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)| ≤ 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)**

<table>
<thead>
<tr>
<th>Function</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>0.00001</td>
<td>0.00002</td>
<td>0.00003</td>
<td>0.00004</td>
</tr>
<tr>
<td>n²</td>
<td>0.0001</td>
<td>0.0004</td>
<td>0.0009</td>
<td>0.0016</td>
</tr>
<tr>
<td>N³</td>
<td>0.1</td>
<td>3.2 s</td>
<td>24.3 s</td>
<td>1.7 min</td>
</tr>
<tr>
<td>2ⁿ</td>
<td>0.001 s</td>
<td>1 s</td>
<td>17.9 min</td>
<td>12.7 day</td>
</tr>
<tr>
<td>3ⁿ</td>
<td>0.059 s</td>
<td>58 min.</td>
<td>6.5 y</td>
<td>3855 cen</td>
</tr>
</tbody>
</table>
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
  \[ PM_i(T_i, C_i^{av}, C_i^{min}, D_i = T_i, r_i = 0) \]
- Cyclic tasks
  \[ TC_i(C_i, T_i^{av}, T_i^{max}, D_i = T_i, r_i = 0) \]
- Periodic tasks
  \[ P_i(T_i, C_i, D_i, r_i) \]
- Sporadic tasks
  \[ S_i(T_i, C_i, D_i, r_i = 0) \]
- 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

<table>
<thead>
<tr>
<th>Task</th>
<th>T</th>
<th>D</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>15</td>
<td>15</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>30</td>
<td>30</td>
<td>10</td>
</tr>
</tbody>
</table>

**Rate Monotonic - Guaranty Function [Liu73]**

- Necessary condition
  \[ \sum_{i=1}^{n} \left( \frac{C_i}{T_i} \right) \leq 1 \]
- Sufficient condition
  \[ \sum_{i=1}^{n} \left( \frac{C_i}{T_i} \right) \leq n \left( 2^{\frac{1}{n}} - 1 \right) \]

- \( n=1 \rightarrow 100\% \); \( n=2 \rightarrow 83\% \); \( n=3 \rightarrow 78\% \);
- \( n \) large \( \rightarrow 69\% \);

**Rate Monotonic - Exact Analysis**

- Due to Joseph and Pandya [Jos86]
- They introduce the idea of worst case response time
  \[ R_i = C_i + \sum_{\forall j \in \text{hp}(i)} \left( \frac{R_j}{T_j} \right) C_j \]
- Iterative calculation
  \[ R_i^{n+1} = C_i + \sum_{\forall j \in \text{hp}(i)} \left( \frac{R_j^n}{T_j} \right) C_j \]

**Rate Monotonic - Exact Analysis (2)**

<table>
<thead>
<tr>
<th>Task</th>
<th>T</th>
<th>D</th>
<th>C</th>
<th>R^0</th>
<th>R^1</th>
<th>R^2</th>
<th>R^3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>10</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>15</td>
<td>15</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>C</td>
<td>30</td>
<td>30</td>
<td>10</td>
<td>10</td>
<td>17</td>
<td>24</td>
<td>27</td>
</tr>
</tbody>
</table>
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)

<table>
<thead>
<tr>
<th>Task</th>
<th>T</th>
<th>D</th>
<th>C</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>10</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>15</td>
<td>15</td>
<td>7</td>
<td>15</td>
</tr>
<tr>
<td>C</td>
<td>30</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

10 20 30
**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

<table>
<thead>
<tr>
<th>Task</th>
<th>T</th>
<th>C</th>
<th>D(RM)</th>
<th>D (DM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>8</td>
<td>1</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>C</td>
<td>10</td>
<td>3</td>
<td>10</td>
<td>7</td>
</tr>
</tbody>
</table>

**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)**

<table>
<thead>
<tr>
<th>Task</th>
<th>T</th>
<th>D</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>15</td>
<td>15</td>
<td>7</td>
</tr>
<tr>
<td>C</td>
<td>30</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>
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.

Definitions
a task that is execution at instant $i$ on $\Pi_i$

a’ task that is execution at instant $i$ on $\Pi_{i+1}$

b task that $\Pi_i$ at instant $i$ is ready, has not yet entirely executed and has closest deadline

c task that $\Pi_{i+1}$ 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 $\Pi_i$.

