Predictable response times in event-driven real-time systems

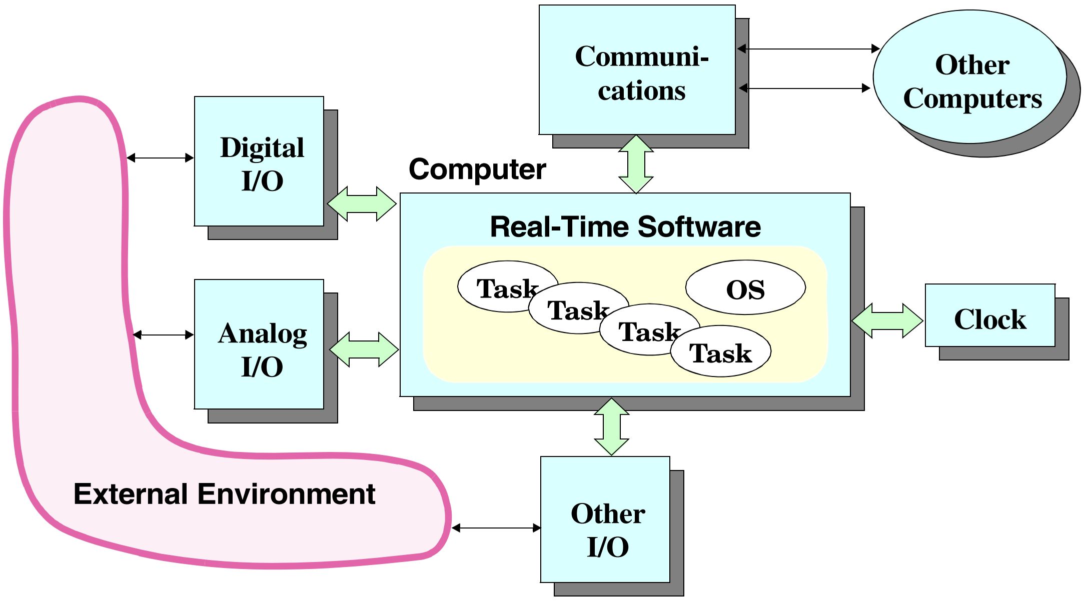
Artist2 Summer School in China 2008

Shanghai, July 2008

Michael González Harbour
`mgh@unican.es`
`www.ctr.unican.es`

Predictable response times in event-driven real-time systems

Elements of a real-time system



Real-time systems

A Real-time system is a combination of a computer, hardware I/O devices, and special-purpose software, in which:

- **there is a strong interaction with the environment**
- **the environment changes with time**
- **the system simultaneously controls and/or reacts to different aspects of the environment**

As a result:

- **timing requirements are imposed on software**
- **software is naturally concurrent**

To ensure that timing requirements are met, the system's timing behavior must be *predictable*

Real-Time Systems

"In Real-time applications, the correctness of computation depends upon not only the results of computation, but also the time at which outputs are generated."

What's important in real-time

Predictability **of the response time**

Criteria **for real-time systems differ from that for time-sharing systems.**

	Time-Share Systems	Real-Time Systems
Capacity	High throughput	Ability to meet timing requirements: Schedulability
Responsiveness	Fast average response	Ensured worst-case latency
Overload	Fairness	Stability of critical part

Worst case cannot be checked by *testing*

Options for managing time

Compile-time schedules:

- time triggered or cyclic executives
- predictability through static schedule
- logical integrity often compromised by timing structure
- difficult to handle aperiodic events & dynamic changes
- difficult to maintain

Run-time schedules:

- priority-based schedulers
- preemptive or non preemptive
- analytical methods needed for predictability
- separates logical structure from timing
- more flexibility

Example: Control in an Automobile

Activities to control:

Speed measurement

C=4 ms.
T=20 ms.
D=5 ms.

ABS Control

C=10 ms.
T=40 ms.
D=40 ms.

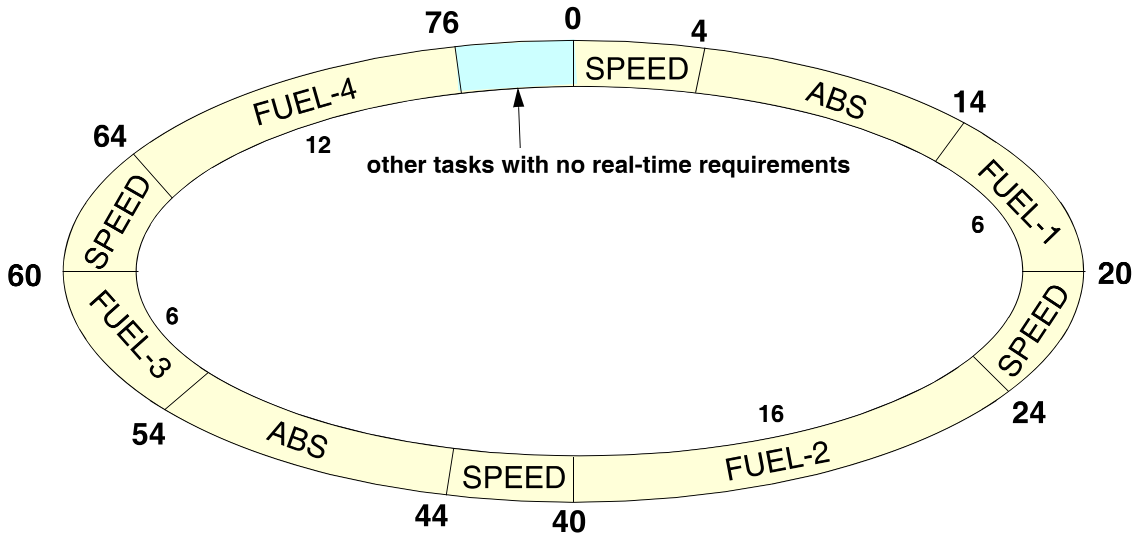
Fuel Injection

C=40 ms.
T=80 ms.
D=80 ms.

C = Execution time
T = Period
D = Deadline

Example: cyclic solution

A cyclic solution implies breaking the longest activity in several parts:



other tasks with no real-time requirements

Example: concurrent solution

The concurrent solution is easier to design and maintain:

Speed measurement

```
loop
   Measure_Speed;
   next:=next+0.02;
   Sleep_until next;
end loop;
```

ABS control

```
loop
   ABS_Control;
   next:=next+0.04;
   Sleep_until next;
end loop;
```

Fuel injection

```
loop
   Fuel_Injection;
   next:=next+0.08;
   Sleep_until next;
end loop;
```

Activities with no timing requirements

```
loop
   do work;
   ...
end loop;
```

Example: maintenance

Suppose that we need to execute an aperiodic activity with a 1ms execution time. The minimum interarrival time is 80ms., but the deadline is 20 ms:

1. **Cyclic solution:**
   - **Sample at least every 19 ms to check if an aperiodic event has arrived**
   - **This implies breaking ABS and FUEL in several parts each**
2. **Concurrent solution:**
   - **Add a new high-priority process and repeat the schedulability analysis**
   - **No modifications to existing code**

Fixed-priority scheduling (FPS)

Fixed-priority preemptive scheduling is very popular for practical applications, because:

- **Timing behavior is simpler to understand**
- **Behavior under transient overload is easy to manage**
- **A complete analytical technique exists**
- **High utilization levels may be achieved (typically 70% to 95% of CPU)**
- **Supported in standard concurrent languages or operating systems:**
  - **Ada 2005’s RT-annex, Java RTSJ**
  - **Real-time POSIX**

Dynamic priority scheduling

In dynamic priority scheduling it is possible make better use of the available resources

We can find two kinds of dynamic priority policies

- *static job priority*: a static priority is assigned to each task job
  - e.g., *EDF* (earliest deadline first): each job is assigned a priority equal to its absolute deadline
  - the absolute deadline of a job does not change
- *dynamic job priority*: the priority of a task job may change before the job is finished
  - e.g., LLF (Least Laxity First): the priority of a job is inverse to the laxity: time left until the end of the job’s absolute deadline, minus the remaining computation time
  - the laxity changes with time

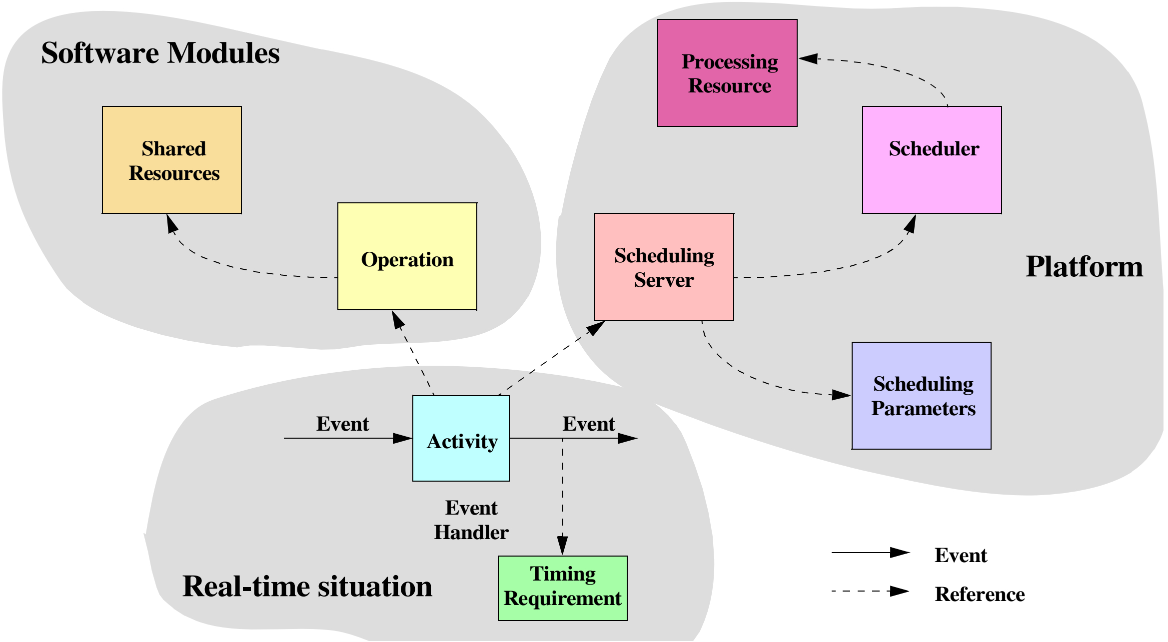
Dynamic priority scheduling

Static job priority policies are more common, because they are simpler

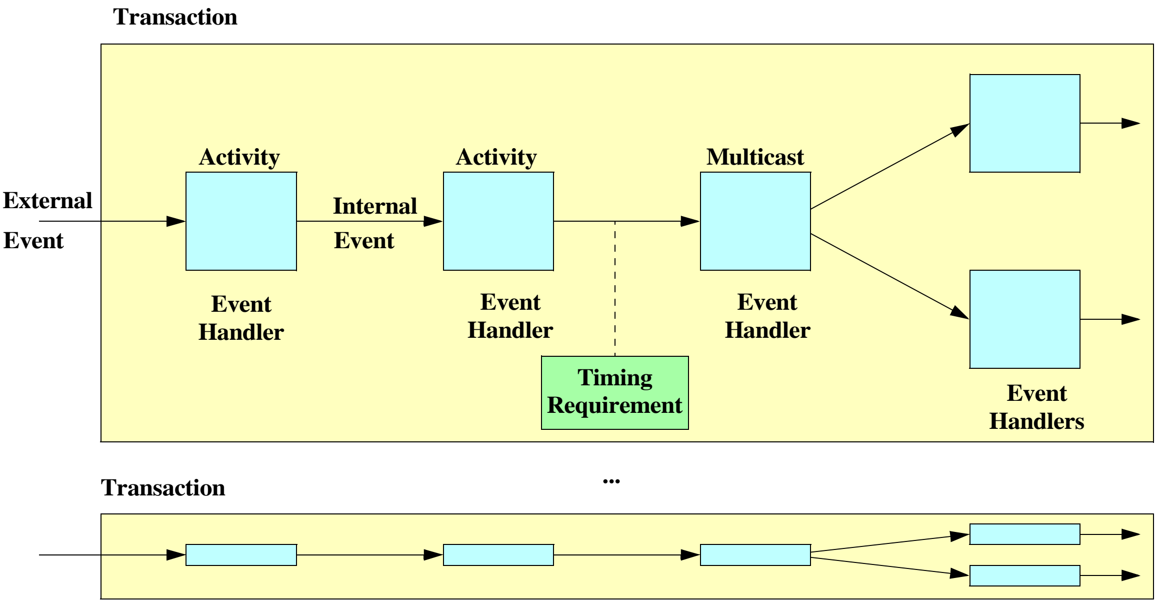
- EDF and its compatible variants are the most common
- treatment of transient overload is more complex
- not supported by standard operating systems
  - supported in Java RTSJ
  - recently added to Ada 2005

Mixed EDF/FPS schemes are possibly the best approach

Real-time system model



Real-time situation



Short History of Real-Time Analysis

Initial paper by Liu and Layland in 1973

- introduced Rate Monotonic and EDF scheduling
- only applicable to a very restrictive case with independent periodic tasks, and deadlines = periods
- optimal priority assignments (when deadlines are at end of period)
- analytic formulas to check schedulability (utilization tests)

Exact schedulability tests (response time analysis)

- developed by Harter (1984) and Joseph and Pandya (1986)

Extended to handle arbitrary deadlines

- Lehoczky (1990)

Extended to handle input jitter

- Tindell (1994)

Extensions to basic theory

Priority inversion and task synchronization

- Immediate priority ceilings
  - Lampson and Redell (1980)
  - Baker (1991)
- shared resources with priority inheritance
  - Sha, Rajkumar, Lehoczky (1990)
- multiprocessor systems
  - Rajkumar, Sha, Lehoczky (1988), Rajkumar (1990)

Aperiodic tasks

- Sporadic server, fixed priorities
  - Sprunt, Sha, Lehoczky (1989)

Multiprocessor and distributed systems, OS issues, etc.

