IN4343 Real-Time Systems, Lecture 7

Non-preemptive Scheduling

Contact

m.nasri@tudelft.nl
(appointment by email)
Sources

Buttazzo’s book, chapter 4, 8

Paper 1:

Paper 2:

Disclaimer: A few slides have been taken from Giorgio Buttazzo’s website:
http://retis.sssup.it/~giorgio/rts-MECS.html
Agenda

• EDF schedulability analysis
  • Optimality of EDF for periodic tasks
  • EDF response-time analysis

• Non-preemptive scheduling
  • Why non-preemptive execution?
  • Existing schedulability analysis for NP-FP and NP-EDF
  • Behind the scenes!
    Things you need to know about work-conserving NP scheduling!

Book: chapter 4
Book: chapter 8
Book: chapter 8
Paper 1, 2
• What is the hyperperiod of 5, 10, 20? 20

• What is the hyperperiod of 5, 10, 21? 210
EDF

Optimality

The proof comes from Dertouzos 1974
(see the proof in Lecture 3)

Schedulability test

- Liu and Layland 1973

A task set $\tau$ is schedulable by EDF if and only if:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

Assumptions:

- Fully preemptive
- Independent tasks
- No self-suspension
- No scheduling overhead
- $\forall i, D_i = T_i$
EDF schedulability analysis for $D < T$

**Processor Demand Criterion** [Baruah ‘90]

In *any interval* of length $L$, the **computational demand** $g(0, L)$ of the task set must be **no greater** than $L$.

$$\forall L > 0, \quad g(0, L) \leq L$$

Applicable to periodic tasks with $\forall i, \phi_i = 0$
Understanding the processor demand ($g$ function)

- The demand in $[t_1, t_2]$ is the computation time of those tasks arrived \textit{at or after} $t_1$ with \textit{deadline less than or equal to} $t_2$:

$$g(t_1, t_2) = \sum_{\forall J_{k,j}, t_1 \leq a_{k,j} < d_{k,j} \leq t_2} C_k$$
Demand of a periodic task set

\[ g_i(0, L) = \max \left\{ 0, \left[ \frac{L - D_i + T_i}{T_i} \right] \cdot C_i \right\} \]

\[ g_i(0, L) = \max \left\{ 0, \left[ \frac{L - D_i}{T_i} + 1 \right] \cdot C_i \right\} \]

\[ g_i(0, L) = \max \left\{ 0, \left( \left[ \frac{L - D_i}{T_i} \right] + 1 \right) \cdot C_i \right\} \]

All three representations of \( g_i \) can be used. Recall: \( \lfloor x \rfloor = \max\{m \in \mathbb{Z} \mid m \leq x\} \)
Examples

\[ g_i(0, L) = \max \left\{ 0, \left[ \frac{L - D_i}{T_i} + 1 \right] \cdot C_i \right\} \]

\[ C_1 = 1, T_1 = 4, D_1 = 3 \]

\[ C_2 = 2, T_2 = 6, D_2 = 4 \]

<table>
<thead>
<tr>
<th>( L )</th>
<th>( g_2(0, L) )</th>
<th>( L )</th>
<th>( g_2(0, L) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>( g_2(0, 4) = \left[ \frac{4 - 4}{6} + 1 \right] \cdot 2 = 2 )</td>
<td>9</td>
<td>( g_2(0, 9) = \left[ \frac{9 - 4}{6} + 1 \right] \cdot 2 = 2 )</td>
</tr>
<tr>
<td>5</td>
<td>( g_2(0, 5) = \left[ \frac{5 - 4}{6} + 1 \right] \cdot 2 = 2 )</td>
<td>10</td>
<td>( g_2(0, 10) = \left[ \frac{10 - 4}{6} + 1 \right] \cdot 2 = 4 )</td>
</tr>
<tr>
<td>6</td>
<td>( g_2(0, 6) = \left[ \frac{6 - 4}{6} + 1 \right] \cdot 2 = 2 )</td>
<td>11</td>
<td>( g_2(0, 11) = \left[ \frac{11 - 4}{6} + 1 \right] \cdot 2 = 4 )</td>
</tr>
<tr>
<td>7</td>
<td>( g_2(0, 7) = \left[ \frac{7 - 4}{6} + 1 \right] \cdot 2 = 2 )</td>
<td>15</td>
<td>( g_2(0, 15) = \left[ \frac{15 - 4}{6} + 1 \right] \cdot 2 = 4 )</td>
</tr>
<tr>
<td>8</td>
<td>( g_2(0, 8) = \left[ \frac{8 - 4}{6} + 1 \right] \cdot 2 = 2 )</td>
<td>16</td>
<td>( g_2(0, 16) = \left[ \frac{16 - 4}{6} + 1 \right] \cdot 2 = 6 )</td>
</tr>
</tbody>
</table>
When does DBF change?

