
- Dependent tasks
  - Scheduling tasks with precedence or mutual exclusion constraints
  - Often find a feasible schedule becomes an NP-complete problem
  - Simpler but also non-exact scheduling analysis may have bad consequences

  - In the previous example: the dependency is on the use of semaphores to enforce mutual exclusion

The problem

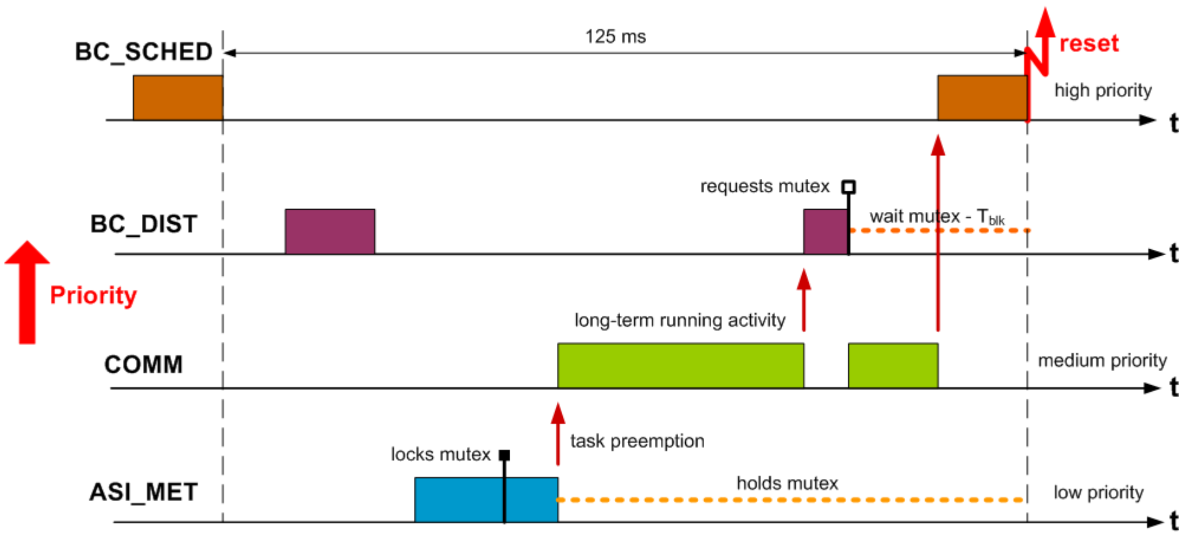
- Priority inversion
  - Shortcoming in scheduling mechanisms allowing a task blocking or delaying the execution of another with a higher urgency
  - Executing medium urgency tasks may prevent the lower urgency task of releasing the resource, thus further delaying the high urgency tasks
  - Example: Mars Pathfinder execution scenario
- Blocking time
  - Time interval a task cannot use a resource (e.g. the processor) because it is held by a less urgent task

Task blocking time



Not.	Designation	Description
T_{int}	max. interference time	maximum time a task can be suspended by more urgent tasks
T_{blk}	max. blocking time	maximum time a task can be blocked by less urgent tasks

