Real-Time Scheduling

Uwe R. Zimmer – The Australian National University
References for this chapter

[Burns01] Alan Burns and Andy Wellings
Real-Time Systems and Programming Languages
Addison Wesley, third edition, 2001

[Murthy01] C. Siva Ram Murthy, G. Manimaran
Resource Management in Real-time Systems and Networks

all references and links are available on the course page
Scheduling in Real-Time Systems

- Concurrency *may* lead to non-determinism
- Non-determinism *may* make it harder to predict the timing behaviour

- RT-Scheduling schemes reduce non-determinism
Scheduling

A scheduling scheme provides two features:

- **Ordering the use of resources** (e.g. CPUs, networks)
- **Predicting the worst-case behaviour** of the system when the scheduling algorithm is applied

The prediction can then be used

☞ **at compile-run:**
  to **confirm the overall temporal requirements** of the application

or

☞ **at run-time:**
  to **permit acceptance** of additional usage/reservation requests.
Scheduling schemes

- **Static**
  all predictions and schedules are done off-line
  often better predictability ⇔ most hard real-time systems

- **Dynamic**
  run-time situation is taken into account
  more flexible, more efficient ⇔ most soft real-time systems
Scheduling as task queuing

- create
- batch
- ready
- suspended
- blocked, suspended
- blocked
- pre-emption or cycle done
- CPU
- term.
- block or synchronize
Real-time scheduling as task queuing

- Create batch
- Ready
- Pre-emption or cycle done
- CPU
- Term.

- Admitting new tasks if set is still schedulable
- Dispatching according to deadlines, priorities, or values
- Blocked
- Block or synchronize
Real-time scheduling

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

Example: Requested times
Real-time scheduling

Example: Deadlines
Dynamic scheduling

Earliest deadline first (EDF)

1. Determine (one of) the process(es) with the closest deadline.

2. Execute this process

   2-a until it finishes

   2-b or until another process’ deadline is found closer than the current one.

☞ Pre-emptive scheme

☞ Dynamic scheme,
   since the dispatched process is selected at run-time, due to the current deadlines.
Dynamic scheduling: Earliest Deadline First (EDF)

Earliest deadline first

1. Schedule the earliest deadline first
2. Avoid task switches (in case of equal deadlines)
Dynamic scheduling: Earliest Deadline First (EDF)

Earliest deadline first: Response times

worst case response times $R_i$ (maximal time in which the request from task $T_i$ is served).
Dynamic scheduling: Earliest Deadline First (EDF)

Earliest deadline first: Response times

worst case response times $R_i$ (maximal time in which the request from task $T_i$ is served):

- can be close or identical to deadlines.

- small or none spare capacity, if any task misses its expected computation time.
Real-Time & Embedded Systems

Dynamic scheduling: Earliest Deadline First (EDF)

Earliest deadline first: Maximal utilization

maximal possible utilization: \( \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \) sufficient & necessary test!

with \( C_i \), \( T_i \) the computation and cycle times of task \( i \)
(the deadlines \( D_i \) are assumed to be identical with the cycles times \( T_i \) here)
Static scheduling

Fixed Priority Scheduling (FPS), rate monotonic

1. Each process is assigned a fixed priority according to its cycle time $T_i$:

$$T_i < T_j \Rightarrow P_i > P_j$$

2. At any point in time: dispatch the process with the highest priority

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

- Rate monotonic ordering is optimal (in the framework of fixed priority schedulers), i.e. if a process set is schedulable under a FPS-scheme, it is also schedulable by applying rate monotonic priorities.
Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Rate monotonic priorities

assign task priorities according to the cycle times $T_i$ (identical to deadline $D_i$).
Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Rate monotonic priorities

assign task priorities according to the cycle times $T_i$ (identical to deadline $D_i$).
Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Rate monotonic priorities

Assign task priorities according to the cycle times $T_i$ (identical to deadline $D_i$).
Real-Time & Embedded Systems

Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Rate monotonic priorities

\[
\sum_{i=1}^{n} \frac{C_i}{T_i} \leq N \left( \frac{1}{2N} - 1 \right)
\]

sufficient, but not necessary test!
Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Rate monotonic priorities

utilization test: \[ \sum_{i=1}^{n} \frac{C_i}{T_i} = 1 > 0.779 \approx N \left( \frac{1}{2^N} - 1 \right) \] not guaranteed!
Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Rate monotonic priorities (reduced requests)

\[
\text{max. utilization test: } \sum_{i=1}^{n} \frac{C_i}{T_i} \leq N \left( \frac{1}{2^N} - 1 \right)
\]
Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Rate monotonic priorities (reduced requests)

