REAL-TIME SYSTEMS
Scheduling

Prof. J.-D. Decotignie
CSEM Centre Suisse d'Electronique et de Microtechnique SA
Jaquet-Droz 1, 2007 Neuchâtel
jean-dominique.decotignie@csem.ch

Outline

- Introduction
- Task taxonomy and Scheduling
- Fixed priority scheduling
- Earliest deadline first scheduling
- Resource sharing
- Sporadic tasks
- Jitter

Outline (cont.)

- Precedence constraints
- Overloads
- Effects of limited priority levels
- Constant bandwidth scheduling
- Non preemptive scheduling

Real-Time Systems

- Strict constraints (hard real-time)
- Soft constraints (soft real-time)

- A number of parallel operations (activities)
  - ☒ which task should be executed, at which instant in order to fulfill the requirements ?
    - Execution time and order
    - Number of processors

Notion of Complexity [Gar79]

- a problem Π is a set of couples (A, I)
  A is an instance and I is a response
- Optimization problem

  Instance: a finite set of elements E={e1, e2, … en}, each one is given a weight p(ei) and a value v(ei). The knapsack has a capacity c.
  Response: subset E'⊆E for which total weight of elements less than c and the total value is maximal.
- Decision problem

Algorithm

- An algorithm is a general procedure that is used step by step to solve a problem
  an algorithm is said to solve a problem Π if, applied to an instance I of Π, it is guaranteed that a solution will be found
- How to compare algorithms ?

Algorithm Complexity

- A function f(n) is O(g(n)) when there exists a constant c such that $|f(n)| \leq c.|g(n)|$
- A polynomial time algorithm is an algorithm whose complexity function is O(g(n)) for a polynomial function g
- Otherwise, it is said solvable in exponential time

Algorithm Complexity (2)

Function	10	20	30	40
n	0.00001	0.00002	0.00003	0.00004
n^2	0.0001	0.0004	0.0009	0.0016
N^5	0.1	3.2 s	24.3 s	1.7 min
2^n	0.001 s	1 s	17.9 min	12.7 day
3^n	0.059 s	58 min.	6.5 y	3855 cen

Problem Classification

- P class
  - A solution can be found in polynomial time
- NP class
  - A solution can be shown correct in polynomial time
- NP-hard class
  - Every NP class problem is a special case
- NP-complete class
  - Intersection of NP and NP-hard

Task Taxonomy

- Continuous permanent tasks $PM_i\left(T_i, C_i^{av}, C_i^{\min}, D_i = T_i, r_i = 0\right)$
- Cyclic tasks $TC_i\left(C_i, T_i^{av}, T_i^{\max}, D_i = T_i, r_i = 0\right)$
- Periodic tasks $P_i\left(T_i, C_i, D_i, r_i\right)$
- Sporadic tasks $S_i\left(T_i, C_i, D_i, r_i = 0\right)$
- Aperiodic tasks
  - Only soft deadlines

Tasks

- Set of real-time tasks
  - Each task consists of an infinite or finite stream of jobs
- Task i parameters
  - Max. computation time: C_i
  - Relative deadline: D_i
  - Period / Minimum interarrival time: T_i
- Job parameter
  - Absolute deadline of a job of task i: d_i

Task Inter Relationship

- Precedence
- Mutual synchronization
- Exclusion
- Resource sharing
- And any combination of above

Scheduling Activities

- Configuration
- Distribution
  - Start and switch tasks
  - Preemptive or not
- A priori analysis
  - Is a given task set schedulable under the given constraints
- Static or dynamic

Deterministic or Predictable

- Deterministic
  - Sequences and their occurring times are known in advance
- Predictable
  - We can predict that the system will behave according to some properties

Scheduling Algorithms

- Fixed priority
- Directly based upon task characteristics
- All based on the notion of worst case execution time C
  - Difficult to evaluate
  - Often too pessimistic

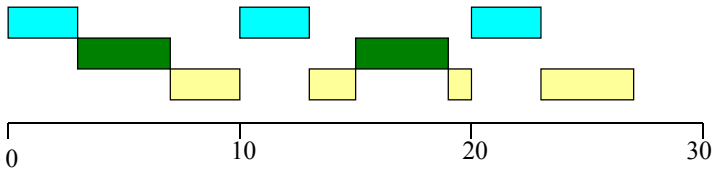
Rate Monotonic (RM)

- Assumptions [Liu73]
  - Periodic tasks (task i period is T_i)
  - Task deadlines D_i are at the end of their period
  - Tasks are preemptive
  - Task cannot block or suspend themselves
  - The execution time C_i of a task i is known and constant

Rate Monotonic (2)

- The shorter the period the higher the priority
- All priorities are different

Task	T	D	C
A	10	10	3
B	15	15	4
C	30	30	10



Rate Monotonic - Guaranty Function [Liu73]

- Necessary condition

$$\sum_{i=1}^{n}\left(\frac{C_i}{T_i}\right) \le 1$$

- Sufficient condition

$$\sum_{i=1}^{n}\left(\frac{C_i}{T_i}\right) \le n\left(2^{1/n}-1\right)$$

  - n=1 → 100%; n=2 → 83%; n=3 → 78%; large n → 69%;

Rate Monotonic - Exact Analysis

- Due to Joseph and Pandya [Jos86]
- They introduce the idea of worst case response time

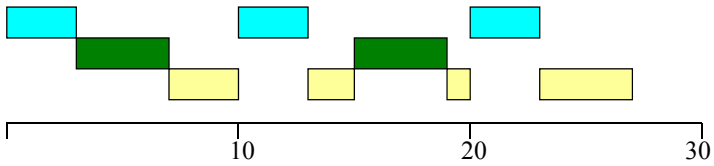
$$R_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

- Iterative calculation

$$R_i^{n+1} = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j$$

Rate Monotonic - Exact Analysis (2)

Task	T	D	C	R^0	R^1	R^2	R^3..R^n
A	10	10	3	3	3	3	3
B	15	15	4	4	7	7	7
C	30	30	10	10	17	24	27



Rate Monotonic - Evaluation

- Advantages
  - Simple
  - Suited to existing operating systems
  - May be used to assign interrupt levels
- Drawbacks
  - Restrictive assumptions
  - Only for preemptive tasks
  - Not suited when deadlines are shorter than period
  - Cannot handle exclusion

Rate Monotonic - Evaluation (2)

- some assumptions may be relaxed
  - Periodicity
  - Constant execution time
  - Deadline at end of period

Deadline Monotonic(DM)

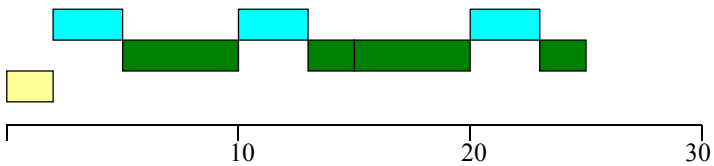
Jackson rule [Jac55]: « Any sequence is optimal that puts the jobs in order of non decreasing due dates »