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 (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]
  \[
  \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \quad \text{and} \quad \sum_{D_i \leq 0} \left( 1 + \left\lfloor \frac{d-D_i}{T_i} \right\rfloor \right) C_i \leq d \quad \text{for} \quad 0 < d \leq L
  \]
  with
  \[
  \begin{cases}
  L(0) = \sum_{i=1}^{n} C_i \\
  L^{(m+1)} = W(L^{(m)}) \\
  W(t) = \sum_{i=1}^{n} \left\lfloor \frac{t}{T_i} \right\rfloor C_i
  \end{cases}
  \]

**EDF - Example**

<table>
<thead>
<tr>
<th>Task</th>
<th>T</th>
<th>D</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>6</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>8</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>16</td>
<td>12</td>
<td>2</td>
</tr>
</tbody>
</table>

**EDF - Exercise**

- If the task set below schedulable?

<table>
<thead>
<tr>
<th>Task</th>
<th>T</th>
<th>D</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>15</td>
<td>15</td>
<td>7</td>
</tr>
<tr>
<td>C</td>
<td>30</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Task</th>
<th>T</th>
<th>D</th>
<th>C</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>6</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>12</td>
<td>12</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>C</td>
<td>30</td>
<td>30</td>
<td>4</td>
<td>20</td>
</tr>
</tbody>
</table>

Priority Inversion

- Example

- Solution
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 Assumptions

- 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 seize and releases semaphores in LIFO order. A request does not exceed system resource
A resource allocation is a triplet \((J, R, \mu)\).

- 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 \(\text{Arrival}(J)\). Execution starts at instant \(\text{Start}(J)\).
- \(\text{Start}(J) \geq \text{Arrival}(J)\); In between task exec is pending.

- \(v_R = \text{number of non allocated units of resource } R\)
- \(J_{\text{cur}} = \text{current task execution}; J_{\text{max}} = \text{oldest of highest priority pending task executions}\)
- 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 \(\pi(J) < \pi(J') \Rightarrow D(J') < D(J)\) 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:

\[ \pi(J) > \pi(J') \text{ or } p(J) \leq p(J') \text{ or } \text{Arrival}(J) \leq \text{Arrival}(J') \]

**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 \([R]\) 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 \(\pi(J) \leq [R]\).
- In particular \([R]_{\nu_R} = \max(\{0\} \cup \{\pi(J)\} \nu_R < \mu_R(J))\).

**SRP - Example**

- In particular \([R]_{\nu_R} = \max(\{0\} \cup \{\pi(J)\} \nu_R < \mu_R(J))\).

<table>
<thead>
<tr>
<th>Task</th>
<th>T</th>
<th>D</th>
<th>C</th>
<th>(\pi)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>6</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>12</td>
<td>12</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>30</td>
<td>30</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(R)</th>
<th>(N)</th>
<th>(\mu(A))</th>
<th>(\mu(B))</th>
<th>(\mu(C))</th>
<th>([R]_{\nu_R})</th>
<th>([R]_{\mu_R})</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>S1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

**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 \(\Pi = \max\{[R_i], i=1..m\}\) then the condition becomes \(\Pi < \pi(J)\)

\[\forall_{i=1..m} R_i \left[R_i\right]_{\nu_R_i} < \pi(J)\]

**SRP - Inversion Avoidance**

\[\Pi_{20-40-60}\]
**SRP - Deadlock Avoidance**

- Sufficient condition (RM, DM, EDF)
  \[ \forall i \left( \sum_{m=1}^{n} \frac{C_m}{D_m} + \frac{B_i}{D_i} \leq 1 \right) \]
  - B<sub>i</sub> is the longest non trivial critical section of jobs J<sub>k</sub> such that D<sub>i</sub> < D<sub>k</sub>
  - \[ B_i = \max_{j,k} \{ L_{j,k} \mid \pi_j < \pi_i \land \left[ R_k \right]_0 \geq \pi_i \} \]

**SRP - A Priori Analysis (2)**

- Necessary and sufficient condition (EDF)
  \[ \sum_{D_i \leq d} \left( 1 + \frac{d - D_i}{T_i} \right) C_i + B_{m(d)} \leq d \quad \text{pour} \ 0 < d \leq L \]
  - B<sub>m(d)</sub> is the longest non trivial critical section of jobs J<sub>m</sub> such that m=n for max(D<sub>n</sub>)≤d
  - \[ B_i = \max_{j,k} \{ L_{j,k} \mid \pi_j < \pi_i \land \left[ R_k \right]_0 \geq \pi_i \} \]

**Jitter**

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

<table>
<thead>
<tr>
<th>Task</th>
<th>T</th>
<th>D</th>
<th>C</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>10</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>15</td>
<td>15</td>
<td>7</td>
<td>15</td>
</tr>
<tr>
<td>C</td>
<td>30</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
Jitter (2)

- Absolute jitter
  \[ GR_i = \max \left\{ T_{i} - T_{i}^{\text{min}}, (T_{i}^{\text{max}} - T_{i}) \right\} \]

- 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 I/O from processing and give higher priority to I/O

Jitter (4)