Utilization: \( \frac{6}{16} + \frac{3}{12} + \frac{1}{4} = 0.875 > 0.779 \approx 3 \left( \frac{1}{2^3} - 1 \right) \); \( \sum_{i=1}^{n} \frac{C_i}{T_i} \leq N \left( \frac{1}{2^N} - 1 \right) \) not guaranteed!
Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Rate monotonic priorities (reduced requests)

utilization: 
\[ \frac{6}{16} + \frac{3}{12} + \frac{1}{4} = 0.875 > 0.779 \approx 3 \left( \frac{3}{2} \right) - 1; \quad \sum_{i=1}^{n} \frac{C_i}{T_i} \leq N \left( \frac{1}{2^N} - 1 \right) \]

not guaranteed!
Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Rate monotonic priorities (further reduced requests)

\[ (T_i, C_i) \]

\[ \begin{align*}
(16,4) & \quad \text{(1)} \\
(12,3) & \quad \text{(2)} \\
(4,1) & \quad \text{(3)}
\end{align*} \]

\[ t \]

max. utilization test:
\[ \sum_{i=1}^{n} \frac{C_i}{T_i} \leq N \left( \frac{1}{2^N} - 1 \right) \]
Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Rate monotonic priorities (further reduced requests)

Utilization: $\frac{4}{16} + \frac{3}{12} + \frac{1}{4} = 0.75 \leq 0.779 \approx 3 \left( \frac{1}{2}^3 - 1 \right); \quad \sum_{i=1}^{n} \frac{C_i}{T_i} \leq N \left( \frac{1}{2^N} - 1 \right)$

Guaranteed!
Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Rate monotonic priorities (further reduced requests)

\[\text{utilization: } \frac{4}{16} + \frac{3}{12} + \frac{1}{4} = 0.75 \leq 0.779 \approx 3 \left( \frac{1}{2^3} - 1 \right) ; \quad \sum_{i=1}^{n} \frac{C_i}{T_i} \leq N \left( \frac{1}{2^N} - 1 \right) \quad \Rightarrow \text{guaranteed!}\]
Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Utilisation based Analysis for FPS rate monotonic

\[ U \equiv \sum_{i=1}^{n} \frac{C_i}{T_i} \leq N \left( \frac{1}{2^N} - 1 \right) \equiv U_{max} \]

- with \( C_i \) the computation time and \( T_i \) the length of the period for task \( i \) out of \( N \) tasks and assuming that the deadline \( D_i = T_i \)

\( \equiv \) sufficient, but not necessary
\( \equiv \) O(n) complexity
Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Response time analysis (further reduced requests)

Calculate the worst case response times for each task individually.
Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Response time analysis (further reduced requests)

for the highest priority task: $R_3 = C_3$
Real-Time & Embedded Systems

Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Response time analysis (further reduced requests)

For other tasks: \( R_i = C_i + I_i = \text{computation } C_i + \text{interference } I_i \)
Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Response time analysis (further reduced requests)

for other tasks: \( R_i = C_i + \sum_{j > i} \left\lceil \frac{R_j}{T_j} \right\rceil C_j \)
Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Response time analysis

\[ R_i = C_i + \sum_{j > i} \left[ \frac{R_j}{T_j} \right] C_j \]

form recurrent equation:

\[ R_{i}^{k+1} = C_i + \sum_{j > i} \left[ \frac{R_j^k}{T_j} \right] C_j \] \hspace{1cm} (1)

starting with \( R_i^0 = C_i \)

Iterate (1) until \( R_i^{k+1} = R_i^k \) or \( R_i^{k+1} > T_i \)
Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Response time analysis

\[ R_i^{k+1} = C_i + \sum_{j>i} \left( \frac{R_i^k}{T_j} \right) C_j (1); R_i^0 = C_i \] 

Iterate (1) until \( R_i^{k+1} = R_i^k \) or \( R_i^{k+1} > T_i \)

Example (further reduced requests):

- set of tasks: \( \{(T_i, C_i)\} = \{(16, 4); (12, 3); (4, 1)\} \) at priorities \( \{1; 2; 3\} \); \( R_3^0 = 1 \)

\[ R_3^0 = 1 (\checkmark) \]
Response time analysis

\[ R_i^{k+1} = C_i + \sum_{j>i} \left[ \frac{R_i^k}{T_j} \right] C_j \] (1);

\[ R_i^0 = C_i - \text{Iterate (1) until } R_i^{k+1} = R_i^k \text{ or } R_i^{k+1} > T_i \]

Example (further reduced requests):

- set of tasks: \( \{(T_i, C_i)\} = \{(16, 4);(12, 3);(4, 1)\} \) at priorities \( \{1;2;3\} \); \( R_2^0 = 3 \)

\[ R_2^1 = 3 + \left[ \frac{3}{4} \right] 1 = 4 \]

\[ R_2^1 = 3 + \left[ \frac{4}{4} \right] 1 = 4(\checkmark) \]
Response time analysis

\[ R_i^{k+1} = C_i + \sum_{j > i} \left( \frac{R_i^k}{T_j} \right) C_j \] (1); \( R_i^0 = C_i \) — Iterate (1) until \( R_i^{k+1} = R_i^k \) or \( R_i^{k+1} > T_i \)

Example (further reduced requests):