Holistic Analysis.

- Tindell (1994) and Tindell and Clark (1994)
- Palencia et al (1997)

Offset Based Analysis

- Tindell (1994)
- Extended to distributed systems by Palencia and González (1998)
- Optimized by Palencia and González (1999), and by Redell (2003)

Priority assignment techniques

- Rate Monotonic for deadlines equal to periods
  - Liu and Layland (1973)
- Deadline monotonic assignment for pre-period deadlines
  - Leung and Layland (1982)
- Deadlines larger than periods
  - Audsley (1991)
- Distributed systems
  - Tindell, Burns, and Wellings (1992)
  - Gutiérrez García and González Harbour (1995)

Dynamic priorities

EDF response time analysis

- Single processor systems
  - Spuri (1996)
- Distributed systems
  - Holistic analysis: Spuri (1996)
  - Offset-based analysis: Palencia and González Harbour (2003)
- Synchronization
  - SRP: Baker (1991)
- Aperiodic tasks
  - Constant bandwidth server, Abeni and Buttazzo (1998)
- Hierarchical scheduling
  - González Harbour and Palencia (2003)

Influence in Standards

IEEE POSIX standards

- Real-time extensions (fixed priorities, priority inheritance protocols)
- Advanced real-time extensions (execution-time clocks, sporadic servers)

Ada

- 1983: fixed priorities
- 1995: priority ceilings in protected objects
- 2005: execution time clocks, EDF

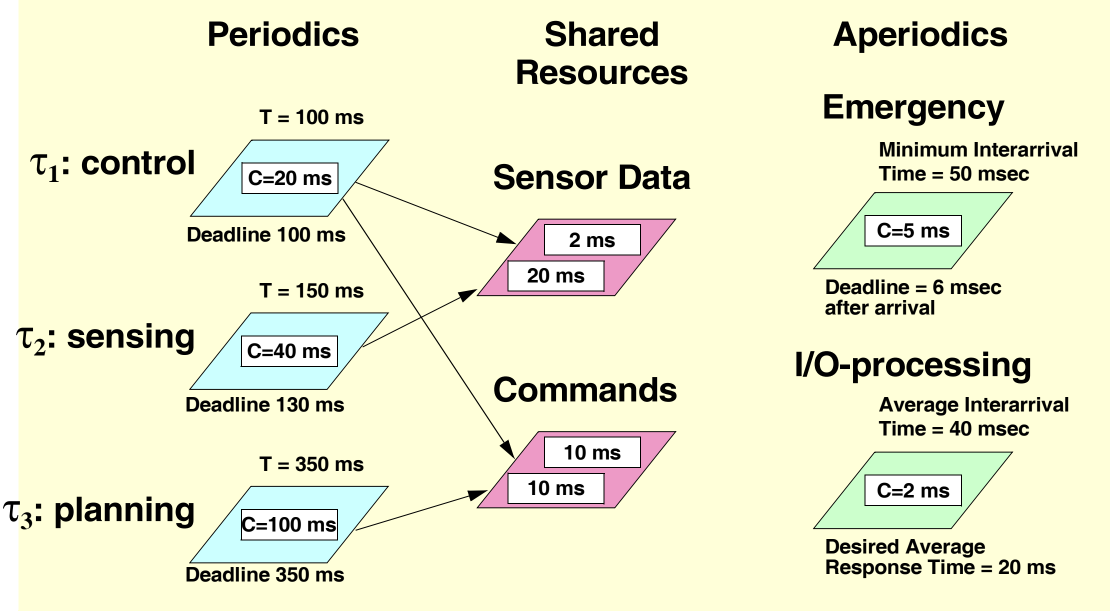
Real-Time Java

- fixed priorities, EDF, priority inheritance, ...

Predictable response times in event-driven real-time systems

1. Introduction
2. ***Basic scheduling***
3. Mutual exclusion
4. Distributed systems
5. Hierarchical scheduling
6. Protection and flexible scheduling frameworks
7. Operating systems
8. Programming languages
9. Modeling and integration into the design process
10. Conclusions

Activities in a basic real-time system: example



Periodics

τ_1: control
T = 100 ms
C=20 ms
Deadline 100 ms

τ_2: sensing
T = 150 ms
C=40 ms
Deadline 130 ms

τ_3: planning
T = 350 ms
C=100 ms
Deadline 350 ms

Shared Resources

Sensor Data
2 ms
20 ms

Commands
10 ms
10 ms

Aperiodics

Emergency
Minimum Interarrival Time = 50 msec
C=5 ms
Deadline = 6 msec after arrival

I/O-processing
Average Interarrival Time = 40 msec
C=2 ms
Desired Average Response Time = 20 ms

Concepts and Definitions - Periodics

Periodic task

- initiated at fixed intervals
- must finish before start of next cycle

Task's CPU utilization: $U_i = C_i / T_i$

- C_i = compute time (execution time) for task τ_i
- T_i = period of task τ_i
- P_i = priority of task τ_i
- D_i = deadline of task τ_i
- ϕ_i = phase of task τ_i
- R_i = response time of task τ_i

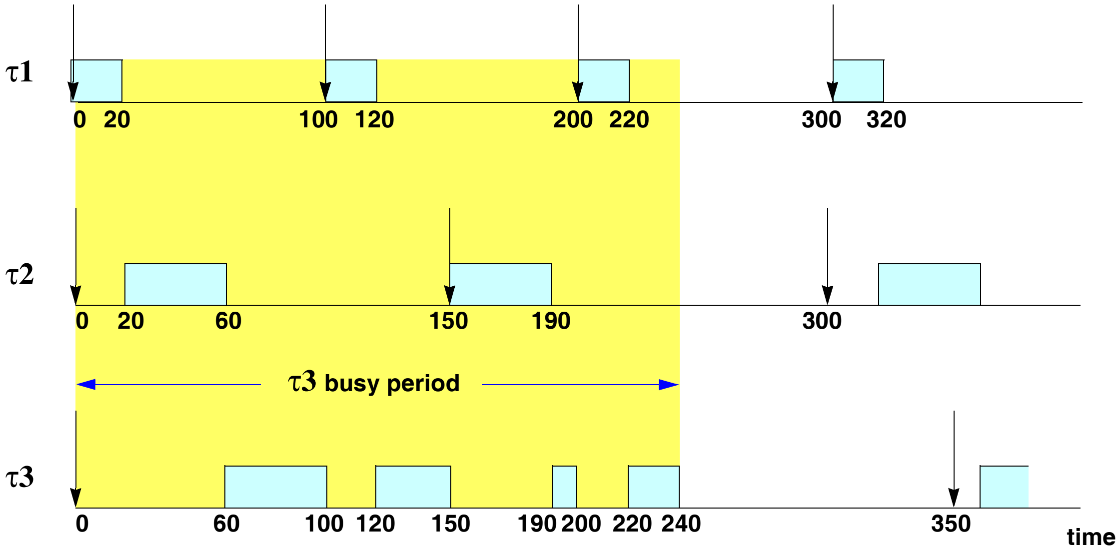
CPU utilization for a set of tasks: $U = U_1 + U_2 + \ldots + U_n$

Basic principles of real-time analysis

Two concepts help building the worst-case condition under fixed priorities:

- *Critical instant*. The worst-case response time for all tasks in the task set is obtained when all tasks are activated at the same time
- *Checking only the deadlines in the worst-case busy period*.
  - for task: interval during which the processor is busy executing τ_i or higher priority tasks

Example of a critical instant



Priority assignment

If $D_i \le T_i$, *deadline monotonic* assignment

- For a set of periodic independent tasks, with deadlines within the period, the optimum priority assignment is the deadline monotonic assignment:
  - Priorities are assigned according to task deadlines
  - A task with a shorter deadline is assigned a higher priority

If $D_i > T_i$ for one or more tasks, Audsley’s algorithm

- iteratively apply analysis, successively ordering tasks by priority: $O(n^2)$ times the analysis

Utilization bound test (D=T) (Liu and Layland, 1973)

Utilization Bound Test: **A set of *n* independent periodic tasks, with deadlines at the end of the periods, scheduled by the rate monotonic algorithm will always meet its deadlines, for all task phasings, if**

$$\frac{C_1}{T_1} + \dots + \frac{C_n}{T_n} \le U(n) \;=\; n(2^{1/n} - 1)$$

U(1) = 1.0	U(4) = 0.756	U(7) = 0.728
U(2) = 0.828	U(5) = 0.743	U(8) = 0.724
U(3) = 0.779	U(6) = 0.734	U(∞) = 0.693

For harmonic task sets, the utilization bound is *U(n)=1.00* for all *n*.

Response time analysis (Harter, 1984; Joseph and Pandya, 1986)

Iterative test (pseudopolynomial time):

$$a_0 \;=\; C_1 + C_2 + \dots + C_i$$

$$a_{k+1} \;=\; W_i(a_k) \;=\; \left\lceil \frac{a_k}{T_1} \right\rceil C_1 + \dots + \left\lceil \frac{a_k}{T_{i-1}} \right\rceil C_{i-1} + C_i + B_i$$

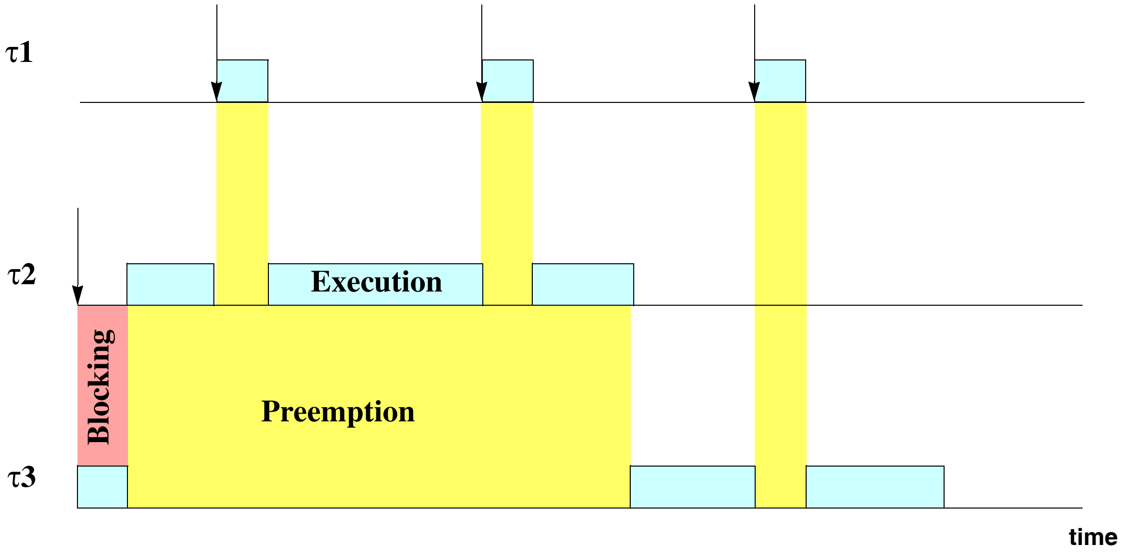
blocking

preemption

execution

Finish when two consecutive results are the same

Elements that influence the response time



Enhancements to response-time analysis (FPS)

Analysis with arbitrary deadlines (Lehoczky, 1990)

- checking first deadline is not enough
- repeat analysis for multiple jobs in a busy period

Analysis with release jitter (Tindell, 1992)

Analysis with overhead effects (context switch)

Methods for controlling input/output jitter

Analysis with arbitrary deadlines and jitter (Tindell, 1992):

We call the maximum release jitter of task i, J_i

Worst case *response time* of a task: found in a busy period in which all tasks:

- are released at the beginning of the busy period,
- have experienced their maximum jitter on the first job
- and experience the minimum jitter on the following jobs that make them happen inside the busy period

---

Analysis with arbitrary deadlines and jitter (cont'd):

Iterative equation used for analysis of task-*i* in one resource:

$$w_i^{n+1}(p) = \sum_{\forall j \in hp(i)} \left\lceil \frac{J_j + w_i^n(p)}{T_j} \right\rceil C_j + pC_i + B_i$$

This is carried out for p=1,2,3,..., until

$$w_i(p) \le pT_i$$

Modeling Task Switching

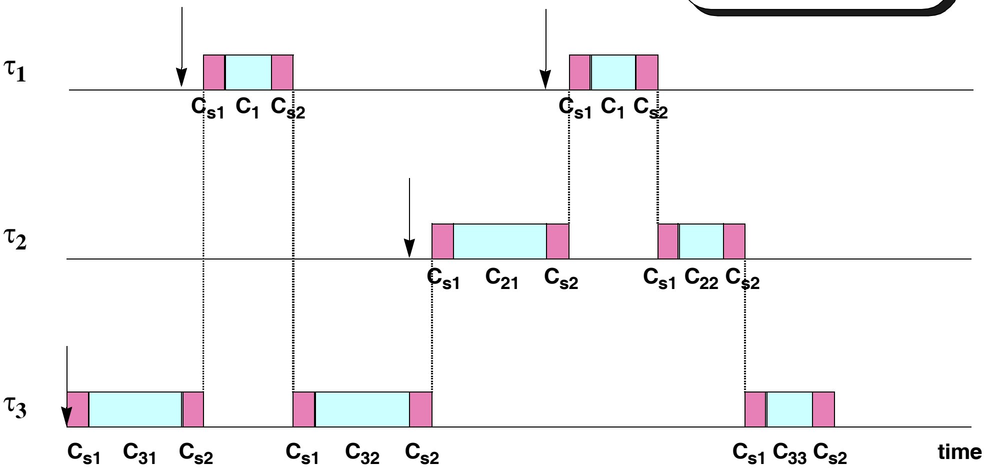
Two scheduling actions per preemptive task
(preemption, and processor relinquishing)

