

RES

Real-Time Scheduling

Uwe R. Zimmer – The Australian National University

RES Real-Time & Embedded Systems

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
MIT Press, Cambridge, Massachusetts,
London, England

all references and links are available on the course page

© 2009 Uwe R. Zimmer, The Australian National University Page 551 of 769 (chapter 7: to 667)

RES Real-Time & Embedded Systems

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

© 2009 Uwe R. Zimmer, The Australian National University Page 552 of 769 (chapter 7: to 667)

RES Real-Time & Embedded Systems

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.

© 2009 Uwe R. Zimmer, The Australian National University Page 553 of 769 (chapter 7: to 667)

RES Real-Time & Embedded Systems

Scheduling schemes

- Static**
all predictions and schedules are done off-line
often better predictability \Rightarrow most hard real-time systems
- Dynamic**
run-time situation is taken into account
more flexible, more efficient \Rightarrow most soft real-time systems

© 2009 Uwe R. Zimmer, The Australian National University Page 554 of 769 (chapter 7: to 667)

RES Real-Time & Embedded Systems

Scheduling as task queuing

© 2009 Uwe R. Zimmer, The Australian National University Page 555 of 769 (chapter 7: to 667)

RES Real-Time & Embedded Systems

Real-time scheduling as task queuing

© 2009 Uwe R. Zimmer, The Australian National University Page 556 of 769 (chapter 7: to 667)

RES Real-Time & Embedded Systems

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.

\Rightarrow 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).

© 2009 Uwe R. Zimmer, The Australian National University Page 557 of 769 (chapter 7: to 667)

RES Real-Time & Embedded Systems

Real-time scheduling

Example: Requested times

© 2009 Uwe R. Zimmer, The Australian National University Page 558 of 769 (chapter 7: to 667)

RES Real-Time & Embedded Systems

Real-time scheduling

Example: Deadlines

© 2009 Uwe R. Zimmer, The Australian National University Page 559 of 769 (chapter 7: to 667)

RES Real-Time & Embedded Systems

Dynamic scheduling

Earliest deadline first (EDF)

- Determine (one of) the process(es) with the closest deadline.
- Execute this process
 - until it finishes
 - 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.

© 2009 Uwe R. Zimmer, The Australian National University Page 560 of 769 (chapter 7: to 667)

RES Real-Time & Embedded Systems

Dynamic scheduling: Earliest Deadline First (EDF)

Earliest deadline first

- Schedule the earliest deadline first
- Avoid task switches (in case of equal deadlines)

© 2009 Uwe R. Zimmer, The Australian National University Page 561 of 769 (chapter 7: to 667)

RES Real-Time & Embedded Systems

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).

© 2009 Uwe R. Zimmer, The Australian National University Page 562 of 769 (chapter 7: to 667)

RES Real-Time & Embedded Systems

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.

© 2009 Uwe R. Zimmer, The Australian National University Page 563 of 769 (chapter 7: to 667)

RES Real-Time & Embedded Systems

Dynamic scheduling: Earliest Deadline First (EDF)

Earliest deadline first: Maximal utilization

\Rightarrow maximal possible utilization: $\sum_{i=1}^n \frac{C_i}{T_i} \leq 1$ \Rightarrow 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)

© 2009 Uwe R. Zimmer, The Australian National University Page 564 of 769 (chapter 7: to 667)

RES Real-Time & Embedded Systems

Static scheduling

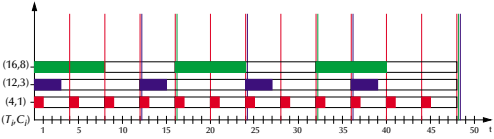
Fixed Priority Scheduling (FPS), rate monotonic

- Each process is assigned a fixed priority according to its cycle time T_i :
 $T_i < T_j \Rightarrow P_i > P_j$
- 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.