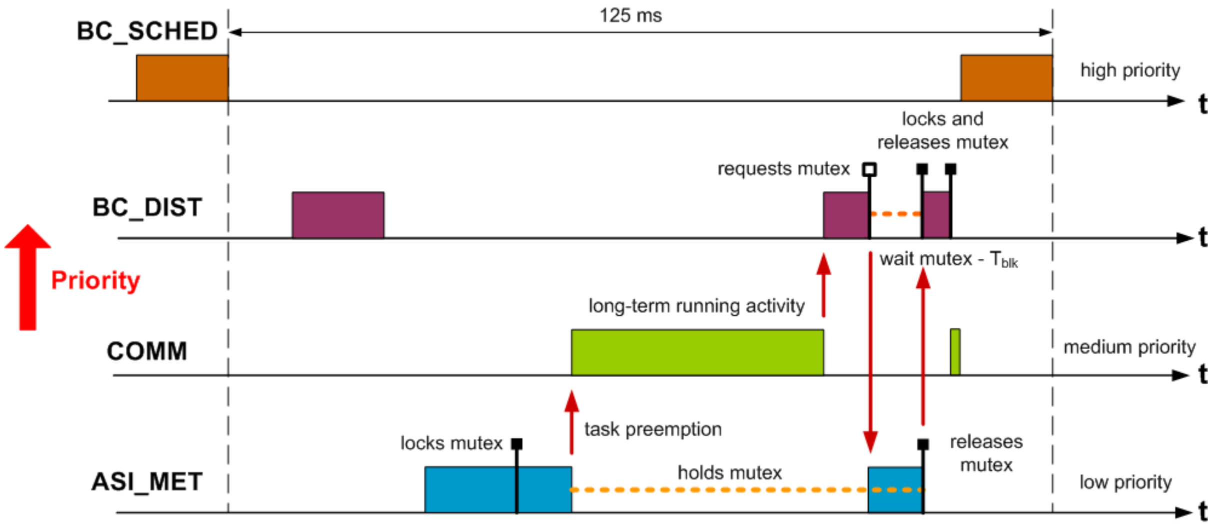
Priority inversion



Priority inheritance

- Priority Inheritance Protocol (PIP)
  - Mechanism allowing a lower priority task to inherit the priority of a higher priority task that becomes blocked in the same resource
  - This mechanism alone does not prevent chained blocking and deadlock

Priority inheritance



Priority inheritance

- Priority Ceiling Protocols
  - Original Ceiling Priority Protocol (OCPP)
  - Immediate Ceiling Priority Protocol (ICPP)
- Characteristics
  - A high priority task is blocked only once by a lower priority task
  - Deadlocks are prevented
  - Transitive blocking is prevented
  - Mutual exclusive access to resources is ensured

Priority inheritance

- Original Ceiling Priority Protocol (OCPP)
  - Each task has a static default priority
  - Each resource has a static ceiling priority, which is equivalent to the maximum priority of the all task using the resource
  - A dynamic priority is assigned to tasks, being the maximum of its own priority and **any it inherits due to blocking higher-priority tasks**
  - A task can only acquire a resource if its dynamic priority is higher than the ceilings of all resources locked by tasks other than the task itself

Priority inheritance

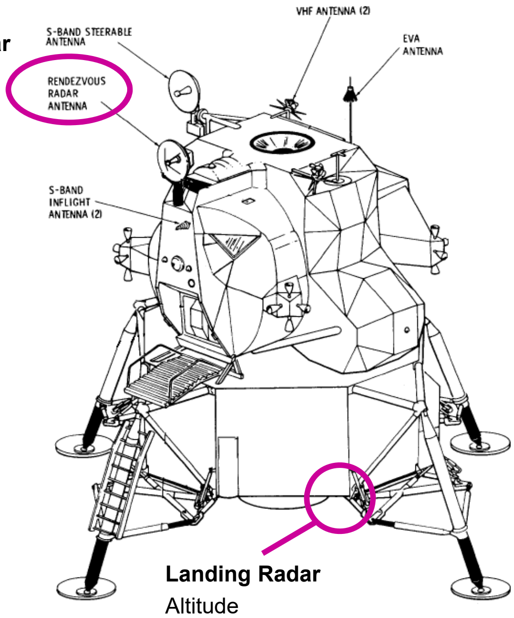
- Immediate Ceiling Priority Protocol (ICPP)
  - Each task has a static default priority
  - Each resource has a static ceiling priority, which is equivalent to the maximum priority of all tasks using the resource
  - A dynamic priority is assigned to tasks, being the maximum of its own priority and **the ceiling of any resources it has locked**
    - Therefore, a task is unable to execute while the resources it needs are being use by another task
  - Simpler to implement; requires less context switches

Scheduling: practical problems

Lunar Module (LM) Infrastructure

LM (Apollo) Guidance Computer
16-bit CPU 100 kHz
32 KiB magnetic core ROM
4 KiB magnetic core RAM
2 KiB priority-based event-driven OS
assembly language programming

Rendezvous Radar
Range
Range rate
Tracking angles

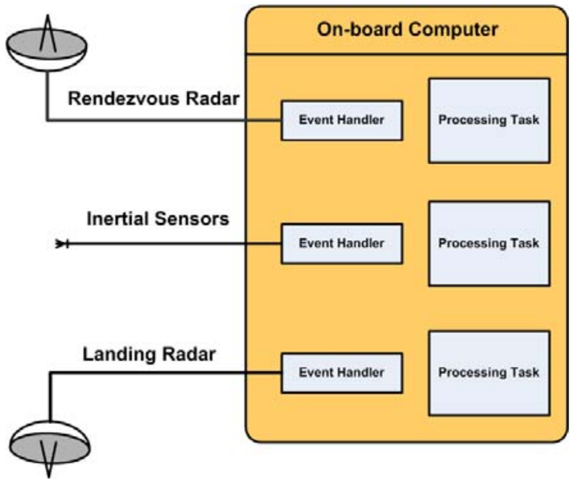


Landing Radar
Altitude
Vertical velocity
Forward and lateral velocity

Incident: on-board computer overload almost caused abortion of the first lunar landing