$C_s = C_{s1} + C_{s2}$ $C_i \Rightarrow C_i + 2C_s$

Legend

🟦 **Task Execution**

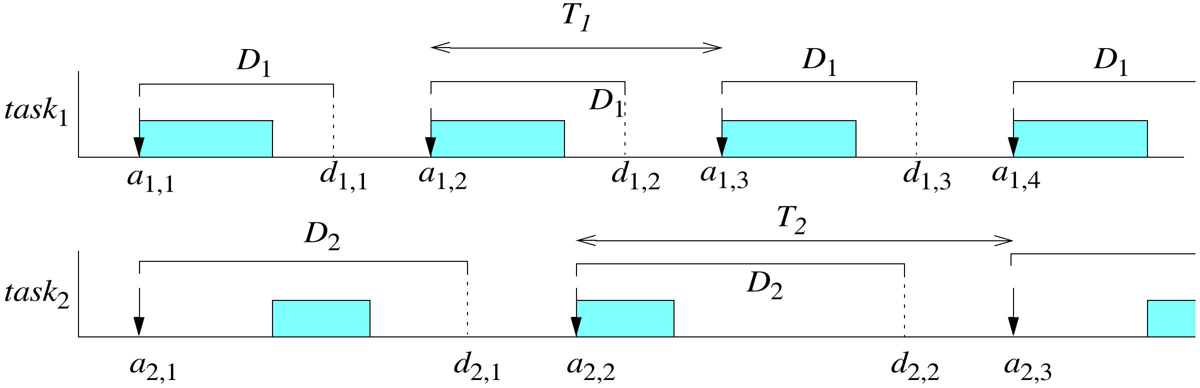
🟪 **Context Switch Activity**



EDF Scheduling Policy

Each task i has a relative period, T_i, and a relative deadline assigned: D_i

Each task-i job j has an absolute activation time, $a_{i,j}$, and an absolute deadline, $d_{i,j}$, used as the inverse of the job priority



Optimality of EDF

In a system with only periodic independent tasks, with a preemptive scheduler, executing in a single processor

- The EDF scheduling policy is optimal (Liu & Layland, 1973)
- 100% utilization may be achieved
- EDF utilization bound test is exact:

$$U = \sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1$$

Response time analysis for EDF

Busy period: interval during which processor is not idle

Worst case *response time* of a task: found in a busy period in which all other tasks:

- are released at the beginning of the busy period,
- and have experienced their maximum jitter

Differences with fixed priorities:

- The task under analysis does not necessarily start with the busy period
- The busy period is longer

Response Time Analysis for EDF

Worst contribution of task τ_i to the busy period at time t, when the deadline of the analyzed task, τ_a, is D:

$$W_i(t, D) = min\left(\left\lceil \frac{t+J_i}{T_i} \right\rceil, \left\lfloor \frac{J_i+D-d_i}{T_i} \right\rfloor + 1\right)_0 \cdot C_i$$

Worst completion time of activation p, if first activation at A:

$$w_a^A(p) = pC_a + \sum_{\forall i \neq a} W_i(w_a^A(p), D^A(p))$$

Worst response time if first activation is A:

$$R^A(p) = w_a^A(p) - A + J_a - (p-1)T_a$$

Response time analysis in a single resource (cont'd)

Set of potential critical instants; L is the longest busy period:

$$\Psi = \cup\{(p-1)T_i - J_i + d_i\} \qquad \forall p = 1\ldots\left\lceil \frac{L-J_a}{T_a} \right\rceil, \forall i \neq a$$

Values of A to check:

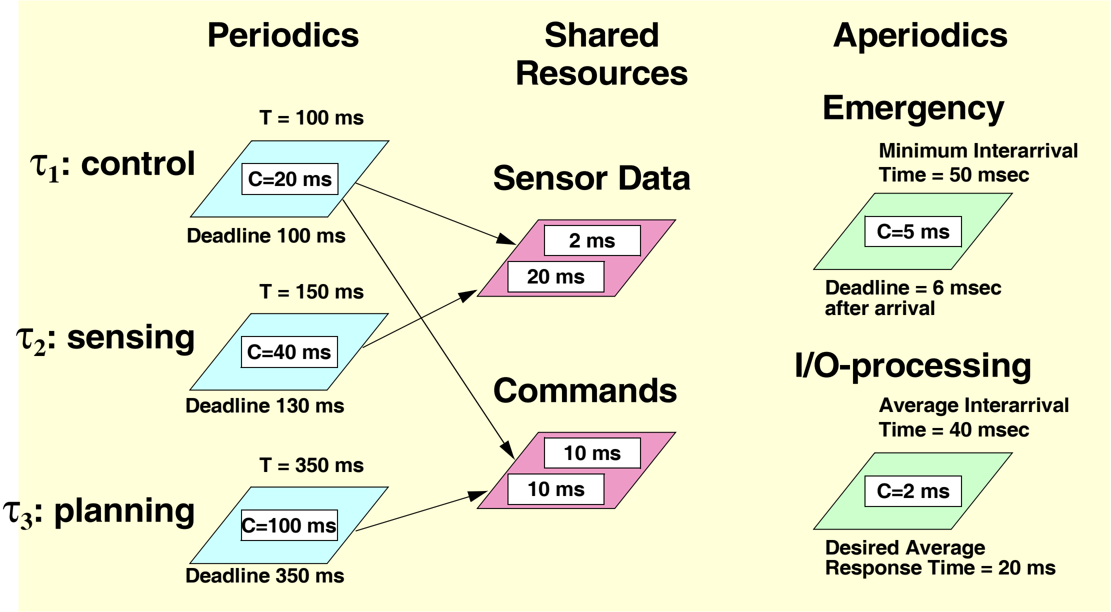
$$\Psi^* = \{\Psi_x \in \Psi | (p-1)T_a - J_a + d_a \le \Psi_x < pT_a - J_a + d_a\}$$

$$A = \Psi_x - [(p-1)T_a - J_a + d_a]$$

Worst-case response time

$$R_a = max[R^A(p)] \qquad \forall p = 1\ldots\left\lceil \frac{L-J_a}{T_a} \right\rceil, \forall A \in \Psi^*$$

Handling aperiodic activities



Periodics

Shared Resources

Aperiodics

τ_1: control

T = 100 ms

C=20 ms

Deadline 100 ms

τ_2: sensing

T = 150 ms

C=40 ms

Deadline 130 ms

τ_3: planning

T = 350 ms

C=100 ms

Deadline 350 ms

Sensor Data

2 ms

20 ms

Commands

10 ms

10 ms

Emergency

Minimum Interarrival Time = 50 msec

C=5 ms

Deadline = 6 msec after arrival

I/O-processing

Average Interarrival Time = 40 msec

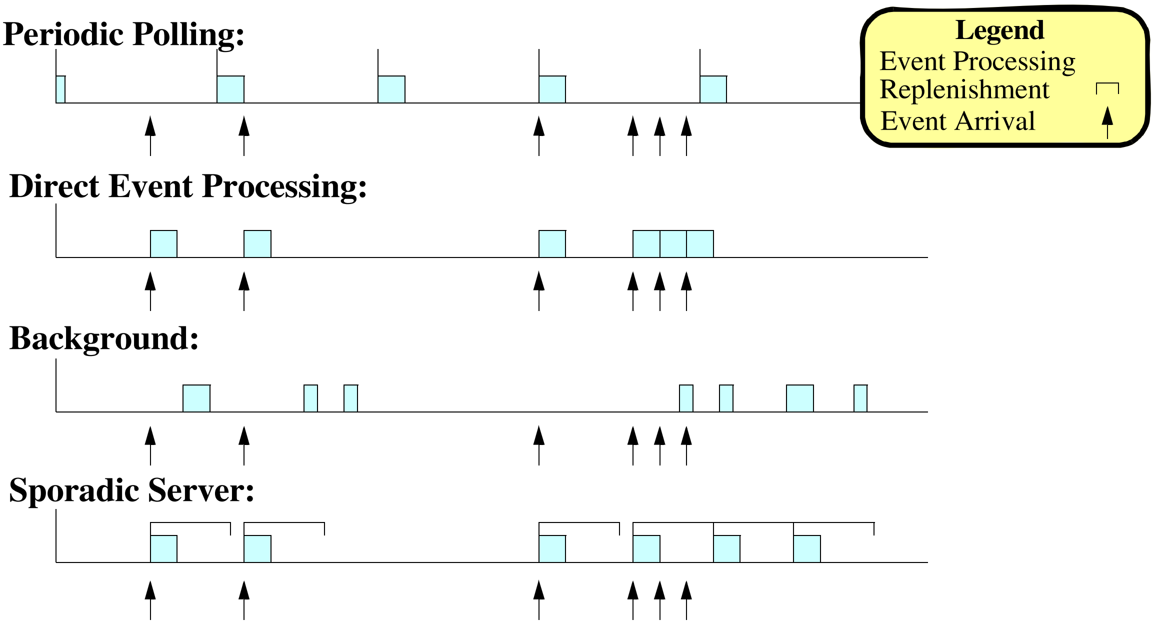
C=2 ms

Desired Average Response Time = 20 ms

Handling aperiodic activities

The main problem is guaranteeing predictability in systems with unbounded aperiodic events

Scheduling aperiodic tasks



Periodic servers

Sporadic server (FPS)

- Characterized by two parameters:
  - initial execution capacity (C_R)
  - replenishment period (T_R)
- It guarantees
  - a minimum bandwidth for aperiodic events
  - bounded preemption on lower priority tasks

Servers also exist for EDF scheduling

- Constant bandwidth servers (CBS)

Constant bandwidth server

The CBS is assigned a service time, Q_s, and a period, T_s. It assigns each job an initial deadline

- If an event arrives at r_k, it is assigned a deadline:

$$d_k = max(r_k, d_{k-1}) + Ts$$

When a task requires more time than what is reserved for it, the absolute deadline is re-evaluated by adding T_s to it

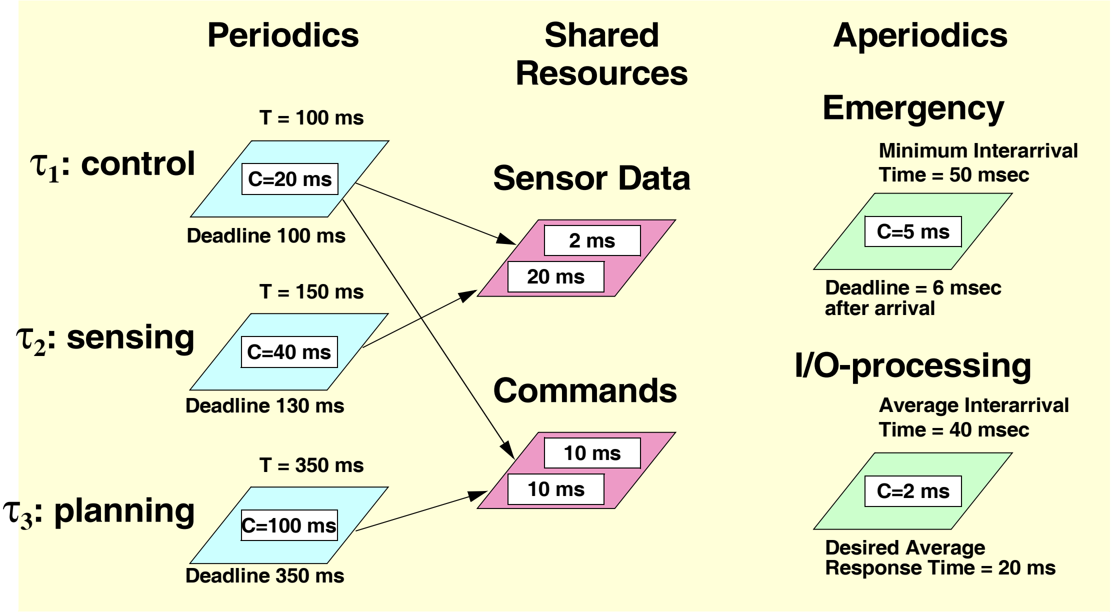
The CBS can be used to bring predictability even if the execution times are unknown or when they exceed the estimated worst-case value

---

Predictable response times in event-driven real-time systems

1. Introduction
2. Basic scheduling
3. ***Mutual exclusion***
4. Distributed systems
5. Hierarchical scheduling
6. Protection and flexible scheduling frameworks
7. Operating systems
8. Programming languages
9. Modeling and integration into the design process
10. Conclusions