- set of tasks: \( \{ (T_i, C_i) \} = \{ (16, 4); (12, 3); (4, 1) \} \) at priorities \( \{ 1; 2; 3 \} \); \( R_1^0 = 4 \)

\[ R_1^1 = 4 + \left[ \frac{4}{12} \right] 3 + \left[ \frac{4}{4} \right] 1 = 8 \]

\[ R_1^2 = 4 + \left[ \frac{8}{12} \right] 3 + \left[ \frac{8}{4} \right] 1 = 9 \]

\[ R_1^3 = 4 + \left[ \frac{9}{12} \right] 3 + \left[ \frac{9}{4} \right] 1 = 10 \] \( R_1^4 = 4 + \left[ \frac{10}{12} \right] 3 + \left[ \frac{10}{4} \right] 1 = 10(\checkmark) \)
Real-Time & Embedded Systems

Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Response time analysis

\[ R_i^{k+1} = C_i + \sum_{j>i} \left( \frac{R_i^k}{T_j} \right) C_j \] (1); \[ R_i^0 = C_i \] — Iterate (1) until \[ R_i^{k+1} = R_i^k \] or \[ R_i^{k+1} > T_i \]

Example (reduced requests):
- set of tasks: \( \{ (T_i, C_i) \} = \{ (16, 6); (12, 3); (4, 1) \} \) at priorities \{ 1; 2; 3 \}; \( R_1^0 = 6 \)

\[ R_1^1 = 6 + \left[ \frac{6}{12} \right] 3 + \left[ \frac{6}{4} \right] 1 = 11 \]

\[ R_1^2 = 6 + \left[ \frac{11}{12} \right] 3 + \left[ \frac{11}{4} \right] 1 = 12 \]

\[ R_1^3 = 6 + \left[ \frac{12}{12} \right] 3 + \left[ \frac{12}{4} \right] 1 = 12(\checkmark) \]
Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

**Response time analysis**

\[ R_i^{k+1} = C_i + \sum_{j>i} \left\lfloor \frac{R_i^k}{T_j} \right\rfloor C_j \] (1); \[ R_i^0 = C_i \] — Iterate (1) until \( R_i^{k+1} = R_i^k \) or \( R_i^{k+1} > T_i \)

Example (full requests):
- set of tasks: \( \{(T_i, C_i)\} = \{(16, 8);(12, 3);(4, 1)\} \) at priorities \( \{1;2;3\} \); \( R_1^0 = 8 \)

\[ R_1^1 = 8 + \left\lfloor \frac{8}{12} \right\rfloor 3 + \left\lfloor \frac{8}{4} \right\rfloor 1 = 13 \]

\[ R_1^2 = 8 + \left\lfloor \frac{13}{12} \right\rfloor 3 + \left\lfloor \frac{13}{4} \right\rfloor 1 = 18(\times) \]

\[ R_1^1 = 8 + \left\lfloor \frac{18}{12} \right\rfloor 3 + \left\lfloor \frac{18}{4} \right\rfloor 1 = 19(\times) \]

\[ R_2^1 = 8 + \left\lfloor \frac{19}{12} \right\rfloor 3 + \left\lfloor \frac{19}{4} \right\rfloor 1 = 19(\times) \]
Dynamic scheduling: Earliest Deadline First (EDF)

Response time analysis

The worst case for EDF is *not* necessarily when all tasks are released at once!
☞ all possible 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 \( \leq 1 \).
- Other tasks need to be considered only, if their deadline is closer or equal to the current task.
Dynamic scheduling: Earliest Deadline First (EDF)

Response time analysis

\[ R_i(a) = \left[ \frac{a}{T_i} + 1 \right] C_i + \sum_{j \neq i_{\text{min}}} \left\{ \left[ \frac{R_i(a)}{T_j} \right], \left[ \frac{a + T_i - T_j}{T_j} \right] + 1 \right\} C_j \]
Dynamic scheduling: Earliest Deadline First (EDF)

Response time analysis

\[ R_i(a) = \left[ \frac{a}{T_i} + 1 \right] C_i + \sum_{j \neq i_{\text{min}}} \left\{ \left[ \frac{R_i(a)}{T_j} \right], \left\{ 0, \left[ \frac{a + T_i - T_j}{T_j} \right] + 1 \right\} \right\} C_j \]

\[ R_i^{k+1}(a) = \left[ \frac{a}{T_i} + 1 \right] C_i + \sum_{j \neq i_{\text{min}}} \left\{ \left[ R_i^k(a) \right], \left\{ 0, \left[ \frac{a + T_i - T_j}{T_j} \right] + 1 \right\} \right\} C_j ^{(2)} \]

\[ \text{starting with } R_i^0(a) = a + C_i \]

\[ \text{Iterate (2) until } R_i^{k+1}(a) = R_i^k(a) \text{ (or } R_i^{k+1}(a) > a + T_i \text{ utilization beyond 100%!) } \]
Dynamic scheduling: Earliest Deadline First (EDF)