- Assumptions [Aud90]
  - Periodic tasks (task i period is T_i)
  - Task deadlines D_i are arbitrary ($\leq T_i$)
  - Tasks are preemptive
  - Task cannot block or suspend themselves
  - maximum execution time C_i of a task i is known

Deadline Monotonic (2)

- The shorter the deadline the higher the priority (priorities are all different)

Task	T	D	C	R
A	10	10	3	5
B	15	15	7	15
C	30	4	2	2



Deadline Monotonic - Evaluation

- Advantages
  - Simple and suited to existing operating systems
  - May be used to assign interrupt levels
  - Suited to periodic tasks with short deadlines
- Drawbacks
  - Only for preemptive tasks
  - Cannot handle exclusion
- Remarks
  - Fixed priorities (see t=10 on example)

Exercise

- Is the task set below schedulable with RM (resp. DM)
  - Necessary condition fulfilled ?
  - Sufficient condition ?
  - Define priorities
  - Calculate response times

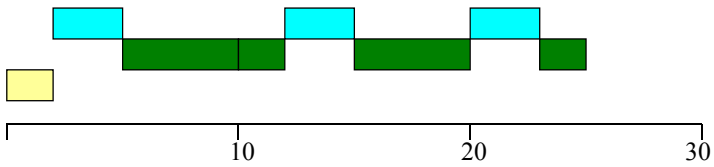
Task	T	C	D(RM)	D (DM)
A	4	2	4	3
B	8	1	8	8
C	10	3	10	7

Earliest Deadline First (EDF)

- Assumptions [Der74]
  - Task deadlines D_i are arbitrary
  - Tasks are preemptive and independent
  - Task cannot block or suspend themselves
  - The worst execution time C_i of a task i is known
- Algorithm
  - At each instant, is under execution the task that has the closest deadline (and is ready)

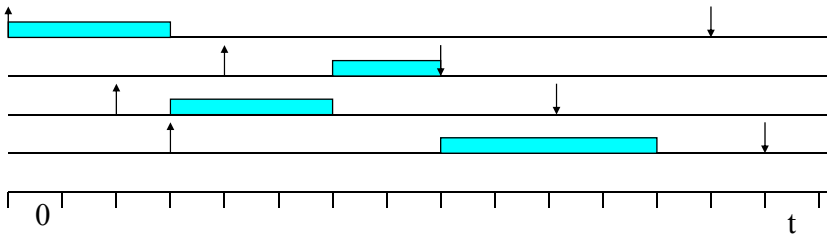
EDF (2)

Task	T	D	C
A	10	10	3
B	15	15	7
C	30	4	2



EDF - Optimality

Theorem [Dertouzos Der74]
EDF algorithm is optimal in that, if there exists an algorithm that can successfully schedule on a single processor a set of tasks with arbitrary deadlines and execution times, EDF will succeed in scheduling the same set



EDF Optimality - Proof

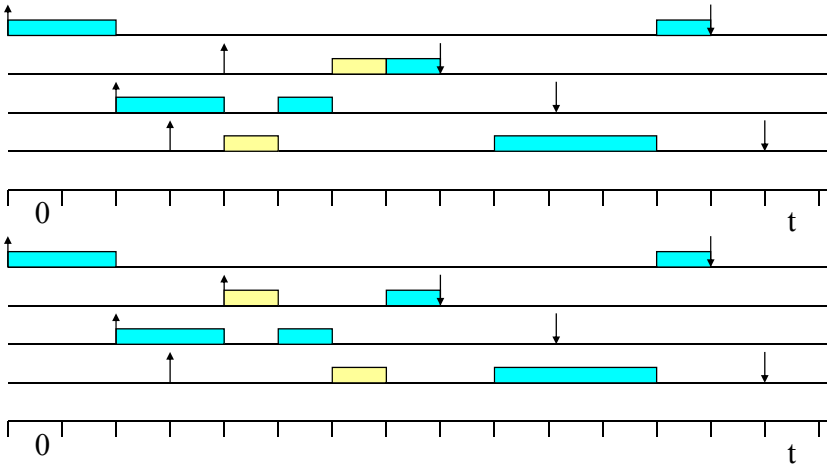
- Definitions
  - a task that is execution at instant i on Π_i
  - a' task that is execution at instant i on Π_{i+1}
  - b task that Π_i at instant i is ready, has not yet entirely executed and has closest deadline
  - c task that Π_{i+1} will be executing at instant T where T is the instant (in future) closest to instant i and at which task b is executing in Π_i

EDF Optimality - Proof (2)

- Proof obtained by a successive transformation algorithm

```
FOR i:=0 TO t DO
        find b
        IF b exists THEN
                IF a=b THEN
                        a'=a
                ELSE
                        a'=b ; c = a
                FI
        ELSE
                a' = a
        FI
OD
```


EDF Optimality - Proof (3)



EDF Optimality - Proof (4)

- Execution time preservation
  - Unchanged state or unit duration swapped
- Deadlines
  - Task b that was executing at instant T and which execution is advanced is the closest deadline one
  - Postponing task a to instant T will not violate its deadline
- Release time
  - A task execution cannot be advance before that instant at which it is ready (by definition of b)

EDF - Guarantee Function

- Necessary and sufficient conditions [Spuri Sta98]

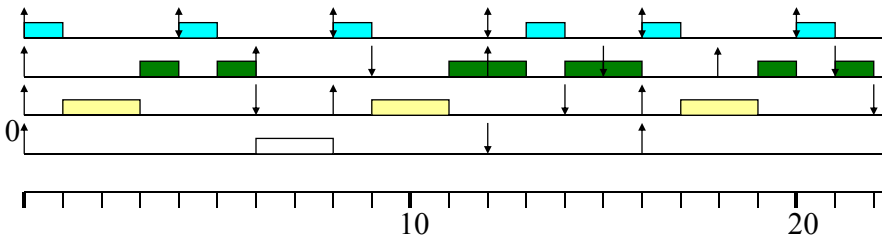
$$\sum_{i=1}^{n} \frac{C_i}{T_i} \le 1 \quad \text{and} \quad \sum_{D_i \le d} \left(1 + \left\lfloor \frac{d - D_i}{T_i} \right\rfloor \right) C_i \le d \quad \text{for} \quad 0 < d \le L$$

with

$$\begin{cases} L^{(0)} = \sum_{i=1}^{n} C_i \\ L^{(m+1)} = W\left(L^{(m)}\right) \end{cases} \quad \text{and} \quad W(t) = \sum_{i=1}^{n} \left\lceil \frac{t}{T_i} \right\rceil C_i$$

EDF - Example

Task	T	D	C
A	4	4	1
B	6	9	2
C	8	6	2
D	16	12	2



EDF - Exercise

- If the task set below schedulable ?