Example: synchronization



Periodics

Shared Resources

Aperiodics

τ_1: control
T = 100 ms
C=20 ms
Deadline 100 ms

τ_2: sensing
T = 150 ms
C=40 ms
Deadline 130 ms

τ_3: planning
T = 350 ms
C=100 ms
Deadline 350 ms

Sensor Data
2 ms
20 ms

Commands
10 ms
10 ms

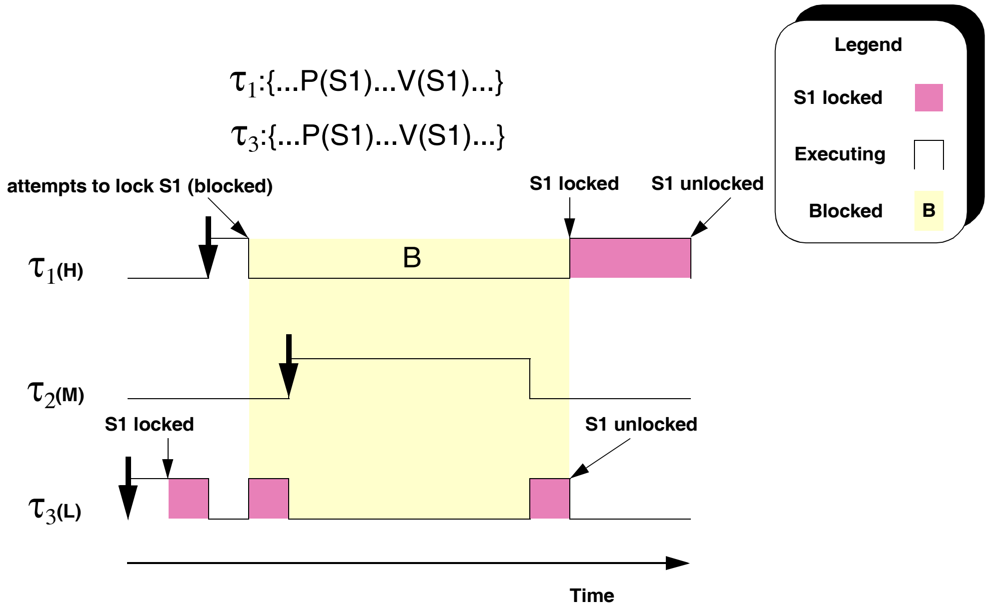
Emergency
Minimum Interarrival Time = 50 msec
C=5 ms
Deadline = 6 msec after arrival

I/O-processing
Average Interarrival Time = 40 msec
C=2 ms
Desired Average Response Time = 20 ms

Unbounded priority inversion

τ_1:{...P(S1)...V(S1)...}

τ_3:{...P(S1)...V(S1)...}



Synchronization protocols

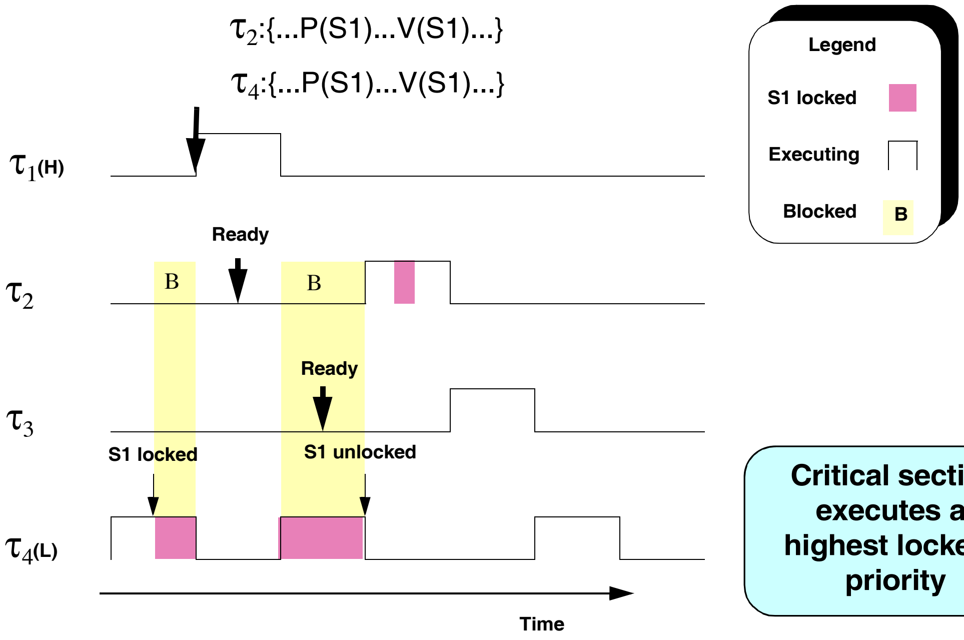
Protocols that prevent unbounded priority inversion:

- ***No preemption***
  - equivalent to raising the priority of the critical section to the highest level
- ***Immediate priority ceiling protocol***
  - raises the priority of the critical section to a value called the "ceiling"
  - lowest blocking time, no extra context switches
  - best for static environments
- ***Priority inheritance***
  - raises the priority only if somebody is waiting
  - best for dynamic environments

Immediate Priority Ceiling (IPC)

τ_2:{...P(S1)...V(S1)...}

τ_4:{...P(S1)...V(S1)...}

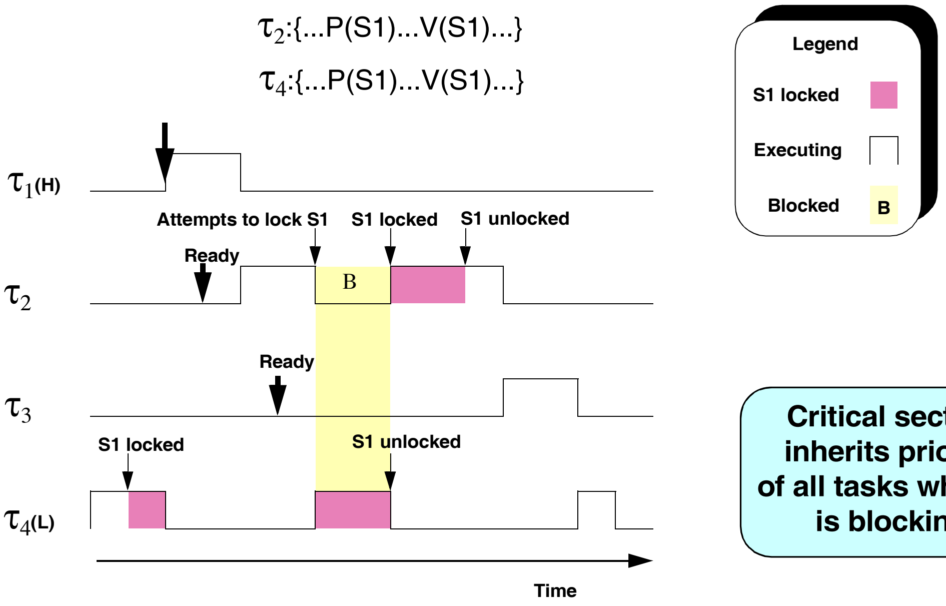


Critical section executes at highest locker's priority

Basic Inheritance Protocol (BIP)

τ_2:{...P(S1)...V(S1)...}

τ_4:{...P(S1)...V(S1)...}



Critical section inherits priority of all tasks which it is blocking

---

Comparison of Synchronization Protocols

Protocol	Non preemption	IPC	BIP
Requires precalculating ceilings	No	Yes	No
Blocked at most once	Yes[2]	Yes[2]	No
Avoids deadlock	Yes[1]	Yes[1]	No
Avoids unnecessary blocking	No	Yes	Yes
Average-case response time	Equal to worst-case	Equal to worst-case	Very low
Worst-case context switch overhead	0	0	$2C_s$ per critical section

Summary of properties

Dynamic **systems (no precalculated ceilings):**

- ***BIP*** **has less blocking than Non-Preemption**

Static **systems (ceilings are precalculated):**

- ***IPC*** **has less overhead than PCP**
- **Critical sections are designed to not suspend themselves**
  - **this is desirable in any case**

Analyzing the Blocking Times

For the no preemption, immediate priority ceiling, or priority ceiling protocols:

- **B_i is the maximum of all the critical sections of lower priority tasks whose ceiling is $\geq P_i$**

For the basic priority inheritance protocol:

- **The critical sections that may cause blocking are those of lower priority tasks whose ceiling is $\geq P_i$; ceiling may be affected by transitive or recursive inheritance.**
- **Only one independent (non overlapped) critical section per semaphore may cause blocking (pick the longest)**
- **Only one independent critical section per lower priority task may cause blocking (pick the longest)**

Synchronization in EDF

Blocking effects similar to unbounded priority inversion exist

Synchronization protocols

- (dynamic) ***priority inheritance***
- ***Baker's protocol***
  - applicable to many scheduling policies
  - immediate priority ceiling is a special case
  - similar properties

SRP Protocol

A preemption level, π_i, is assigned to each task i

- in reverse order relative to the relative deadlines, D_i

Each resource *R* has a preemption ceiling, C_R. For binary resources (with mutual exclusion)

$$C_R = max(\pi_i | \tau_i \text{ uses } R)$$

The dynamic system preemption ceiling is defined as

$$\Pi = max(C_R | R \text{ is locked})$$

SRP Protocol

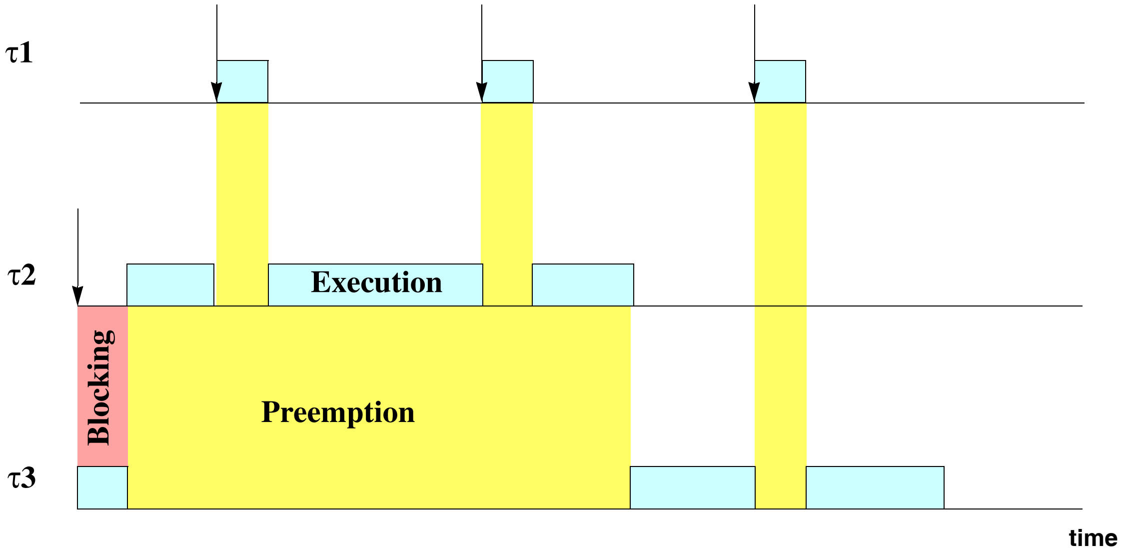
The scheduler is modified to add a new preemption rule:

- a task is only allowed to execute if its job priority is the highest of all the executable tasks, and $\pi_i > \Pi$

The properties of SRP are the same as for the immediate priority ceiling. Assuming no suspension inside critical sections, and a single-processor system:

- if a task blocks, it does it only before starting the execution of its job
- a task can only be blocked once per job
- there are no deadlocks caused by SRP resources

Elements that influence the response time



Lessons learned from basic real-time theory

In fixed-priority systems, the worst-case response time of a task is a function of:

- ***preemption***: time waiting for higher-priority tasks
- ***execution***: time to do its own work
- ***blocking***: time delayed by lower-priority tasks, usually because of mutual exclusion synchronization

Deadlines are missed when utilization is over 100% (seems obvious, but happens in reality):

- Solutions:
  - change the computation requirements
  - change the periods or interarrival times
  - buy a faster CPU

Lessons learned from basic real-time theory (cont'd)

Deadlines are missed because there is too much preemption:

- A task or a task's portion runs at a higher priority than it should
  - this happens typically with interrupt service routines
- There are not enough priorities
- Solutions:
  - preemption can be minimized by choosing an optimum priority assignment
  - sometimes a task or an ISR must be split into parts whose priorities can be independently assigned
  - buy a system with user-defined priorities for I/O drivers
  - buy a system with at least 32 priority levels

Lessons learned from basic real-time theory (cont'd)

Deadlines are missed because there is too much blocking:

- Normal semaphores suffer from unbounded priority inversion
  - this causes horrendous worst-case blocking times
- Solutions
  - disable preemption in the critical sections (may still have too much blocking)
  - priority inheritance protocol
  - priority ceiling or Baker's protocol
  - split long critical sections

Predictable response times in event-driven real-time systems

Real-time networks

Very few networks guarantee real-time response

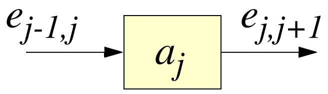
- many protocols allow collisions
- no priorities or deadlines in the protocols

Some solutions

- CAN bus and other field busses
- Priority-based token passing
- Time-division multiplexed access
- Point to point networks

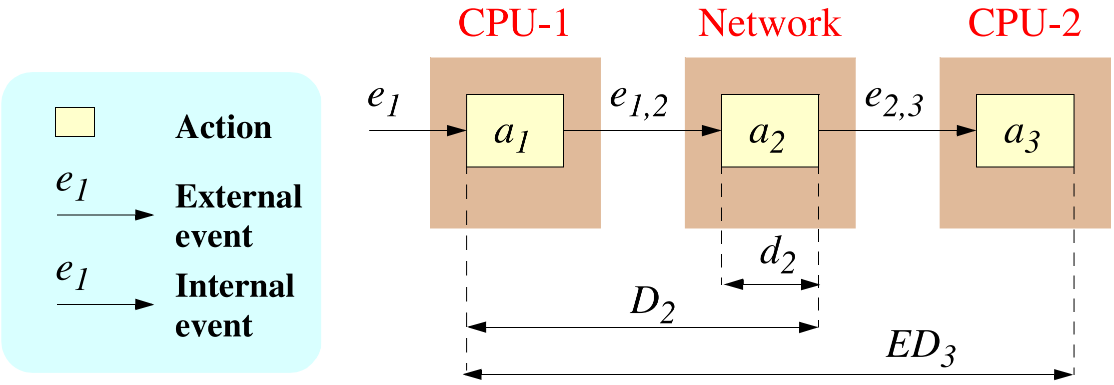
Distributed system model

Linear Action:



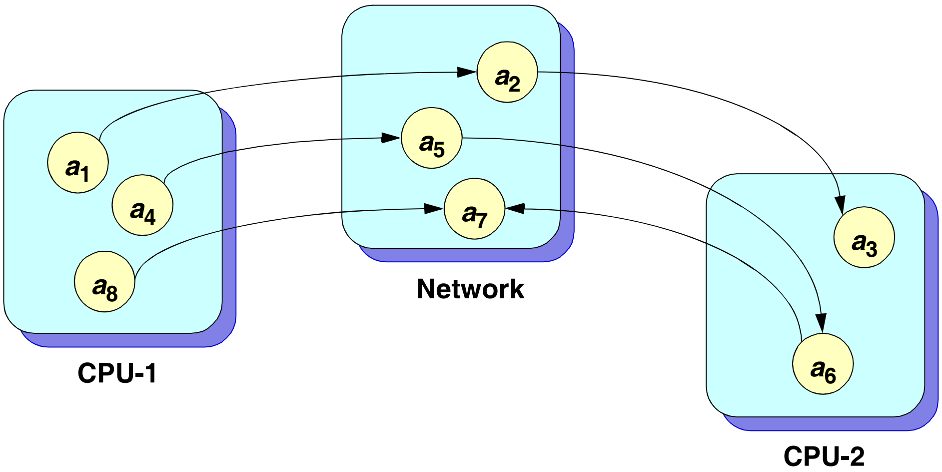
$T_{j-1,j} = T_j = T_{j,j+1}$

Linear Response to an Event:



Release jitter in distributed systems

In the example below, actions a_2,a_3 and a_5,a_6,a_7,a_8 have jitter, even if a_1 and a_4 are purely periodic:



The problem

Release jitter in one resource depends on the response times in other resources

Response times depend on release jitter

Holistic analysis technique

Developed at the University of York

Each resource is analyzed separately (CPU’s and communication networks):

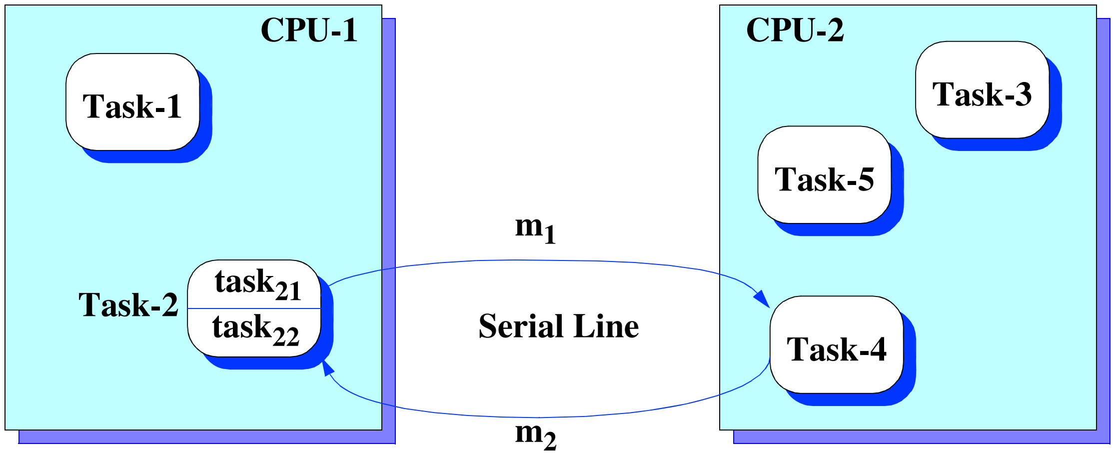
- the analysis is repeated *iteratively*, until a stable solution is achieved
- the method converges because of the *monotonic* relation between jitters and response times

Analysis provides guarantees on schedulability, but is very pessimistic

- independent critical instants may not be possible in practice

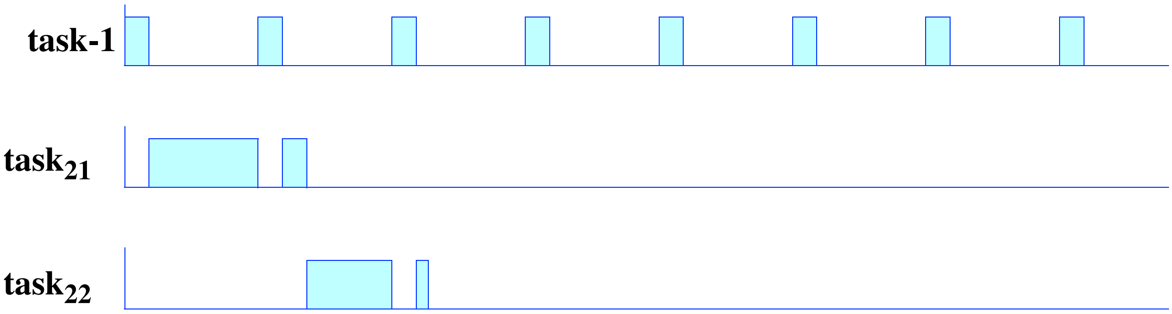
Pessimism in holistic analysis

Holistic analysis technique assumes independent task activations in each resource



Independent task analysis

Execution timeline for $task_{22}$ in previous example:

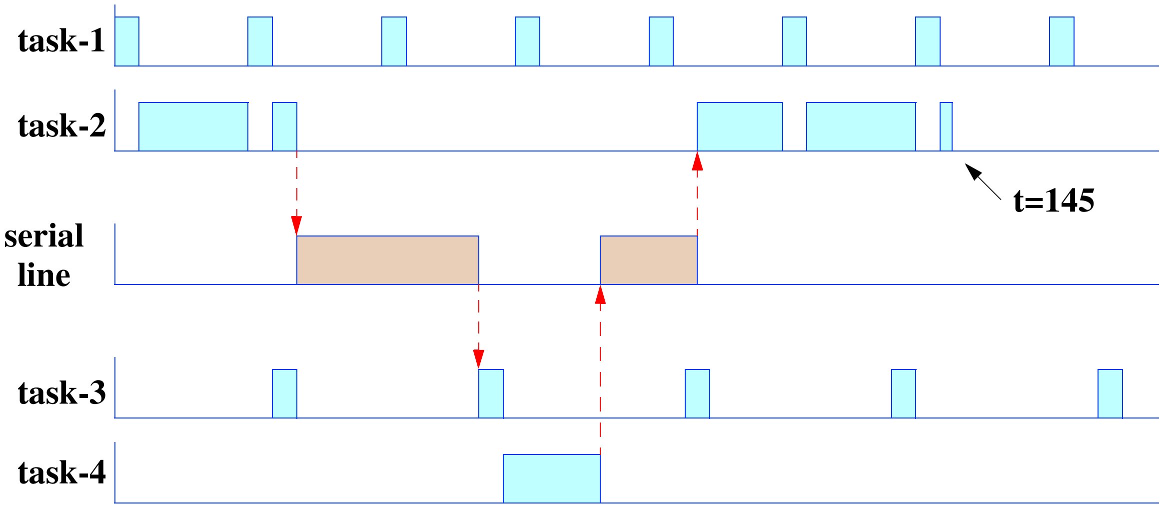


Response time for task-2:

- includes times for $task_{21}$, m_1, task-4, m_2 and $task_{22}$
- total is 270

Using offsets to reduce pessimism

Execution timeline for analysis of task-2:



Offset-based techniques

They reduce the pessimism of the worst-case analysis in multiprocessor and distributed systems:

- by considering offsets in the analysis

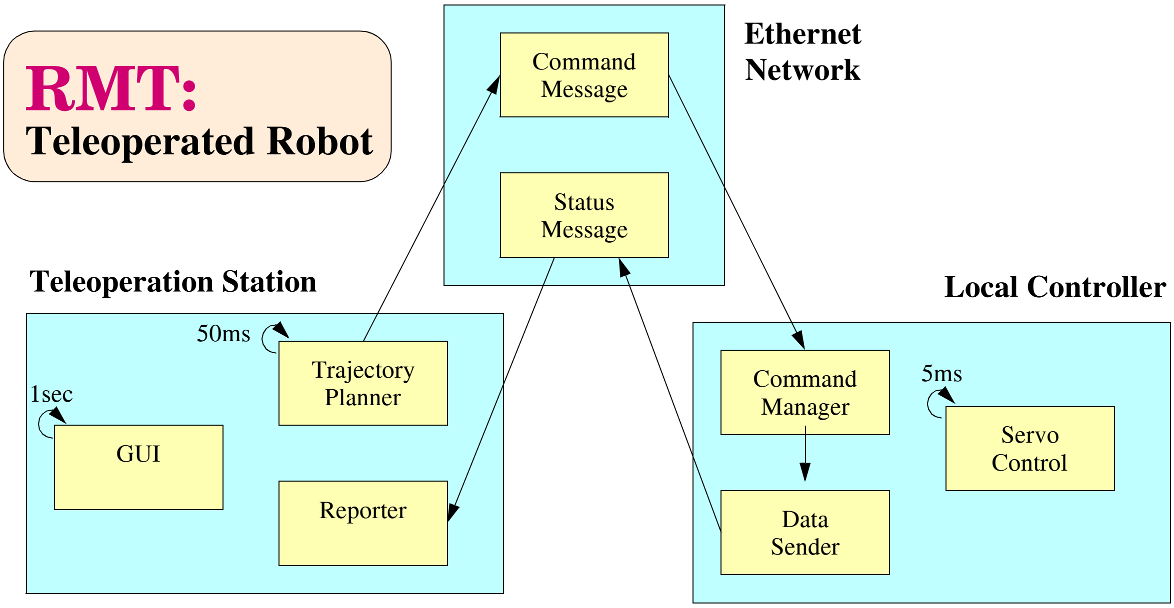
Exact analysis is intractable:

- The task that generates the worst-case contribution of a given transaction is unknown
- The analysis has to check all possible combinations of tasks

Upper bound approximation

- This technique is pessimistic, but polynomial
- In 93% of the tested cases, the response times were exact

Example



RMT:
Teleoperated Robot

Priority assignment techniques

No known optimum priority assignment for distributed systems

Simulated Annealing

- Standard optimization technique

HOPA

- Heuristic algorithm based on successively applying the analysis
- Much faster than simulated annealing
- Usually finds better solutions

None of them guarantees finding the solution

Summary

Kind of Analysis	Deadlines	Number of processors	Fixed Priorities	EDF
`Utilization test`	D=T	1	☑	☑
`Response Time Analysis`	Arbitrary	1	☑	☑
`Holistic Analysis`	Arbitrary	Many	☑	☑
`Offset-Based Analysis`	Arbitrary	Many	☑	☑

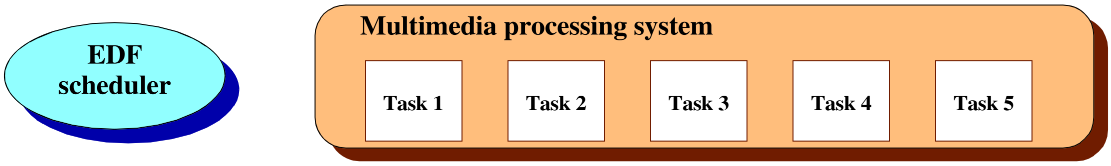
Distributed model also applicable to:

- signal & wait synchronization
- activities that suspend themselves (i.e., delays, I/O, ...)

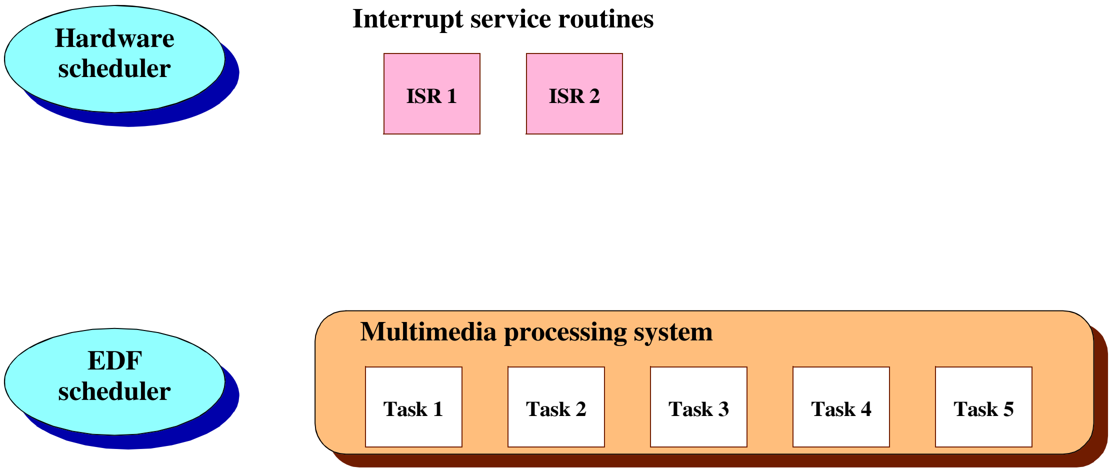
Predictable response times in event-driven real-time systems

1. Introduction
2. Basic scheduling
3. Mutual exclusion
4. Distributed systems
5. *Hierarchical scheduling*
6. Protection and flexible scheduling frameworks
7. Operating systems
8. Programming languages
9. Modeling and integration into the design process
10. Conclusions