The problem

- **Problem:** Just a few minutes to go before landing, on-board computer started to repeatedly display a “1202 alarm”.
- **Mission Control Engineers Advise:** ignore the alarm and go for landing.
- **Cause of the alarm:** events produced by the rendezvous radar introduced an unexpectedly high overhead in the event (interrupt) handling processing.
- **Possible Solution**: mode change



Load control mechanisms

Mode change

- Allocation of resources is driven by the operational requirements of a task set
- Different operating modes
  - Example: on a spacecraft cruise and landing
- Change of schedules on mode change

Event load control

Response-time analysis

- Account for the interference of (interrupt) event processing

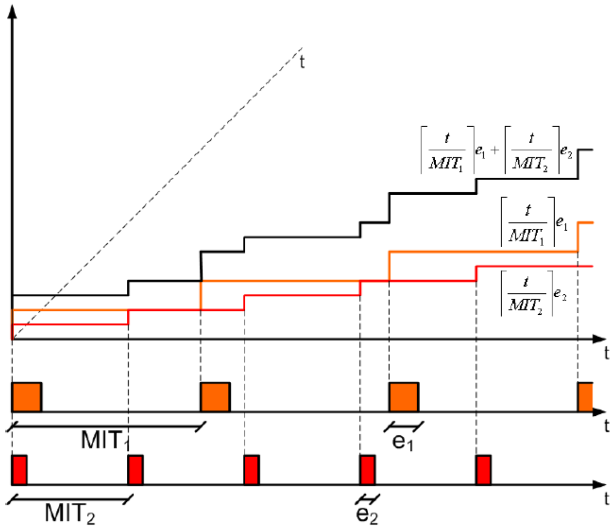
$$R_i^0 = c_i$$

$$R_i^k = c_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^{k-1}}{T_j} \right\rceil \cdot c_j + \sum_{l \in hpi(i)} \left\lceil \frac{R_i^{k-1}}{m_l} \right\rceil \cdot c_l$$

Event load control

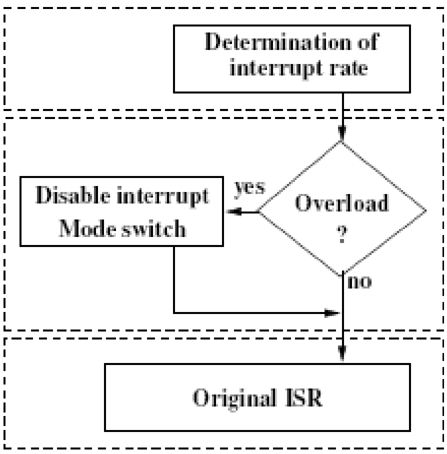
Workload

- Illustrating the workload due to the interference of (interrupt) event processing



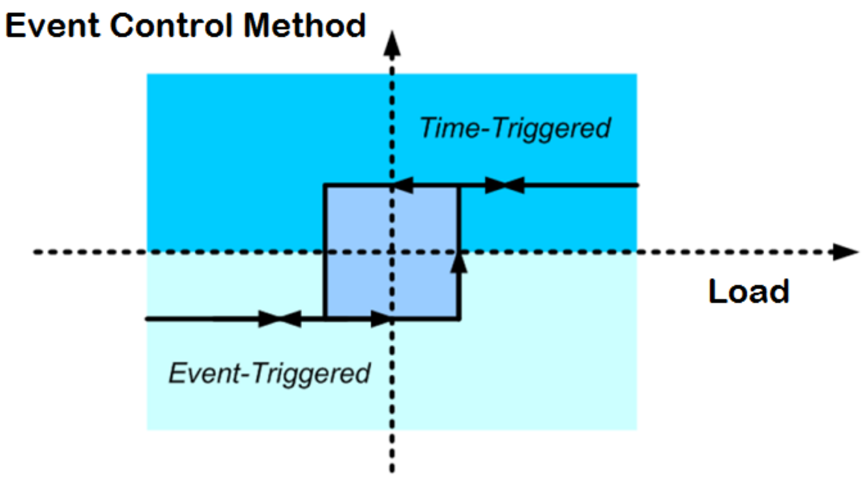
Event load control

- Account for interrupt event metrics (e.g. rate, etc)
- In the presence of interrupt overload, perform mode change on interrupt service routine