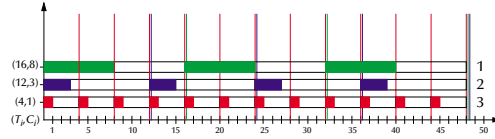
© 2009 Uwe R. Zimmer, The Australian National University Page 565 of 769 (chapter 7: to 667)

Rate monotonic priorities



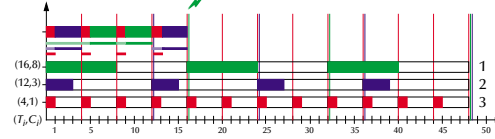
assign task priorities according to the cycle times T_i (identical to deadline D_i).

Rate monotonic priorities



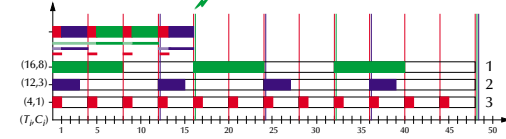
assign task priorities according to the cycle times T_i (identical to deadline D_i).

Rate monotonic priorities



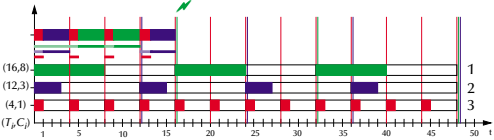
assign task priorities according to the cycle times T_i (identical to deadline D_i).

Rate monotonic priorities



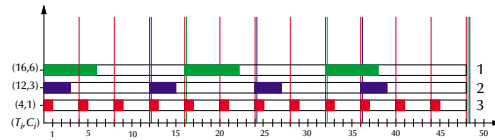
max. utilization test: $\sum_{i=1}^n \frac{C_i}{T_i} \leq N \left(\frac{1}{2^N} - 1 \right)$ sufficient, but not necessary test!

Rate monotonic priorities



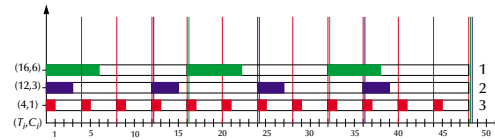
utilization test: $\sum_{i=1}^n \frac{C_i}{T_i} = 1 > 0.779 = N \left(\frac{1}{2^N} - 1 \right)$ not guaranteed!

Rate monotonic priorities (reduced requests)



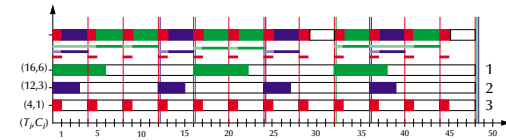
max. utilization test: $\sum_{i=1}^n \frac{C_i}{T_i} \leq N \left(\frac{1}{2^N} - 1 \right)$

Rate monotonic priorities (reduced requests)



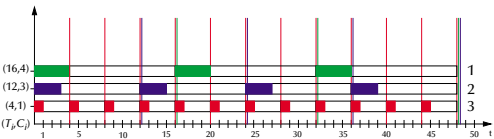
utilization: $\frac{6}{16} + \frac{3}{12} + \frac{1}{4} = 0.875 > 0.779 = 3 \left(\frac{1}{2^3} - 1 \right); \sum_{i=1}^n \frac{C_i}{T_i} \leq N \left(\frac{1}{2^N} - 1 \right)$ not guaranteed!

Rate monotonic priorities (reduced requests)



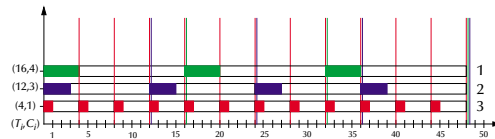
utilization: $\frac{6}{16} + \frac{3}{12} + \frac{1}{4} = 0.875 > 0.779 = 3 \left(\frac{1}{2^3} - 1 \right); \sum_{i=1}^n \frac{C_i}{T_i} \leq N \left(\frac{1}{2^N} - 1 \right)$ not guaranteed!

