CMPE 117

Lectures:
Aperiodic Task Scheduling

© Luca de Alfaro, 2002-4
The Goal of Scheduling

• Ensure that activities (jobs) are executed within their deadlines in a real-time application.
• One must know the execution times!
**Execution Times, or WCET**

**WCET: Worst-Case Execution Time**

- In order to schedule embedded code, we need to know the WCET of the code.
- This is difficult, since most programming abstractions have been designed for non-real-time programming, and they are ill-suited to an application area, such as embedded systems, where real-time performance and predictability are important.
Programming Abstractions

• DMA
  - Unpredictable execution time due to cycle stealing.
  - Sometimes, DMA slots are reserved (the time-slice method).

• Semaphores
  - They can cause job suspension.
  - We will talk about them at length later on.
Programming Abstractions

• Cache
  - The page fault rate depends on which code has been executed before.
  - Sophisticated analysis techniques can look at a program and estimate the page faults.
  - Harder to reason about this if there are interrupts (these disrupt the cache); easier in the time-triggered setting.
WCET Prediction

• New tools, coming from program analysis, can predict it to 10% or less.
• E.g., the AbsInt tool (Reinhard Wilhelm et al) used among others by Airbus.
• Out of scope how the tool works!
Interrupts

• Interrupts are used to know what goes on in the real world, but can cause unbounded delays unless their frequency can be bounded.

• There are various approaches...
Approach 1: Do Nothing

• Typically, interrupt service routines (aka ISRs) call device drivers, resulting in non-negligible CPU time.

• Unless the frequency of interrupts is known, the timing becomes very hard to analyze.
Approach 2: No Interrupts!

Each job should poll its own devices.

• Predictable, but polling is inefficient.

• It can be used in some cases, when the “right moment when to poll” is somehow known (some serial sensor communication, for instance).

• Jobs must know the details of the devices (unless a library is used).
Approach 3: Timer Only

There is a (high-frequency) periodic timer interrupt, and the OS checks the devices during this interrupt.

• As in the lego
• Sometimes two periods, for fast and slow devices.
• All details of devices can be encapsulated in the kernel.
Approach 4: Light-Weight ISR

- Have the ISR do minimal work: typically, just activate a job and maybe copy some minimal amount of information. The real device handling is done under priority/OS control by the activated job (which waits if it is not the highest priority job).

- Unpredictability not totally eliminated, but much reduced.
Scheduling: vocabulary

\( a \): arrival time

\( C \): computation time

\( d \): deadline

\( s \): starting time

\( f \): finishing time

\( L = f - d \): lateness

\( E = \max(0, L) \): tardiness

\( X = d - a - C \): slack
Scheduling: types

• Preemptive vs. non-preemptive.
• Synchronous (all processes arrive at once) or asynchronous (processes arrive one by one)
• Static (a, C, d of all processes known in advance) or on-line.
• Independent, or with dependency relation.
• Aperiodic vs. periodic vs. hybrid (mixed)
Scheduling Anomalies

• Theorem [Graham, 1976]:
  If a task set is optimally scheduled on a multiprocessor with some priority assignment, fixed execution times, and precedence constraints, then the following can increase the schedule length:
  • increasing the number of processors
  • reducing execution times
  • weakening the precedence relation
Scheduling anomalies - example

Adding one processor

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

0 4 8 12
Scheduling anomalies - example
Decreasing the running time by 1

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>4</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td></td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0  4  8  12
Scheduling anomalies - example
Weakening the precedence relation
Where's the catch?

• Theorem [Graham, 1976]:
  If a task set is optimally scheduled on a multiprocessor with some priority assignment, fixed execution times, and precedence constraints, then the following can increase the schedule length...

