ARTIST2 Summer School 2008 in Europe
Autrans (near Grenoble), France
September 8-12, 2008

Real-Time Scheduling and Resource Management

Lecturer: Giorgio Buttazzo
Full Professor
Scuola Superiore Sant’Anna

http://www.artist-embedded.org/
Outline

- Importance of scheduling in embedded systems
- Review of main scheduling algorithms
- Schedulability analysis
- Taking into account shared resources
- Preemptive vs. Non preemptive scheduling
- Bounding delays and jitter
- Managing overloads
- Design issues: integrating RT and control theory
Typical task structure

buffer → <read data>

<process data>

<write data> → buffer

<wait for next activation>
Activation modes

**Periodic task** (time driven)
A task is automatically activated by the kernel at regular time intervals

**Aperiodic task** (event driven)
A task is activated upon the arrival of an event (interrupt or explicit activation)
Complex control applications

- Hierarchical design
- Many periodic activities running at different rates
- Many event-driven routines
Task scheduling

When more tasks are ready to execute, the order of execution is decided by the scheduler:
Importance of scheduling

- It affects task response times
- It affects delay and jitter in control loops
- It affects execution times (preemptions destroy cache data and prefetch queues)
- It can be used to cope with overload conditions
- It can be used to optimize resource usage
- It can be used to save energy in processors with voltage scaling (energy-aware scheduling)
Periodic Task Scheduling

We have \( n \) periodic tasks: \( \{\tau_1, \tau_2 \ldots \tau_n\} \)

\[ \tau_i (C_i, T_i, D_i) \]

- **Period**: \( T_i \)
- **Relative deadline**: \( r_{i,k} \)
- **Absolute deadline**: \( d_{i,k} \)

<table>
<thead>
<tr>
<th>( r_{i,1} = \Phi_i )</th>
<th>( r_{i,k} )</th>
<th>( d_{i,k} )</th>
<th>( r_{i,k+1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Goal**

- Execute all tasks within their deadlines
- Verify feasibility before runtime

\[
\begin{align*}
    r_{i,k} &= \Phi_i + (k-1) T_i \\
    d_{i,k} &= r_{i,k} + D_i
\end{align*}
\]
Optimal Priority Assignments

• **Rate Monotonic (RM):**
  \[ p_i \propto \frac{1}{T_i} \text{ (static)} \]
  \[ D_i = T_i \]

• **Deadline Monotonic (DM):**
  \[ p_i \propto \frac{1}{D_i} \text{ (static)} \]
  \[ D_i \neq T_i \]

• **Earliest Deadline First (EDF):**
  \[ p_i \propto \frac{1}{d_i} \text{ (dynamic)} \]
  \[ d_{i,k} = r_{i,k} + D_i \]
Basic results

Assumptions:

- Independent tasks
- $\Phi_i = 0$
- $D_i = T_i$

In 1973, Liu & Layland proved that a set of $n$ periodic tasks can be feasibly scheduled

under RM if

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

under EDF if and only if

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$
Schedulability bound

\[ \text{for } n \to \infty \quad U_{\text{lub}} \to \ln 2 \simeq 0.69 \]

CPU% 

<table>
<thead>
<tr>
<th>n</th>
<th>CPU%</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>90</td>
</tr>
<tr>
<td>3</td>
<td>80</td>
</tr>
<tr>
<td>4</td>
<td>70</td>
</tr>
<tr>
<td>5</td>
<td>60</td>
</tr>
<tr>
<td>6</td>
<td>50</td>
</tr>
<tr>
<td>7</td>
<td>40</td>
</tr>
<tr>
<td>8</td>
<td>30</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>
An unfeasible RM schedule

\[ U_p = \frac{3}{6} + \frac{4}{9} = 0.944 \]
EDF vs. RM Schedule

EDF

\[
\begin{align*}
\tau_1 & \quad 0 \rightarrow 3 \rightarrow 6 \rightarrow 9 \rightarrow 12 \rightarrow 15 \rightarrow 18 \\
\tau_2 & \quad 0 \rightarrow 3 \rightarrow 6 \rightarrow 9 \rightarrow 12 \rightarrow 15 \rightarrow 18
\end{align*}
\]

