Priority-driven Scheduling of Periodic Tasks (1)

Embedded Real-Time Software
Lecture 5
Lecture Outline

• Assumptions
• Fixed-priority algorithms
  – Rate monotonic
  – Deadline monotonic
• Dynamic-priority algorithms
  – Earliest deadline first
  – Least slack time
• Relative merits of fixed- and dynamic-priority scheduling
• Schedulable utilization and proof of schedulability
Assumptions

- Priority-driven scheduling of periodic tasks on a single processor
- Assume a restricted periodic task model:
  - A fixed number of independent periodic tasks exist
    - Jobs comprising those tasks:
      - Are ready for execution as soon as they are released
      - Can be pre-empted at any time
    - Never suspend themselves
    - New tasks only admitted after an acceptance test; may be rejected
    - The period of a task defined as minimum inter-release time of jobs in task
  - There are no aperiodic or sporadic tasks
  - Scheduling decisions made immediately upon job release and completion
    - Algorithms are event driven, not clock driven
    - Never intentionally leave a resource idle
  - Context switch overhead negligibly small; unlimited priority levels
Dynamic versus Static Systems

- **In static systems,**
  - Jobs are partitioned into subsystems, each subsystem bound statically to a processor
  - The scheduler for each processor schedules the jobs in its subsystem independent of the schedulers for the other processors
  - Priority-driven uniprocessor systems are applicable to each subsystem of a static multiprocessor system

- **In dynamic systems,**
  - Jobs are scheduled on multiple processors, and a job can be dispatched to any of the processors
  - Difficult to determine the best- and worst-case performance of dynamic systems, so most hard real-time systems built are static

- **In most cases, the performance of dynamic systems is superior to static system**

- **In the worst case, the performance of priority-driven algorithm can be very poor**
Fixed- and Dynamic-Priority Algorithms

• **A priority-driven scheduler is an on-line scheduler**
  – It does *not pre-compute a schedule of tasks/jobs: instead assigns priorities to jobs* when released, places them on a run queue in priority order
  – When pre-emption is allowed, a scheduling decision is made whenever a job is released or completed
  – At each scheduling decision time, the scheduler updates the run queues and executes the job at the head of the queue

• **Assignment of priority**
  – Fixed-priority algorithm: to assign the same priority all jobs in each task
  – Dynamic-priority algorithm: to assign different priorities to the individual jobs in each task. Once assigned, the priority of the job does not change (job-level fixed-priority)
  – Job level dynamic-priority: to vary the priority of a job after it has started. It is usually very inefficient
Rate Monotonic Scheduling (RM)

- **Best known fixed-priority algorithm**
- **Assigns priorities to tasks based on their periods**
  - The shorter the period, the higher the priority
  - The *rate* (of job releases) is the inverse of the period, so jobs with shorter period have higher priority
- **Very widely studied and used**
- **For example, consider a system of 3 tasks:**
  - $T_1 = (4, 1)$ $\Rightarrow$ rate = $1/4$
  - $T_2 = (5, 2)$ $\Rightarrow$ rate = $1/5$
  - $T_3 = (20, 5)$ $\Rightarrow$ rate = $1/20$
  - Relative priorities: $T_1 > T_2 > T_3$
Example: Rate Monotonic Scheduling
Deadline Monotonic Scheduling (DM)

- To assign task priority according to relative deadlines
  - the shorter the relative deadline, the higher the priority
- When relative deadline of every task matches its period, then rate monotonic and deadline monotonic give identical results
- When the relative deadlines are arbitrary:
  - Deadline monotonic can sometimes produce a feasible schedule in cases where rate monotonic cannot
  - But, rate monotonic always fails when deadline monotonic fails
  - Deadline monotonic preferred to rate monotonic
    - If deadline ≠ period
Example: Deadline Monotonic

- \( T_1 (50, 50, 25, 100), \ T_2 (0, 62.5, 10, 20), \ T_3 (0, 125, 25, 50) \)
  - Relative priority: \( T_2 > T_3 > T_1 \)
Dynamic-Priority Algorithms

• **Earliest deadline first (EDF)**
  – To assign priorities to jobs in the tasks according to their absolute deadline