• Priorities are a bad idea! They confuse two concepts:
  - How important is something?
  - Should it be scheduled next?
Scheduling without Precedence Constraints
Earliest Due Date (EDD) | 1 | synch | Lmax

Use when:

• Synch: all jobs have r=0 (arrive immediately)
• Optimizes maximum lateness
• No precedence relations
• No preemption necessary
Earliest Due Date (EDD)

Given a set of processes 
\{[ C_1, d_1], [ C_2, d_2], ..., [ C_m, d_m]\},
in order to minimize \(L_{\text{max}}\) it suffices to schedule the processes in order of increasing deadlines.

- **EDD Algorithm:**
  1) Sort the deadlines in increasing order
  2) Schedule the processes in order of increasing deadline.

- **Schedulability criterion:**
  Once ordered so that \(d_1 < d_2 < ... < d_m\), check that \(\sum_{i=1}^{k} C_i \leq d_k\) for all \(k \in \{1,...,m\}\).
EDD Optimality

Assume \( d_a < d_b \)

Recall: \( L = f - d \)

What about the max lateness in \( S' = \max(L'_a, L'_b) \)?

- \( L'_a = f'_a - d_a < f_a - d_a = L_a \)
- \( L'_b = f'_b - d_b = f_a - d_b < f_a - d_a = L_a \)

Hence, \( \max(L'_a, L'_b) < \max(L_a, L_b) \)
Earliest Deadline First (EDF) 1|asynch|L_{max}

Use when:

- Tasks can have arbitrary arrival time.
- They are preemptable. This is important!
EDF Algorithm

[Horn]: to minimize $L_{\text{max}}$, always execute the process that has the earliest deadline
Proof of EDF Optimality
Scheduling with Precedence Constraints
Scheduling with precedence constraints (1 | prec, syn | Lmax):

LDF (Latest Deadline First)

• Build the schedule from last to first, by recursively:
  - Let Q be the subset of jobs all whose descendants have been scheduled.
  - Pick the job p from Q that has the latest deadline.
  - Add p to the front of the schedule.
• Should be really called LDL (Latest deadline last).
• Optimal wrt. Lmax.
LDF Example

\[ J_i^{r_c} \]

\[
\begin{array}{c}
J_1 \\
J_2 \\
J_3 \\
J_4 \\
J_5 \\
J_6 \\
\end{array}
\]

\[
\begin{array}{c}
0 & 2 & 3 & 0 & 2 & 0 & 0 \\
3 & 5 & 0 & 4 & 1 & 16 & 10 \\
8 & 1 & 0 & 7 & 0 & 0 & 0 \\
0 & 7 & 0 & 2 & 16 & 0 & 0 \\
0 & 4 & 0 & 2 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]
LDF Example

\[ J_1 \rightarrow J_3 \rightarrow J_5 \]
\[ J_3 \rightarrow J_4 \rightarrow J_2 \]
\[ J_4 \rightarrow J_6 \rightarrow J_5 \]

\[ J_i^r \rightarrow C \]

[Diagram showing nodes and arrows with weights]
LDF Example

Graph with nodes $J_1$ to $J_6$ and edges with weights 0, 2, 3, 5, 7, 10, 16.
LDF Example

\[
\begin{array}{c}
J_1 \\
J_2 \\
J_3 \\
J_4 \\
J_5 \\
J_6 \\
\end{array}
\]

\[
\begin{array}{c}
0 \\
2 \\
3 \\
2 \\
4 \\
0 \\
\end{array}
\]

\[
\begin{array}{c}
0 \\
5 \\
3 \\
4 \\
7 \\
0 \\
\end{array}
\]

\[
\begin{array}{c}
0 \\
2 \\
8 \\
0 \\
16 \\
10 \\
\end{array}
\]

\[
\begin{array}{c}
J_i \\
D \\
J_3 \\
J_6 \\
J_5 \\
\end{array}
\]
LDF Example

\[ J_{i}^{c} \]

\[ D \]

\[ J_{1}^{0} 2 \]
\[ J_{2}^{0} 2 \]
\[ J_{3}^{0} 3 \]
\[ J_{4}^{0} 4 \]
\[ J_{5}^{0} 16 \]
\[ J_{6}^{0} 10 \]

\[ J_{4} J_{3} J_{6} J_{5} \]
LDF Example

\[ J_i \]

\[ J_D \]

\[ J_2 J_4 J_3 J_6 J_5 \]
LDF Example

Schedule:

\[
\begin{array}{c}
J_1\ J_2\ J_4\ J_3\ J_6\ J_5
\end{array}
\]
LDF: Proof of Optimality