Task	T	D	C
A	10	10	3
B	15	15	7
C	30	4	2

EDF - Evaluation

- Advantages
  - Simple
  - Good resource use
  - Suited to sporadic and periodic tasks with short relative deadlines
- Drawbacks
  - Only for preemptive tasks
  - Cannot handle exclusion
  - Rather bad in case of overloads

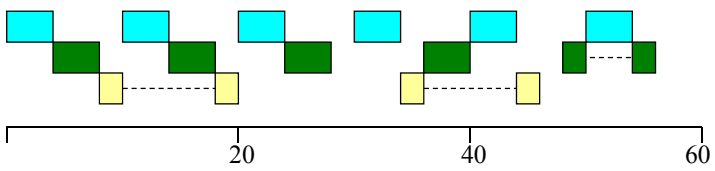
Shared Resources

- Principle
  - a task cannot access to a shared resource unless it has seized the mutex that protects this resource
  - To seize it, it has to wait until the mutex is free
  - When the task no longer needs the resource, it must release the mutex
- Shared resources protected by mutexes may lead to problems such as priority inversions and deadlocks

Shared Resource - Example

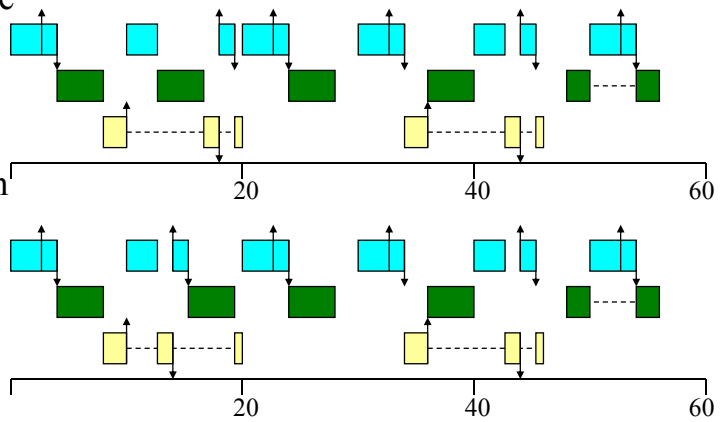
- Schedule according to DM

Task	T	D	C	R
A	10	6	4	4
B	12	12	4	8
C	30	30	4	20

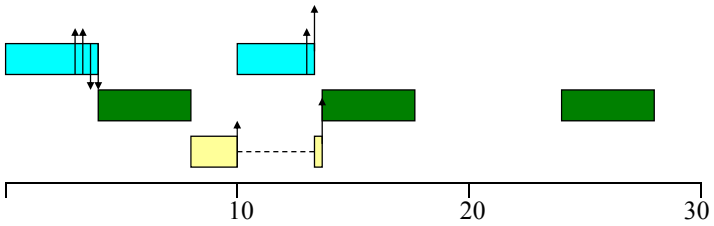


Priority Inversion

- example
- solution



Deadlock



Mutual Exclusion

Theorem [Mok83]: When there are mutual exclusion constraints, it is impossible to find a totally on-line optimal runtime scheduler

Theorem [Mok83]: The problem of deciding whether it is possible to schedule a set of periodic processes that use semaphores only to enforce mutual exclusion is NP-hard

Mutual Exclusion (2)

- 2 approaches
  - try to have blocks with identical execution times
  - Find a non optimal solution that avoids priority inversions and deadlocks while guaranteeing a minimum of performances
    - "Priority Ceiling Protocol" [Sha90]
    - "Immediate Inheritance Protocol"
    - "Stack Resource Policy" [Bak91]

SRP Asumptions

- Tasks J_i are preemptive and known in advance
- Exec. Times C_i bounded, deadline D_i arbitrary
- Tasks cannot wait by themselves
- Resource R_i (i=1..M) cannot be preempted and is made of a finite number N_R of units
- Execution priority p(*J*) of a task set according to one of 3 algorithms: EDF, RM, DM
- Each task seizes and releases semaphores in LIFO order. A request does not exceed system resource

SRP Definitions

- A resource allocation is a triplet (J, R, μ)
- A task waiting for an allocation is said blocked
- When it is granted the allocation (this one is said running), the task owns it until it releases it
- A task execution *J* is a particular execution instance of a task J created as a response to an exec. Request. The former occurs at Arrival(*J*). Execution starts at instant Start(*J*)
- Start(*J*)≥Arrival(*J*); In between task exec. is pending

SRP Definitions (2)

- ν_R= number of non allocated units of resource R
- J_{cur} = current task execution; J_{max} oldest of highest priority pending task executions
- Each task exec. has a preemption level $\pi(J_i)$
- A critical resource is said trivial if it is related to a resource that cannot cause any blocking
- An external non trivial critical section is a critical section that is included in no other non trivial critical section

SRP - Premption Levels

- Preemption levels are such that
$$\pi(J) < \pi(J') \Leftrightarrow D(J') < D(J)$$
with EDF, RM and DM, a task exec. *J'* that has a preemption level higher than the level of another task exec. *J* will never be preempted by the latter
- This result is true as long as:

$$\pi(J) > \pi(J') \text{ or } p(J) \leq p(J') \text{ or } \text{Arrival}(J) \leq \textit{Arrival}(J')$$

SRP - Basic Idea

To avoid deadlocks, a task should not be allowed to start unless the resource available at this instant are sufficient to satisfy all its needs

- Multiple inversions = job blocked longer that the duration of the external non trivial critical section of a job of lower priority

To avoid multiple priority inversions, a task should not be allowed to start unless the resource available at this instant are sufficient to satisfy the needs of each task execution that might preempt it

SRP - The Algorithm

- A resource has a current ceiling $\lceil R \rceil$ that is a function of the current set of allocations of R
- Ceilings and task preemption levels are related in such a way that: if *J* is running or may preempt the running task and if *J* may request a resource allocation to R that would be blocked by the current R allocations then $\pi(J) \leq \lceil R \rceil$
- In particular $\lceil R \rceil_{\nu R} = \max(\{0\} \cup \{\pi(J) | \ \nu_R < \mu_R(J)\})$

SRP - Example

- In particular $\lceil R \rceil_{\nu R} = \max(\{0\} \cup \{\pi(J) | \ \nu_R < \mu_R(J)\})$

Task	T	D	C	π
A	10	6	4	3
B	12	12	4	2
C	30	30	4	1

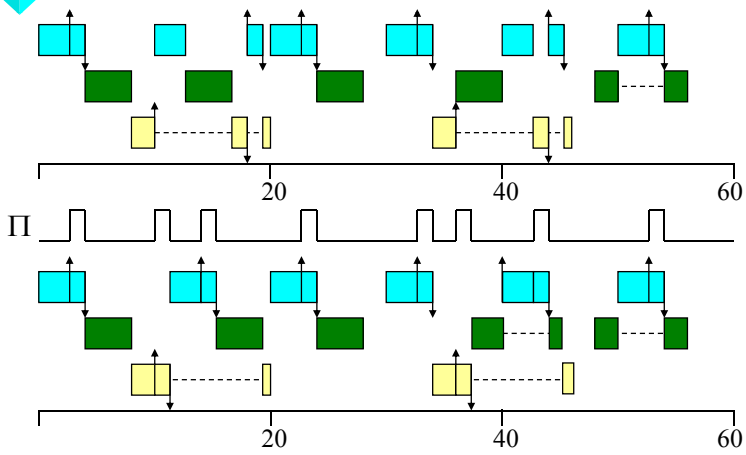
R	N	μ(A)	μ(B)	μ(C)	$\lceil R \rceil_0$	$\lceil R \rceil_1$
S	1	1	0	1	3	0
S 1	1	1	0	1	3	0