Rate monotonic priorities (further reduced requests)



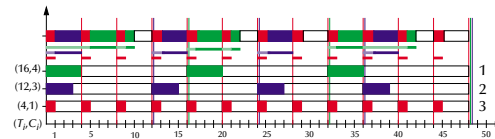
max. utilization test: $\sum_{i=1}^n \frac{C_i}{T_i} \leq N \left(\frac{1}{2^N} - 1 \right)$

Rate monotonic priorities (further reduced requests)



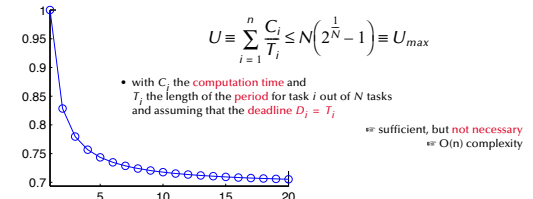
utilization: $\frac{4}{16} + \frac{3}{12} + \frac{1}{4} = 0.75 \leq 0.779 = 3 \left(\frac{1}{2^3} - 1 \right); \sum_{i=1}^n \frac{C_i}{T_i} \leq N \left(\frac{1}{2^N} - 1 \right)$ guaranteed!

Rate monotonic priorities (further reduced requests)

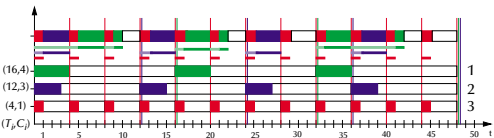


utilization: $\frac{4}{16} + \frac{3}{12} + \frac{1}{4} = 0.75 \leq 0.779 = 3 \left(\frac{1}{2^3} - 1 \right); \sum_{i=1}^n \frac{C_i}{T_i} \leq N \left(\frac{1}{2^N} - 1 \right)$ guaranteed!

Utilisation based Analysis for FPS rate monotonic

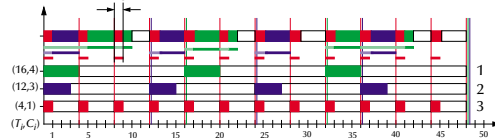


Response time analysis (further reduced requests)



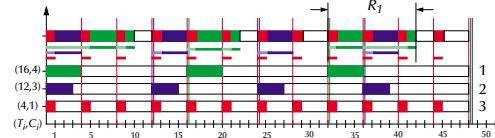
calculate the worst case response times for each task individually.

Response time analysis (further reduced requests)



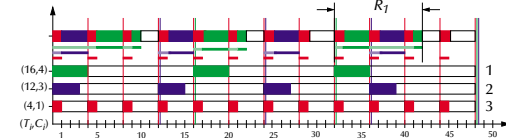
for the highest priority task: $R_3 = C_3$

Response time analysis (further reduced requests)



for other tasks: $R_j = C_j + I_j =$ computation $C_j +$ interference I_j

Response time analysis (further reduced requests)



for other tasks: $R_j = C_j + \sum_{i>j} \left\lceil \frac{R_j}{T_i} \right\rceil C_i$

Response time analysis

$$R_i = C_i + \sum_{j>i} \left\lceil \frac{R_j}{T_j} \right\rceil C_j$$

≡ fixed-point equation!

≡ form recurrent equation: $R_i^{k+1} = C_i + \sum_{j>i} \left\lceil \frac{R_j^k}{T_j} \right\rceil C_j(1)$

≡ starting with $R_i^0 = C_i$
 ≡ Iterate (1) until $R_i^{k+1} = R_i^k$ or $R_i^{k+1} > T_i$

Response time analysis

$$\equiv R_i^{k+1} = C_i + \sum_{j>i} \left\lceil \frac{R_j^k}{T_j} \right\rceil C_j(1); R_i^0 = C_i - \text{Iterate (1) until } R_i^{k+1} = R_i^k \text{ or } R_i^{k+1} > T_i$$

Example (further reduced requests):

• set of tasks: $\{(T_p, C_p)\} = \{(16, 4); (12, 3); (4, 1)\}$ at priorities $\{1; 2; 3\}$; $R_3^0 = 1$
 $R_3^0 = 1 (\checkmark)$

Response time analysis

$$\equiv R_i^{k+1} = C_i + \sum_{j>i} \left\lceil \frac{R_j^k}{T_j} \right\rceil C_j(1); R_i^0 = C_i - \text{Iterate (1) until } R_i^{k+1} = R_i^k \text{ or } R_i^{k+1} > T_i$$

Example (further reduced requests):