<table>
<thead>
<tr>
<th>$L$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g_1(0, L)$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>$g_2(0, L)$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>$g(0, L)$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>10</td>
</tr>
</tbody>
</table>
\[ g(0, L) = \sum_{i=1}^{n} g_i(0, L) \quad g_i(0, L) = \max\left\{0, \left\lfloor \frac{L - D_i}{T_i} + 1 \right\rfloor \cdot C_i \right\} \]

### How can we make it faster?

- **L** is a step function
- **g** is a step function
- **L** is continuous
Bounding complexity

• Since $g(0, L)$ is a step function, we can check feasibility only at deadline points, where $g$ changes.

• If tasks are synchronous and $U \leq 1$, we can check the feasibility up to the hyperperiod $H$:

$$ H = lcm(T_1, \ldots, T_n) $$
Bounding complexity: finding a safe upper bound for the analysis window

\[
g(0, L) = \sum_{i=1}^{n} \left[ \frac{L + T_i - D_i}{T_i} \right] \cdot C_i \leq G(0, L) = \sum_{i=1}^{n} \left( \frac{L + T_i - D_i}{T_i} \right) \cdot C_i
\]

\[
G(0, L) = \sum_{i=1}^{n} L \frac{C_i}{T_i} + \sum_{i=1}^{n} (T_i - D_i) \frac{C_i}{T_i}
\]

\[
= L \cdot U + \sum_{i=1}^{n} (T_i - D_i) \cdot U_i
\]

\[g(0, L) < G(0, L)\]
Limiting $L$

For any $L \geq L^*$, it is meaningless to check if $g(0, L) \leq L$ because it will obviously be smaller! Hence, we can stop the search at $L^*$. 

$$G(0, L) = L \cdot U + \sum_{i=1}^{n} (T_i - D_i) \cdot U_i$$

$$L^* = \frac{\sum_{i=1}^{n} (T_i - D_i) \cdot U_i}{1 - U}$$

For $L > L^*$

$$g(0, L) \leq G(0, L) < L$$
Processor Demand Test

∀L ∈ D,  \quad g(0, L) ≤ L

Where \( D \) is the set of deadline points for which \( g(0, L) \) must be calculated:

\[
D = \{d_{i,j} | d_{i,j} \leq \min\{H, L^*\}\}
\]

\[
H = \text{lcm}(T_1, \ldots, T_n)
\]

\[
L^* = \frac{\sum_{i=1}^{n} (T_i - D_i) \cdot U_i}{1 - U}
\]
Embedded and Networked Systems

FP v.s. EDF

Anime: Naruto
EDF v.s. FP

Context switches

<table>
<thead>
<tr>
<th>$\tau_i$</th>
<th>$C_i$</th>
<th>$T_i$</th>
<th>$D_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>2</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>4</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>4</td>
<td>20</td>
<td>17</td>
</tr>
</tbody>
</table>

How many preemptions?
3 preemptions in jobs $J_{2,1}, J_{3,1}, J_{2,2}$

How many preemptions?
1 preemption in job $J_{3,1}$

In average, FP has more context switches
**EDF v.s. FP**

What do you think will happen for FP if \( U > 1 \)?

- **FP under permanent overload** ➔ **starvation for low priority tasks**

\[
U = \frac{4}{8} + \frac{6}{12} + \frac{5}{20} = 1.25
\]

- High priority tasks execute at the proper rate
- Low priority tasks are completely blocked
EDF v.s. FP

EDF under permanent overload

\[
U = \frac{4}{8} + \frac{6}{12} + \frac{5}{20} = 1.25
\]