L_{\text{max}} = \max \{L_A, L_B, L_a, L_b\}

- \(L_A\) unchanged
- \(L'_B < L_B\) as \(B\) starts earlier
- \(L'_b < L_b\) as \(J_b\) starts earlier
- \(L'_a = f'_a - d_a < f_b - d_b\) (as \(d_a > d_b\)) = \(L_b\)

So \(L'_{\text{max}} \leq L_{\text{max}}\).

The argument is concluded by induction.
EDF with precedence constraints
( 1 | async, prec, preem | Lmax)

- Algorithm by Chetto, Silly, Bouchenouf (1990).
- Idea: transform timing constraints to enforce precedence (cannot start before predecessors, cannot pre-empt successors), then use EDF.
EDF with precedence constraints

(1 | async, prec, preem | Lmax)

\[ r, C, D, \text{precedence} \]

transform the release times and deadlines

\[ r^*, C^*, D \]

schedule by EDF

\[ \text{schedule} \]
EDF with precedence constraints

(1 | async, prec, preem | Lmax)

Original problem schedulable iff EDF finds a schedule.
EDF with precedence constraints
(1 | async, prec, preem | Lmax)

• When $J_a \triangleleft J_b$, then:
  - $s_b \leq r_b$ (so $J_b$ must start after being released)
  - $s_b \leq r_a + C_a$ (so $J_a$ has time to finish)
• Thus, replace $r_b$ by $r_b^* = \max\{r_b, r_a + C_a\}$
EDF with precedence constraints 
( 1 | async, prec, preem | Lmax) 

- Algorithm for release times: 
  - For all roots $J_a$ of dependency graph, let $r^{*}_a = r_a$. 
  - Select a task $J_b$ such that all predecessors of $J_b$ have already been modified; let $H$ be this set. 
  - Let $r^{*}_b = \max \{r_b, \max_{c \in H} (r^{*}_c + C_c)\}$ 
- And recur.
EDF: Modification of the Deadlines

- Given $J_a$ and $J_b$ s.t. $J_a \leq J_b$, then the following conditions must hold:
  - $f_a \geq d_a$ (or else we miss the deadline)
  - $f_a \geq d_b - C_b$ (or there won't be enough time for $a$ to finish).
- Hence, replace $d_a$ by $d^*_a = \min\{d_a, d_b - C_b\}$
EDF: Modification of the Deadlines

• Algorithm for deadline modification:
  - For all sinks $J_a$ of dependency graph, let $d^*_{a} = d_a$.
  - Select a task $J_b$ such that all successors of $J_b$ have already been modified; let $H$ be this set.
  - Let $d^*_{b} = \min \{d_b, \min_{c \in H} (d^*_c - C_c)\}$
• And recur.
EDF Modification Example
EDF Modification Example
EDF Modification Example

\[ J_1 \rightarrow J_3 \rightarrow J_5 \rightarrow J_6 \]
\[ J_2 \rightarrow J_4 \rightarrow J_5 \rightarrow J_6 \]

\[ J_1 \rightarrow J_2 \rightarrow J_4 \rightarrow J_6 \]

Arrows represent dependencies, numbers indicate task durations.
EDF Modification Example

Graph showing nodes labeled $J_1, J_2, J_3, J_4, J_5, J_6$ with edges labeled with weights 1, 2, 3, 4, and 5.
EDF Modification Example