• **Least slack time first (LST)**
  – To check all ready jobs each time a new job is released and
  – To order the new job and the existing jobs on their slack time
  – Two variations:
    • Strict LST – scheduling decisions are made also whenever a queued job’s slack time becomes smaller than the executing job’s slack time – *huge* overheads, not used
    • Non-strict LST – scheduling decisions made only when jobs release or complete

• **First in, first out (FIFO)**
  – Job queue is first-in-first-out by release time

• **Last in, first out (LIFO)**
  – Job queue is last-in-first-out by release time

• **Focus on EDF as commonly used example**
Example: EDF

- \textbf{T1 (2, 0.9), T2 (5, 2.3)}
Relative Merits

- Fixed- and dynamic-priority scheduling algorithms have different properties; neither appropriate for all scenarios.
- Algorithms that do not take into account the urgencies of jobs in priority assignment usually perform poorly.
  - E.g. FIFO, LIFO.
- The EDF algorithm gives higher priority to jobs that have missed their deadlines than to jobs whose deadline is still in the future.
  - Not necessarily suited to systems where occasional overload unavoidable.
- Dynamic algorithms like EDF can produce feasible schedules in cases where RM and DM cannot. However, it is difficult for the dynamic algorithms to predict which task will miss their deadlines during overloads.
  - But fixed priority algorithms often more predictable, lower overhead.
Example: Comparing Different Algorithms

- Compare performance of RM, EDF, LST and FIFO scheduling
- Assume a single processor system with 2 tasks:
  - $T_1 = (2, 1)$
  - $T_2 = (5, 2.5)$  $H = 10$

- The total utilization is 1.0 ⇒ no slack time
  - Expect some of these algorithms to lead to missed deadlines!
  - This is one of the cases where EDF works better than RM/DM
Example: RM, EDF, LST and FIFO

- Demonstrate by exhaustive simulation that LST and EDF meet deadlines, but FIFO and RM don’t
Schedulability Tests

• Simulating schedules is both tedious and error-prone... can we demonstrate correctness without working through the schedule?

• Yes, in some cases. This is a schedulability test
  – A test to demonstrate that all deadlines are met, when scheduled using a particular algorithm
  – An efficient schedulability test can be used as an on-line acceptance test; clearly exhaustive simulation is too expensive
Schedulable Utilization

- **Schedulable utilization, definition**  
  - A scheduling algorithm can feasibly schedule any set of periodic tasks on a processor if and only if the total utilization of the tasks is equal to or less than the schedulable utilization of the algorithm.

- **Utilization, definition**  
  - A periodic task $T_i$ is defined by the 4-tuple $(\varphi_i, p_i, e_i, D_i)$ with utilization $u_i = \frac{e_i}{p_i}$
  - Total utilization of the system $U = \sum_{t=1}^{n} u_t$ where $0 \leq U \leq 1$

- **A scheduling algorithm can feasibly schedule any system of periodic tasks on a processor if $U$ is equal to or less than the maximum schedulable utilization of the algorithm, $U_{ALG}$**  
  - If $U_{ALG} = 1$, the algorithm is optimal.

- **Why is knowing of $U_{ALG}$ important?**  
  - It gives a schedulability test, where a system can be validated by showing that $U \leq U_{ALG}$
Schedulable Utilization: EDF

- **Theorem 6.1**
  - A system of independent preemptable periodic tasks with $D_i = p_i$ can be feasibly scheduled on one processor using EDF if and only if $U \leq 1$
- **Proof:**
  - The “only if” part is clear. If $U > 1$, then they are not schedulable.
  - To prove “if” part, we will show that if the system fails to meet some deadline then its total utilization is larger than 1.
  - Assume that
    - The job $J_{i,c}$ of task $T_i$ misses its deadline at time $t$.
    - Prior to $t$, the process was never idles
  - Two cases
    1. The current period of every task begins at or after $r_{i,c}$
    2. The current period of some tasks begin before $r_{i,c}$
      - $r_{i,c}$ is the release time of the job that misses its deadline
Case 1: EDF Schedulable Utilization