Response time analysis

\[ R_i(a) = \left[ \frac{a}{T_i} + 1 \right] C_i + \sum_{j \neq i_{\text{min}}} \left\{ \left[ \frac{R_i(a)}{T_j} \right], \max \left\{ 0, \left[ \frac{a + T_i - T_j}{T_j} \right] + 1 \right\} \right\} C_j \]

\[ R_i^{k+1}(a) = \left[ \frac{a}{T_i} + 1 \right] C_i + \sum_{j \neq i_{\text{min}}} \left\{ \left[ R_i^k(a) \right], \max \left\{ 0, \left[ \frac{a + T_i - T_j}{T_j} \right] + 1 \right\} \right\} C_j \quad (2) \]

Starting with \( R_i^0(a) = a + C_i \)

Iterate (2) until \( R_i^{k+1}(a) = R_i^k(a) \)

\[ R_i = \max \{ R_i(a) - a \} ; \quad \text{where} \ A = \text{scm}\{T_i\} \]
Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Response time analysis (further reduced requests)

\[ R_i = C_i + \sum_{j \neq i} \left[ \frac{R_j}{T_j} \right] C_j; \quad R_3 = 1; \quad R_2 = 4; \quad R_1 = 10 \text{ and } \sum_{i=1}^{n} \frac{C_i}{T_i} \leq N \left( \frac{1}{2^N} - 1 \right) \]
Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Response time analysis (reduced requests)

\[ R_i = C_i + \sum_{j>i} \left[ \frac{R_j}{T_j} \right] C_j; \ R_3 = 1 \checkmark; \ R_2 = 4 \checkmark; \ R_1 = 12 \checkmark \text{ but } \sum_{i=1}^{n} \frac{C_i}{T_i} > N \left( 2^{\frac{1}{N}} - 1 \right) \]
Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Response time analysis (full requests)

\[ R_i = C_i + \sum_{j>i} \left[ \frac{R_j}{T_j} \right] C_j; \quad R_3 = 1; \quad R_2 = 4; \quad R_1 = 19 \quad \text{and} \quad \sum_{i=1}^{n} \frac{C_i}{T_i} > N \left( \frac{1}{2^N} - 1 \right) \]
Dynamic scheduling: Earliest Deadline First (EDF)

Response time analysis (full requests)

testing all combinations in a hyper-period: LCM of \( \{T_i\} \) — here: 48
Dynamic scheduling: Earliest Deadline First (EDF)

Response time analysis (full requests)

Testing all combinations in a hyper-period: LCM of \( \{T_i\} \) — here: 48

\[
R_\text{-} : 16 \leq 16 \checkmark = T_\text{-} ; \quad R_\text{-} : 12 \leq 12 \checkmark = T_\text{-} ; \quad R_\text{-} : 4 \leq 4 \checkmark = T_\text{-}
\]
Dynamic scheduling: Earliest Deadline First (EDF)

Response time analysis (reduced requests)

relaxed task-set changes:

\[ R_\rightarrow : 16 \rightarrow 12 \leq 16 \checkmark = T \checkmark ; \quad R_\rightarrow : 12 \rightarrow 8 \leq 12 \checkmark = T \checkmark ; \quad R_\rightarrow : 4 \rightarrow 1 \leq 4 \checkmark = T \checkmark \]
Dynamic scheduling: Earliest Deadline First (EDF)

Response time analysis (further reduced requests)

Further relaxed task-set changes:

\[ R_- : 12 \rightarrow 10 \leq 16 \checkmark = T_- ; \quad R_- : 8 \rightarrow 6 \leq 12 \checkmark = T_- ; \quad R_- : 1 \rightarrow 1 \leq 4 \checkmark = T_- \]
### Real-time scheduling

#### Response time analysis (comparison)

<table>
<thead>
<tr>
<th></th>
<th>Fixed Priority Scheduling</th>
<th>Earliest Deadline First</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>utilization test</td>
<td>response times {R_i}</td>
</tr>
<tr>
<td>{(T_i, C_i)} = {(16, 8);(12, 3);(4, 1)}</td>
<td>✗ (1.000)</td>
<td>{ ✗, 4, 1}</td>
</tr>
<tr>
<td>{(T_i, C_i)} = {(16, 6);(12, 3);(4, 1)}</td>
<td>✗ (0.875)</td>
<td>{ 12, 4, 1}</td>
</tr>
<tr>
<td>{(T_i, C_i)} = {(16, 4);(12, 3);(4, 1)}</td>
<td>✓ (0.750)</td>
<td>{ 10, 4, 1}</td>
</tr>
</tbody>
</table>

\[
\sum_{i=1}^{n} \frac{C_i}{T_i} \leq N \left( \frac{1}{2^N} - 1 \right), \quad C_i + \sum_{j>i} \left\lfloor \frac{R_i}{T_j} \right\rfloor C_j, \quad \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \text{ check full hyper-cycle}
\]
Real-Time & Embedded Systems

Real-time scheduling

**Fixed Priority Scheduling ↔ Earliest Deadline First**

- EDF can handle higher (full) utilization than FPS.
- FPS is easier to implement and implies less run-time overhead

- Graceful degradation features (resource is over-booked):
  - FPS: processes with lower priorities will always miss their deadlines first.
  - EDF: any process can miss its deadline and can trigger a cascade of failed deadlines.