- All tasks execute at a slower rate
- No task is blocked
**EDF v.s. FP**

**Schedulability analysis**

<table>
<thead>
<tr>
<th></th>
<th>(D_i = T_i)</th>
<th>(D_i \leq T_i)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RM</strong></td>
<td><strong>Suff.: polynomial (O(n))</strong></td>
<td><em>pseudo-polynomial</em> Response Time Analysis</td>
</tr>
<tr>
<td></td>
<td>(\sum U_i \leq n(2^{1/n} - 1))</td>
<td>(\forall i \quad R_i \leq D_i)</td>
</tr>
<tr>
<td></td>
<td><strong>HB: (\prod(U_i + 1) \leq 2)</strong></td>
<td>(R_i = C_i + \sum_{k=1}^{i-1} \left\lfloor \frac{R_i}{T_k} \right\rfloor C_k)</td>
</tr>
<tr>
<td></td>
<td><strong>Exact (pseudo-polynomial)</strong></td>
<td>(\text{RTA})</td>
</tr>
<tr>
<td><strong>EDF</strong></td>
<td><strong>polynomial: (O(n))</strong></td>
<td><em>pseudo-polynomial</em> Processor Demand Analysis</td>
</tr>
<tr>
<td></td>
<td>(\sum U_i \leq 1)</td>
<td>(\forall L &gt; 0, \quad g(0, L) \leq L)</td>
</tr>
</tbody>
</table>

**FP’s response-time analysis is usually faster than EDF**
**EDF v.s. FP**

**Implementation overhead**

- Both EDF and FP can be implemented using a binary min-heap (see wiki)
  
  https://en.wikipedia.org/wiki/Binary_heap

  The root always points to the smallest value

  Each node has a value that is less than or equal to the values of its children

<table>
<thead>
<tr>
<th>3</th>
<th>5</th>
<th>9</th>
<th>6</th>
<th>8</th>
<th>20</th>
<th>10</th>
<th>12</th>
<th>18</th>
<th>9</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>...</td>
</tr>
</tbody>
</table>

**Heap Insertion**

- A new element is added to the **end** of the list, and then the tree is updated (bubble-up operation) to restore the heap property, i.e., each node is smaller than its children.

**Examples:**

http://condor.depaul.edu/ntomuro/courses/00winter/notes/lecture13.html
EDF v.s. FP

Implementation overhead

- Both EDF and FP can be implemented using a binary heap (see wiki)
  https://en.wikipedia.org/wiki/Binary_heap

Possible operations on the heap
Example: inserting a newly arrived task into the ready queue

<table>
<thead>
<tr>
<th>Operation</th>
<th>find-min</th>
<th>delete-min</th>
<th>insert</th>
<th>decrease-key</th>
<th>merge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log'n)$</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Leftist</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>Binomial</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(1)^{[c]}$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)^{[d]}$</td>
</tr>
<tr>
<td>Fibonacci</td>
<td>$O(1)$</td>
<td>$O(\log n)^{[c]}$</td>
<td>$O(1)$</td>
<td>$O(1)^{[c]}$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Pairing</td>
<td>$O(1)$</td>
<td>$O(\log n)^{[c]}$</td>
<td>$O(1)$</td>
<td>$O(\log n)^{[c][e]}$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Brodal</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Rank-pairing</td>
<td>$O(1)$</td>
<td>$O(\log n)^{[c]}$</td>
<td>$O(1)$</td>
<td>$O(1)^{[c]}$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Strict Fibonacci</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>2-3 heap</td>
<td>?</td>
<td>$O(\log n)^{[c]}$</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>?</td>
</tr>
</tbody>
</table>

Different binary-heap algorithms to be used instead of the ready-queue

How many insertion at time 0?
3 insertions at time 0
$n \times O(1) \rightarrow O(n)$
EDF v.s. FP

Implementation overhead

• Both EDF and FP can be implemented using a binary heap (see wiki)
  https://en.wikipedia.org/wiki/Binary_heap

• However,
  Using a bitmap data structure + most-significant-bit instruction
  ⇒ FP can be implemented in $O(1)$
  (provided that the number of priority levels is limited by a constant)

Implementing FP using bitmaps:
1. Whenever a task is released, set its bit in the flag to 1
2. Whenever a task completes, set its bit in the flag to 0
3. To find the highest-priority ready task, calculate the most-significant-bit of the flag
EDF v.s. FP

**EDF**
- Higher schedulability
- Reduces context switches
- During overloads does not starve low-priority tasks

**FP**
- Simpler to implement
- Widely implemented in most operating systems
- More predictable during overloads

How does DM compare to EDF?
Agenda

• EDF schedulability analysis
  • Optimality of EDF for periodic tasks
  • EDF response-time analysis