SRP - The Algorithm (2)

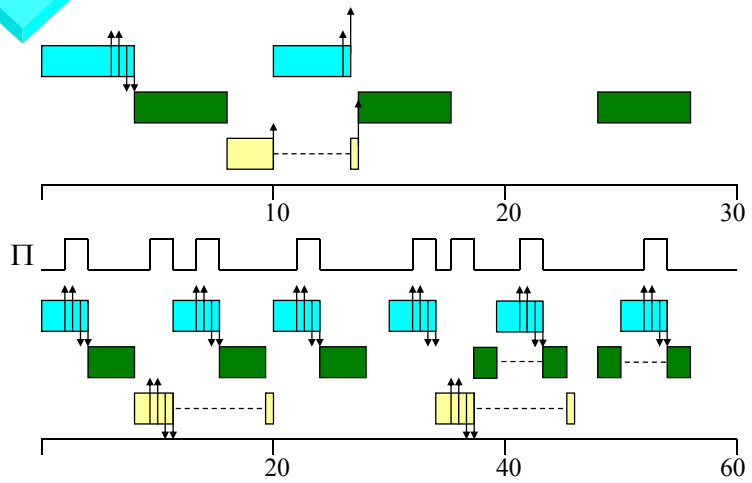
- A task execution request is not allowed to start unless *J* is the oldest pending request among the highest priority ones and, if *J* would preempt a running task exec.
- If $\Pi = \max\{\lceil R_i \rceil, i=1..m\}$ then the condition becomes $\Pi < \pi(J)$

$$\forall_{i=1..m} \ R_i \ \ \lceil R_i \rceil_{\nu_{R_i}} < \pi(J)$$

SRP - Inversion Avoidance



SRP - Deadlock Avoidance



SRP - A Priori Analysis

- Sufficient condition (RM, DM, EDF)

$$\forall_{i=1..n}\ i \bullet \left(\sum_{m=1}^{i} \frac{C_m}{D_m} \right) + \frac{B_i}{D_i} \le 1$$

B_i is the longest non trivial critical section of jobs J_k such that $D_i<D_k$

$$B_i = \max_{j,k} \left\{ L_{j,k} \middle| \pi_j < \pi_i \bullet \lceil R_k \rceil_0 \ge \pi_i \right\}$$

SRP - A Priori Analysis (2)

- Necessary and sufficient condition (EDF)

$$\sum_{D_i \le d} \left(1 + \left\lfloor \frac{d - D_i}{T_i} \right\rfloor \right) C_i + B_{m(d)} \le d \quad \text{pour } 0 < d \le L$$

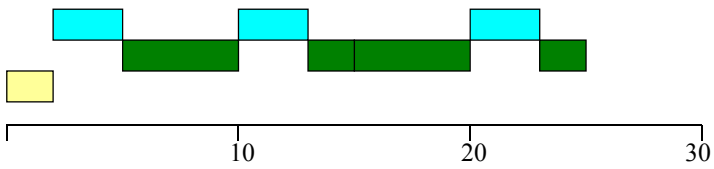
$B_{m(d)}$ is the longest non trivial critical section of jobs J_m such that m=n for max(D_n)≤d

$$B_i = \max_{j,k} \left\{ L_{j,k} \middle| \pi_j < \pi_i \bullet \lceil R_k \rceil_0 \ge \pi_i \right\}$$

Jitter

- Variation in execution period for periodic tasks
- Example: DM case

Task	T	D	C	R
A	10	10	3	5
B	15	15	7	15
C	30	4	2	2



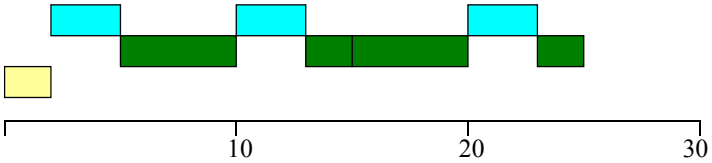
Jitter (2)

- Absolute jitter

$$GR_i = \max\left\{\left(T_i - T_i^{\min}\right), \left(T_i^{\max} - T_i\right)\right\}$$

- Relative jitter

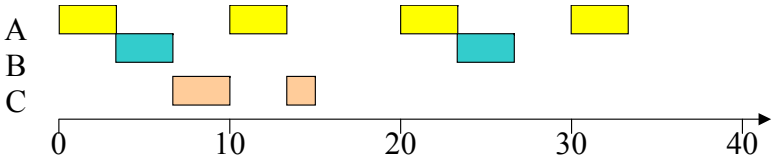
$$GA_i = GR_i / T_i$$



Jitter (3)

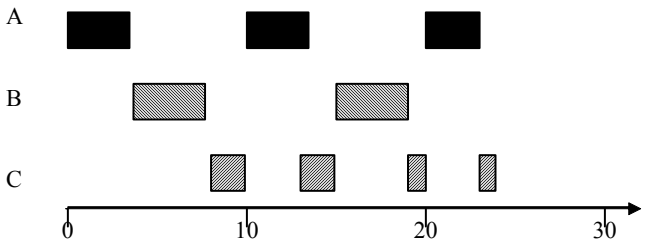
- Possible solutions
  - Give highest priority to the task that has jitter constraints
  - Use periods that are in harmonic relationship and all other tasks with lower priorities
  - Separate task IO from processing and give higher priority to IO

Jitter (4)



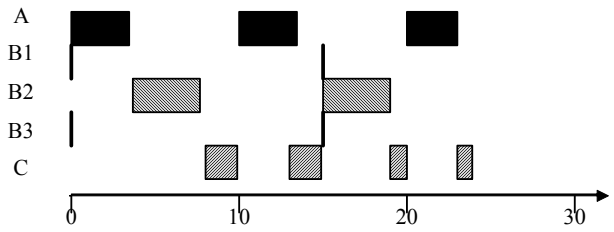
Task	T	D	C
A	10	10	3.3
B	20	20	3.3
C	40	40	5

Jitter – before task split



Task	T	D	C
A	10	10	3.3
B	15	15	4
C	30	30	5

Jitter (5)