<table>
<thead>
<tr>
<th>Task</th>
<th>T</th>
<th>D</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>10</td>
<td>3.3</td>
</tr>
<tr>
<td>B</td>
<td>20</td>
<td>20</td>
<td>3.3</td>
</tr>
<tr>
<td>C</td>
<td>40</td>
<td>40</td>
<td>5</td>
</tr>
</tbody>
</table>

Jitter – before task split

<table>
<thead>
<tr>
<th>Task</th>
<th>T</th>
<th>D</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>10</td>
<td>3.3</td>
</tr>
<tr>
<td>B</td>
<td>15</td>
<td>15</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>30</td>
<td>30</td>
<td>5</td>
</tr>
</tbody>
</table>
**Jitter (5)**

A B1 B2 B3 C

<table>
<thead>
<tr>
<th>Task</th>
<th>T</th>
<th>D</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>10</td>
<td>3.3</td>
</tr>
<tr>
<td>B1</td>
<td>15</td>
<td>2</td>
<td>0.001</td>
</tr>
<tr>
<td>B2</td>
<td>15</td>
<td>15</td>
<td>3.998</td>
</tr>
<tr>
<td>B3</td>
<td>15</td>
<td>1</td>
<td>0.001</td>
</tr>
<tr>
<td>C</td>
<td>30</td>
<td>30</td>
<td>5</td>
</tr>
</tbody>
</table>

**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

$$LT = LCM(T_j) - LCM(T_j) \sum_{k} \left( \frac{C_k}{T_k} \right)$$

**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 $C_{sp} \leq C_s$, sporadic request completed within two period task periods (one for waiting, one for executing)
  - $2*T_s \leq D_{sp}$
- Arbitrary execution times:
  - $T_s + \left\lceil \frac{C_{sp}}{C_s} \right\rceil T_s \leq D_{sp}$
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 if \( D_i \leq T_i \)
  \[
  \forall i \quad R_i \leq D_i \quad \text{with} \quad R_i = C_i + \sum_{j \in h(i)} \left[ \frac{R_i}{T_j} \right] C_j
  \]

- EDF necessary and sufficient condition for general task set (D and T unrelated)
  \[
  \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \quad \text{and} \quad \sum_{D_i \leq d} \left( 1 + \frac{d - D_i}{T_i} \right) C_i \leq d \quad \text{for} \quad 0 < d \leq L
  \]

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 background
  - 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 (\( P_i < P_{s} \) if \( i > k \))
  - \( RT_i \): replenishment instant for priority level \( P_i \)
  - Active: a given priority level \( P_i \) is active if \( P_i \leq P_s \)
  - 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**

![Example Diagram](Diagram)

**Scheduling 69**

**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 $\geq \lceil \text{period/\text{min_interarrival\_time}} \rceil$
- Use the deadline oriented versions

**Scheduling 70**
**Precedence Constraints**

- Tasks are preemptive or not
- All tasks characteristics are known 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 share 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:

$$d_i^* = \min \left\{ d_i, \min_{S_i \rightarrow S_j} \left( d_j^* - C_j \right) \right\}$$

and

$$r_i^* = \max \left\{ r_i, \max_{S_j \rightarrow S_i} \left( r_j^* + C_j \right) \right\}$$

$S_i \rightarrow S_j$ 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].

### DoVer 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 $r \ (0 \leq r \leq 1)$ if it guarantees to obtain a cumulative value at least $r$ times the one achieved by clairvoyant one.
Overloads - Some Results

- Best competitive factor is: \[ \frac{1}{(1 + \sqrt{k})^2} \]
- 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.

Dover 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 \( t: r_i \leq t \leq d_i \); not completed nor abandoned

Dover Algorithm - Definitions (2)

- \( \Gamma \) collection of tasks \( T_1,T_2 ... T_n \)
- smallest importance of a task in \( \Gamma \) 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
Dover Algorithm - Definitions (3)

- $\Gamma = \{\text{current}, \text{recently-preempted } \Gamma_{rp}, \text{ others } \Gamma_o\}$
- a preempted task goes into $\Gamma_{rp}$
- if a task is scheduled under latest_start_time event, all ready tasks go into $\Gamma_o$
- $\text{AvailT}$: 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 $\Gamma_{rp}$ to miss their deadlines

Dover Algorithm - Variables

- 2 queues ordered by deadline: $Q_{recent}, Q_{other}$
- $Q_{lst}$ (all tasks) ordered by latest start time
- recentval: running value of tasks in $Q_{recent}$
- now(): returns current time
- $Laxity(T_i) = d_i - [\text{now()+remain_{comp\_time}(T_i)]}$
- Schedule($T_i$): gives the processor to task $T_i$
- $Q_{recent} = (T, \text{PreviousTime}, \text{PreviousAvail})$

