Real-time Scheduling of Periodic Tasks (2)

Advanced Operating Systems
Lecture 3
Lecture Outline

- The rate monotonic algorithm (cont’d)
  - ...
  - Maximum utilisation test

- The deadline monotonic algorithm

- The earliest deadline first algorithm
  - Definition
  - Optimality
  - Maximum utilisation test

- The least slack time algorithm

- Discussion
Rate Monotonic: Other Scheduling Tests

• Exhaustive simulation and time-demand analysis complex and error prone

• Simple scheduling tests derived for some cases:
  • Simply periodic systems
  • Maximum utilisation test
Simply Periodic Systems

• In a *simply periodic* system, the periods of all tasks are integer multiples of each other
  
  • \( p_k = n \cdot p_i \) for all \( i, k \) such that \( p_i < p_k \) where \( n \) is a positive integer
  
  • True for many real-world systems, since easy to engineer around multiples of a single run loop
Simply Periodic Rate Monotonic Tasks

- Rate monotonic optimal for simply periodic systems
  - A set of *simply periodic*, independent, preemptable tasks with $D_i \geq p_i$ can be scheduled on a single processor using RM provided $U \leq 1$

- Proof follows from time-demand analysis:
  - A simply periodic system, assume tasks in phase
    - Worst case execution time occurs when tasks in phase
  - $T_i$ misses deadline at time $t$ where $t$ is an integer multiple of $p_i$
    - Again, worst case $\Rightarrow D_i = p_i$
  - Simply periodic $\Rightarrow t$ integer multiple of periods of all higher priority tasks
  - Total time required to complete jobs with deadline $\leq t$ is $\sum_{k=1}^{i} \frac{e_k}{p_k} t = t \cdot U_i$
  - Only fails when $U_i > 1$
Maximum Utilisation Tests

• Simply periodic systems have a simple *maximum utilisation* test

• Possible to generalise the result to general rate monotonic systems
  • Derive a maximum utilisation, such that it is guaranteed a feasible schedule exists provided the maximum is not exceeded
RM Maximum Utilisation Test: $D_i = p_i$

- A system of $n$ independent preemptable periodic tasks with $D_i = p_i$ can be feasibly scheduled on one processor using rate monotonic if $U \leq n \cdot (2^{1/n} - 1)$

- $U_{RM}(n) = n \cdot (2^{1/n} - 1)$
- For large $n \rightarrow \ln 2$
  (i.e., $n \rightarrow 0.69314718056…$)

See Jane W. S. Liu, “Real-time systems”, Section 6.7 for proof

- $U \leq U_{RM}(n)$ is a sufficient, but not necessary, condition – i.e., a feasible rate monotonic schedule is guaranteed to exist if $U \leq U_{RM}(n)$, but might still be possible if $U > U_{RM}(n)$
RM Maximum Utilisation Test: $D_i = v \cdot p_i$

- Maximum utilisation varies if relative deadline and period differ
- For $n$ tasks, where the relative deadline $D_k = v \cdot p_k$ it can be shown that:

$$U_{RM}(n, v) = \begin{cases} 
  v & \text{for } 0 \leq v \leq 0.5 \\
  n((2v)^{\frac{1}{n}} - 1) + 1 - v & \text{for } 0.5 \leq v \leq 1 \\
  v(n - 1)[(\frac{v+1}{v})^{\frac{1}{n}} - 1] & \text{for } v = 2, 3, \ldots
\end{cases}$$

(you are not expected to remember this formula – but should understand how the utilisation changes in general terms)
**RM Maximum Utilisation Test:** $D_i = v \cdot p_i$