• Non-preemptive scheduling
  • Why non-preemptive execution?
  • Existing schedulability analysis for NP-FP and NP-EDF
  • Behind the scenes!
    Things you need to know about work-conserving NP scheduling!
Disadvantages of preemptions

1- Context switch cost

It is the time taken by the scheduler to suspend the running task, switch the context, and dispatch the new incoming task.
Disadvantages of preemptions

2- Cache-related preemption delay (CRPD)

It is the delay introduced by high-priority tasks that evict cache lines containing data used in the future:

\[
\begin{align*}
\tau_1 & \quad \text{(high priority)} \\
\tau_2 & \quad \text{(low priority)} \\
\text{write A} & \quad \text{(A is loaded into cache)} \\
\text{read A} & \quad \text{(Cache hit)}
\end{align*}
\]
Disadvantages of preemptions

2- Cache-related preemption delay (CRPD)

It is the delay introduced by high-priority tasks that evict cache lines containing data used in the future:

Extra time is needed for reading $A$. It increases the WCET of $\tau_2$. 
Disadvantages of preemptions

3- larger worst-case execution time

Preemptions cause CRPD which in turn increases the WCET of the task.

The amount of CRPD depends on the number of preemptions a job suffers.

Task experiencing preemptions by higher priority tasks:

\[
\tau_i \quad \text{WCET}_i \quad C_i = C_i^{NP} + \text{CRPD}
\]
Disadvantages of preemptions

4- Pipeline cost

time to flush the pipeline when a task is interrupted and to refill it when task is resumed.

5- Bus cost

time spent waiting for the bus due to additional conflicts with I/O devices, caused by extra accesses to the RAM for the extra cache misses.
Disadvantages of preemptions

A preemption may cause more preemptions
the extra execution time also increases the number of preemptions:

Preemption overhead is zero

Preemption overhead is NOT zero
Disadvantages of preemptions

Preemption cost can be very large

- WCETs may increase up to 35% in the presence of preemptions (less efficiency)!

In preemptive execution, the execution times are more variable (less predictability).
Advantages of NP scheduling

• It reduces **context-switch overhead**:
  • Making WCETs smaller and more predictable

• It simplifies the **access to shared resources**:
  • No semaphores are needed for critical sections

• It reduces **stack size**:
  • Task can share the same stack, since no more than one task can be in execution

• It allows achieving **small I/O Jitter**:
  • “finishing_time – start_time” has a low variation
Advantages of NP scheduling

- In fixed-priority systems, non-preemptive execution can even improve schedulability (in some cases)
So, why preemptive execution?
Disadvantages of NP scheduling

- In general, NP scheduling reduces schedulability because of introducing **blocking delays** on the high-priority tasks.
Disadvantages of NP scheduling

• The utilization bound under non-preemptive scheduling drops to zero

\[ C_1 = \epsilon \sim 0 \quad T_1 = 10 \Rightarrow U_1 = \frac{\epsilon}{T_1} \sim 0 \]

\[ C_2 = 2T_1 \quad T_2 = \infty \Rightarrow U_2 = \frac{2T_1}{\infty} \sim 0 \]

\[ U_1 + U_2 \sim 0 \]

Infeasible! Despite having \( U \sim 0 \)
Disadvantages of NP scheduling

Anomalies

Anomaly is a situation in which a deadline miss happens when we don’t expect it to happen!

Example: the task set is feasible on the current processor but when we use a faster one, it becomes unschedulable!
Agenda

• EDF schedulability analysis
  • Optimality of EDF for periodic tasks
  • EDF response-time analysis

• Non-preemptive scheduling
  • Why non-preemptive execution?
  • Existing schedulability analysis for NP-FP and NP-EDF
  • Behind the scenes!
    Things you need to know about work-conserving NP scheduling!

These are sufficient schedulability tests for periodic tasks
Challenges of analyzing non-preemptive systems

- Analysis of non-preemptive systems is more complex, because the largest-response time may not occur in the first job after the critical instant.
Challenges of analyzing non-preemptive systems

**Self-pushing phenomenon**

- High-priority jobs activated during non-preemptive execution of lower priority tasks are pushed ahead and introduce higher delays in subsequent jobs of the same task.

---

Delayed execution of $J_{2,1}$ caused a long delay in the start time of $J_{2,2}$
Challenges of analyzing non-preemptive systems

• Hence, the analysis of $\tau_i$ must be carried out for multiple jobs, until all tasks with higher priority than $P_i$ are completed.