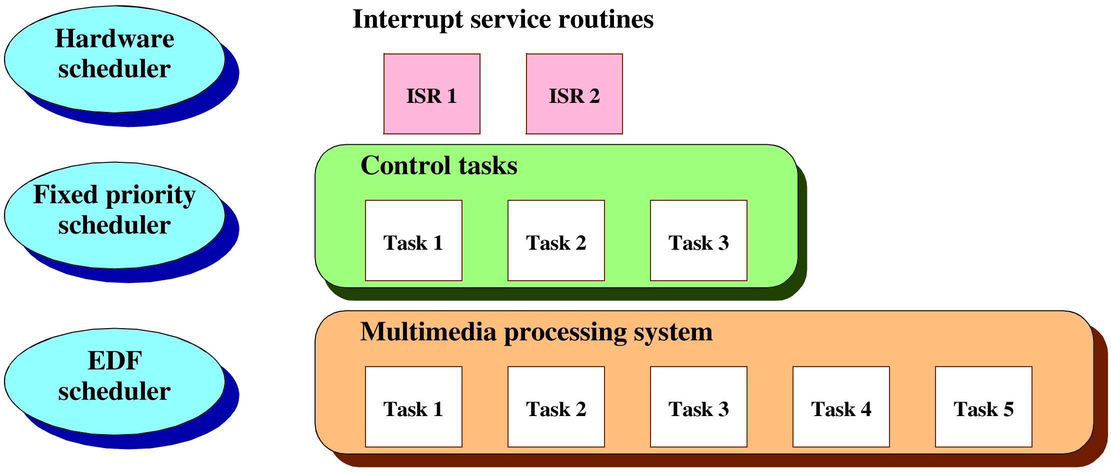
Mixing EDF and fixed priorities



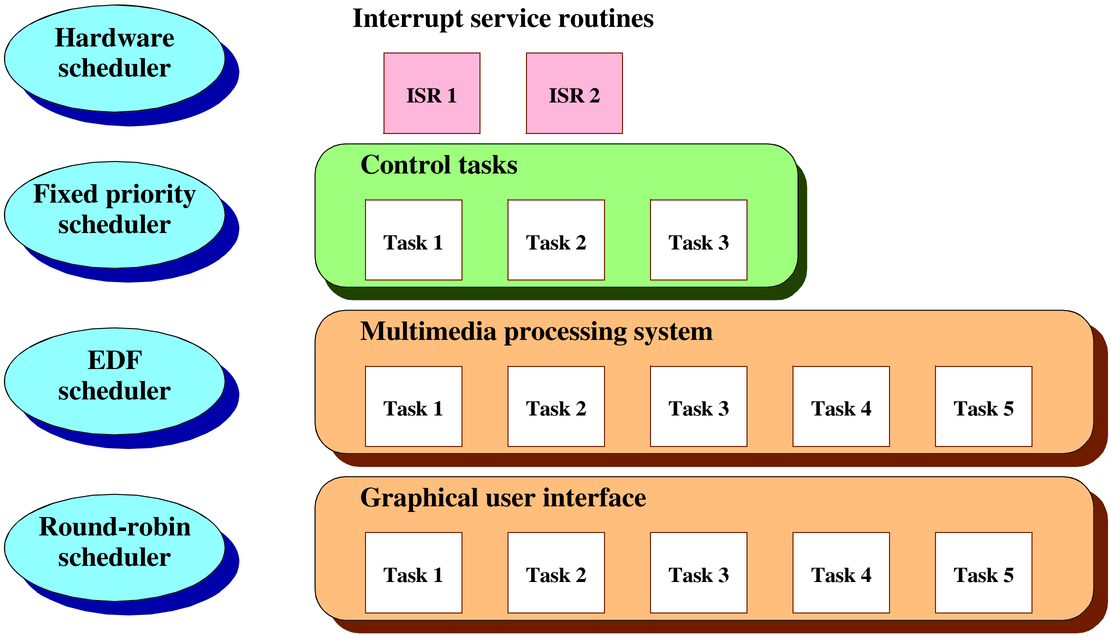
Mixing EDF and fixed priorities



Mixing EDF and fixed priorities



Mixing EDF and fixed priorities



RTA for EDF-within-priorities

Techniques have developed to obtain a worst-case response time analysis in systems with hierarchical schedulers

- underlying fixed priority scheduler
- EDF schedulers running tasks at a given priority level

We call this approach "*EDF within fixed priorities*"

Predictable response times in event-driven real-time systems

Motivation for time protection

Response-time analysis is heavily based on knowing the worst-case execution times (WCET) of each individual task

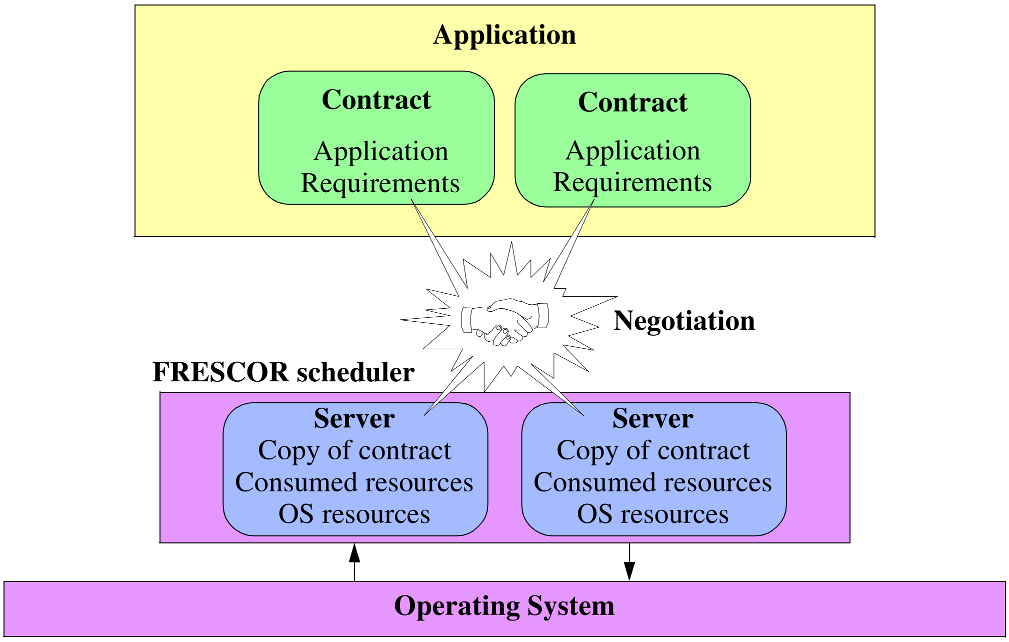
- tools based on precise processor models
- tools based on measurement + probabilistic methods

Precise WCET is very different from average case

Time protection helps by detecting execution-time overruns

Independently developed components sharing resources require time protection

Contract-based scheduling



Application

Contract
Application Requirements

Contract
Application Requirements

Negotiation

FRESCOR scheduler

Server
Copy of contract
Consumed resources
OS resources

Server
Copy of contract
Consumed resources
OS resources

Operating System

The FRESCOR flexible scheduling framework project

A contract specifies:

- the minimum resource usage requirements
- how to make use of spare resources

FRESCOR Increases the level of abstraction for RT services

- automatic admission test
- independence of scheduling algorithm
- dynamic reclamation of unused resources
- coexistence of hard real-time, quality of service

Temporal encapsulation of subsystems

Closes the gap between theory and practice

Example: Periodic task with budget and deadline control

With OS API

```
set priority
create budget signal handler
create deadline signal handler
create budget timer
create deadline timer

while (true) {
   reset deadline timer
   set budget timer
   do useful things
   reset budget timer
   set deadline timer
   wait for next period
}
```

With FRSH API

```
create contract with {C,T}
negotiate the contract
while (true) {
   do useful things
   frsh_timed_wait
}
```

Example: 3-version algorithm

```
create contract with
   {{C1,T},{C2,T},{C3,T}}
negotiate the contract
while (1) {
   if (current_budget<C2) {
         do_version_1
   } else if (current_budget<C3) {
         do_version_2
   } else {
         do_version_3
   }
   frsh_timed_wait
}
```

Predictable response times in event-driven real-time systems

1. Introduction
2. Basic scheduling
3. Mutual exclusion
4. Distributed systems
5. Hierarchical scheduling
6. Protection and flexible scheduling frameworks
7. *Operating systems*
8. Programming languages
9. Modeling and integration into the design process
10. Conclusions

Real-time operating systems

In the past, many real-time systems did not need an operating system

Today, many applications require operating system services such as:

- **concurrent programming, communication networks, file system, etc.**

The timing behavior of a program depends strongly on the operating system's behavior

POSIX definition of real-time in operating systems:
"The ability of the operating system to provide a required level of service in a ***bounded response time*****."**

What is POSIX?

Portable Operating System Interface

Based on UNIX operating systems

The goal is portability of

- applications (at the source code level)
- programmers

Sponsored by the IEEE and The Open Group

Real-Time Features in POSIX

Real-time Features	Support in POSIX
Priority preemptive scheduler	SCHED_FIFO scheduling policy
Sufficient number of priorities	At least 32 priority levels
Priorities modifiable at run-time	Yes
Known context switch times, etc.	Not provided
Synchronization primitives free of unbounded priority inversion	PRIO_INHERIT and PRIO_PROTECT for mutexes
Periodic task activation	Yes
Execution-time budgets	Execution time clocks and timers
Sporadic server	SCHED_SPORADIC scheduling policy
Synchronization primitives free of remote blocking	Immediate priority protocol for mutexes

POSIX real-time profiles (POSIX.13)

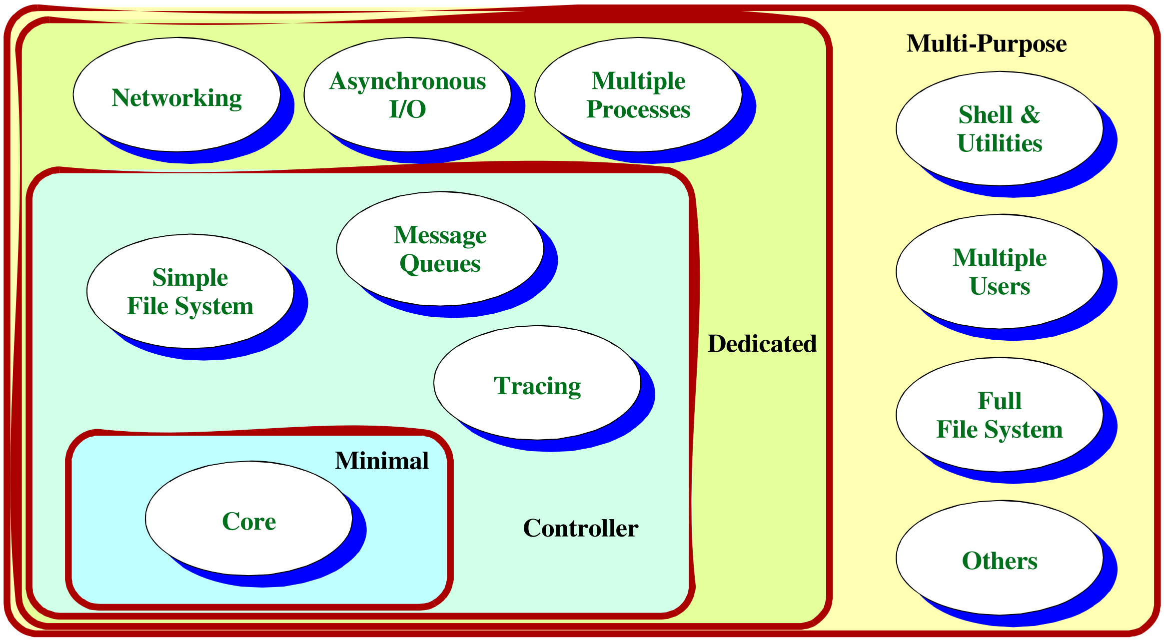
The POSIX Standard:

- Allows writing portable real-time applications
- Very large: inappropriate for embedded real-time systems

POSIX.13:

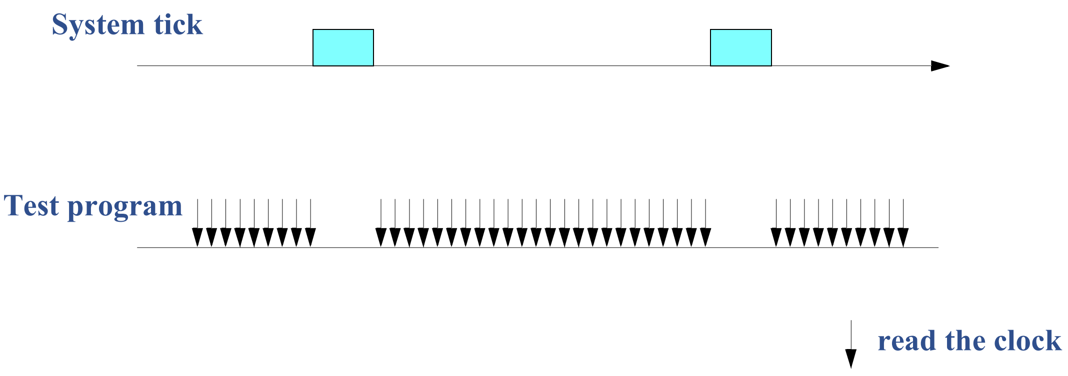
- Defines four real-time system subsets (profiles)
- C and Ada language options

Summary of RT Profiles



System tick and other OS events

Can be measured with a program that constantly reads the clock and records differences large than usual

