Scheduling in Real-Time Systems

Motivation

Concurrency may lead to non-determinism.

Non-determinism may make it harder to predict the timing behaviour.

Real-Time Scheduling schemes reduce non-determinism.

Deployment

Static versus Dynamic

Scheduling in Real-Time Systems

Rigid:
- All schedules are set off-line.
- Full predictability (many high integrity real-time systems).

Static:
- Schedule relations are statically ordered off-line.
- Predictable response to disturbances (many real-time systems).

Dynamic:
- Schedules depend on run-time situation.
- More flexible, more efficient (most soft real-time systems).

Scheduling in Real-Time Systems

Assumptions

A simple process model

- The number of processes in the system is fixed.
- All processes are periodic and all periods are known.
- All processes are independent.
- The task-switching overhead is negligible.
- All deadlines are identical with the process cycle times (periods).
- The worst case execution time is known for all processes.
- All processes are released at once.

This model can only be applied to a specific group of hard real-time systems. Extensions to this model will be discussed later in this chapter.
Real-time scheduling:

### Earliest Deadline First (EDF)

1. **Determine** (one of) the process(es) with the earliest deadline.
2. **Execute** this process
   - 2a. **until** deadlines
   - 2b. **until** another process' deadline is found earlier than the current one.

- Pre-emptive scheme
- Dynamic scheme, since the dispatched process is selected at run-time, due to the current deadlines.

If multiple deadlines coincide, other means are needed to select a process, i.e.
- Avoid unnecessary task switches.
- Dispatch by task id (out of the currently qualifying processes).

### Rate monotonic ordering

In the framework of fixed priority schedulers:

- If a process set is schedulable under an FPS-scheme, then it is also schedulable under FPS with rate monotonic priorities.

### Worst case response times

Response time: Time from scheduler request to process completion

### Times to deadlines

Gives an idea how "critical" the schedule is.

### Maximal utilization

The maximal utilization for EDF becomes: \( \sum_{i} \frac{C_i}{T_i} \leq 1 \) (sufficient and necessary test)

Pre-emptive scheme.

Static scheme, since the order dispatch order of processes is fixed and calculated off-line.

Rate monotonic ordering is optimal (in the framework of fixed priority schedulers):

- A process set is schedulable under an FPS-scheme, then it is also schedulable under FPS with rate monotonic priorities.
Scheduling

Real-time scheduling: Fixed Priority Scheduling

Execute FPS schedule $\Rightarrow$ Fails!

Reduced task set

Further reduced task set

Maximal utilization

Utilization test fails, schedulability not guaranteed.

Utilization test succeeds, schedulability guaranteed.

Further reduced task set
Scheduling

Real-time scheduling: Fixed Priority Scheduling

Worst case response times

for the highest priority task: \( R_1 = C_1 \)

for others tasks: \( R_i = C_i + t_i \) (interference from higher priority tasks)

with:

\[
R_j = C_j + \sum_{k \in \text{min}} \left[ \frac{R_k}{T_k} \right] C_k
\]

Recurrence form: \( R^{j+1}_j = C_j + \sum_{k \in \text{min}} \left[ \frac{R^{j+1}_k}{T_k} \right] C_k \) with \( R^{j+1}_j = C_j \)

Iterate the recurrence form until \( R^{j+1}_j = R^j_j \) or \( R^{j+1}_j > D_j \)

Scheduling

Real-time scheduling: Earliest Deadline First

Response time analysis

\[
R_j(a) = \left[ \frac{2}{T_j} + 1 \right] C_j + \sum_{k \in \text{min}} \left[ \frac{R_j(a)}{T_k} \right] \max \left[ 0, \frac{a + T_j - T_k}{T_j} + 1 \right] \cdot C_k
\]

Recurrence form:

\[
R^{j+1}_j(a) = \left[ \frac{2}{T_j} + 1 \right] C_j + \sum_{k \in \text{min}} \left[ \frac{R^{j+1}_j(a)}{T_k} \right] \max \left[ 0, \frac{a + T_j - T_k}{T_j} + 1 \right] \cdot C_k
\]

Iterate until: \( R^{j+1}_j(a) = R^j_j(a) \) or \( R^{j+1}_j(a) > a + D_j \)

Scheduling

Real-time scheduling: Earliest Deadline First

Response time analysis

The worst case for Earliest Deadline First is not necessarily when all tasks are released at once!