- Any current whose deadline is after $t$ is not given any processor time to execute before $t$, and
- The total processor time for completing $J_{i,c}$ and all the job with deadlines at or before $t$ exceed the total available time $t$.
- Each term in the sum is the total amount of time before $t$ required to complete jobs that are in a task $T_k$ other than $T_i$ have deadline at or before $t$.

\[
\begin{align*}
t &< \frac{(t - \phi_i)e_i}{p_i} + \sum_{k \neq i} \left[ \frac{t - \phi_k}{p_k} \right] e_k \\
&\leq \frac{(t - \phi_i)e_i}{p_i} + t \sum_{k \neq i} \frac{e_k}{p_k} \\
&= t \sum_{k=1}^{n} u_k = tU \\
U &> 1
\end{align*}
\]
Case 2: EDF Schedulable Utilization

- $T'$: the subset of tasks whose current jobs were released before $r_{i,c}$ and have deadlines after $t$.

- Some processor time before $r_{i,c}$ was given to the current job of some tasks in $T'$, say them as $T_l$.

- $t_{-1}$: the end of the latest time interval $I$ (shown as a black box).

- $\varphi_i'$: the release time of the first job of task $T_k$ in $T-T'$

- Apply the similar to the previous inequality except $t$ is replaced by $t-t_{-1}$. 

\[
t - t_{-1} < \frac{(t - t_{-1} - \varphi_i')e_i}{p_i} + \sum_{T_k \in T-T'} \left[ \frac{t - t_{-1} - \varphi'}{p_k} \right] e_k
\]
Schedulable Utilization: EDF

- **Facts**
  - $U_{\text{EDF}} = 1$ for independent, preemptable periodic tasks with $D_i = p_i$
  - Corollary: The result also holds if deadline longer than period:
    - $U_{\text{EDF}} = 1$ for independent preemptable periodic tasks with $D_i \geq p_i$

- **Notes:**
  - Result is independent of $\varphi_i$
  - Result can also be shown to apply to strict LST
Schedulable Utilization: EDF

• What happens if $D_i < p_i$ for some $i$? The test doesn’t work...
  – E.g. $T_1 = (2, 0.8)$, $T_2 = (5, 2.3, 3)$

  ![Diagram showing task preemption and deadline miss]

• 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 EDF 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)
Schedulable Utilization: EDF

• **How can you use this in practice?**
  – Assume using EDF to schedule multiple periodic tasks, known execution time for all jobs

  ⇒ Choose the periods for the tasks such that the schedulability test is met

• **Example: a simple digital controller:**
  – Control-law computation task, $T_1$, takes $e_1 = 8$ ms, sampling rate is 100 Hz (i.e. $p_1 = 10$ ms)

  ⇒ $u_1$ is 0.8
  ⇒ the system is guaranteed to be schedulable
  – Want to add a built-in self test task, $T_2$, taking 50ms - will the system still work?
Schedulable Utilization of RM

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

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

  - [Proof in book Sec 6.7]

- $U \leq U_{RM}(n)$ 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 $U > U_{RM}(n)$
Schedulable Utilization of RM

• What happens if the relative deadlines for tasks are not equal to their respective periods?
• Assume the deadline is some multiple \( v \) of the period: \( D_k = v p_k \)
• It can be shown that:

\[
U_{RM}(n,v) = \begin{cases} 
  v & 0 \leq v \leq 0.5 \\
  n((2v)^{\frac{1}{n}} - 1) + 1 - v & 0.5 \leq v \leq 1 \\
  v(n-1)\left[\left(\frac{v + 1}{v}\right)^{\frac{1}{n-1}} - 1\right] & v = 2, 3, \ldots
\end{cases}
\]
# Schedulable Utilization of RM

<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>

\[
D_i > p_i \Rightarrow \text{Schedulable utilization increases}
\]

\[
D_i = p_i
\]

\[
D_i < p_i \Rightarrow \text{Schedulable utilization decreases}
\]
Summary

Key points:
• Different priority scheduling algorithms
  – Earliest deadline first, least slack time, rate monotonic, deadline monotonic
  – Each has different properties, suited for different scenarios
• Scheduling tests, concept of maximum schedulable utilization
  – Examples for different algorithms

Next lecture: practical factors, more schedulability tests...