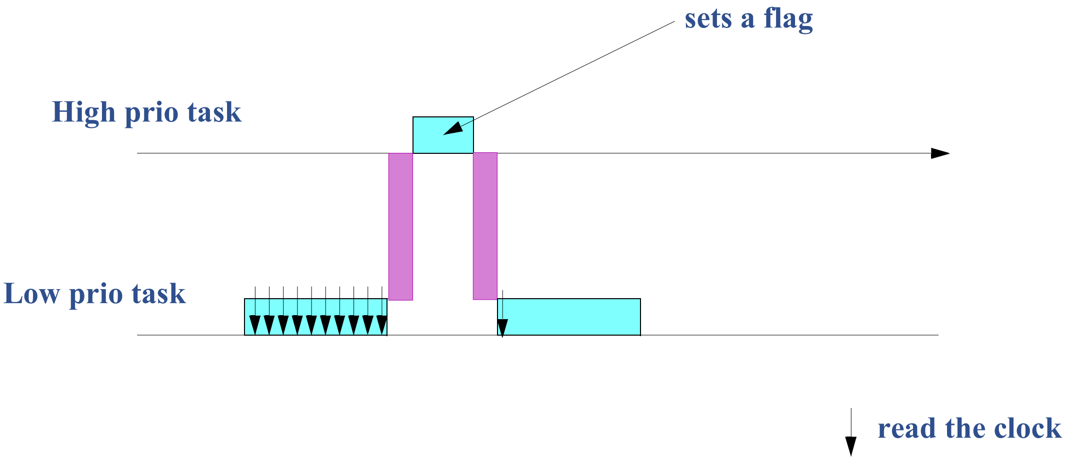


System tick

Test program

read the clock

Measuring context switches



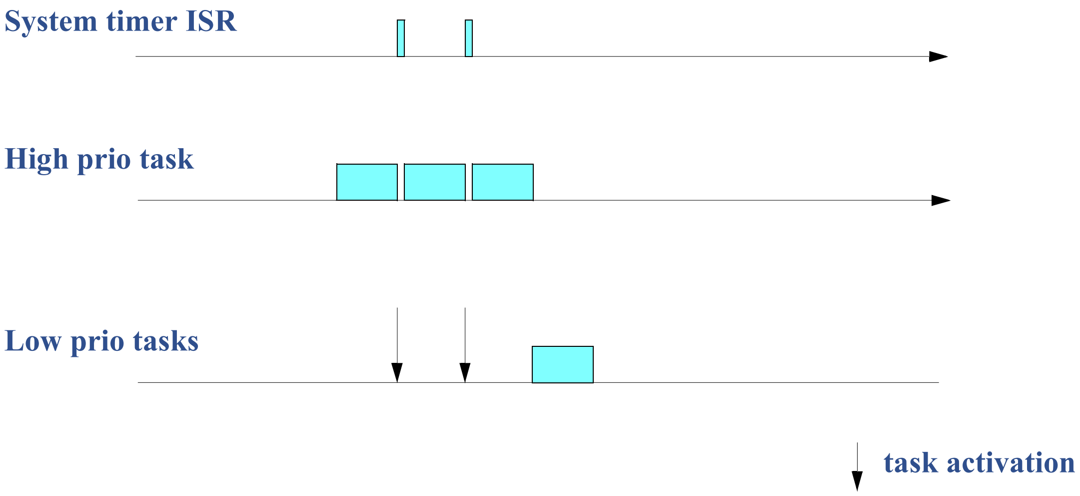
sets a flag

High prio task

Low prio task

read the clock

Delay expirations



System timer ISR

High prio task

Low prio tasks

task activation

Predictable response times in event-driven real-time systems

1. Introduction
2. Basic scheduling
3. Mutual exclusion
4. Distributed systems
5. Hierarchical scheduling
6. Protection and flexible scheduling frameworks
7. Operating systems
8. *Programming languages*
9. Modeling and integration into the design process
10. Conclusions

Programming languages for real-time systems

Some languages do not provide support for concurrency or real-time

- C, C++
- support achieved through OS services (POSIX)

Other languages provide support in the language itself

- Java (concurrency) and RTSJ (real-time)
  - API based
- Ada: concurrency and real-time
  - integrated into the core language

Programming languages for real-time systems

Language support helps in

- increased reliability
- more expressive power
- reduced development and maintenance costs

Ada Real-Time Features

Real-time Features	Support in Ada
Priority preemptive scheduler	Default policy in the RT Annex
Sufficient number of priorities	At least 31 priority levels defined in the Real-Time Annex
Priorities modifiable at run-time	Yes, in the Real-Time Annex
Known context switch times, etc.	Not provided
Synchronization primitives free of unbounded priority inversion	Immediate priority ceiling protocol for protected objects
Periodic task activation	Absolute delay (delay until)
Execution-time budgets	Ada.Execution_Time
Sporadic server	Not provided
Synchronization primitives free of remote blocking	Immediate priority ceiling protocol for protected objects

---

Predictable response times in event-driven real-time systems

1. Introduction
2. Basic scheduling
3. Mutual exclusion
4. Distributed systems
5. Hierarchical scheduling
6. Protection and flexible scheduling frameworks
7. Operating systems
8. Programming languages
9. ***Modeling and integration into the design process***
10. Conclusions

Motivation

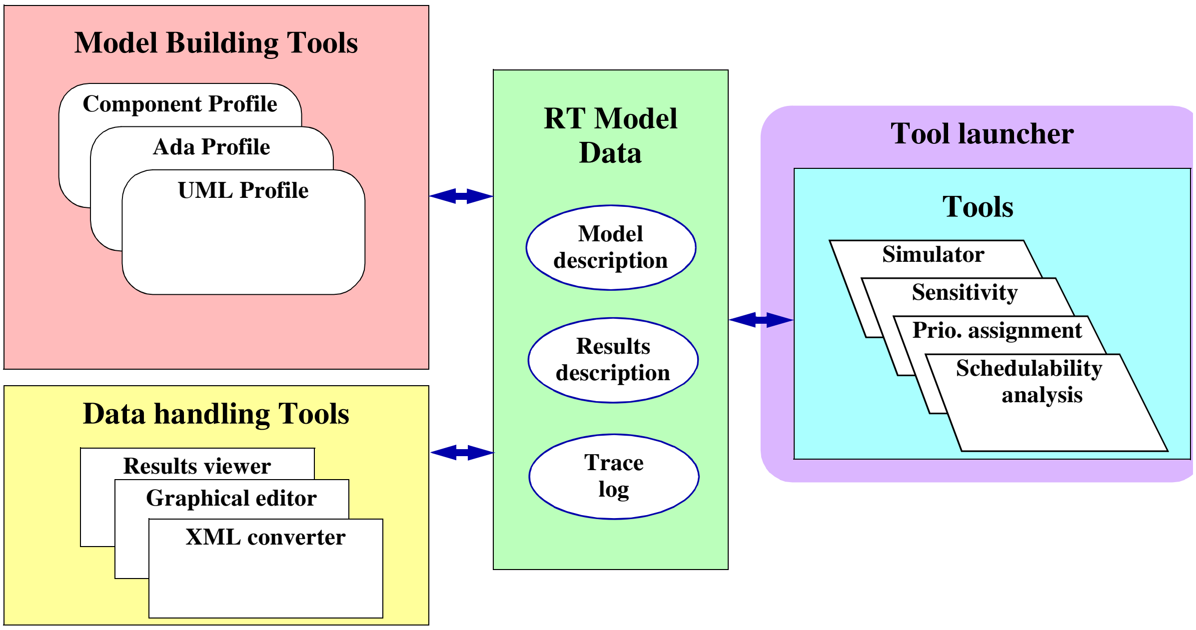
The latest schedulability analysis techniques are difficult to apply by hand

Need to integrate all the techniques in a single tool suite

- schedulability analysis tools
- priority assignment
- sensitivity analysis

A model of the real-time behavior is necessary

MAST environment



Model Building Tools

Component Profile

Ada Profile

UML Profile

Data handling Tools

Results viewer

Graphical editor

XML converter

RT Model Data

Model description

Results description

Trace log

Tool launcher

Tools

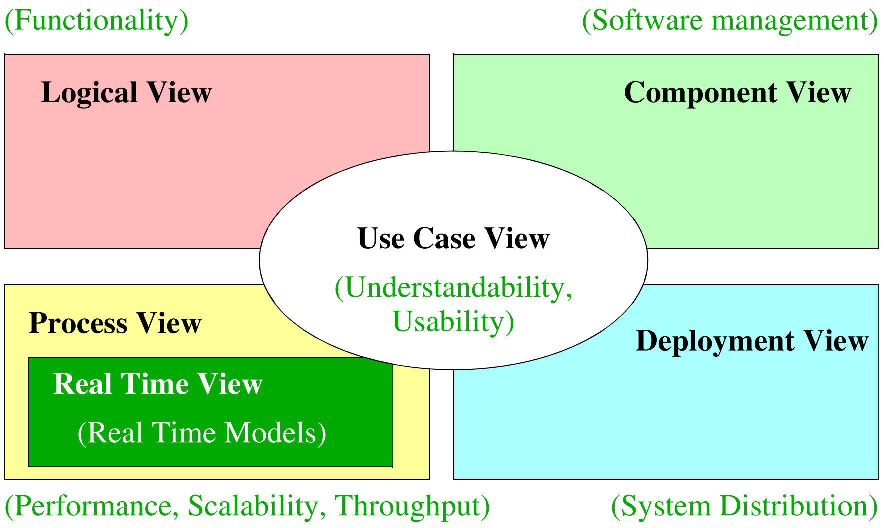
Simulator

Sensitivity

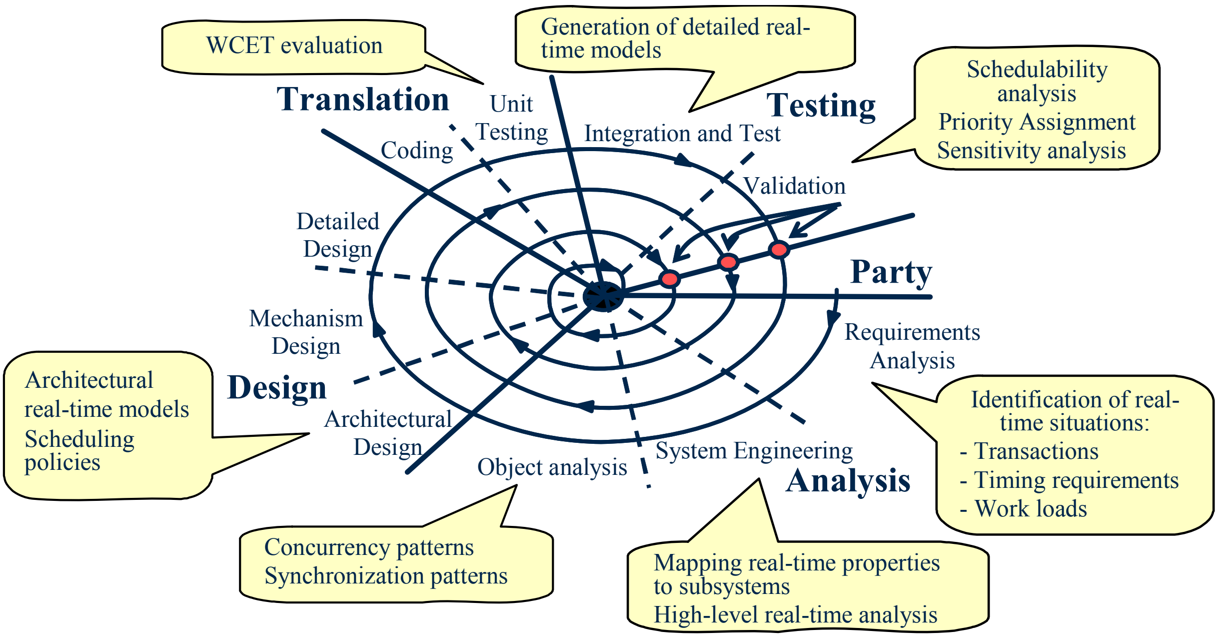
Prio. assignment

Schedulability analysis

The MAST_RT_View in the “4+1 View”



Integration into the design process



WCET evaluation

Generation of detailed real-time models

Schedulability analysis
Priority Assignment
Sensitivity analysis

Architectural real-time models
Scheduling policies

Identification of real-time situations:
- Transactions
- Timing requirements
- Work loads

Concurrency patterns
Synchronization patterns

Mapping real-time properties to subsystems
High-level real-time analysis

Predictable response times in event-driven real-time systems

Conclusions

Priority-based run-time scheduling supports

- predictability of response time, via static analysis
- flexibility and separation of concerns
- dynamic and fixed priorities can be combined together

Real-time theory is now capable of analyzing complex real-time systems

Tools are available to

- automatically analyze a system
- provide sensitivity analysis
- design space exploration

Conclusions (cont'd)

Flexible scheduling frameworks

- integrate hard real-time with quality of service requirements
- increased use of system resources
- higher level of abstraction

Future goals

Integration of schedulability analysis with design tools and methods

- component-based approaches

Improved methods for WCET estimation

New networks and protocols that use real-time scheduling

Integration of real-time response into general-purpose operating systems

- kernel support, including I/O drivers
- flexible scheduling frameworks

URLs

MAST

http://mast.unican.es

FRESCOR

http://frescor.org

MaRTE OS

http://marte.unican.es

References

Books

[1] M.H. Klein, T. Ralya, B. Pollak, R. Obenza, and M. González Harbour. "A practitioner's Handbook for Real-Time Analysis". Kluwer Academic Pub., 1993.

[2] A. Burns and A. Wellings. "Real-Time Systems and Programming Languages". Second Edition. Addison-Wesley, 2002.

[3] G. Buttazzo. "Hard Real-Time Computing Systems". Kluwer Academic Pub., 1997.

[4] Liu J.S.W., "Real Time Systems". Prentice Hall, 2000.

Real-Time analysis in single processor systems

[5] J.P. Lehoczky, L. Sha, J.K. Strosnider, and H. Tokuda. "Fixed-Priority Scheduling for Hard Real-Time Systems". In *Foundations of Real-Time Computing: Scheduling and Resource Management*, Van Tilborg, Andre and Koob, Gary M., Ed., Kluwer Academic Publishers, 1991, pp. 1-30.

References (continued)

[6] L. Sha, M.H. Klein, and J.B. Goodenough. “Rate Monotonic Analysis for Real-Time Systems”. In *Foundations of Real-Time Computing: Scheduling and Resource Management*, Van Tilborg, Andre and Koob, Gary M., Ed., Kluwer Academic Publishers, 1991, pp. 129-155.