<table>
<thead>
<tr>
<th>$n$</th>
<th>$v = 4.0$</th>
<th>$v = 3.0$</th>
<th>$v = 2.0$</th>
<th>$v = 1.0$</th>
<th>$v = 0.9$</th>
<th>$v = 0.8$</th>
<th>$v = 0.7$</th>
<th>$v = 0.6$</th>
<th>$v = 0.5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.944</td>
<td>0.928</td>
<td>0.898</td>
<td>0.828</td>
<td>0.783</td>
<td>0.729</td>
<td>0.666</td>
<td>0.590</td>
<td>0.500</td>
</tr>
<tr>
<td>3</td>
<td>0.926</td>
<td>0.906</td>
<td>0.868</td>
<td>0.779</td>
<td>0.749</td>
<td>0.708</td>
<td>0.656</td>
<td>0.588</td>
<td>0.500</td>
</tr>
<tr>
<td>4</td>
<td>0.917</td>
<td>0.894</td>
<td>0.853</td>
<td>0.756</td>
<td>0.733</td>
<td>0.698</td>
<td>0.651</td>
<td>0.586</td>
<td>0.500</td>
</tr>
<tr>
<td>5</td>
<td>0.912</td>
<td>0.888</td>
<td>0.844</td>
<td>0.743</td>
<td>0.723</td>
<td>0.692</td>
<td>0.648</td>
<td>0.585</td>
<td>0.500</td>
</tr>
<tr>
<td>6</td>
<td>0.909</td>
<td>0.884</td>
<td>0.838</td>
<td>0.734</td>
<td>0.717</td>
<td>0.688</td>
<td>0.646</td>
<td>0.585</td>
<td>0.500</td>
</tr>
<tr>
<td>7</td>
<td>0.906</td>
<td>0.881</td>
<td>0.834</td>
<td>0.728</td>
<td>0.713</td>
<td>0.686</td>
<td>0.644</td>
<td>0.584</td>
<td>0.500</td>
</tr>
<tr>
<td>8</td>
<td>0.905</td>
<td>0.878</td>
<td>0.831</td>
<td>0.724</td>
<td>0.709</td>
<td>0.684</td>
<td>0.643</td>
<td>0.584</td>
<td>0.500</td>
</tr>
<tr>
<td>9</td>
<td>0.903</td>
<td>0.876</td>
<td>0.829</td>
<td>0.720</td>
<td>0.707</td>
<td>0.682</td>
<td>0.642</td>
<td>0.584</td>
<td>0.500</td>
</tr>
<tr>
<td>$\infty$</td>
<td>0.892</td>
<td>0.863</td>
<td>0.810</td>
<td>0.693</td>
<td>0.687</td>
<td>0.670</td>
<td>0.636</td>
<td>0.582</td>
<td>0.500</td>
</tr>
</tbody>
</table>

$D_i > p_i \Rightarrow$ Maximum utilisation increases

$D_i < p_i \Rightarrow$ Maximum utilisation decreases

$D_i = p_i$
The Deadline Monotonic Algorithm

• Assign priorities to jobs in each task based on the relative deadline of that task
  • Shorter relative deadline → higher the priority
  • If relative deadline equals period, schedule is identical to rate monotonic
  • When the relative deadlines and periods differ: deadline monotonic can sometimes produce a feasible schedule in cases where rate monotonic cannot; rate monotonic always fails when deadline monotonic fails
  • Hence deadline monotonic preferred if deadline ≠ period

• Not widely used – periodic systems typically have relative deadline equal to their period
The Earliest Deadline First Algorithm

- Assign priority to jobs based on deadline: earlier deadline = higher priority
- Rationale: do the most urgent thing first

Dynamic priority algorithm: priority of a job depends on relative deadlines of all active tasks
- May change over time as other jobs complete or are released
- May differ from other jobs in the task
# Earliest Deadline First: Example

<table>
<thead>
<tr>
<th>Time</th>
<th>Ready to run</th>
<th>Running</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$J_{2,1}$</td>
<td>$J$</td>
</tr>
<tr>
<td>1</td>
<td>$J$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$J_{2,1}$</td>
<td>$J$</td>
</tr>
<tr>
<td>3</td>
<td>$J$</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$J_{1,3}$</td>
<td>$J$</td>
</tr>
<tr>
<td>4.5</td>
<td>$J$</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>$J_{2,2}$</td>
<td>$J$</td>
</tr>
<tr>
<td>5.5</td>
<td>$J$</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>$J_{2,2}$</td>
<td>$J$</td>
</tr>
<tr>
<td>7</td>
<td>$J$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>Ready to run</th>
<th>Running</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>$J_{2,2}$</td>
<td>$J$</td>
</tr>
<tr>
<td>9</td>
<td>$J$</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>$J_{2,3}$</td>
<td>$J$</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