- Response time analysis and utilization tests:
  - FPS: $O(n)$ utilization test — response time analysis: fixed point equation
  - EDS: $O(n)$ utilization test — response time analysis: fixed point equation in hyper-cycle
Scheduling

Constraints which we used up to here:

- tasks are periodic
- deadlines are identical with task’s period time \((D = T)\)
- pre-emptive scheduling
- worst case execution times are known
- tasks are independent
Scheduling

Extensions which we will introduce:

• 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

• pre-emptive scheduling
  ☞ we will introduce (briefly) cooperative scheduling

• worst case execution times are known
  ☞ we will introduce fault tolerant scheduling

• tasks are independent
  ☞ we will introduce schedules for interacting tasks
Real-Time & Embedded Systems

Scheduling — real-world considerations

Including

aperiodic, sporadic & ‘soft’ real-time tasks
Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Hard real-time tasks
Real-Time & Embedded Systems

Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Introducing soft real-time tasks
Introducing soft real-time tasks

- set can be scheduled using average computation and period times
- hard real-time tasks can be scheduled under worst case conditions (including worst case behaviours of soft real-time tasks)
Static scheduling: FPS, rate monotonic + server

Introducing a server task

Server is established at a high priority
Introducing a server task: Deferrable Server

Deferrable Server (DS): Capacity replenished every $T_s$ (here: 8)
Introducing a server task: Sporadic Server

Sporadic Server (SS): Capacity replenished $T_s$ units after $t_s$

 POSIX
Introducing dual priorities

Start hard RT-tasks in low priorities; promote them in time to higher ones.
Dynamic scheduling: Earliest Deadline First+ aperiodic server

Introducing a server task to EDF

\[(T_i, C_i, D_i)\]
Introducing a server task to EDF
Dynamic scheduling: Earliest Deadline First + aperiodic tasks

Switching between EDF & Earliest Deadline Last (EDL)

Switching between EDF & EDL
Scheduling — real-world considerations

Including

tasks with deadlines shorter than their cycle time
Tasks with $D < T$

(Deadline earlier than inter-arrival period)

In fixed priority scheduling (FPS): change from:

Rate Monotonic Priority Ordering (RMPO)

to:

Deadline Monotonic Priority Ordering (DMPO)

Lemma

Any task set $Q$ which is schedulable by a FPS scheme $W$, is also schedulable by DMPO!
Any task set $Q$ which is schedulable by a FPS scheme $W$, is also schedulable by DMPO!

**Proof**

1. $i, j$ are two tasks in $Q$, with $(P_i = P_j + 1) \land (D_i > D_j)$ in $W \not\Rightarrow \text{DMPO}$

2. Generate $W'$ by swapping $P_i$ and $P_j$, i.e. $(P_i' < P_j') \land (D_i > D_j) \Rightarrow \text{DMPO}$

3. $W'$ is scheduling $Q$ because:
   
   3-a all $t_k \in Q$ with $P_k > P_i$ or $P_k < P_j$ are unaffected
   
   3-b $t_j$ is schedulable in $W'$ because $P_j' > P_i \Rightarrow R_j' \leq R_j \leq D_j$
   
   3-c $t_i$ is schedulable in $W'$ because …
Proof

1. \( t_i, t_j \) are two tasks in \( Q \), with \( P_i > P_j \) and \( D_i > D_j \) in \( W \) \( \not\in \text{DMPO} \)

2. Generate \( W' \) by swapping \( P_i \) and \( P_j \), i.e. \((P_i', < P_j') \wedge (D_i > D_j) \not\in \text{DMPO}\)

3. \( W' \) is scheduling \( Q \) because:

   3-a all \( t_k \in Q \) with \( P_k > P_i \) or \( P_k < P_j \) are unaffected

   3-b \( t_j \) is schedulable in \( W' \) because \( P_j' > P_i \Rightarrow R_j' \leq R_j \leq D_j \)

   3-c \( t_i \) is schedulable in \( W' \) because

   in \( W \): \( R_j \leq D_j < D_i \leq T_i \Rightarrow R_j < T_i \) i.e. \( t_i \) interfered only once with \( t_j \)

   in \( W \): \( t_j \) released once in \( R_j \), and \( R_i < R_j \)

   \( \not\in \) in \( W' \): \( t_j \) interferes only once with \( t_i \) \( \not\in \) \( R_i' = R_j \leq D_j < D_i \Rightarrow R_i' < D_i \)
Static scheduling: Fixed Priority Scheduling (FPS), DMPO

$t_i$ is still schedulable in $W'$ because:

\[ R_j \leq D_j < D_i \leq T_i \Rightarrow R_j < T_i \]

i.e. $t_i$ interfered only once with $t_j$

in $W$: $t_j$ released once in $R_j$, and $R_i < R_j$

in $W'$ …
Static scheduling: Fixed Priority Scheduling (FPS), DMPO

$t_i$ is still schedulable in $W'$ because:

- in $W$: $t_j$ released once in $R_j$, and $R_i < R_j$
- in $W'$: $t_j$ interferes only once with $t_i \iff R_i' = R_j \leq D_j < D_i \Rightarrow R_i' < D_i$
Static scheduling: Fixed Priority Scheduling (FPS), DMPO

Any task set $Q$ which is schedulable by a FPS scheme $W$, is also schedulable by DMPO.