[7] M.H. Klein, J.P. Lehoczky, and R. Rajkumar. “Rate-Monotonic Analysis for Real-Time Industrial Computing”. IEEE Computer, Vol. 27, No. 1, January 1994.

[8] J.A. Stankovic. “Misconceptions About Real-Time Computing”. IEEE Computer, Vol. 21, No. 10, October 1988.

[9] J.A. Stankovic and K. Ramamritham “What is Predictability for Real-Time Systems?”. Real-Time Systems Journal, Vol. 2, 247-254, 1990.

[10] T.P. Baker, A. Shaw. “The Cyclic executive Model and Ada”. *Proceedings of the IEEE Real-Time Systems Symposium*, December 1988.

References (continued)

[11] C.L. Liu, and J.W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment”. Journal of the ACM, 20 (1), pp 46-61, 1973.

[12] P.K. Harter, "Response times in level-structured systems". ACM Transactions on Computer Systems, vol. 5, no. 3, pp. 232-248, Aug. 1984.

[13] M. Joseph and P. Pandya, "Finding Response Times in a Real-Time System". The Computer Journal (British Computing Society) 29, 5, pp. 390-395, October 1986.

[14] J. Leung, and J.W. Layland. “On Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks”. *Performance Evaluation 2*, 237-50, 1982.

[15] J.P. Lehoczky, L. Sha, and Y. Ding. “The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior”. *Proceedings of the IEEE Real-Time Systems Symposium*, 1989.

[16] J.P. Lehoczky. “Fixed Priority Scheduling of Periodic Tasks Sets with Arbitrary Deadlines”. *IEEE Real-Time Systems Symposium*, 1990.

References (continued)

[17] M. González Harbour, M.H. Klein, and J.P. Lehoczky. “Fixed Priority Scheduling of Periodic Tasks with Varying Execution Priority”. *Proceedings of the IEEE Real-Time Systems Symposium*, December 1991, pp. 116-128.

[18] M. González Harbour, M.H. Klein, and J.P. Lehoczky. “Timing Analysis for Fixed-Priority Scheduling of Hard Real-Time Systems”. IEEE Trans. on Software Engineering, Vol. 20, No.1, January 1994.

[19] K. Tindell, “An Extendible Approach for Analysing Fixed Priority Hard Real-Time Tasks”. Journal of Real-Time Systems, Vol. 6, No. 2, March 1994.

[20] M. Spuri. "Analysis of Deadline Scheduled Real-Time Systems". RR-2772, INRIA, France, 1996.

[21] M. González Harbour and J.C. Palencia, “Response Time Analysis for Tasks Scheduled under EDF within Fixed Priorities”, Proceedings of the 24th IEEE Real-Time Systems Symposium, Cancun, México, December, 2003.

References (continued)

[22] M.H. Klein, and T. Ralya. “An Analysis of Input/Output Paradigms for Real-Time Systems”. Technical Report CMU/SEI-90-TR-19, Software Engineering Institute, July 1990.

[23] Audsley, N.C., “Optimal Priority Assignment and Feasibility of Static Priority Tasks with Arbitrary Start Times”, Dept. Computer Science, University of York, December 1991.

Aperiodic Scheduling

[24] B. Sprunt, L. Sha, and J.P. Lehoczky. “Aperiodic Task Scheduling for Hard Real-Time Systems”. *The Journal of Real-Time Systems*, Vol. 1, 1989, pp. 27-60.

[25] M. González Harbour, J.J. Gutiérrez García, J.C. Palencia Gutiérrez. “Implementing Application-Level sporadic Server Schedulers in Ada 95”. International Conference on Reliable Software Technologies: Ada-Europe'97. London, UK, 1997.

[26] L. Abeni and G. Buttazzo. “Integrating Multimedia Applications in Hard Real-Time Systems”. Proceedings of the IEEE Real-Time Systems Symposium, Madrid, Spain, December 1998

References (continued)

Distributed Real-Time Systems

[27] L. Sha, and S.S Sathaye. “A Systematic Approach to Designing Distributed Real-Time Systems”. IEEE Computer, Vol 26., No. 9, September 1993.

[28] K.W. Tindell, A. Burns, and A.J. Wellings. “Allocating Real-Time Tasks. An NP-Hard Problem Made Easy”. *Real-Time Systems Journal*, Vol. 4, No. 2, May 1992. pp. 145- 166.

[29] K.W. Tindell, and J. Clark. "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems". Microprocessing and Microprogramming, Vol. 50, Nos. 2-3, pp 117-134, April 1994.

[30] M. Spuri. "Holistic Analysis of Deadline Scheduled Real-Time Distributed Systems". RR-2873, INRIA, France, 1996.

[31] J.C. Palencia Gutiérrez, J.J. Gutiérrez García, M. González Harbour. “On the schedulability analysis for distributed hard real-time systems”. 9th Euromicro Workshop on Real-Time Systems. Toledo, 1997.

References (continued)

[32] Gutiérrez García J.J. and González Harbour M.: "Optimized Priority Assignment for Tasks and Messages in Distributed Real-Time Systems". Proceedings of the 3rd Workshop on Parallel and Distributed Real-Time Systems, Santa Barbara, California, pp. 124-132, April 1995.

[33] Palencia Gutiérrez J.C., Gutiérrez García J.J., González Harbour M. : "Best-Case Analysis for Improving the Worst-Case Schedulability Test for Distributed Hard Real-Time Systems". Proceeding of the 10th IEEE Euromicro Workshop on Real-Time Systems, Berlin, Germany, June 1998.

[34] K. Tindell. "Adding Time-Offsets to Schedulability Analysis", Technical Report YCS 221, Dept. of Computer Science, University of York, England, January 1994.

[35] Palencia Gutiérrez J.C., González Harbour M. : "Schedulability Analysis for Tasks with Static and Dynamic Offsets". Proceedings of the 18th. IEEE Real-Time Systems Symposium, Madrid, Spain, December 1998.

References (continued)

[36] J.C. Palencia, and M. González Harbour, "Exploiting Precedence Relations in the Schedulability Analysis of Distributed Real-Time Systems". Proceedings of the 20th IEEE Real-Time Systems Symposium, 1999.

[37] J.J. Gutiérrez García, J.C. Palencia Gutiérrez, and M. González Harbour, "Schedulability Analysis of Distributed Hard Real-Time Systems with Multiple-Event Synchronization". Euromicro Conference on Real-Time Systems, Stockholm, Sweden, 2000.

[38] J.C. Palencia and M. González Harbour, "Offset-Based Response Time Analysis of Distributed Systems Scheduled under EDF". Euromicro conference on real-time systems, Porto, Portugal, June 2003.

[39] Redell, O. Response Time Analysis for Implementation of Distributed Control Systems, Doctoral Thesis, TRITA-MMK 2003:17, ISSN 1400-1179, ISRN KTH/MMK/R--03/17--SE, 2003

References (continued)

Synchronization

[40] L. Sha, R. Rajkumar, and J.P. Lehoczky. "Priority Inheritance Protocols: An approach to Real-Time Synchronization". IEEE Trans. on Computers, September 1990.

[41] Baker T.P., "Stack-Based Scheduling of Realtime Processes", Journal of Real-Time Systems, Volume 3, Issue 1 (March 1991), pp. 67–99.

[42] J.B. Goodenough, and L. Sha. "The Priority Ceiling Protocol: A Method for Minimizing the Blocking of High Priority Ada Tasks". *Proceedings of the 2nd International Workshop on Real-Time Ada Issues*, June 1988.

[43] R. Rajkumar, L. Sha, and J.P. Lehoczky. "Real-Time Synchronization Protocols for Multiprocessors". *IEEE Real-Time Systems Symposium*, December 1988.

[44] R. Rajkumar. "Real-Time Synchronization Protocols for Shared Memory Multiprocessors". *Proceedings of the 10th International Conference on Distributed Computing*, 1990.

[45] L. Sha, R. Rajkumar, J.P. Lehoczky. "Real-Time Computing with IEEE Futurebus+". *IEEE Micro*, June 1991, pp. 30-33, and 95-100.

References (continued)

Priority assignment

[46] J. Leung, and J.W. Layland. “On Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks”. Performance Evaluation 2, 237-50, 1982.

[47] N.C. Audsley, “Optimal Priority Assignment and Feasibility of Static Priority Tasks with Arbitrary Start Times”, Dept. Computer Science, University of York, December 1991.

[48] K.W. Tindell, A. Burns, and A.J. Wellings. “Allocating Real-Time Tasks. An NP-Hard Problem Made Easy”. Real-Time Systems Journal, Vol. 4, No. 2, May 1992. pp. 145- 166.

[49] J.J. Gutiérrez García and M. González Harbour. “Optimized Priority Assignment for Tasks and Messages in Distributed Real-Time Systems”. Proceedings of 3rd Workshop on Parallel and Distributed Real-Time Systems, Santa Barbara, California, pp. 124-132, 1995.

References (continued)

Initial work on RMA

[50] C.L. Liu and J.W. Layland. “Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment”. Journal of the ACM, Vol. 20, No. 1, 1973, pp. 46-61.

[51] M. Joseph and P. Pandya, “Finding Response Times in a Real-Time System,” *BCS Computer Journal*, Vol. 29, no. 5, October 1986, pp 390-395.

[52] B.W. Lampson, and D.D. Redell. “Experience with Processes and Monitors in Mesa”. *Communications of the ACM* 23, 2, February 1980, pp. 105-107.

References (continued)

Real-Time operating systems and networks

[53] K.Tindell, A. Burns and A. Wellings. “Calculating Controller Area Network (CAN) message response times”. Proc. of the 1994 IFAC Workshop on Distributed Computer Control Systems (DCCS), Toledo, Spain.

[54] M. Aldea and M. González. “MaRTE OS: An Ada Kernel for Real-Time Embedded Applications”. Proceedings of the International Conference on Reliable Software Technologies, Ada-Europe-2001, Leuven, Belgium, Lecture Notes in Computer Science, LNCS 2043, May, 2001.

[55] Juan López Campos, J. Javier Gutiérrez and M. González Harbour. “CAN-RT-TOP: Real-Time Task-Oriented Protocol over CAN for Analyzable Distributed Applications”. 3rd International Workshop on Real-Time Networks (RTN), Catania, Sicily (Italy), July 2004.

[56] José María Martínez and Michael González Harbour. “RT-EP: A Fixed-Priority Real Time Communication Protocol over Standard Ethernet”. to appear in the Proceedings of the International Conference on Reliable Software Technologies, Ada-Europe-2005, York, UK, June 2005.

References (continued)

[57] ISO/IEC 9945-1:2003. Standard for Information Technology -Portable Operating System Interface (POSIX).

[58] IEEE Std. 1003.13-2003. Information Technology -Standardized Application Environment Profile- POSIX Realtime and Embedded Application Support (AEP). The Institute of Electrical and Electronics Engineers.

[59] CAN Specification Version 2.0. 1991, Robert Bosch GmbH, Postfatch 30 02 40, D-70442 Stuttgart.

[60] MaRTE OS home page. `http://marte.unican.es/`

[61] SHaRK home page. `http://shark.sssup.it/`

References (continued)

Ada

[62] Ada Language Reference Manual, ANSI, 1983.

[63] International Standard ISO/IEC 8652:1995, “Information Technology, Programming Languages, Ada Reference Manual”, January 1995

References (continued)

POSIX: Standard Operating System Interface

[64] M. González Harbour y C..D. Locke. “Tostadores y POSIX”. Novática, Vol. 129, Octubre 1997.

[65] B.O. Gallmeister, and C. Lanier. “Early Experience with POSIX 1003.4 and POSIX 1003.4a”. *Proceedings of the IEEE Real-Time Systems Symposium*, December 1991, pp. 190-198.

[66] B.O. Gallmeister. “POSIX.4: Programming for the Real World”. O´Reilly $ Associates, Inc., 1995.

[67] B. Nichols, D. Buttlar and J. Proulx Farrell. “Pthreads Programming”. O’Reilly & associates, Inc., 1996

[68] S. Kleiman, et al. , “Programing with Threads”. Prentice Hall, 1996.

References (continued)

[69] ISO/IEC Standard 9945-1:1996, *"Information Technology -Portable Operating System Interface (POSIX)- Part 1: System Application Program Interface (API) [C Language]"*. Institute of Electrical and electronic Engineers, 1996.

[70] IEEE Standard 1003.1d:1999, "*Standard for Information Technology -Portable Operating System Interface (POSIX)- Part 1: System Application Program Interface (API) [C Language]. Realtime Extension*". The Institute of Electrical and Electronics Engineers, 1999.

[71] IEEE Standards Project P1003.1j, "*Draft Standard for Information Technology -Portable Operating System Interface (POSIX)- Part 1: System Application Program Interface (API) [C Language]. Advanced Realtime Extension*". Draft 9. The Institute of Electrical and Electronics Engineers, October 1999

[72] IEEE Standard 1003.13:1998, "*Standard for Information Technology -Standardized Application Environment Profile- POSIX Realtime Application Support (AEP)*". The Institute of Electrical and Electronics Engineers, 1998

References (concluded)

MAST

[73] J.M. Drake, M. González Harbour J.L. Medina: "MAST Real-Time View: Graphic UML tool for modeling object-oriented real time systems." Group of Computers and Real-Time Systems. University of Cantabria (Internal report), 2000.

[74] M. González Harbour, J.J. Gutiérrez, J.C. Palencia and J.M. Drake. "MAST: Modeling and Analysis Suite for Real-Time Applications". Proceedings of the Euromicro Conference on Real-Time Systems, Delft, The Netherlands, June 2001.

[75] MAST home page. http://mast.unican.es

FRESCOR project

[76] FRESCOR home page. http://frescor.org

Acknowledgements

Part of the material in this tutorial is extracted from the tutorial on Rate Monotonic Analysis by the *Software Engineering Institute* of *Carnegie Mellon University*