- All possible release combinations in a full hyper-cycle need to be considered!

- The response times are bounded by the cycle times as long as the maximal utilization is ≤ 1.

- Other tasks need to be considered only, if their deadlines closer or equal to the current task.

R_j(R_{min}, t, C) = \left[ \frac{2}{T_j} + 1 \right] C_j + \sum_{k \in \text{min}} \left[ \frac{R_j(a)}{T_k} \right] \max \left[ 0, \frac{a + T_j - T_k}{T_j} + 1 \right] \cdot C_k
Real-time scheduling: Earliest Deadline First

Worst case response times

\[ R_3 = 1 \leq 4 \land R_2 = 4 \leq 12 \land R_1 = 12 \leq 16 \land \sum_{i=1}^{n} \frac{C_i}{T_i} = N \left( \frac{2^n - 1}{2} \right) \]

Real-time scheduling: Comparison

Response Time Analysis

<table>
<thead>
<tr>
<th>Fixed Priority Scheduling</th>
<th>Earliest Deadline First</th>
</tr>
</thead>
<tbody>
<tr>
<td>Utilization Test</td>
<td>Response Times [R_i]</td>
</tr>
<tr>
<td>[ \sum_{i=1}^{n} \frac{C_i}{T_i} \leq N \left( \frac{2^n - 1}{2} \right) ]</td>
<td>[ \sum_{i=1}^{n} \frac{C_i}{T_i} \leq N \left( \frac{2^n - 1}{2} \right) ]</td>
</tr>
<tr>
<td>(16, 12, 4)</td>
<td>(16, 12, 4)</td>
</tr>
<tr>
<td>(16, 6, 12, 4)</td>
<td>(16, 12, 4)</td>
</tr>
<tr>
<td>(16, 4, 12, 4)</td>
<td>(16, 12, 4)</td>
</tr>
<tr>
<td>(16, 4, 12, 4)</td>
<td>(16, 12, 4)</td>
</tr>
<tr>
<td>(16, 4, 12, 4)</td>
<td>(16, 12, 4)</td>
</tr>
</tbody>
</table>

Real-World Extension

Simplistic Assumptions

- Tasks are periodic
- Deadlines are identical with task's period time (D = 1)
- Tasks are independent
- Pre-emptive scheduling
- Worst case execution times are known
Scheduling

Real-World Extension

More Realistic Assumptions

- Tasks are periodic
  - we will introduce sporadic and aperiodic processes
- Deadlines are identical with task’s period time \( (D = T) \)
  - we will introduce arbitrary deadlines
- Tasks are independent
  - we will introduce schedules for interacting tasks
- Pre-emptive scheduling
  - we will introduce (briefly) cooperative scheduling
- Worst case execution times are known
  - we will introduce fault tolerant scheduling

Sporadic and Aperiodic Processes

FPS for hard real-time tasks

Introducing a soft real-time task

FPS lowest priority for soft real-time task

Introducing a server task on highest priority

FPS server task as normal task

Sporadic task utilizing deferrable server

H 2 / 7
© 2015 Uwe R. Zimmer, The Australian National University page 749 of 925 (chapter 7: “Scheduling” up to page 801)
Tasks with arbitrary deadlines

Proof of DMPO optimality

Swap two priorities out of W which violate DMPO:
1. \( t_i \) and \( t_j \) are two tasks in Q with \( P_i > P_j \) and \( D_i > D_j \) in W \( \Rightarrow \) DMPO
2. Generate \( W' \) by swapping \( P_i \) and \( P_j \) in \( W \) \( \Rightarrow \) DMPO
3. \( W' \) schedules Q because:
   - All \( t_k \in Q \) with \( P_k > P_j \) or \( P_k > P_i \) are unaffected.
   - \( t_i \) is schedulable in \( W' \) because \( P_i' > P_j' > P_i \) and \( D_i > D_j \)
   - \( t_j \) is schedulable in \( W' \) because:
     - in \( W' \) \( R_i > D_j \) \( \Rightarrow \) \( R_i > T_i \) \( \Rightarrow \) \( R_i > T_j \) 
     - in \( W' \) \( R_j > D_i \) \( \Rightarrow \) \( R_j > T_i \) \( \Rightarrow \) \( R_j > T_j \) 