- Swap all $t_i, t_j$ in $Q$, with $P_i > P_j$ and $D_i > D_j$ in $W$ resulting in all $t_i, t_j$ in $Q$, with $P_i > P_j$ have $D_i < D_j$

  Deadline Monotonic Priority Ordering (DMPO)

- Since the each swapping operation keeps schedulability, the final priority scheme (DMPO) is also schedulable.

  If FPS-DMPO is not schedulable, there is no schedulable FPS-scheme.
Scheduling — real-world considerations

Including

task interdependencies
Schedule for independent tasks

(independent task set)
Scheduling: Interdependencies

Schedule for independent tasks

(independent task set)
Scheduling: Interdependencies

Synchronized via lock

(Interdependent task set \(\Rightarrow\) lock \(\Rightarrow\) shared between \(\Rightarrow\) and \(\Rightarrow\))
Scheduling: Interdependencies

Synchronized via lock

.priority inversion

(interdependent task set lock shared between and )
Task $t_i$ inherits the priority of $t_j$, if:

1. $P_i < P_j$

2. task $t_i$ has locked a resource $Q$

3. task $t_j$ is blocked waiting for resource $Q$ to be released
Scheduling: Interdependencies

Priority inheritance

Maximal blocking time for task $t_i$: \( B_i = \sum_{r=1}^{R} \text{usage}(r, i)C(r) \)

- with $R$ the number of critical sections
- \text{usage}(r, i) a boolean \((0/1)\) function indicating that $r$ is used by at least one $t_j$ with $P_j < P_i$ and at least one $t_k$ with $P_k \geq P_i$
- $C(r)$ is the worst case computation time in critical section $r$

a task can only be blocked once for each employed resource!
Scheduling: Interdependencies

Priority inheritance

(inherit priority of \( \text{\textcolor{green}{\textit{\textbullet}}} \), when \( \text{\textcolor{green}{\textit{\textbullet}}} \) is in lock and \( \text{\textcolor{blue}{\textit{\textbullet}}} \) is dispatched)
Without priority inheritance

Priority inversion

(interdependent task set lock shared between and )
Scheduling: Interdependencies

A more complex example

(independent task set)
Scheduling: Interdependencies

A more complex example

(independent task set)
Scheduling: Interdependencies

Interdependencies
Real-Time & Embedded Systems

Scheduling: Interdependencies

Interdependencies

Priority inversion
Real-Time & Embedded Systems

Scheduling: Interdependencies

Priority inheritance

( and inherit priority of , when in lock and is dispatched)
Scheduling: Interdependencies

Priority inheritance

( and inherit priority of , when in lock and is dispatched)
One additional lock request
Scheduling: Interdependencies

One additional lock request

Deadlock
Real-Time & Embedded Systems

**Scheduling: Interdependencies: Priority ceiling protocols**

**Immediate ceiling priority protocol** *(POSIX, Ada, RT-Java)*

- Each task $t_i$ has static default priority $P_i$.
- Each resource (lock, monitor) $R_k$ has a static ceiling priority $C_k$, which is the maximum of priorities of the tasks $t_i$ which employ this resource.

$$C_k = \max_i \{\text{employ}(i, k) \cdot P_i\}$$

- Each task $t_i$ has a dynamic priority $P_i^D$, which is the maximum of its own static priority and the ceiling priorities of any resource it has locked.

$$P_i^D = \max\{P_i, \max_k \{\text{locked}(i, k) \cdot C_k\}\}$$
Immediate ceiling priority protocol (POSIX, Ada, RT-Java)

( and inherit the ceiling priority of or when entering the lock)
Scheduling: Interdependencies: Priority ceiling protocols

Immediate ceiling priority protocol (POSIX, Ada, RT-Java)

(, , and inherit the ceiling priority of or when entering the lock)
Scheduling: Interdependencies: Priority ceiling protocols

**Immediate ceiling priority protocol** (POSIX, Ada, RT-Java)

- Tasks are dispatched only if all employed resources are available.
- Deadlocks are prevented.
- Number of context switches is reduced.
Immediate ceiling priority protocol \((\text{POSIX, Ada, RT-Java})\)

Maximal blocking time: \(B_i = \max_r \left\{ \sum_{i=1}^{R} \text{usage}(r, i) \cdot C(r) \right\} \)

- with \(R\) the number of critical sections

- \(\text{usage}(r, i)\) a boolean \((0/1)\) function indicating that \(r\) is used by at least one \(t_j\) with \(P_j < P_i\) and at least one \(t_k\) with \(P_k \geq P_i\)

- \(C(r)\) is the worst case computation time in critical section \(r\)

A task can only be blocked once by any lower priority task!
Scheduling — real-world considerations

