Priority-driven Scheduling of Periodic Tasks (1)

Advanced Operating Systems (M)
Lecture 4
Priority-driven Scheduling

- Assign priorities to jobs, based on their deadline or other timing constraint
  - Make scheduling decisions based on the priorities, when events such as releases and job completions occur
  - Jobs are placed in one or more queues; at each event, the ready job with the highest priority is executed
  - The assignment of jobs to priority queues, along with rules such as whether preemption is allowed, completely defines a priority scheduling algorithm

- Priority-driven algorithms make locally optimal decisions about which job to run
  - Locally optimal scheduling decisions are often not globally optimal
  - Priority-driven algorithms never intentionally leave any resource idle; leaving a resource idle is not locally optimal
Advantages and Disadvantages

• Priority-driven scheduling has many advantages over clock-driven scheduling
  • Better suited to applications with varying time and resource requirements, since needs less a priori information
  • Run-time overheads are small

• But, harder to validate for correctness:
  • Scheduling anomalies can occur for multiprocessor systems, if preemption is disallowed, or if there is contention for resources
    • Reducing the execution time of a job in a task can increase the total response time of the task: not sufficient to show correctness with worse-case execution times, must simulate with all possible execution times for all jobs comprising a task
  • Can be proved that anomalies do not occur for independent, jobs with fixed release times, where preemption is allowed, executed using any priority-driven scheduler on a single processor
Priority-driven Scheduling

• Many priority-driven real-time scheduling algorithms exist
  • Earliest deadline first
  • Least slack time
  • Rate monotonic
  • Deadline monotonic

• Each has different characteristics
Fixed- and Dynamic-Priority Algorithms

- A priority-driven scheduler is an on-line scheduler
  - It does not pre-compute a schedule: 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

- The priority of jobs within a task may vary:
  - Jobs in a task may be assigned the same priority (task level fixed-priority) or different priorities (task level dynamic-priority)
  - The priority of each job is usually fixed (job level fixed-priority); but some systems vary the priority of a job after it has started (job level dynamic-priority)
Rate Monotonic Scheduling

- Well 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 higher rate have higher priority

- For example, consider a system of 3 tasks:
  - $T_1 = (4, 1) \Rightarrow \text{rate} = 1/4$
  - $T_2 = (5, 2) \Rightarrow \text{rate} = 1/5$
  - $T_3 = (20, 5) \Rightarrow \text{rate} = 1/20$
  - Relative priorities: $T_1 > T_2 > T_3$
## Example: Rate Monotonic Scheduling

<table>
<thead>
<tr>
<th>Time</th>
<th>Ready to run</th>
<th>Running</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></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>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- $T_1 = (4, 1)$
- $T_2 = (5, 2)$
- $T_3 = (20, 5)$
Deadline Monotonic Scheduling

- The deadline monotonic algorithm assigns 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; rate monotonic always fails when deadline monotonic fails.
  - Hence deadline monotonic preferred if deadline ≠ period.
The EDF and LST Scheduling Algorithms

• Two popular dynamic priority algorithms

• Earliest deadline first (EDF)
  • Assign priority to jobs based on deadline: earlier deadline = higher priority
  • Simple, just requires knowledge of deadlines

• 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 $t_{rem} = e_i - (t - r_i)$
  • Assign priority based on least slack time, $t_{slack} = d_i - t - t_{rem}$
  • 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
Example: Earliest Deadline First

<table>
<thead>
<tr>
<th>Time</th>
<th>Ready to run</th>
<th>Running</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></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></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$T_1 = (2, 1)$

$T_2 = (5, 2.5)$

Example: Earliest Deadline First
Optimality of EDF and LST

• The EDF and LST algorithms are optimal
  • On a single processor, as long as preemption is allowed and jobs do not contend for resources; can fail to schedule a feasible set of jobs if there are multiple processors, or if preemption is allowed
Optimality of EDF and LST: 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 $\Rightarrow$ they’re already in EDF order
    • The release time of $J_k$ is before the end of the interval in which $J_i$ executes:
      • Swap $J_i$ and $J_k$ (this is always possible, since $J_i$’s deadline is later than $J_k$’s)
      • Move any jobs following idle periods forward into the 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]
Relative Merits

• Fixed- and dynamic-priority scheduling algorithms have different properties; neither appropriate for all scenarios

• 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
  • 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 utilisation is 1.0; there is 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 Utilisation

• Recall: a periodic task $T_i$ is defined by the 4-tuple $(\phi_i, p_i, e_i, D_i)$ with utilisation $u_i = e_i / p_i$

• Total utilisation of system $U = \sum_{i=1}^{n} u_i$ 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 utilisation of the algorithm, $U_{ALG}$

• This gives a schedulability test, where a system can be validated by showing that $U \leq U_{ALG}$
  
  • If $U_{ALG} = 1$, the algorithm is optimal
Theorem: 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 $U \leq 1$

- $U_{EDF} = 1$ for independent, preemptable periodic tasks with $D_i = p_i$
- Corollary: result also holds if deadline longer than period: $U_{EDF} = 1$ 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 Utilisation: EDF

- 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 $\Delta = \delta_1 + \delta_2 + ... + \delta_n$
  - 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)
Schedulable Utilisation: 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)
    \( \Rightarrow u_1 \) is 0.8
    \( \Rightarrow \) the system is guaranteed to be schedulable
  • Want to add another task, \( T_2 \), taking 50ms - will the system still work?
Schedulable Utilisation of RM

- 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 \( n \to \ln 2 \) (i.e., \( n \to 0.69314718056\ldots \))

- \( U \leq U_{RM}(n) \) 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 Utilisation of RM

• What happens if the relative deadlines for tasks are not equal to their respective periods?

• If the deadline is a multiple $\nu$ of the period: $D_k = \nu \cdot p_k$

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

<table>
<thead>
<tr>
<th>( n )</th>
<th>( \nu = 4.0 )</th>
<th>( \nu = 3.0 )</th>
<th>( \nu = 2.0 )</th>
<th>( \nu = 1.0 )</th>
<th>( \nu = 0.9 )</th>
<th>( \nu = 0.8 )</th>
<th>( \nu = 0.7 )</th>
<th>( \nu = 0.6 )</th>
<th>( \nu = 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>

\( D_i > p_i \Rightarrow \text{Schedulable utilisation increases} \)

\( D_i < p_i \Rightarrow \text{Schedulable utilisation decreases} \)

\( D_i = p_i \)
Summary

- Different priority-driven scheduling algorithms
  -Earliest deadline first, least slack time, rate- and deadline-monotonic
  -Each has different properties, suited for different scenarios

- Scheduling tests, concept of maximum schedulable utilisation
  -Examples for different algorithms

- Next lecture: practical factors, more schedulability tests…