RM

\[
\begin{align*}
\tau_1 & \quad 0 \rightarrow 3 \rightarrow 6 \rightarrow 9 \rightarrow 12 \rightarrow 15 \rightarrow 18 \\
\tau_2 & \quad 0 \rightarrow 3 \rightarrow 6 \rightarrow 9 \rightarrow 12 \rightarrow 15 \rightarrow 18
\end{align*}
\]

deadline miss
Schedulability region

A more useful approach is to identify a region in the space of task parameters where the system is schedulable by an algorithm.
Schedulability region

The U-space

\[ \sum_{i=1}^{n} U_i \leq 1 \]

\[ \sum_{i=1}^{n} U_i \leq n(2^{1/n} - 1) \]
Schedulability region

The U-space

<table>
<thead>
<tr>
<th></th>
<th>$C_i$</th>
<th>$T_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>

\[ U_p = \frac{3}{6} + \frac{4}{9} = 0.94 \]
The Hyperbolic Bound

- In 2000, Bini et al. proved that a set of $n$ periodic tasks is schedulable with RM if:

$$\prod_{i=1}^{n} (U_i + 1) \leq 2$$
Schedulability region

The U-space

\[ \sum_{i=1}^{n} U_i \leq 1 \]

\[ \sum_{i=1}^{n} U_i \leq n(2^{1/n} - 1) \]
The U-space

Schedulability region

\[ \sum_{i=1}^{n} U_i \leq 1 \]

\[ \sum_{i=1}^{n} U_i \leq n(2^{1/n} - 1) \]

\[ \prod_{i=1}^{n} (U_i + 1) \leq 2 \]
Handling tasks with D < T

Scheduling algorithms

- Deadline Monotonic: $p_i \propto 1/D_i$ (static)
- Earliest Deadline First: $p_i \propto 1/d_i$ (dynamic)
How to guarantee feasibility?

- **Fixed priority**: Response Time Analysis (RTA)
- **EDF**: Processor Demand Criterion (PDC)
Response Time Analysis

[Audsley, 1990]

- For each task $\tau_i$ compute the interference due to higher priority tasks:
  \[ I_i = \sum_{D_k < D_i} C_k \]

- Compute its response time as
  \[ R_i = C_i + I_i \]

- Verify if
  \[ R_i \leq D_i \]
Computing the interference

Interference of $\tau_k$ on $\tau_i$ in the interval $[0, R_i]$: 

$$I_{ik} = \left\lceil \frac{R_i}{T_k} \right\rceil C_k$$

Interference of high priority tasks on $\tau_i$: 

$$I_i = \sum_{k=1}^{i-1} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$$
Computing the response time

\[ R_i = C_i + \sum_{k=1}^{i-1} \left( \frac{R_i}{T_k} \right) C_k \]

Iterative solution:

\[
\begin{align*}
R_i^{(0)} &= C_i \\
R_i^{(s)} &= C_i + \sum_{k=1}^{i-1} \left( \frac{R_i^{(s-1)}}{T_k} \right) C_k
\end{align*}
\]

iterate until \( R_i^{(s)} > R_i^{(s-1)} \)
Processor Demand Criterion

[Baruah, Howell, Rosier 1990]

For checking the existence of a feasible schedule under EDF

In any interval of time, the computation demanded by the task set must be no greater than the available time.

\[ \forall t_1, t_2 > 0, \quad g(t_1, t_2) \leq (t_2 - t_1) \]
The demand in \([t_1, t_2]\) is the computation time of those jobs started at or after \(t_1\) with deadline less than or equal to \(t_2\): 

\[
g(t_1, t_2) = \sum_{r_i \geq t_1}^{d_i \leq t_2} C_i
\]
Processor Demand

For synchronous task sets we can only analyze intervals \([0,L]\)

\[
g(0, L) = \sum_{i=1}^{n} \left[ \frac{L - D_i + T_i}{T_i} \right] C_i
\]
Processor Demand Test

\[ \forall L > 0 \quad \sum_{i=1}^{n} \left| \frac{L - D_i + T_i}{T_i} \right| C_i \leq L \]
Example