Example:

Any task set Q which is schedulable by a FPS scheme \( W_i \) is also schedulable under DMPO.
Scheduling

Proof of DMPO optimality

Swap two priorities out of W which violate DMPO:

in W: $R_i < D_i < D_j \leq T_i = R_j < T_j$, $i$ interfered only once with $j$
also $i$, released once in $R_j$, and $R_j < R_i$

Swap two priorities out of W which violate DMPO:

Tasks with D > T

Deadline later than cycle time

Assumption every task $i$ is released only after the former release of $i$ is completed.

In case that $R_j > T_j$ for a specific scheduling situation, the following release of task $j$ is delayed by $R_j - T_j$.

Mind that $R_j > T_j$ cannot hold for all release situations, otherwise the task is not schedulable.

The worst case response time $R_i$ might thus be longer than $T_j$ but must still be shorter than $D_j$.

Proof of DMPO optimality

Swap all priorities out of W which violate DMPO:

Assumption every task $i$ is released only after the former release of $i$ is completed.

Since the response time $R_i$ can now be potentially greater than the cycle time $T$.

More Realistic Assumptions

Independent tasks

Interdependent tasks

Deadlines identical to cycle times as DMPO or RMPO.

Real-World Extension

More Realistic Assumptions

- Tasks are periodic
- Deadlines are identical with task’s period time ($D = T$)
- Tasks are independent
- Pre-emptive scheduling
- Worst case execution times are known

Tasks with arbitrary deadlines

Deadline later than cycle time

Assumption every task $i$ is released only after the former release of $i$ is completed.

Since the response time $R_i$ can now be potentially greater than the cycle time $T$.

More than one release $q$ of the task $i$ needs to be considered:

$R_i(q) = R_i + q \cdot C_i + \sum_{j = 1}^{q} \frac{D_j}{2} \leq \sum_{j = 1}^{q} \frac{D_j}{2}$

where $\forall R_i(q) - q - 1 = T_j \geq D_j$

$R_i$ is the blocking time; $q$ is the number of releases.

$R_i = \max \{R_i(q) - q - 1 = T_j \geq D_j\}$ and $q_{\max} = \left\lceil \frac{R_i}{T} \right\rceil$. 

Schedulable under DMPO or RMPO.
Priority inheritance

Maximal blocking time for task $t_j$: $B_j = \sum_{r=1}^{R} \text{usage}(r, i) C(r)$ with:
- $R$ denoting the number of critical sections,
- $\text{usage}(r, i)$ being a boolean function returning "1" for true and indicating the $r$ is used by:
  - at least one $t_j$ with $P_j < P_i$
  - and at least one $t_k$ with $P_k \geq P_i$
- $C(r)$ denoting the worst case computation time in critical section $r$

Each task can only be blocked once for each employed resource!

Task $t_j$ inherits priority $P_k$ of task $t_k$ if:
1. $P_j < P_k$.
2. Task $t_j$ has locked a resource $Q$.
3. Task $t_k$ is blocked waiting for the release of resource $Q$.

Task dependencies with multiple locks

Scheduling by DMPO/RMPO results in blocking for the higher priority tasks.

Scheduling by DMPO/RMPO with priority inheritance does not improve the result.
being a boolean function returning "1" for true if task.

Each task has a static priority

D has a dynamic priority:

Deadlocks are prevented (no hold and wait).

Number of context switches are reduced.

Avoids the deadlock!

Implications:

Tasks are dispatched only if all employed resources are available.

Deadlocks are prevented (no hold and wait).

Number of context switches are reduced.

Avoids the deadlock!

Implications:

Tasks are dispatched only if all employed resources are available.

Deadlocks are prevented (no hold and wait).