• set of tasks: $\{(T_p, C_p)\} = \{(16, 4); (12, 3); (4, 1)\}$ at priorities $\{1; 2; 3\}$; $R_2^0 = 3$

$$\equiv R_2^1 = 3 + \left\lceil \frac{3}{4} \right\rceil 1 = 4$$

$$\equiv R_2^1 = 3 + \left\lceil \frac{4}{4} \right\rceil 1 = 4 (\checkmark)$$

Response time analysis

$$\equiv R_i^{k+1} = C_i + \sum_{j>i} \left\lceil \frac{R_j^k}{T_j} \right\rceil C_j(1); R_i^0 = C_i - \text{Iterate (1) until } R_i^{k+1} = R_i^k \text{ or } R_i^{k+1} > T_i$$

Example (further reduced requests):

• set of tasks: $\{(T_p, C_p)\} = \{(16, 4); (12, 3); (4, 1)\}$ at priorities $\{1; 2; 3\}$; $R_1^0 = 4$

$$\equiv R_1^1 = 4 + \left\lceil \frac{4}{12} \right\rceil 3 + \left\lceil \frac{4}{4} \right\rceil 1 = 8$$

$$\equiv R_1^2 = 4 + \left\lceil \frac{8}{12} \right\rceil 3 + \left\lceil \frac{8}{4} \right\rceil 1 = 9$$

$$\equiv R_1^3 = 4 + \left\lceil \frac{9}{12} \right\rceil 3 + \left\lceil \frac{9}{4} \right\rceil 1 = 10 \Rightarrow R_1^4 = 4 + \left\lceil \frac{10}{12} \right\rceil 3 + \left\lceil \frac{10}{4} \right\rceil 1 = 10 (\checkmark)$$

Response time analysis

$$\equiv R_i^{k+1} = C_i + \sum_{j>i} \left\lceil \frac{R_j^k}{T_j} \right\rceil C_j(1); R_i^0 = C_i - \text{Iterate (1) until } R_i^{k+1} = R_i^k \text{ or } R_i^{k+1} > T_i$$

Example (reduced requests):

• set of tasks: $\{(T_p, C_p)\} = \{(16, 6); (12, 3); (4, 1)\}$ at priorities $\{1; 2; 3\}$; $R_1^0 = 6$

$$\equiv R_1^1 = 6 + \left\lceil \frac{6}{12} \right\rceil 3 + \left\lceil \frac{6}{4} \right\rceil 1 = 11$$

$$\equiv R_1^2 = 6 + \left\lceil \frac{11}{12} \right\rceil 3 + \left\lceil \frac{11}{4} \right\rceil 1 = 12$$

$$\equiv R_1^3 = 6 + \left\lceil \frac{12}{12} \right\rceil 3 + \left\lceil \frac{12}{4} \right\rceil 1 = 12 (\checkmark)$$

Response time analysis

$$\equiv R_i^{k+1} = C_i + \sum_{j>i} \left\lceil \frac{R_j^k}{T_j} \right\rceil C_j(1); R_i^0 = C_i - \text{Iterate (1) until } R_i^{k+1} = R_i^k \text{ or } R_i^{k+1} > T_i$$

Example (full requests):

• set of tasks: $\{(T_p, C_p)\} = \{(16, 8); (12, 3); (4, 1)\}$ at priorities $\{1; 2; 3\}$; $R_1^0 = 8$

$$\equiv R_1^1 = 8 + \left\lceil \frac{8}{12} \right\rceil 3 + \left\lceil \frac{8}{4} \right\rceil 1 = 13$$

$$\equiv R_1^2 = 8 + \left\lceil \frac{13}{12} \right\rceil 3 + \left\lceil \frac{13}{4} \right\rceil 1 = 18 (\star)$$

$$\equiv R_1^2 = 8 + \left\lceil \frac{18}{12} \right\rceil 3 + \left\lceil \frac{18}{4} \right\rceil 1 = 19 (\star) \Rightarrow R_1^2 = 8 + \left\lceil \frac{19}{12} \right\rceil 3 + \left\lceil \frac{19}{4} \right\rceil 1 = 19 (\star)$$

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 ≤ 1 .
- Other tasks need to be considered only, if their deadline is closer or equal to the current task.