**NOTE**
Analysis can reduce to the first job of each task if and only if
1. the task set is feasible under preemptive scheduling;
2. All deadlines are less than or equal to periods.
Response-time analysis of NP-FP

Maximum blocking (caused by the lower-priority tasks):
\[ B_i = \max\{C_j \mid \forall \tau_j, P_i < P_j\} \]

Maximum interference (caused by the higher-priority tasks):
\[ I_i \]

Worst-case occupied time (worst-case start time): due to blocking and interference
\[ WO_i = \max\{s_{i,k} - a_{i,k} \mid \forall j, 1 \leq k \leq \infty\} \]

This analysis is designed for “preemptively feasible task sets” with \( D \leq T \)
Response-time analysis of NP-FP

The solution is based on fixed-point iterations:

Starting point:
\[ WO_i^{(0)} = B_i + \sum_{k=1}^{i-1} C_k \]

Iterate until:
\[ WO_i^{(n)} = WO_i^{(n-1)} \]

This analysis is designed for “preemptively feasible task sets” with \( D \leq T \)

**NOTE:** the end of \( l_i \) cannot coincide with the activation of a higher-priority task, because it would increase \( l_i \).

**Hence:** instead of \([x]\) we must use \([x] + 1\)
History of the schedulability analysis of NP-FP?

In 1994, Tindel et al. have introduced the test we saw earlier. They did not know that their test is correct only if the following condition holds:

“task set must be “preemptively feasible” with $D \leq T$”

In the late 90’s, this test was used in CAN controllers in the cars for any type of task set!

Only in 2007, Dr. Reinder Bril from TU/e noticed a bug in the test!

However, thankfully, no car accident happened because of this bug.

Why?

Because engineers decided to “simplify” their life and instead of using

$$B_i = \max \{C_j \mid \forall \tau_j, P_i < P_j\}$$

used

$$B_i = \max \{C_j \mid \forall \tau_j\}$$
Schedulability analysis for NP-EDF

Jeffay’91 test
Task set \( \tau \) with periodic tasks sorted in a non-decreasing order of their periods is schedulable if \( U \leq 1 \) and
\[
\forall \tau_i, 1 < i \leq n, \forall L, T_1 < L < T_i:
\]
\[
L \geq C_i + \sum_{j=1}^{i-1} \left\lfloor \frac{L - 1}{T_j} \right\rfloor \cdot C_j
\]

Jeffay’s test is **exact** (necessary and sufficient) for non-preemptive **sporadic** tasks

Jeffay’s test is just a **sufficient** test for non-preemptive **periodic** tasks

Why Jeffay’s test is just sufficient for periodic tasks?

• Because it rejects the following task set which is perfectly schedulable by NP-EDF

\[ \forall i, 1 < i \leq n, \forall L, T_1 < L < T_i: \]
\[ L \geq C_i + \sum_{j=1}^{i-1} \left\lfloor \frac{L - 1}{T_j} \right\rfloor \cdot C_j \]

\[ i = 2, L = 6, 5 < L < 20: \]
\[ 6 \not\geq 8 + \left\lfloor \frac{6 - 1}{5} \right\rfloor \cdot 1 \]

The first counter example was introduced in:

We learned how to “analyze” schedulability of non-preemptive tasks for NP-EDF and NP-FP!

Should we be happy?

Anime: Naruto shippuden
NP-EDF and NP-FP in practice
How effective are NP-EDF and NP-FP schedulability analyses?

- NP-FP classic test
How effective are those schedulability analyses?

- NP-FP classic test

Many task sets do not pass the test

Is it because of the **pessimism of the test**?
Or
Because the NP-FP is a **bad algorithm** to schedule these task sets?

Automotive benchmarks
We didn’t know, so we had to invent an “exact test” for non-preemptive tasks in the next lecture.
Still, many task sets are not schedulable.

Are these task sets feasible at all? To answer that, we need to check all schedules, OR, build a supper clever scheduling algorithm.

About 40% more schedulable task sets are found.

Automotive benchmarks
We found the “clever” trick!

Online non-work-conserving policies
Non-work-conserving scheduling!

- NP-FP classic test
- Precautious-RM (non-work-conserving)
- Our exact test for NP-EDF
- Our exact test for NP-FP
- CW-EDF (non-work-conserving)

Automotive benchmarks
Next lecture: successful stories