Real-time Scheduling

Klaus-Peter Zauner

COMP2215: Computer Systems II
Real-time: Deadlines

- Tasks have deadlines
- Tasks can be periodic or aperiodic
- Task duration can be constant or not

**Deadline**

The latest time by which a task has to be completed.

- Often require predictable behaviour $\rightarrow$ guarantees
CPU Utilization $U$

\[ U = c_{\text{total}} - c_{\text{idle}} \leq 1 \]

$c_{\text{total}} \rightarrow$ Total CPU time available (100%)
$c_{\text{idle}} \rightarrow$ Fraction of CPU time spent in idle task or sleeping
Assumptions for Analysis

1. Tasks are periodic
   ▶ convert aperiodic tasks by polling

2. The deadline for a task is its next invocation

3. Context switches take no time
   ▶ leave some margin in duration and deadline
Load from Set of Periodic Tasks

\[ U = c_{\text{total}} - c_{\text{idle}} \leq 1 \]

Given a set of tasks \( T_1 \cdots T_n \) with periodicity \( p_i \) and fixed CPU time \( c_i \) for \( T_i \) the utilization is:

\[ U = \sum_{i=1}^{n} \frac{c_i}{p_i} \]
Schedulability

Requirement: All tasks meet their deadlines all the time

A Real-time System is schedulable if

\[ U = \sum_{i=1}^{n} \frac{c_i}{p_i} \leq 1 \]

Assuming periodic tasks, with the next invocation as deadline and no overhead for context switching.
How to schedule it?
How to schedule it?

... we need **predictable** worst case performance.
All Tasks can finish within the period of the most frequent task. ⇒ Static cyclic scheduling possible: use a table to assign time slots to tasks.
Priority

1. Fixed (static) Priority
⇒ assigned at compile time

2. Dynamic Priority
⇒ changes during runtime
Priority

1. Fixed (static) Priority
   ⇒ assigned at compile time

2. Dynamic Priority
   ⇒ changes during runtime
Fixed Priority Scheduling

- Simple
  - Code (and changes) relatively easy to verify
- Well understood
  - Confidence in behaviour

Optimal fixed priority scheme: Rate Monotonic Scheduling
Rate Monotonic Scheduling

Requirements for RMS

1. Tasks are independent
   - no blocking for each other
2. Tasks have fixed CPU requirement
3. Free context switching
4. Deadline is task period
Rate Monotonic Scheduling

If the conditions for RMS are met, than it is optimal to assign fixed priorities according to the period:

- **Most frequent Task has highest priority**

Guaranteed scheduling for

$$ U = \sum_{i=1}^{n} \frac{c_i}{p_i} \leq n\left(2^{1/n} - 1\right) $$

$$ n \rightarrow \infty \Rightarrow U \leq \ln 2 \approx 69\% $$

Liu and Layland, 1973
Rate Monotonic Scheduling

If the conditions for RMS are met, than it is optimal to assign fixed priorities according to the period:

- **Most frequent Task has highest priority**

Guaranteed scheduling for

\[
U = \sum_{i=1}^{n} \frac{c_i}{p_i} \leq n\left(2^{1/n} - 1\right)
\]

\[
n \rightarrow \infty \Rightarrow U \leq \ln 2 \approx 69\%
\]

Liu and Layland, 1973
Rate Monotonic Scheduling

If the conditions for RMS are met, than it is optimal to assign fixed priorities according to the period:

- **Most frequent Task has highest priority**

Guaranteed scheduling for

\[
U = \sum_{i=1}^{n} \frac{c_i}{p_i} \leq n (2^{1/n} - 1)
\]

\[n \to \infty \implies U \leq \ln 2 \approx 69\%
\]

Liu and Layland, 1973
RMS Priority

Example

<table>
<thead>
<tr>
<th>Task</th>
<th>Importance</th>
<th>Frequency</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Very high (critical)</td>
<td>2 Hz</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>High</td>
<td>0.1 Hz</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>Medium</td>
<td>50 Hz</td>
<td></td>
</tr>
</tbody>
</table>

Note: The priority in RMS is directly derived from the frequency of the task—and has nothing to do with the importance of the task!
## RMS Priority

### Example

<table>
<thead>
<tr>
<th>Task</th>
<th>Importance</th>
<th>Frequency</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Very high (critical)</td>
<td>2 Hz</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>High</td>
<td>0.1 Hz</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>Medium</td>
<td>50 Hz</td>
<td>0 high</td>
</tr>
</tbody>
</table>

**Note:** The priority in RMS is directly derived from the frequency of the task—and has nothing to do with the importance of the task!
RMS is optimal for fixed priorities

- Use RMS, your own priority assignment cannot outperform it

- What if your task set does not satisfy the $U$ bound?
RMS is optimal for fixed priorities

- Use RMS, your own priority assignment cannot outperform it

- What if your task set does not satisfy the $U$ bound?
RMS is optimal for fixed priorities

- Use RMS, your own priority assignment cannot outperform it
- What if your task set does not satisfy the $U$ bound?