Released

$J_{1,1}$  $J_{1,2}$  $J_{1,3}$  $J_{1,4}$  $J_{1,5}$  $J_{1,6}$  $J_{2,1}$  $J_{2,2}$  $J_{2,2}$  $J_{2,2}$  $J_{2,3}$

$T_1 = (2, 1)$  
$T_2 = (5, 2.5)$
Earliest Deadline First is Optimal

- EDF is optimal, provided the system has a single processor, preemption is allowed, and jobs don’t contend for resources
  - That is, it will find a feasible schedule *if one exists*, not that it will always be able to schedule a set of tasks
- EDF is not optimal with multiple processors, or if preemption is not allowed
Earliest Deadline First is Optimal: Proof

• Any feasible schedule can be transformed into an EDF schedule

• If $J_i$ is scheduled to run before $J_k$, but $J_i$’s deadline is later than $J_k$’s either:
  • The release time of $J_k$ is after the $J_i$ completes ⇒ they’re already in EDF order
  • The release time of $J_k$ is before the end of the interval when $J_i$ executes:
  
  ![Diagram showing two jobs in sequence with $J_i$ before $J_k$ and release time of $J_k$ before $J_i$ completion]

  • Swap $J_i$ and $J_k$ (this is always possible, since $J_i$’s deadline is later than $J_k$’s)
  
  ![Diagram showing swapped $J_i$ and $J_k$]

  • Move any jobs following idle periods forward into the idle period
  
  ![Diagram showing moved jobs into idle period]

  • The result is an EDF schedule

• So, if EDF fails to produce a feasible schedule, no such schedule exists
  • If a feasible schedule existed it could be transformed into an EDF schedule, contradicting the statement that EDF failed to produce a feasible schedule [proof for LST is similar]
Maximum Utilisation Test: $D_i \geq p_i$

- **Theorem:**
  - A system of independent preemptable periodic tasks with $D_i \geq p_i$ can be feasibly scheduled on one processor using EDF if and only if $U \leq 1$
  - Note: result is independent of $\phi_i$

- **Proof follows from optimality of the system**
  - [Proof in the book, Section 6.3.1]
Maximum Utilisation Test: $D_i < p_i$

- Test fails if $D_i < p_i$ for some $i$
  - E.g. $T_1 = (2, 0.8), T_2 = (5, 2.3, 3)$

- However, there is an alternative test:
  - The density of the task, $T_i$, is $\delta_i = e_i / \min(D_i, p_i)$
  - The density of the system is $\Delta = \delta_1 + \delta_2 + \ldots + \delta_n$
  - Theorem: A system $T$ of independent, preemptable periodic tasks can be feasibly scheduled on one processor using EDT if $\Delta \leq 1$.

- Note:
  - This is a sufficient condition, but not a necessary condition – i.e. a system is guaranteed to be feasible if $\Delta \leq 1$, but might still be feasible if $\Delta > 1$ (would have to run the exhaustive simulation to prove)
The Least Slack Time Algorithm

• Least Slack Time first (LST)

• A job $J_i$ has deadline $d_i$, execution time $e_i$, and was released at time $r_i$
• At time $t < d_i$: remaining execution time $t_{\text{rem}} = e_i - (t - r_i)$
• Assign priority based on least slack time, $t_{\text{slack}} = d_i - t - t_{\text{rem}}$
• Two variants:
  • Strict LST – scheduling decision made whenever a queued job’s slack time becomes smaller than the executing job’s slack time – high overhead, not used;
  • Non-strict LST – scheduling decisions made only when jobs release or complete
• More complex, requires knowledge of execution times and deadlines
• Infrequently used, since has similar behaviour to EDF, but more complex
Discussion

• EDF is optimal, and simpler to prove correct – why use RM?
  • RM more widely supported since easier to retro-fit to standard fixed priority scheduler, and support included in POSIX real-time APIs
  • RM more predictable: worst case execution time of a task occurs with worst case execution time of the component jobs – not always true for EDF, where speeding up one job can increase overall execution time (known as a “scheduling anomaly”)
Summary

• The rate monotonic algorithm
  • Simply periodic systems
  • Maximum utilisation test

• The earliest deadline first algorithm
  • Optimality
  • Maximum utilisation tests

• Other algorithms
  • Deadline monotonic
  • Least slack time