$\tau_1$

$\tau_2$

$g(0, L)$
## Summarizing: RM vs. EDF

<table>
<thead>
<tr>
<th></th>
<th>RM</th>
<th>EDF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$D_i = T_i$</td>
<td>$D_i \leq T_i$</td>
</tr>
<tr>
<td></td>
<td><strong>Suff.: polynomial</strong> $\mathcal{O}(n)$</td>
<td><strong>pseudo-polynomial</strong></td>
</tr>
<tr>
<td></td>
<td>LL: $\sum U_i \leq n(2^{1/n} - 1)$</td>
<td><strong>Response Time Analysis</strong></td>
</tr>
<tr>
<td></td>
<td>HB: $\Pi(U_i+1) \leq 2$</td>
<td>$\forall_i R_i \leq D_i$</td>
</tr>
<tr>
<td></td>
<td><strong>Exact pseudo-polynomial</strong></td>
<td><strong>RTA</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Processor Demand Analysis</strong></td>
<td>$R_i = C_i + \sum_{k=1}^{i-1} \left\lfloor \frac{R_i}{T_k} \right\rfloor C_k$</td>
</tr>
<tr>
<td></td>
<td>$\sum U_i \leq 1$</td>
<td><strong>pseudo-polynomial</strong> $\mathcal{O}(n)$</td>
</tr>
<tr>
<td></td>
<td>$\forall L &gt; 0$, $g(0,L) \leq L$</td>
<td><strong>polynomial:</strong> $\mathcal{O}(n)$</td>
</tr>
</tbody>
</table>
Handling shared resources

Problems caused by mutual exclusion
Critical sections

\[ \tau_1 \]

wait(s)
\[ x = 3; \]
\[ y = 5; \]
signal(s)

\[ \tau_2 \]

wait(s)
\[ a = x+1; \]
\[ b = y+2; \]
signal(s)

Write
\[ \text{int } x; \]
\[ \text{int } y; \]

Read
\[ c = x+y; \]
Blocking on a semaphore

It seems that the maximum blocking time for $\tau_1$ is equal to the length of the critical section of $\tau_2$, but …
Priority Inversion

Occurs when a high priority task is blocked by a lower-priority task for an unbounded interval of time.
Resource Access Protocols

Under fixed priorities

- Non Preemptive Protocol (NPP)
- Highest Locker Priority (HLP)
- Priority Inheritance (PIP)  [Sha-Rajkumar-Lehoczcky, 90]
- Priority Ceiling (PCP)     [Sha-Rajkumar-Lehoczcky, 90]

Under EDF

- Non Preemptive Protocol (NPP)
- Dynamic Priority Inheritance (D-PIP) [Spuri, 98]
- Dynamic Priority Ceiling (D-PCP)    [Chen-Lin, 90]
- Stack Resource Policy (SRP)         [Baker, 90]
Guarantee when $D = T$

- Compute the maximum blocking time for each task
- Inflate $C_i$ by $B_i$

**Extended LL test:**

#### RM

$$\forall i \quad \sum_{k=1}^{i-1} \frac{C_k}{T_k} + \frac{C_i + B_i}{T_i} \leq i(2^{1/i} - 1)$$

#### EDF

$$\forall i \quad \sum_{k=1}^{i-1} \frac{C_k}{T_k} + \frac{C_i + B_i}{T_i} \leq 1$$
Guarantee when $D \leq T$

Under **DM** a task set is schedulable if $\forall i \ R_i \leq D_i$

$$R_i = C_i + B_i + \sum_{k=1}^{i-1} \left( \frac{R_i}{T_k} \right) C_k$$

Under **EDF** a task set is schedulable if $U < 1$ and

$$\forall i \ \forall L \quad B_i + \sum_{k=1}^{n} \left( \frac{L + T_k - D_k}{T_k} \right) C_k \leq L$$
Non-preemptive scheduling

It is a special case of preemptive scheduling where all tasks share a single resource for their entire duration.

The max blocking time for task $\tau_i$ is given by the largest $C_k$ among the lowest priority tasks:

$$B_i = \max\{C_k : P_k < P_i\}$$
Advantages of NP scheduling

• It reduces runtime overhead
  - Less context switches
  - No semaphores are needed for critical sections

• It reduces stack size, since no more than one task can be in execution.

• It preserves program locality, improving the effectiveness of
  - Cache memory
  - Pipeline mechanisms
  - Prefetch queues
Advantages of NP scheduling

- As a consequence, task execution times are
  - Smaller
  - More predictable
Advantages of NP scheduling

In fixed priority systems can improve schedulability:

\[ U = \frac{2}{5} + \frac{4}{7} \approx 0.97 \]

RM

NP-RM

deadline miss
Disadvantages of NP scheduling

- In general, NP scheduling reduces schedulability.
- The utilization bound under non preemptive scheduling drops to zero:

\[ U = \frac{\varepsilon}{T_1} + \frac{C_2}{\infty} \rightarrow 0 \]
Trade-off solutions

Tunable Preemptive Systems

- Compute the longest non-preemptive section that allows a feasible schedule [Baruah-Bertogna, 08].
- Allow preemption only in certain points in the code.
Handling Jitter & Delay

Jitter for an event

The maximum time variation in the occurrence of a particular event in two consecutive jobs.