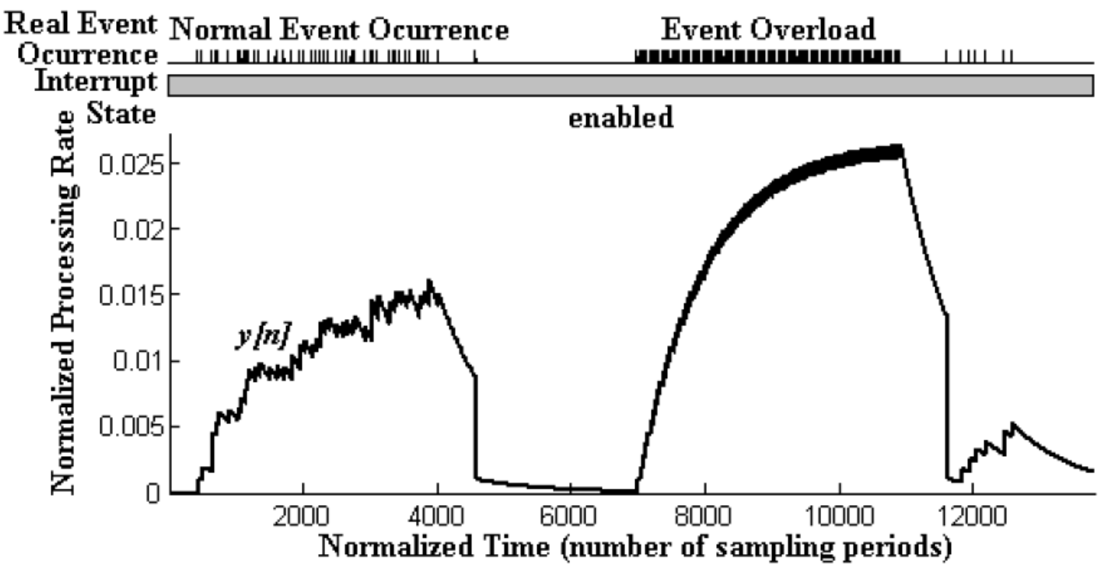
Event load control

- Avoidance of control instability by hysteresis



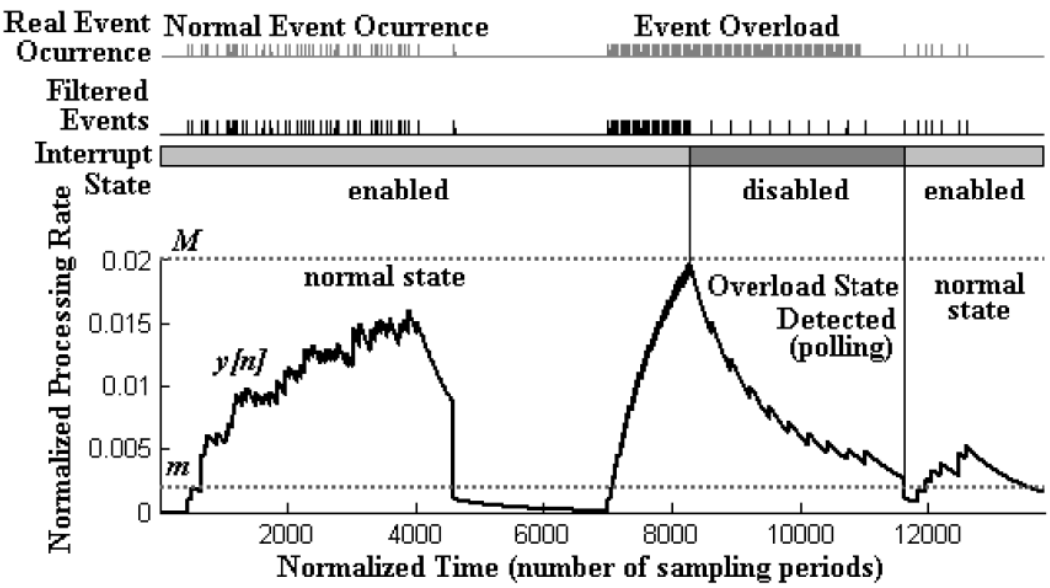
Event load control

Example: event overload occurrence



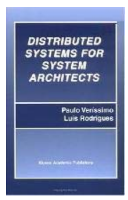
Event load control

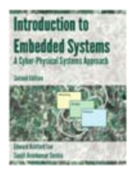
Example: overload control by mode change



Bibliography

- Textbook:
  - P. Veríssimo and L. Rodrigues, Distributed Systems for System Architects, Kluwer Academic Publishers, 2001, 650pp., Part III – Real-Time.
    - Section 12.7
  - Hermann Kopetz, Real-Time Systems: Design Principles for Distributed Embedded Applications, Kluwer Academic Publishers, 1997.
    - Chapter 10
  - Edward A. Lee and Sanjit A. Seshia, Introduction to Embedded Systems, A Cyber-Physical Systems Approach, Second Edition, http://LeeSeshia.org
    - Chapter 12





Thank you for your participation.
Final questions?

https://moodle.ciencias.ulisboa.pt
Sistemas Embebidos e de Tempo-Real

The Disintegration of the Persistence of Memory
Salvador Dali Museum, St. Petersburg, Florida
Salvador Dali, 1952-54