Task	T	D	C
A	10	10	3.3
B1	15	2	0.001
B2	15	15	3.998
B3	15	1	0.001
C	30	30	5

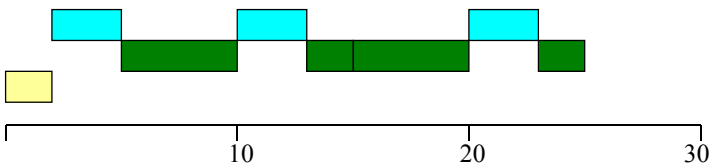
Sporadic Tasks

- Response time made of 2 components: detection time and reaction time
- Sporadic task C_{sp}, D_{sp}, T_{sp}
- Simple techniques
  - Background processing
  - Polling
  - Using RM, DM or EDF

Background Processing for sporadic tasks

- Background - use leftover CPU time

$$LT = LCM(T_j) - LCM(T_j)\sum_k (C_k/T_k)$$



Polling sporadic tasks (simple server)

- sporadic task C_{sp}, D_{sp}, T_{sp}
- Server (periodic task) C_s, D_s, T_s
- worst case:
  - sporadic request misses the server
  - has to wait until next instance
  - if $C_{sp} <= C_s$, sporadic request completed within two period task periods (one for waiting, one for executing)
  - $2*T_s <= D_{sp}$
- arbitrary execution times:
  - $T_s + \lceil C_{sp}/C_s \rceil T_s <= D_{sp}$

Using RM, DM or EDF for sporadic tasks

- Joseph and Pandya give necessary and sufficient condition for periodic and sporadic tasks under RM and DM iif Di ≤ Ti

$$\forall i \;\; R_i \le D_i \;\; \text{with } R_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

- EDF necessary and sufficient condition for general task set (D and T unrelated)

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \le 1 \;\; \text{and} \sum_{D_i \le d} \left(1 + \left\lfloor \frac{d - D_i}{T_i} \right\rfloor \right) C_i \le d \;\; \text{for} \;\; 0 < d \le L$$

Sporadic tasks: analysis

- Background
  - Only leftover time is available for servicing
  - Not good for short deadline
  - No risk for other tasks
- Polling
  - Can provide better guarantees than backgound
  - No risk for other tasks
- Use normal scheduler
  - Guaranteed deadline
  - No protection against misuse

Special Purpose Algorithms for Sporadic Tasks

- Fixed priority
  - Deferrable Server [Str95]
  - Sporadic Server [Spr89]
- Dynamic Priority
  - Deadline Deferrable Server [Gha95]
  - Deadline Sporadic Server [Gha95]
  - Deadline Exchange Server [Gha95]

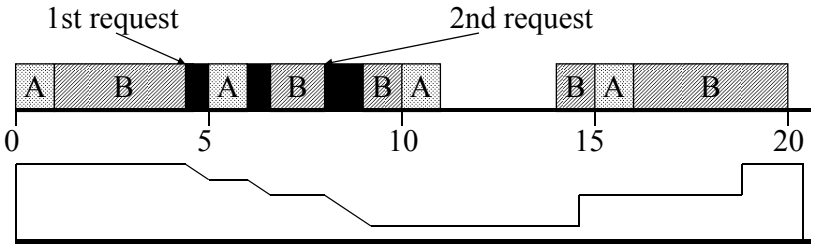
Sporadic Server - Definitions

- A periodic task that keeps its execution time until a request occurs
- Definitions
  - P_s: priority level of the running task
  - P_i: a given priority level ($P_i<P_k$, if i>k)
  - RT_i: replenishment instant for priority level P_i
  - Active: a given priority level P_i is active if $P_i \le P_s$
  - Idle: a given priority level P_i is idle if $P_i > P_s$

Sporadic Server - Algorithm

- If the server still has available execution time, the replenishment instant RT_i is decided when the level P_i becomes active. Its value is the current instant increased by the server period
- If its capacity is exhausted, RT_i will be decided when the capacity becomes non empty and P_i is active
- Capacity replenishment will take place at RT_i if priority level becomes idle or capacity exhausted
- Replenishment capacity is equal to used capacity since the last transition from idle to active

Sporadic Server - Example



Task	T	C	use
A	5	1	20%
SpS	10	2.5	25%
B	14	6	42.9%

Sporadic Server - Evaluation

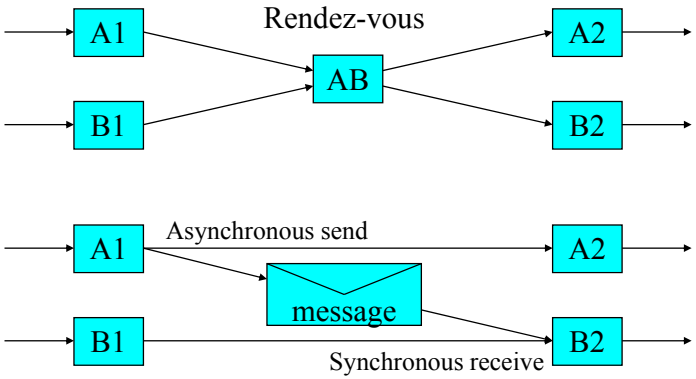
Theorem [Spr89]: A set of periodic tasks that may be scheduled by RM with a periodic task X, is also schedulable if task X is replaced by a sporadic server that has the same period and execution time

- Smaller detection time
- Capacity is preserved indefinitely
- Fair
- Known effect on low priority tasks (RM classical a priori analysis can be used)
- More than one server may be used

Sporadic Tasks

- Use RM, DM or EDF but beware of overloads
- Use Sporadic Server
  - Its period should be lower than the sporadic task deadline
  - Execution time $\geq \lceil$ period/min_interarrival_time $\rceil$
- Use the deadline oriented versions

Precedence Constraints

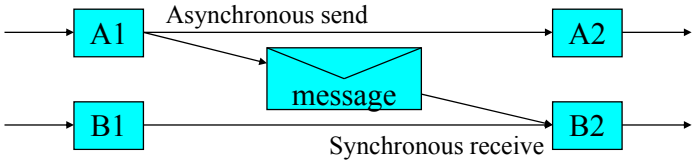


Precedence Constraints (2)

- Tasks are preemptive or not
- All tasks characteristics are know in advance
- Task deadlines are arbitrary
- There is a cost function p(t) monotonously increasing that gives the cost related to a task when it finishes at time t
- Task execution times are bounded
- Tasks do not shared resources
- All tasks are ready at the same time

Lawler Algorithm [Law73]

Schedule tasks from last to first by selecting the task among the available tasks (their successors are already selected) the task that has the latest deadline



Blazewicz Method

Theorem [Bla76]: EDF is optimal for preemptive tasks that have arbitrary precedence relationships and different release times and deadlines if release times and deadlines are modified in the following manner:

$$d_i^* = \min\left\{d_i, \min_{S_i \to S_j}\left(d_j^* - C_j\right)\right\} \text{ and } r_i^* = \max\left\{r_i, \max_{S_j \to S_i}\left(r_j^* + C_j\right)\right\}$$