Response time analysis

$$R_i(a) = \left\lceil \frac{a}{T_i} + 1 \right\rceil C_i + \sum_{j \neq i} \min \left\{ \left\lceil \frac{R_j(a)}{T_j} \right\rceil, \left\lceil \frac{a + T_j - T_j}{T_j} + 1 \right\rceil \right\} C_j$$

Response time analysis

$$R_i(a) = \left\lceil \frac{a}{T_i} + 1 \right\rceil C_i + \sum_{j \neq i} \min \left\{ \left\lceil \frac{R_j(a)}{T_j} \right\rceil, \max \left\{ 0, \left\lceil \frac{a + T_j - T_j}{T_j} + 1 \right\rceil \right\} \right\} C_j$$

$$\equiv R_i^{k+1}(a) = \left\lceil \frac{a}{T_i} + 1 \right\rceil C_i + \sum_{j \neq i} \min \left\{ \left\lceil \frac{R_j^k(a)}{T_j} \right\rceil, \max \left\{ 0, \left\lceil \frac{a + T_j - T_j}{T_j} + 1 \right\rceil \right\} \right\} C_j(2)$$

≡ starting with $R_i^0(a) = a + C_i$
 ≡ Iterate (2) until $R_i^{k+1}(a) = R_i^k(a)$ (or $R_i^{k+1}(a) > a + T_i$, \Rightarrow utilization beyond 100%)

Response time analysis

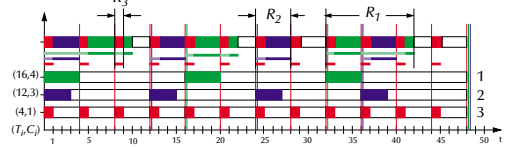
$$R_i(a) = \left\lceil \frac{a}{T_i} + 1 \right\rceil C_i + \sum_{j \neq i} \min \left\{ \left\lceil \frac{R_j(a)}{T_j} \right\rceil, \max \left\{ 0, \left\lceil \frac{a + T_j - T_j}{T_j} + 1 \right\rceil \right\} \right\} C_j$$

$$\equiv R_i^{k+1}(a) = \left\lceil \frac{a}{T_i} + 1 \right\rceil C_i + \sum_{j \neq i} \min \left\{ \left\lceil \frac{R_j^k(a)}{T_j} \right\rceil, \max \left\{ 0, \left\lceil \frac{a + T_j - T_j}{T_j} + 1 \right\rceil \right\} \right\} C_j(2)$$

≡ starting with $R_i^0(a) = a + C_i$
 ≡ Iterate (2) until $R_i^{k+1}(a) = R_i^k(a)$

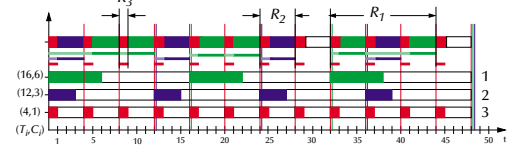
$$\equiv R_i = \max_{a \in A} \{R_i(a) - a\} \text{ where } A = \text{scm}\{T_i\}$$

Response time analysis (further reduced requests)



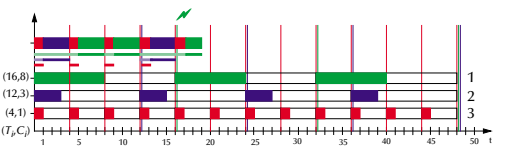
$$\equiv R_j = C_j + \sum_{i>j} \left\lceil \frac{R_i}{T_i} \right\rceil C_i; R_3 = 1 \checkmark; R_2 = 4 \checkmark; R_1 = 10 \checkmark \text{ and } \sum_{i=1}^n \frac{C_i}{T_i} \leq N \left(2^{\frac{1}{N}} - 1 \right) \checkmark$$

Response time analysis (reduced requests)



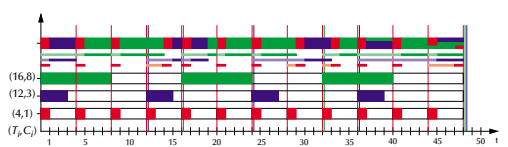
$$\equiv R_j = C_j + \sum_{i>j} \left\lceil \frac{R_i}{T_i} \right\rceil C_i; R_3 = 1 \checkmark; R_2 = 4 \checkmark; R_1 = 12 \checkmark \text{ but } \sum_{i=1}^n \frac{C_i}{T_i} > N \left(2^{\frac{1}{N}} - 1 \right) \star$$