In many control applications, delay and jitter can cause instability or jerky behavior.
Definitions

Start time delay (Input Latency): \( \text{INL}_{i,k} = s_{i,k} - r_{i,k} \)

\[
\tau_i \\
\begin{array}{cccccc}
& r_{i,1} & S_{i,1} & r_{i,2} & S_{i,2} & r_{i,3} & S_{i,3} \\
\end{array}
\]

Start time Jitter (Input Jitter):

Absolute: \( \text{INJ}_{i}^{\text{abs}} = \max_k (s_{i,k} - r_{i,k}) - \min_k (s_{i,k} - r_{i,k}) \)

Relative: \( \text{INJ}_{i}^{\text{rel}} = \max_k \left| (s_{i,k} - r_{i,k}) - (s_{i,k-1} - r_{i,k-1}) \right| \)
Definitions

Response Time (Output Latency): \( R_{i,k} = f_{i,k} - r_{i,k} \)

Response Time Jitter (Output Jitter):

Absolute: \( RTJ_{i}^{\text{abs}} = \max_k (f_{i,k} - r_{i,k}) - \min_k (f_{i,k} - r_{i,k}) \)

Relative: \( RTJ_{i}^{\text{rel}} = \max_k \left| (f_{i,k} - r_{i,k}) - (f_{i,k-1} - r_{i,k-1}) \right| \)
**Definitions**

**Input-Output Latency:** \( IOL_{i,k} = f_{i,k} - s_{i,k} \)

**Input-Output Jitter:**

**Absolute:**
\[
IOJ_{i}^{\text{abs}} = \max_{k} (f_{i,k} - s_{i,k}) - \min_{k} (f_{i,k} - s_{i,k})
\]

**Relative:**
\[
IOJ_{i}^{\text{rel}} = \max_{k} \left| (f_{i,k} - s_{i,k}) - (f_{i,k-1} - s_{i,k-1}) \right|
\]
Cause of delays and jitter

- task parameters
- number of tasks
- total load
- activation phases
- scheduling algorithm
Jitter under RM

Low priority tasks experience very high delay and jitter
Jitter under EDF

For a little increase of RTJ\(_1\), RTJ\(_3\) decreases a lot

IOJ = 0 for all the tasks
Jitter under RM and EDF

Normalized Avg. RTJ

U = 0.9   N = 10

Task number (ordered by increasing periods)
How to handle delay and jitter

Two main methods can be used to reduce the effect of delay and jitter:

1. compensate them by proper control actions;
2. reduce them as much as possible.

Even when compensation is used, reducing delay and jitter improves system performance.

Hence we concentrate on reduction methods.
Jitter Reduction methods

Three methods can be used to reduce the jitter caused by task interference:

1. Task Splitting
2. Advancing Deadlines
3. Non Preemptive Scheduling
Reducing Jitter by Task Splitting

The idea is to force input and output parts to execute in a time-triggered fashion, using timers:
Reducing Jitter by Task Splitting

Advantages

1. Jitter is reduced at the minimum possible value;

2. If input and output parts are small, this method is effective for any task, independently of the scheduler and task parameters.
Reducing Jitter by Task Splitting

Disadvantages

1. Extra effort to be implemented;
2. Jitter is reduced at the expense of delay;
3. Input and output parts create extra interference which complicates the analysis and reduces schedulability;
4. Input and output parts may compete and need to be scheduled with some policy.
Reducing Jitter by Task Splitting

Interfering I/O parts

\(\tau_1\)

\(\tau_2\)

\(\tau_3\)

\(\tau_4\)
Reducing Jitter by Advancing Deadlines

The idea is to advance task deadlines to reduce the active window in which jobs can be executed:
Reducing Jitter by Advancing Deadlines

Advantages

1. Easy to implement (no special support is required from the OS);

2. No extra interference caused by additional timer interrupts;

3. Both delay and jitter are reduced!!
Reducing Jitter by Advancing Deadlines

Disadvantages

1. Not all tasks can reduce jitter to zero. A further reduction can be achieved by proper offsets, but the analysis requires exponential complexity.