$S_i \to S_j$ means that task j follows task i

Overloads

- A system is said underloaded if all deadlines are met. Otherwise it is overloaded
- Many good algorithms (EDF, Smallest Slack Time) perform poorly in case of overloads
- Practical systems are prone to intermittent overloads
- A good on-line algo. should give performance guarantee in such a case (also if underloaded)

Overloads (2)

- Fixed Priority Systems
  - overload unaware: low priority tasks never executed
  - overload aware: Myopic Slack Manager [Mar95]
- Dynamic Priority Systems
  - overload unaware: may exhibit bad behavior
  - overload aware: D-over [Kor95], Robust EDF [But93]

D^{over} Algorithm

- Due to Koren & Shasha [Kor95]
- Applicable to soft & firm RT systems
  - firm: no value is given if deadline missed
  - soft: less value is given if deadline is missed
- On-line (no info on task prior arrival)
- Tasks (release r_i, deadline d_i, value v_i, range of computation time c_i, preemptive, independant)

Soft RT - Definitions

- value density = value / computation time
- importance ratio k of a collection of tasks = max. value density / min. value density
- clairvoyant scheduler has a priori knowledge of all tasks and their parameters
- on-line algo competitive factor r ($0 \le r \le 1$) iif it guarantees to obtain a cumulative value at least r times the one achieved by clairvoyant one

Overloads - Some Results

- Best competitive factor is: with importance ration k $\frac{1}{(1+\sqrt{k})^2}$
- This is not efficiency
- 0.25 in case of uniform value density
- There exists other algorithms in case of uniform value density that achieve the max. (r=0.25 in overloads and 100% of possible value in underloads)

Overloads - Some Results (2)

- in absence of guarantee mechanisms, the best algo is HDF Highest Density(=v/c) First
- in case of rejection at task arrival if overload occurs, best algo is EDF (acceptance at arrival gives bad results for algo. based on task values)
- in case of more sophisticated control, variant of EDF seem to perform best

G. Buttazo et al., RTSS 1995.

D^{over} Algorithm - Definitions

- executable period $\Delta_i=[r_i,d_i]$
- completed task: has received an amount of execution time equal to C_i before its deadline
- abandoned task: not completed and will never be scheduled again
- preempted task: not executing, might be later
- ready task at t: $r_i \le t \le d_i$; not completed nor abandoned

D^{over} Algorithm - Definitions (2)

- Γ collection of tasks $T_1,T_2 \ldots T_n$
- smallest importance of a task in Γ is 1
- 3 kinds of events
  - task completion (high priority)
  - task release (normal priority)
  - task latest start time (normal priority)
- an event may be suppressed by another

D^{over} Algorithm - Definitions (3)

- Γ={current, recently-preempted Γ_{rp} , others Γ_o}
- a preempted task goes into Γ_{rp}
- if a task is scheduled under latest_start_time event, all ready tasks go into Γ_o
- AvailT: when a new task is released which has the earliest deadline, this is the max. comp. time that can be taken without causing the current one or any in Γ_{rp} to miss their deadlines

D^{over} Algorithm - Variables

- 2 queues ordered by deadline: Qrecent, Qother
- Qlst (all tasks) ordered by latest start time
- recentval: running value of tasks in Qrecent
- now(): returns current time
- Laxity(T_i)= d_i -[now()+remain_comp_time(T_i)]
- Schedule(T_i): gives the processor to task T_i
- Qrecent = (T, PreviousTime, PreviousAvail)

D^{over} - The Algorithm

- task completion
  - if Qrecent ≠ Ø & Qother ≠ Ø
    - (Tqrec,tprev,availprev)=Min(Qrecent); AvailT=availprev-(now-tprev); Tqother=Min(Qother);
    - if d(Tqother) < d(Tqrec) & AvailT ≥remain(Tqother)
      - Dequeue(Qother); AvailT =AvailT-remain(Tqother); AvailT = min(AvailT, laxity(Tqother)); schedule(Tqother);
    - else
      - Dequeue(Qrecent); recentval = recentval-value(Tqrecent); schedule(Tqrecent);

D^{over} - The Algorithm (2)

- task completion
  - if Qrecent=Ø & Qother≠Ø
    - Tcur=Dequeue(Qother); AvailT=laxity(Tcur); schedule(Tcur);
  - if Qrecent ≠ Ø & Qother=Ø
    - (Tcur,tprev,availprev)=Dequeue(Qrecent); AvailT=availprev-(now-tprev); recentval=recentval-value(Tcur); schedule(Tcur);
  - if Qrecent=Ø & Qother=Ø
    - idle=true; AvailT=∞

D^{over} - The Algorithm (3)

- task release (new task is Tarr)
  - if idle
    - Tcur=Tarr; AvailT=laxity(Tcur); idle= false; sche(Tcur);
  - else
    - if d(Tarr) < d(Tcur) & AvailT ≥ c(Tarr)
      - insert Tcur in Qlst; insert (Tcur, now, AvailT) in Qrecent; AvailT = AvailT- c(Tarr); AvailT = min(AvailT, laxity(Tarr)); recentval = recentval+value(Tcur); Tcur = Tarr; schedule(Tcur);
    - else
      - insert Tarr in Qlst and Qother

D^{over} - The Algorithm (4)

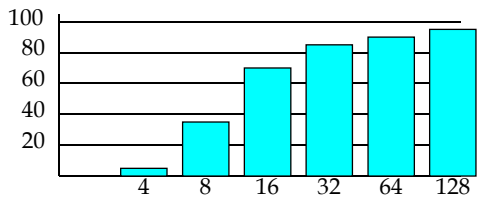
- latest start time event
  - Tnext = Dequeue(Qlst); Vnext= value(Tnext);
  - if (Vnext > (1+sqrt(k))(recentval+value(Tcur))
    - insert Tcur in Qlst and Qother; move all tasks from Qrecent to Qother; recentval = 0; AvailT = 0; Tcur = Tnext; schedule(Tcur);
  - else
    - abandon Tnext;

D^{over} Algorithm - Analysis

- behaves as EDF in absence of overload
- reaches the optimal competitive factor
- generalizes to soft deadlines
- generalizes to varying computation times

Fixed Priority Scheduling with Limited Priority levels

- In some cases (networks, interrupt driven systems, operating systems), the number of available priority levels is limited
- DM and RM assume that all tasks have a different priority



Relative schedulability vs number of priorities [Kat95]

Fixed Priority Scheduling with Limited Priority levels (2)

- m distinct priority levels, n tasks (τ_1 .. τ_n)
- tasks are grouped in m groups (g_1 .. g_m) in decreasing priority order
- necessary and sufficient condition (τ_i in g_k, r tasks in groups of higher priority, $D_i \leq T_i$)

$$\min_{o<t\leq D_i} W_i(t)/t \leq 1 \text{ with } W_i(t) = \sum_{j=1}^{r} C_j \left\lceil \frac{t}{T_j} \right\rceil + \sum_{\forall l \in g_k} C_l$$

Fixed Priority Scheduling with Limited Priority levels (3)

- Exercise 1

Task	T	D	C	R
A	10	10	3	5
B	15	15	7	12
C	30	4	2	2

- Exercise 2

Task	T	D	C
A	10	10	3
B	15	15	4
C	30	30	10

Constant Bandwidth Scheduling

- Continuous Media (CM) activities need real-time support
- Not well handled by conventional techniques
  - execution times vary widely
  - difficult to estimate WCET
  - interarrival time not deterministic
  - rather dynamic

Constant Bandwidth Scheduling (2)

- The main idea is to provide a fair share of the processor time without jeopardizing the other real-time tasks
- a number of approaches
  - Constant Utilization Server [Den97]
  - Total Bandwidth Server [Spu94]
  - Constant Bandwidth Server CBS [Abe98]
  - ….