Response time analysis (full requests)



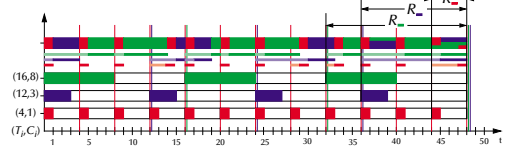
$$\equiv R_j = C_j + \sum_{i>j} \left\lceil \frac{R_i}{T_i} \right\rceil C_i; R_3 = 1 \checkmark; R_2 = 4 \checkmark; R_1 = 19 \star \text{ and } \sum_{i=1}^n \frac{C_i}{T_i} > N \left(2^{\frac{1}{N}} - 1 \right) \star$$

Response time analysis (full requests)



≡ testing all combinations in a hyper-period: LCM of $\{T_i\}$ – here: 48

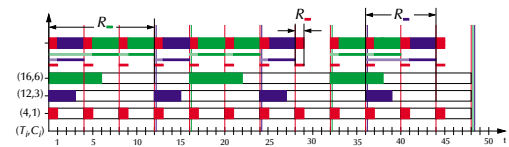
Response time analysis (full requests)



≡ testing all combinations in a hyper-period: LCM of $\{T_i\}$ – here: 48

$$R_{..}: 16 \leq 16 \checkmark = T_{..}; \quad R_{..}: 12 \leq 12 \checkmark = T_{..}; \quad R_{..}: 4 \leq 4 \checkmark = T_{..}$$

Response time analysis (reduced requests)

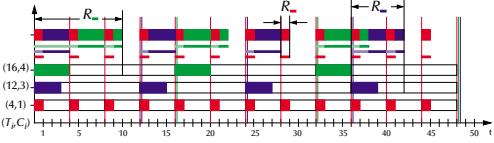


≡ relaxed task-set changes:

$$R_{..}: 16 \rightarrow 12 \leq 16 \checkmark = T_{..}; \quad R_{..}: 12 \rightarrow 8 \leq 12 \checkmark = T_{..}; \quad R_{..}: 4 \rightarrow 1 \leq 4 \checkmark = T_{..}$$

Dynamic scheduling: Earliest Deadline First (EDF)

Response time analysis (further reduced requests)



further relaxed task-set changes:
 $R_m: 12 \rightarrow 10 \leq 16 \checkmark = T_m; \quad R_m: 8 \rightarrow 6 \leq 12 \checkmark = T_m; \quad R_m: 1 \rightarrow 1 \leq 4 \checkmark = T_m$

Real-time scheduling

Response time analysis (comparison)

	Fixed Priority Scheduling		Earliest Deadline First	
	utilization test	response times $\{R_i\}$	utilization test	response times $\{R_i\}$
$\{(T_p, C_i)\} = \{(16, 8); (12, 3); (4, 1)\}$	✘ (1.000)	$\{8, 4, 1\}$	✓ (1.000)	$\{16, 12, 4\}$
$\{(T_p, C_i)\} = \{(16, 6); (12, 3); (4, 1)\}$	✘ (0.875)	$\{12, 4, 1\}$	✓ (0.875)	$\{12, 8, 1\}$
$\{(T_p, C_i)\} = \{(16, 4); (12, 3); (4, 1)\}$	✓ (0.750)	$\{10, 4, 1\}$	✓ (0.750)	$\{10, 6, 1\}$
$\sum_{i=1}^n \frac{C_i}{T_i} \leq N \left(\frac{1}{2^N} - 1 \right)$	$C_i + \sum_{j>1} \left\lceil \frac{R_j}{T_j} \right\rceil C_i$	$\sum_{i=1}^n \frac{C_i}{T_i} \leq 1$	check full hyper-cycle	