2. Advancing deadlines reduces system schedulability.
Reducing Jitter by Non Preemption

Disabling preemtions a task can be delayed, but once started cannot be interrupted:

\[ \forall k \left\{ \begin{array}{l}
\text{IOL}_{i,k} = C_i \\
\text{IOJ}_{i,k} = 0
\end{array} \right. \]
Reducing Jitter by Non Preemption

Example with 3 tasks
Reducing Jitter by Non Preemption

Advantages

1. IOJ$_i$ = 0 for all tasks;

2. IOL$_i$ = C$_i$ for all tasks, simplifying the use of delay compensation techniques;

3. Non preemptive execution also simplifies resource management (there is no need to protect critical sections).

4. Non preemptive execution allows stack sharing.
Reducing Jitter by Non Preemption

Disadvantages

1. Non preemption reduces schedulability (analysis must take blocking times into account);

2. The utilization upper bound drops to zero:

\[ U_1 \to 0 \]
\[ U_2 \to 0 \]
\[ C_1 \to 0 \]
\[ C_2 > T_1 \]
\[ T_2 \to \infty \]
Scheduling under overload conditions
RM under overloads

\[ U = \frac{4}{8} + \frac{6}{12} + \frac{5}{20} = 1.25 \]

- High priority tasks execute at the proper rate
- Low priority tasks are completely blocked
EDF under overloads

\[ U = \frac{4}{8} + \frac{6}{12} + \frac{5}{20} = 1.25 \]

- All tasks execute at a slower rate
- No task is blocked
EDF under overloads

**Theorem (Cervin ‘03)**

If $U > 1$, EDF executes tasks with an average period $T'_i = T_i U$. 

$U = 1.25$

<table>
<thead>
<tr>
<th>$\tau_1$</th>
<th>$\tau_2$</th>
<th>$\tau_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_i$</td>
<td>$T'_i$</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>25</td>
<td></td>
</tr>
</tbody>
</table>
Exploiting control flexibility

Relaxing timing constraints

• The idea is to reduce the load by increasing deadlines and/or periods.

• Each task must specify a range of values in which its period must be included.

• Periods are increased during overloads, and reduced when the overload is over.

Many control applications allow timing flexibility
Examples: altimeter reading

- The smaller the altitude, the higher the acquisition rate:
Obstacle avoidance

- The closer the obstacle, the higher the acquisition rate:
Engine control

- Some tasks need to be activated at specific angles of the motor axis:
  \[ \Rightarrow \text{the higher the speed, the higher the rate}. \]
Elastic task model

- A periodic task $\tau_i$ is characterized by:
  $$(C_i, T_{i-min}, T_{i-max}, E_i)$$

- Tasks’ utilizations are treated as elastic springs

- The resistance of a task to a period variation is controlled by an **elastic coefficient** $E_i$
Compression algorithm

During overloads, utilizations must be compressed to bring the load below a desired value $U_d$. 
Solution for tasks

\[ U_i = U_{io} - (U_0 - U_d) \frac{E_i}{E_s} \]

then:

\[ T_i = \frac{C_i}{U_i} \]
Solution with constraints

Iterative solution \( O(n^2) \):
Control design issues

Design of control laws

Mapping to periodic tasks

Schedulability analysis

Feasible?

Yes

Implementation

Run time monitoring

Meet constraints?

No

Yes

Traditional approach

Disadvantages

➢ Repetitive development process

➢ Suboptimal performance

➢ Suboptimal use of the resources
Control performance

Performance as a function of periods
Sensitivity Analysis

Feasible region
RT & Control codesign

Performance

$T_1$ $T_2$
Codesign as optimization

Generic control law

System performance characterization

Resource constraints characterization

Optimization process

Sampling period

Delay

Jitter

Performance

Task parameters
Conclusions

When designing complex embedded systems:

1. Split your system in components, follow a modular design, with hierarchical control levels with precise interface:
Conclusions

2. Organize software as a set of control tasks with precise timing and resource constraints:
Conclusions

3. Estimate worst-case computation times of tasks, using specific tools and testing.

4. Select an appropriate scheduling algorithm and a suitable resource access protocol.

5. Estimate maximum blocking times due non preemptive sections or mutually exclusive resources.

6. Apply schedulability analysis to verify feasibility.
Conclusions

7. If possible, use sensitivity analysis and integrate real-time with control issues at early design stages.

8. Exploit system flexibility defining admissible ranges of parameters to cope with overloads.

9. Perform extensive testing and simulation at each implementation step under worst-case scenarios.