Considering

non-pre-emptive scheduling
Cooperative Scheduling

- All schemes up to here used pre-emptive dispatching.
- In interdependent task sets maximal blocking times $B_i$ can be determined for each task $t_i$ when employing a priority ceiling protocol.
- If the overall maximal blocking time $B_{max}$ can be accepted by all tasks, the number of pre-emptions can significantly reduced by:

**Deferred pre-emption – Cooperative Scheduling**

- Each task $t_i$ is divided in $k$ non-pre-emptive blocks of $C_{i,k} \leq B_{max}$
- All critical sections are completely enclosed in one code-block
- Each task calls a ‘de-scheduling’ kernel routine at the end of each code-block, i.e. ‘offering’ a task-switch.
Cooperative Scheduling

Deferred pre-emption – Cooperative Scheduling

- Number of task switches is reduced
- Caches, pre-fetching, and pipelines are more efficient
- Execution times are (a bit) easier to predict
- Schedules are simpler
- Interdependent task sets are schedulable deadlock free
Cooperative Scheduling

Deferred pre-emption – Cooperative Scheduling

Response times:

\[ R_i = R_i^n + F_i, \text{ with } R_i^{k+1} = B_{\text{max}} + C_i - F_i + \sum_{j > i} \left[ \frac{R_i^k}{T_j} \right] C_j \]

and \( F_i \) the execution time of the final code-block

… in the simplified case \( C = C_i = C_j = F_i = B_{\text{max}} \):

\[ R_i = R_i^n, \text{ with } R_i^{k+1} = C + \sum_{j > i} \left[ \frac{R_i^k}{T_j} \right] C \]

… and with even \( T = T_j \forall i \):

\[ R_i = C + \sum_{j > i} C \]
Cooperative Scheduling

Deferred pre-emption – Cooperative Scheduling

What’s the cost?

• Code block division need to be done very thoroughly.

• Additional protection against badly behaving (non-cooperative) tasks:
  ☞ Scheduler pre-empt tasks, which fail to offer a ‘de-schedule’ themselves.

• Due to a central $B_{max}$ value, additional tasks need to be engineered to participate in a specific cooperative schedule.

• Requires that a $B_{max}$ value can be accepted by all tasks
  ☞ very short and reactive tasks are excluded or will be treated specially.
Scheduling — real-world considerations

Considering
deadlines beyond the release period
Tasks with $D > T$

(Deadline later than inter-arrival period)

(a cross-over of a hard, periodic and a soft real-time task)

Assuming that a task $t_i$ is released only after a former release of $t_i$ is completed.

☞ In case that $R > T$ for a specific scheduling situation, the following release of task $t_i$ is delayed until completion of the former release.

☞ Mind that $R > T$ cannot hold for all release situations, otherwise the task is not schedulable.

☞ The worst case response time $R_i$ might thus be longer than $T_i$ but must still be shorter than $D_i$. 


**Tasks with \( D > T \)**

(Deadline later than inter-arrival period)

Since the response time \( R \) can now be potentially greater than the cycle time \( T \):

more than one release \( q \) of the task \( t_i \) needs to be considered:

\[
R_i(q) = B_i + qC_i + \sum_{j > i} \left[ \frac{R_j(q)}{T_j} \right] C_j \quad \text{where } \forall q \left( R_i(q) - (q - 1)T_i \leq D_i \right)!
\]

with \( q \): number of releases

\[
R_i = \max \left\{ R_i(q) - (q - 1)T_i \mid q \in \{1 \ldots q_{max}\} \right\} \quad \text{and} \quad q_{max} = \min \left\{ q \left| \frac{R_i(q)}{q} \leq T_i \right. \right\}
\]
Scheduling — real-world considerations

Considering

‘fault-tolerance’
(additional CPU-time for exception handling and recovery)
Fault Tolerance

Exceptions and Recoveries

Task $t_i$ needs extra CPU-time $C_i^f$ for error recovery or exception handling (done at $P_i$) and the minimum inter-arrival time between faults is $T_f$:

$$R_i = B_i + C_i + \sum_{j>i} \left\lceil \frac{R_i}{T_j} \right\rceil C_j + \max_{k \geq i} \left\lceil \frac{R_i}{T_f} \right\rceil C_k^f$$

if error recovery or exception handling is performed at the highest priority:

$$R_i = B_i + C_i + \sum_{j>i} \left\lceil \frac{R_i}{T_j} \right\rceil C_j + \max_k \left\lceil \frac{R_i}{T_f} \right\rceil C_k^f$$
Considering

task sets with release offsets
Task sets with offsets

Some task sets can be scheduled by introducing offsets to the release times, but …

☞ without any further restrictions this problem is NP-hard!