CBS - Assumptions

- Task τ_i (C_i,D_i) consists in a sequence of jobs $J_{i,k}$
- $r_{i,k}$ is the arrival time of $J_{i,k}$
- deadline $d_{i,k} = r_{i,k} + D_i$
- objective is to minimize mean tardiness of soft tasks without jeopardizing real-time tasks
- finishing time $f_{i,k}$
- tardiness $E_{i,k} = \max\{0, f_{i,k} - d_{i,k}\}$

CBS - Assumptions (2)

- Real-time tasks scheduled by EDF
- soft real-time tasks handled by a dedicated server CBS
- server period T_s and maximum budget Q_s
- server deadline $d_{s,j}$ ($d_{s,0}$=0)
- schedulable iif $U_p + U_s \leq 1$ ($U_s = Q_s / T_s$)
- $U_p = \sum (C_i / T_i)$ for all real-time tasks

CBS - Algorithm

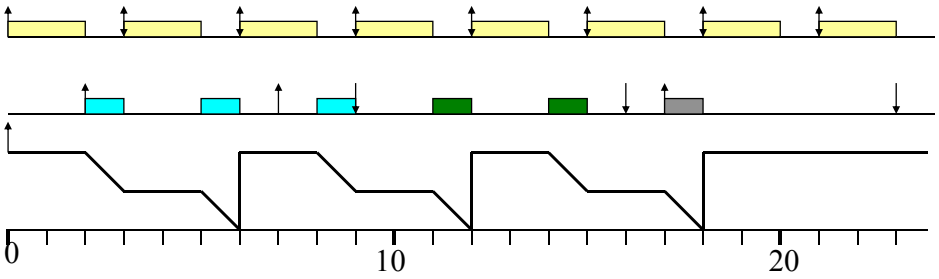
- Each served job is assigned a deadline $d_{i,k}$ equal to current server deadline $d_{s,j}$
- whenever a served job executes the server budget c_s is decreased by same amount
- when c_s=0, budget recharged to maximum Q_s and deadline changed to $d_{s,j+1} = d_{s,j} + T_s$
- CBS said active if pending jobs, idle otherwise
- if CBS active, new arriving job queued in FIFO

CBS - Algorithm (2)

- If CBS idle and job arrives,
  - if $c_s \geq (d_{s,j} - r_{i,k}) U_s$ the server generates a new deadline $d_{s,j+1} = r_{i,k} + T_s$ and c_s is recharged to max.
  - otherwise the job is served with the last deadline and remaining budget
- when a job is finished, the next job is served with current deadline and budget (idle if none)
- at any instant a job is assigned the last server deadline

CBS - Example

CBS(2,7), τ1(2,3), τ2 (soft, variable execution time)



CBS - Exercise

- CBS features

  CBS(2,7), τ1(2,3), τ2 (soft, variable execution time)
- $T2 = 7$
- $r_{2,1} = 0, C_{2,1} = 2$
- $r_{2,2} = 7, C_{2,2} = 3$
- $r_{2,3} = 14, C_{2,3} = 2$

CBS - Analysis

- Works well if hard real-time tasks have deadline greater or equal to period
- automatically reclaims spare time caused by early completions
- does not guarantee the deadlines for soft tasks (scheduled by CBS)
- some statistical guarantees can however be obtained

Non Preemptive Scheduling

- Sometimes necessary or desired
- Exclusive access to shared resource is guaranteed
- Implementation is simpler

- Unfortunately more complex to analyze

Non Preemptive EDF [Jef91]

- Asumptions
  - Tasks are periodic P_i (T_i,C_i,D_i=T_i,r_i) or sporadic S_i (T_i,C_i,D_i=T_i,r_i)
  - Tasks are independent
- Definitions
  - Task set τ={S_i} or {P_i}
  - Concrete task set ω= {S_i ,r_i} or {P_i ,r_i}
  - A task set can be scheduled if all sets ω are

Non Preemptive EDF (2)

- Universality
  - An algorithm is universal for a concrete task set if it can schedule all schedulable sets
  - An algorithm is universal for a task set if it can schedule every concrete task set that has been generated from this task set

Non preemptive EDF is universal for all task sets and for all sporadic concrete task sets

It is unlikely that there exists a universal algorithm for concrete periodic task sets

Non Preemptive EDF (3)

- Under the following conditions, a periodic or sporadic task set may be scheduled

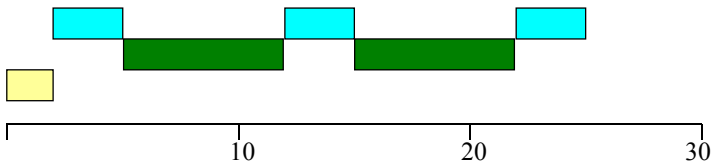
$$\sum_{i=1}^{n} \frac{C_i}{T_i} \le 1 \text{ and } \forall i : 1 < i \le n; \forall L : T_1 < L < T_i; L \ge C_i + \sum_{k=1}^{i-1} \left\lfloor \frac{L-1}{T_k} \right\rfloor C_k$$

- Some concrete periodic task sets may be scheduled even if the 2nd condition is not satisfied
- algorithm

When a task ends or the processor is idle, start the execution of the task that has the closest deadline

Non Preemptive EDF (4)

Task	T	D	C	R
A	10	10	3	5
B	15	15	7	12
C	30	4	2	2



Non Preemptive EDF: exercise

- Is it possible to schedule the following task set in a non preemptive manner ?