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**

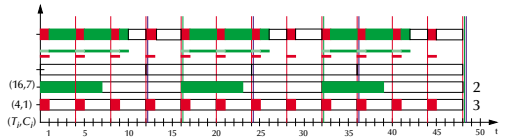
Scheduling — real-world considerations

Including

aperiodic, sporadic & 'soft' real-time tasks

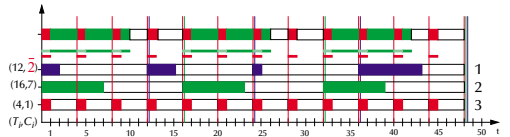
Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Hard real-time tasks



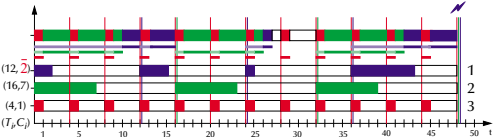
Static scheduling: Fixed Priority Scheduling (FPS), rate monotonic

Introducing soft real-time tasks



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

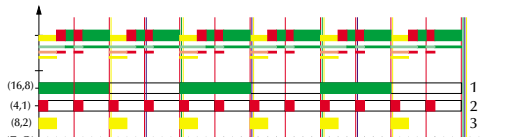
Introducing soft real-time tasks



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

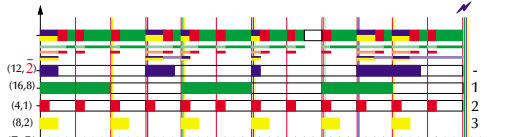
Introducing a server task



Server is established at a high priority

Static scheduling: FPS, rate monotonic + server

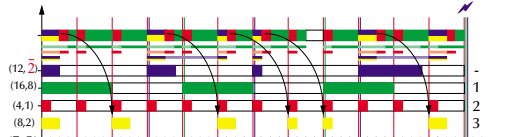
Introducing a server task: Deferrable Server



Deferrable Server (DS): Capacity replenished every T_s (here: 8)

Static scheduling: FPS, rate monotonic + server

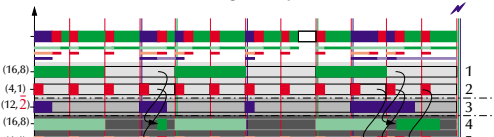
Introducing a server task: Sporadic Server



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

Static scheduling: Fixed Priority Scheduling (FPS), dual-priorities

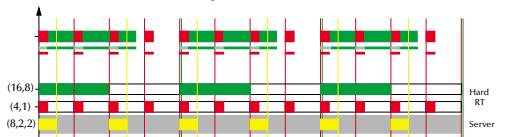
Introducing dual priorities



start hard rt-tasks in low priorities; promote them in time to higher ones

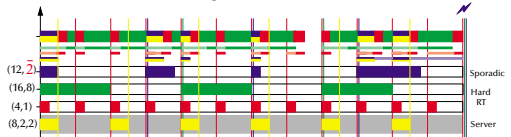
Dynamic scheduling: Earliest Deadline First+ aperiodic server

Introducing a server task to EDF



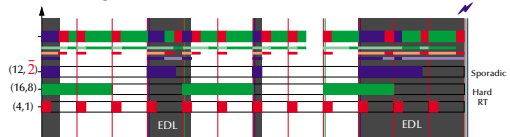
Dynamic scheduling: Earliest Deadline First + aperiodic server

Introducing a server task to EDF



Dynamic scheduling: Earliest Deadline First + aperiodic tasks

Switching between EDF & Earliest Deadline Last (EDL)



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!

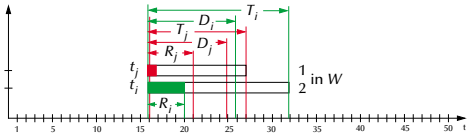
Proof

- i, j are two tasks in Q, with $(P_i = P_j + 1) \wedge (D_i > D_j)$ in $W \Rightarrow$ -DMPO
- Generate W' by swapping P_i and P_j , i.e. $(P_i' < P_j') \wedge (D_i > D_j) \Rightarrow$ DMPO
- W' is scheduling Q because:
 - all $t_k \in Q$ with $P_k > P_i$ or $P_k < P_j$ are unaffected
 - t_j is schedulable in W' because $P_j' > P_i \Rightarrow R_j' \leq R_i \leq D_j$
 - t_i is schedulable in W' because ...