Number of context switches are reduced.

Avoids the deadlock!

Implications:

Tasks are dispatched only if all employed resources are available.

Deadlocks are prevented (no hold and wait).

Number of context switches are reduced.

Avoids the deadlock!

Implications:

Tasks are dispatched only if all employed resources are available.

Deadlocks are prevented (no hold and wait).

Number of context switches are reduced.

Avoids the deadlock!

Implications:

Tasks are dispatched only if all employed resources are available.

Deadlocks are prevented (no hold and wait).

Number of context switches are reduced.

Avoids the deadlock!

Implications:

Tasks are dispatched only if all employed resources are available.

Deadlocks are prevented (no hold and wait).

Number of context switches are reduced.

Avoids the deadlock!

Implications:

Tasks are dispatched only if all employed resources are available.

Deadlocks are prevented (no hold and wait).

Number of context switches are reduced.

Avoids the deadlock!

Implications:

Tasks are dispatched only if all employed resources are available.

Deadlocks are prevented (no hold and wait).

Number of context switches are reduced.

Avoids the deadlock!

Implications:

Tasks are dispatched only if all employed resources are available.

Deadlocks are prevented (no hold and wait).

Number of context switches are reduced.

Avoids the deadlock!

Implications:

Tasks are dispatched only if all employed resources are available.

Deadlocks are prevented (no hold and wait).

Number of context switches are reduced.

Avoids the deadlock!

Implications:

Tasks are dispatched only if all employed resources are available.

Deadlocks are prevented (no hold and wait).

Number of context switches are reduced.

Avoids the deadlock!

Implications:

Tasks are dispatched only if all employed resources are available.

Deadlocks are prevented (no hold and wait).

Number of context switches are reduced.

Avoids the deadlock!

Implications:

Tasks are dispatched only if all employed resources are available.

Deadlocks are prevented (no hold and wait).

Number of context switches are reduced.

Avoids the deadlock!

Implications:

Tasks are dispatched only if all employed resources are available.

Deadlocks are prevented (no hold and wait).

Number of context switches are reduced.

Avoids the deadlock!

Implications:

Tasks are dispatched only if all employed resources are available.

Deadlocks are prevented (no hold and wait).

Number of context switches are reduced.

Avoids the deadlock!

Implications:

Tasks are dispatched only if all employed resources are available.

Deadlocks are prevented (no hold and wait).

Number of context switches are reduced.

Avoids the deadlock!

Implications:

Tasks are dispatched only if all employed resources are available.

Deadlocks are prevented (no hold and wait).

Number of context switches are reduced.

Avoids the deadlock!
Scheduling

Non pre-emptive scheduling

Cooperative Scheduling

Response times:

\[ R_i = R_i^0 + f_i \]

with \( R_i^0 \) the pre-emption time of the first block.

For the simplified case of \( C_i = C - C_i = f_i - R_i^0 \):

\[ R_j = R_j^0 + f_j \]

with \( R_j^0 \) the execution time of the final block.

If error recovery is performed at the highest priority:

\[ R_j = R_j^0 + C_j + \sum_{i=j}^{\infty} \frac{B_i}{T_i} C_i + \frac{B_j}{T_j} C_j \]

If error recovery is performed at the second highest priority:

\[ R_j = R_j^0 + C_j + \sum_{i=j}^{\infty} \frac{B_i}{T_i} C_i + \max_{i \neq j} \frac{B_i}{T_i} C_i \]

If error recovery is performed at the lowest priority:

\[ R_j = R_j^0 + C_j + \sum_{i=j}^{\infty} \frac{B_i}{T_i} C_i + \max_{i \neq j} \frac{B_i}{T_i} C_i \]

Scheduling

Real-World Extension

More Realistic Assumptions

- Tasks are periodic
- Deadlines are identical with task’s period time \( (D - T) \)
- Tasks are independent
- Pre-emptive scheduling

Real-World Extension

Exceptions and Recoveries

Task \( T_j \) needs extra CPU-time \( C_j^0 \) for error recovery or exception handling and the minimum inter-arrival time between faults is \( T_j^0 \):

\[ R_j = R_j^0 + C_j + \sum_{i=j}^{\infty} \frac{B_i}{T_i} C_i + \frac{B_j}{T_j} C_j \]