Dover - The Algorithm

- task completion
  - if $Q_{recent} \neq \emptyset$ & $Q_{other} \neq \emptyset$
    - $(T_{qrec}, t_{prev}, \text{availprev}) = \text{Min}(Q_{recent}); \text{AvailT} = \text{availprev} - (\text{now}-t_{prev}); T_{qother} = \text{Min}(Q_{other})$
    - if $d(T_{qother}) < d(T_{qrec})$ & $\text{AvailT} \geq \text{remain}(T_{qother})$
      - Dequeue($Q_{other}$); $\text{AvailT} = \text{AvailT} - \text{remain}(T_{qother})$; $\text{AvailT} = \text{min}(\text{AvailT}, \text{laxity}(T_{qother}))$; schedule($T_{qother}$);
    - else
      - Dequeue($Q_{recent}$); currentval = currentval-value($T_{qrecent}$); schedule($T_{qrecent}$);
  - if $Q_{recent} = \emptyset$ & $Q_{other} \neq \emptyset$
    - $T_{cur} = \text{Dequeue}(Q_{other}); \text{AvailT} = \text{laxity}(T_{cur}); \text{schedule}(T_{cur})$
  - if $Q_{recent} \neq \emptyset$ & $Q_{other} = \emptyset$
    - $T_{cur} = \text{Dequeue}(Q_{recent}); \text{AvailT} = \text{availprev} - (\text{now}-t_{prev}); \text{recentval} = \text{recentval-value}(T_{cur}); \text{schedule}(T_{cur})$
  - if $Q_{recent} = \emptyset$ & $Q_{other} = \emptyset$
    - $\text{idle} = \text{true}; \text{AvailT} = \infty$

Dover - The Algorithm (2)
Dover - 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

Dover - 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;

Dover 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 ($\tau_1 .. \tau_n$)
- tasks are grouped in m groups ($g_1 .. g_m$) in decreasing priority order
- necessary and sufficient condition ($\tau_i$ in $g_k$, r tasks in groups of higher priority, $D_t \leq T_i$)

$$\min_{0 < t \leq D_i} W_i(t)/t \leq 1 \text{ with } W_i(t) = \sum_{j=1}^{r} C_j \left[ \frac{t}{T_{ji}} \right] + \sum_{l \in g_k} C_l$$

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

Exercise 1

<table>
<thead>
<tr>
<th>Task</th>
<th>T</th>
<th>D</th>
<th>C</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>10</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>15</td>
<td>15</td>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>C</td>
<td>30</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Exercise 2

<table>
<thead>
<tr>
<th>Task</th>
<th>T</th>
<th>D</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>15</td>
<td>15</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>30</td>
<td>30</td>
<td>10</td>
</tr>
</tbody>
</table>

Continuous Media (CM) activities need real-time support

- execution times vary widely
- difficult to estimate WCET
- interarrival time not deterministic
- rather dynamic

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 $\tau_i$ $(C_i, D_i)$ 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 if $U_p + U_s \leq 1$ ($U_s = Q_s / T_s$)
- $U_p = \sum (C_i / T_i)$ 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
  - r2,1 = 0, C2,1 = 2
  - r2,2 = 7, C2,2 = 3
  - r2,3 = 14, C2,3 = 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 \( \tau = \{S_i\} \) or \( \{P_i\} \)
  - Concrete task set \( \omega = \{S_i, r_i\} \) or \( \{P_i, r_i\} \)
  - A task set can be scheduled if all sets \( \omega \) are

Non Preemptive EDF (2)

- **Universality**
  - An algorithm is universal for a concrete task set if it can schedule all schedulable sets

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
  \[
  \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \quad \forall i : 1 < i \leq n; \quad \forall L : T_1 < L < T_i; L \geq C_i + \sum_{k=1}^{i-1} \left| \frac{L-1}{T_k} \right| C_k
  \]

- 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)

<table>
<thead>
<tr>
<th>Task</th>
<th>T</th>
<th>D</th>
<th>C</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>10</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>15</td>
<td>15</td>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>C</td>
<td>30</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

When a task ends or the processor is idle, start the execution of the task that has the closest deadline
Non Preemptive EDF: exercise

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

<table>
<thead>
<tr>
<th>Task</th>
<th>T</th>
<th>D</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>14</td>
<td>14</td>
<td>6</td>
</tr>
<tr>
<td>B</td>
<td>20</td>
<td>20</td>
<td>6</td>
</tr>
<tr>
<td>C</td>
<td>30</td>
<td>30</td>
<td>6</td>
</tr>
</tbody>
</table>

Many techniques
- independent, 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

Journal Articles

References (2)