…

by introducing further assumptions about the granularity of the period times and deadlines:

☞ the schedulability analysis’ complexity can be reduced to be realistic.
Scheduling

Scheduling support

in different

real-time languages / environments
Ada95 provides:

- Task and interrupt priorities (static, dynamic, active)
- Task attributes
- Prioritized entry queues
- Priority ceiling locking (ICPP)
- Schedulers (at least FIFO within priorities (pre-emptive) is requested)

Ada95 does not provide:

- Earliest Deadline First (EDF)
- Sporadic servers (a Ada95-implementation of a sporadic server is on the course page)
- Direct task execution time measurements e.g. POSIX or VxWorks timers
package System is

    subtype Any_Priority is Integer
        range implementation-defined;

    subtype Priority is Any_Priority
        range Any_Priority'First .. implementation-defined;

    subtype Interrupt_Priority is Any_Priority
        range Priority'Last+1 .. Any_Priority'Last;

    Default_Priority : constant Priority :=
        (Priority'First + Priority'Last)/2;

end System;

package Ada.Dynamic_Priorities is

    procedure Set_Priority (Priority : in System.Any_Priority;
        T        : in Ada.Task_Identification.Task_ID
            := Ada.Task_Identification.Current_Task);

    function Get_Priority (T        : Ada.Task_Identification.Task_ID
        return System.Any_Priority;

end Ada.Dynamic_Priorities;
Environment Support for Scheduling: POSIX

POSIX provides:

- Task and interrupt priorities (static, dynamic, active)
- Prioritized message queues
- Priority ceiling locking (ICPP)
- Schedulers, priority based with at least:
  - FIFO, Round-Robin, Sporadic Server, possibly others
- Threads can be
  - ‘system contented’ or
  - ‘process contented’ (priority scheduling unclear in this case)
- Timers
Real-Time & Embedded Systems

Language Support for Scheduling: Real-Time JAVA

Real-Time Java provides:

- Task priorities (static, dynamic, active)
- Prioritized message queues
- Priority ceiling locking (ICPP)
- Schedulable objects (associated with threads) with
  - memory, release, and scheduling parameters
- Pre-emptive priority-oriented dispatching, possibly with a feasibility analysis
- An extendible scheduler class dynamic scheduling

Real-Time Java does not (necessarily) provide:

- Earliest Deadline First (EDF)
- Sporadic servers
- Direct task execution time measurements (might be provided)
public abstract class Scheduler
{
    protected Scheduler ();
    protected abstract boolean addToFeasibility (Schedulable s);
    public abstract void fireSchedulable (Schedulable s);
    public abstract boolean isFeasible ();
    protected abstract boolean removeFromFeasibility (Schedulable s);
    public boolean setIfFeasible (Schedulable s, ReleaseParameters r, MemoryParameters m);

    ...
}

Formulates an on-line schedulability analysis!
public class PriorityScheduler extends Scheduler {
    public static final int MAX_PRIORITY;
    public static final int MIN_PRIORITY;

    protected PriorityScheduler ();
    protected boolean addToFeasibility (Schedulable s);
    public    void    fireSchedulable (Schedulable s);
    public    boolean isFeasible ();
    protected boolean removeFromFeasibility (Schedulable s);
    public    boolean setIfFeasible (Schedulable s,
                                      ReleaseParameters r,
                                      MemoryParameters m);

    ...
}
Real-time Java

```java
public abstract class SchedulingParameters
{
    public SchedulingParameters ();
}
public class PriorityParameters extends SchedulingParameters
{
    public PriorityParameters (int priority);
    public int  getPriority ();
    public void setPriority (int priority) throws …;
    ...
}
```

‘Priority’ is the only default default scheduling parameter
Real-time Java

```java
public abstract class ReleaseParameters {
    protected ReleaseParameters (RelativeTime cost, RelativeTime deadline, AsyncEventHandler overrunHandler, AsyncEventHandler missHandler);
    public RelativeTime      getCost();
    public AsyncEventHandler getCostOverrunHandler();
    public RelativeTime      getDeadline();
    public AsyncEventHandler getDeadlineMissHandler();
}
```

Cost is an estimate of the max. execution time

Measuring execution time is not requested, i.e. the overrunHandler might never be activated!
public class PeriodicParameters extends ReleaseParameters
{
    public PeriodicParameters (HighResolutionTime start,
                                RelativeTime period,
                                RelativeTime cost, 
                                RelativeTime deadline,
                                AsyncEventHandler overrunHandler,
                                AsyncEventHandler missHandler);

    public RelativeTime getPeriod ();
    public HighResolutionTime getStart ();
    public void setPeriod (RelativeTime period);
    public void setStart (HighResolutionTime start);
}

most frequently used release parameters
public class AperiodicParameters extends ReleaseParameters
{
    public AperiodicParameters
        (RelativeTime cost,
         RelativeTime deadline,
         AsyncEventHandler overrunHandler,
         AsyncEventHandler missHandler);
}

these are the minimum release parameters (while cost might be used for feasibility analysis only)

the deadline-missHandler need to be supplied in any implementation
public class SporadicParameters extends AperiodicParameters
{
    public SporadicParameters
        (RelativeTime minInterarrival,
        RelativeTime cost,
        RelativeTime deadline,
        AsyncEventHandler overrunHandler,
        AsyncEventHandler missHandler);

    public RelativeTime getMinimumInterarrival ();
    public void          setMinimumInterarrival (RelativeTime minimum);
}
Summary

Scheduling

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

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

- Language support
  - Ada95, POSIX static, off-line analysis mostly — RT-Java on-line, dynamic scheduling