Task	T	D	C
A	14	14	6
B	20	20	6
C	30	30	6

Conclusion

- Many techniques
  - independant, resource sharing, precedence
  - sporadic and multimedia tasks
  - jitter, temporary overloads
- Other phenomena may be taken into account
  - power consumption of processor, memory, communication
- Multiprocessor scheduling

References

- Books
  - [Kle93] M. Klein, "A Practioners 's Handbook for Real-Time Analysis: Guide to Rate Monoto-nic Analysis for Real-Time Systems", Kluwer Academic, Boston, 1993, ISBN 0-7923-9361-9
  - [Raj91] R. Rajkumar, « Synchronization in Real-Time Systems - A Priority Inheritance Approach », Kluwer Academic, Boston, 1991, ISBN 0-7923-9211-6
  - [Sta98] J. Stankovic et al., « Deadline Scheduling for Real-Time Systems », Kluwer Aacademic, Boston, 1998, ISBN 0-7923-8269-2
  - [But97] G. Buttazzo, "Hard Real-Time Computing Systems", Kluwer Academic, Boston, 1997.
- Journal Articles
  - [Abe98] L. Abeni et al., "Integrating Multimedia Applications in Hard Real-Time Systems", Proc. RTSS, Madrid, Dec. 2-4.1998, pp.4-13.
  - [Aud90] N. Audsley, "Deadline Monotonic Scheduling", report YCS 146, York University, 1990.
  - [Bak91] T. Baker, "Stack-Based Scheduling of Real-Time Processes", Real-Time Systems 3 (1), pp.67-99, 1991.

References (2)

- [Bla76] J. Blazewicz, "Scheduling Dependant Tasks with Different Arrival Times to Meet Deadlines", in Modeling and Performance Evaluation of Computer Systems, North Holland, Amsterdam, pp. 57-65, 1976.
- [But93] G. Buttazzo and J. Stankovic, "RED: Robust Earliest Deadline Scheduling,", 3rd Int. Workshop On Responsive Computing Systems, Austin, 1993.
- [Che88] S. Cheng, et al., "Scheduling Algorithms for Hard Real-Time Systems - A Brief Survey", in Hard Real-Time Systems, J. Stankovic, K. Ramamritham eds, IEEE Computer Society Press, 1988.
- [Che90] M. Chen, K. Lin, "Dynamic priority ceilings: a concurrency control protocol for real-time systems", Real-Time Systems 2 (4), pp.325-46, 1990.
- [Che90a] H. Chetto, M. Silly, T. Bouchentouf, "Dynamic scheduling of real-time tasks under precedence constraints", Real-Time Systems 2 (3), pp.181-94, 1990.
- [Der74] M. Dertouzos, "Control Robotics; The Procedural Control of Physical processes", Information Processing 74, North Holland, pp. 807-813, 1974.
- [Den97] Z. deng et al., « A Scheme for Scheduling Hard Real-Time Applications in Open System Environment », 9th Euromicro Workshop on Real-Time Systems, Toledo, June 11-13, 1997, pp.191-9
- [Gar79] M. Garey, D. Johnson, "Computers and Intractability - a guide to the Theory of NP-Completeness", Freeman and Co, New-York, 1979.

References (3)

- [Gar81] M. Garey et al., "Scheduling Unit-Time Tasks with Arbitrary Release Times and Deadlines", SIAM Journal of Computing 10(2), pp. 1981.
- [Gha95] T. Ghazalie, T. Baker, "Aperiodic servers in a deadline scheduling environment", Real-Time Systems 9 (1), pp.31-67, 1995.
- [Hom94] N. Homayoun, P. Ramanathan, "Dynamic priority scheduling of periodic and aperiodic tasks in hard real-time systems", Real-Time Systems 6 (2), pp.207-32, 1994.
- [Jac55] J.Jackson, "Scheduling a Production Line to Minimize Maximum Tardiness", Research report 43, Management Science Research Project, University of California, Los Angeles, 1955.
- [Jef91] K. Jeffay, D. Stanat, C. Martel, "On Non-Preemptive Scheduling of Periodic and Sporadic Tasks", 12th Real-Time Systems Symposium, San Antonio, Texas, Dec. 4-6,1991, pp.129-139.
- [Jos86] M. Joseph, P. Pandya, "Finding Response Times in a real-Time System", The Computer Journal 29 (5), 1986, pp.390-395.
- [Kat95] D. Katcher et all, "Fixed Priority Scheduling with Limited Priority Levels", IEEE Trans. On Computers 44 (9), pp.1140-4
- [Kle94] M. Klein, J. Lehoczky, R. Rajkumar, "Rate-Monotonic Analysis for Real-Time Industrial Computing", IEEE Computer 2 (1), pp. 24-33, 1994.

References (4)

- [Kor95] G. Koren, D. Shasha, "D^{over}: an optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems", SIAM Journal on Computing 24 (2), pp.318-39, 1995.
- [Law73] E. Lawler, "Optimal Sequencing of Single machine Subject to Precedence Constraints", Management Science 19 , pp. , 1973.
- [Law78] E. Lawler, J. Labetoulle, "On preemptive scheduling of unrelated parallel processors by linear programming", Journal of the ACM, 25 (4), pp.612-19, 1978.
- [Law83] E. Lawler, "Recent Results in the Theory of Machine Scheduling", in Mathematical Programming; the State of the Art, A. Bachen ed., Springer, New-York, 1983.
- [Leh87] J. Lehoczky, L. Sha, J. Strosnider, "Enhanced Aperiodic Responsiveness in Hard Real-Time Systems", Proc. IEEE RTSS, San Jose, CA, USA, 1-3 Dec. 1987, pp. 261-70.
- [Leu92] J. Leung, J. Whitehead, "On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks", Performance Evaluation 2(4), 1982, pp.237-250.
- [Liu73] C. Liu, J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", Journal of the ACM 20 (1), 1973, pp.40-61.
- [Mar95] M. Marucheck et al., “An evaluation of the graceful degradation properties of real-time schedulers”,25th Annual Int. Symp. on Fault-Tolerant Computing, June 1995.

References (5)

- [Moo68] J. Moore, " An n Job, one Machine Sequencing Algorithm for Minimize the Number of Late Jobs", Management Science 15 (1), 1968.
- [Sha86] L. Sha, J. Lehoczky, R. Rajkumar, "Solutions for Some Practical Problems in Prioritized Preemptible Scheduling", Proc. IEEE RTSS, New Orleans, LA, pp.181-191, 1986.
- [Sha90] L. Sha, R. Rajkumar, J. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization", IEEE Trans. on Computers 39 (9), pp. , 1990.
- [Spr89] B. Sprunt, L. Sha, J. Lehoczky, "Aperiodic Task Scheduling for Hard-Real-Time Systems", Real Time systems 1(1), pp.27-60, 1989.
- [Spu94] M. Spuri et al., « Efficient Aperiodic Service under Earliest Deadline Scheduling », Proc. IEEE Real-Time Systems Symp, Pisa, Dec.5-7, 1995, pp.210-9
- [Sta95] J. Stankovic, M. Spuri, M. Di Natale, G. Buttazzo, "Implications of Classical Scheduling Results for Real-Time Systems", IEEE Computer 28 (6), pp.18-25, 1995.
- [Str95] J. Strosnider et al., "The deferrable server algorithm for enhanced aperiodic respon-siveness in hard real-time environments", IEEE Trans. on Computers 44 (1), 1995, pp.73-91.
- [Xiu93] J. Xiu, D. Parnas, "On Satisfying Timing Constraints in Hard-real-Time Systems", IEEE trans. on Software Engineering 19, No.1, pp.70-84, 1993.