To guarantee scheduling for any task set that satisfies the conditions the bound has to assume a worst-case task set.

- your task set may still be RMS schedulable
- but you don’t get the guarantee that it is
- requires analysis of specific task set
Wasting 30% of the CPU time?

- Run non-real time tasks as low priority in remaining CPU time
- Optimize task periods to achieve a regular execution pattern
Wasting 30% of the CPU time?

- Run non-real time tasks as low priority in remaining CPU time
- Optimize task periods to achieve a regular execution pattern
Harmonic Task Sets I

⇒ Every task period is a multiple of the period of any higher priority task

<table>
<thead>
<tr>
<th>Non-harmonic</th>
<th>Harmonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1 = 17$ ms</td>
<td>$p_1 = 15$ ms</td>
</tr>
<tr>
<td>$p_2 = 31$ ms</td>
<td>$p_2 = 30$ ms</td>
</tr>
<tr>
<td>$p_3 = 130$ ms</td>
<td>$p_3 = 120$ ms</td>
</tr>
</tbody>
</table>
Harmonic Task Sets I

⇒ Every task period is a multiple of the period of any higher priority task

<table>
<thead>
<tr>
<th>Non-harmonic</th>
<th>Harmonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1 = 17$ ms</td>
<td>$p_1 = 15$ ms</td>
</tr>
<tr>
<td>$p_2 = 31$ ms</td>
<td>$p_2 = 30$ ms</td>
</tr>
<tr>
<td>$p_3 = 130$ ms</td>
<td>$p_3 = 120$ ms</td>
</tr>
</tbody>
</table>
Harmonic Task Sets I

⇒ Every task period is a multiple of the period of any higher priority task

<table>
<thead>
<tr>
<th>Non-harmonic</th>
<th>Harmonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1 = 17$ ms</td>
<td>$p_1 = 15$ ms</td>
</tr>
<tr>
<td>$p_2 = 31$ ms</td>
<td>$p_2 = 30$ ms</td>
</tr>
<tr>
<td>$p_3 = 130$ ms</td>
<td>$p_3 = 120$ ms</td>
</tr>
</tbody>
</table>
Harmonic Task Sets II

- If possible make your task sets harmonic and schedule with RMS
- RMS can reach 100% utilization with harmonic task sets
- Harmonic task sets are easy to analyze
  - regular execution pattern
- Note: shorter deadlines may be better!
  - decrease periods to make harmonic
Harmonic Task Sets II

▶ If possible make your task sets harmonic and schedule with RMS

▶ RMS can reach 100% utilization with harmonic task sets

▶ Harmonic task sets are easy to analyze
  ▶ regular execution pattern

▶ Note: shorter deadlines may be better!
  → decrease periods to make harmonic
RMS Issues to consider...

- What happens in RMS scheduling in an overload situation?
- Are the assumptions made for RMS analysis realistic?
- Deadlines can be used instead of periods → useful if deadlines are earlier than next period
RMS Issues to consider...

- What happens in RMS scheduling in an overload situation?
- Are the assumptions made for RMS analysis realistic?
- Deadlines can be used instead of periods
  → useful if deadlines are earlier than next period
RMS Issues to consider...

- What happens in RMS scheduling in an overload situation?
- Are the assumptions made for RMS analysis realistic?
- Deadlines can be used instead of periods
  → useful if deadlines are earlier than next period
Dynamic Priority Scheduling
Earliest Deadline First

Is a high utilization possible with non-harmonic task sets?
Earliest Deadline First

Is a high utilization possible with non-harmonic task sets?

- Dynamic Change of Priority
- Run most urgent task first
- Higher complexity in scheduler
- Utilization up to 100% possible
- With and without preemption
EDF

\[ U = \sum_{i=1}^{n} \frac{c_i}{p_i} \leq 1 \]

- Scheduler more complicated
- Scheduler has more overhead
- Can handle changing importance of tasks
- Can accommodate new tasks at runtime
- Can handle variable execution times
- Not stable under overload
- EDF is optimal