If error recovery is performed at the highest priority:

\[ R_j = R_j^0 + C_j + \sum_{i=j}^{\infty} \frac{B_i}{T_i} C_i + \max_{i \neq j} \frac{B_i}{T_i} C_i \]

If error recovery is performed at the second highest priority:

\[ R_j = R_j^0 + C_j + \sum_{i=j}^{\infty} \frac{B_i}{T_i} C_i + \max_{i \neq j} \frac{B_i}{T_i} C_i \]

If error recovery is performed at the lowest priority:

\[ R_j = R_j^0 + C_j + \sum_{i=j}^{\infty} \frac{B_i}{T_i} C_i + \max_{i \neq j} \frac{B_i}{T_i} C_i \]

Scheduling

Real-World Extension

More Realistic Assumptions

- Tasks are periodic
- Deadlines are identical with task’s period time \( (D - T) \)
- Tasks are independent
- Pre-emptive scheduling

Real-World Extension

Exceptions and Recoveries

Task \( T_j \) needs extra CPU-time \( C_j^0 \) for error recovery or exception handling and the minimum inter-arrival time between faults is \( T_j^0 \):

\[ R_j = R_j^0 + C_j + \sum_{i=j}^{\infty} \frac{B_i}{T_i} C_i + \frac{B_j}{T_j} C_j \]

If error recovery is performed at the highest priority:

\[ R_j = R_j^0 + C_j + \sum_{i=j}^{\infty} \frac{B_i}{T_i} C_i + \max_{i \neq j} \frac{B_i}{T_i} C_i \]

If error recovery is performed at the second highest priority:

\[ R_j = R_j^0 + C_j + \sum_{i=j}^{\infty} \frac{B_i}{T_i} C_i + \max_{i \neq j} \frac{B_i}{T_i} C_i \]

If error recovery is performed at the lowest priority:

\[ R_j = R_j^0 + C_j + \sum_{i=j}^{\infty} \frac{B_i}{T_i} C_i + \max_{i \neq j} \frac{B_i}{T_i} C_i \]

Scheduling

Real-World Extension

More Realistic Assumptions

- Tasks are periodic
- Deadlines are identical with task’s period time \( (D - T) \)
- Tasks are independent
- Pre-emptive scheduling

Real-World Extension

Exceptions and Recoveries

Task \( T_j \) needs extra CPU-time \( C_j^0 \) for error recovery or exception handling and the minimum inter-arrival time between faults is \( T_j^0 \):

\[ R_j = R_j^0 + C_j + \sum_{i=j}^{\infty} \frac{B_i}{T_i} C_i + \frac{B_j}{T_j} C_j \]

If error recovery is performed at the highest priority:

\[ R_j = R_j^0 + C_j + \sum_{i=j}^{\infty} \frac{B_i}{T_i} C_i + \max_{i \neq j} \frac{B_i}{T_i} C_i \]

If error recovery is performed at the second highest priority:

\[ R_j = R_j^0 + C_j + \sum_{i=j}^{\infty} \frac{B_i}{T_i} C_i + \max_{i \neq j} \frac{B_i}{T_i} C_i \]

If error recovery is performed at the lowest priority:

\[ R_j = R_j^0 + C_j + \sum_{i=j}^{\infty} \frac{B_i}{T_i} C_i + \max_{i \neq j} \frac{B_i}{T_i} C_i \]
Summary

Scheduling

- Basic real-time scheduling
  - Fixed Priority Scheduling (FPS) with Rate Monotonic (RMPO) and Deadline Monotonic Priority Ordering (DMPO).
  - Earliest Deadline First (EDF).

- Real-world extensions
  - Aperiodic, sporadic, soft real-time tasks.
  - Deadlines different from period.
  - Synchronized tasks (priority inheritance, priority ceiling protocols).
  - Cooperative and deferred pre-emption scheduling.
  - Fault tolerance in terms of exception handling considerations.

- Language support
  - Ada, POSIX