- **$J_1$**
  - 1
  - 2
  - 3
  - 4

- **$J_2$**
  - 2

- **$J_3$**
  - 3
  - 4
  - 8

- **$J_4$**
  - 2
  - 4

- **$J_5$**
  - 1
  - 6
  - 16

- **$J_6$**
  - 2
  - 8
  - 10
EDF Modification Example

Diagram:

- Nodes: $J_1, J_2, J_3, J_4, J_5, J_6$
- Edges with labels:
  - $J_1$ to $J_2$: 2
  - $J_1$ to $J_4$: 3
  - $J_2$ to $J_4$: 5
  - $J_3$ to $J_4$: 4
  - $J_3$ to $J_5$: 8
  - $J_4$ to $J_5$: 7
  - $J_4$ to $J_6$: 4
  - $J_5$ to $J_6$: 16
  - $J_5$ to $J_1$: 1
  - $J_6$ to $J_1$: 10
  - $J_6$ to $J_2$: 2
EDF Modification Example
EDF Modification Example

J_1 \rightarrow J_3 \rightarrow J_5 \rightarrow J_4 \rightarrow J_2 \rightarrow J_6
EDF Modification Example
Sometimes we can see right away that the problem is not schedulable.
Another Example
Another Example
Another Example

Diagram:

- Node $J_1$: 1
- Node $J_2$: 2
- Node $J_3$: 4
- Node $J_4$: 3
- Node $J_5$: 8
- Node $J_6$: 2

Connections:
- $J_1$ to $J_3$: 3, 31
- $J_3$ to $J_5$: 4, 20
- $J_4$ to $J_3$: 4
- $J_4$ to $J_2$: 4
- $J_5$ to $J_6$: 8
- $J_2$ to $J_6$: 2

Values:
- $J_1$: 1
- $J_2$: 2
- $J_3$: 4
- $J_4$: 3
- $J_5$: 8
- $J_6$: 2
Another Example
Another Example
Another Example
Another Example

Schedule:

Schedulable.
EDF + prec: Proof of optimality

• The algorithm provides a translation EDF+Prec→modified-EDF.

• We can show that the translation preserves schedulability.

• First direction: if EDF+Prec is schedulable, then modified-EDF is also schedulable (the same schedule works).
EDF + prec: Proof of optimality

- **Converse:** if modified-EDF is schedulable, the same schedule shows that EDF+Prec is schedulable.

- **Clearly,** the schedule produced by modified EDF meets all release times and deadlines of the original problem; we must only prove that the precedence relation is respected.

- **Assume** $J_a \triangleleft J_b$, and $C_b > 0$. Then, $D_{\star a} < D_{\star b}$, so EDF schedules $J_a$ before $J_b$. 
Non-Preemptive Scheduling

• Jobs have release times, arrive dynamically, and there are no precedence relations.

• There are two kinds of algorithms:
  - Non-idle, that always execute a job as long as there is one ready
  - May-Idle, that may choose to do nothing even though there are jobs ready.

• There are cases when may-idle algorithms outperform non-idle ones....
Being idle is sometimes a good idea

But: unless the arrival times are known, are may-idle algorithms reasonable? They require knowledge about the future!
EDF is best of non-idle 1|async|\(L_{\text{max}}\) algorithms

- This is a theorem by Jeffay, Stanat, and Martel (1991).
- Intuition: if you can't help being active, at least do the most useful thing at hand, that is, the one with the earliest deadline.
### Aperiodic scheduling: summary

<table>
<thead>
<tr>
<th></th>
<th>Sync</th>
<th>Preempt</th>
<th>Non-preempt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Independent</td>
<td>EDD</td>
<td>EDF</td>
<td>if non-idle, EDF</td>
</tr>
<tr>
<td>Prec Constr.</td>
<td>LDF</td>
<td>EDF with modifications</td>
<td>Search</td>
</tr>
</tbody>
</table>