Proof

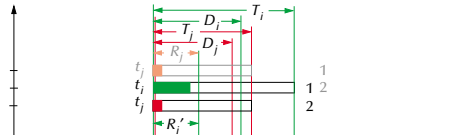
- t_i, t_j are two tasks in Q, with $P_i > P_j$ and $D_i > D_j$ in $W \Rightarrow$ -DMPO
- Generate W' by swapping P_i and P_j , i.e. $(P_i' < P_j') \wedge (D_i > D_j) \Rightarrow$ DMPO
- W' is scheduling Q because:
 - all $t_k \in Q$ with $P_k > P_i$ or $P_k < P_j$ are unaffected
 - t_j is schedulable in W' because $P_j' > P_i \Rightarrow R_j' \leq R_i \leq D_j$
 - 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_j interfered only once with t_i
 - in W : t_j released once in R_j , and $R_j < R_j$
 - \Rightarrow in W' : t_j interferes only once with $t_i \Rightarrow R_j' = R_j \leq D_j < D_i \Rightarrow R_j' < D_i$

t_j is still schedulable in W' because:



in W : $R_j \leq D_j < D_i \leq T_i \Rightarrow R_j < T_i$ i.e. t_j interfered only once with t_i
in W : t_j released once in R_j , and $R_j < R_j$
 \Rightarrow in W' ...

t_i is still schedulable in W' because:



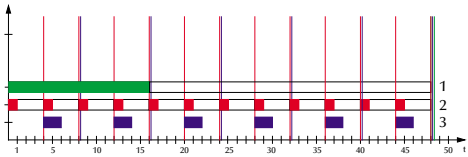
in W : t_j released once in R_j , and $R_i < R_j$
 \Rightarrow in W' : t_j interferes only once with $t_i \Rightarrow R_j' = R_j \leq D_j < D_i \Rightarrow R_j' < D_i$

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 $D_i < D_j$
 \Rightarrow Deadline Monotonic Priority Ordering (DMPO)
 - Since the each swapping operation keeps schedulability, the final priority scheme (DMPO) is also schedulable.
- \Rightarrow If FPS-DMPO is not schedulable, there is no schedulable FPS-scheme.

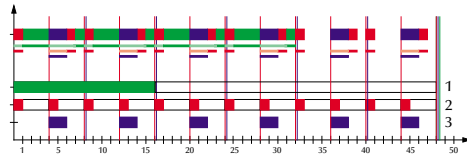
Including
task interdependencies

Schedule for independent tasks



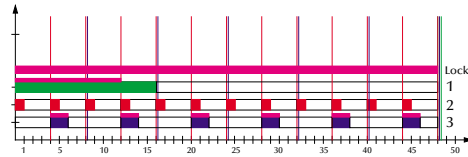
(independent task set)

Schedule for independent tasks



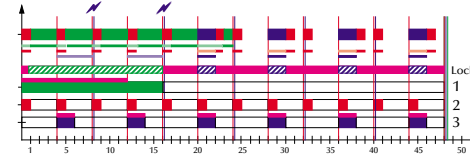
(independent task set)

Synchronized via lock



(interdependent task set \Rightarrow lock shared between 1 and 2)

Synchronized via lock



\Rightarrow Priority inversion
(interdependent task set \Rightarrow lock shared between 1 and 2)

Priority inheritance

Task t_j inherits the priority of t_i , if:

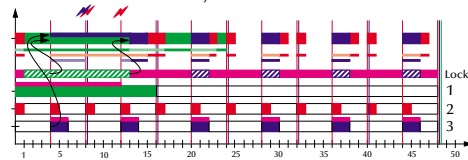
- $P_j < P_i$
- task t_j has locked a resource Q
- task t_j is blocked waiting for resource Q to be released

Priority inheritance

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

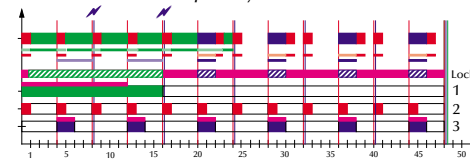
- with R the number of critical sections
- $\text{usage}(r, i)$ a boolean (0/1) function indicating that r is used by at least one t_j with $P_j < P_i$ and at least one t_k with $P_k \geq P_i$
- $C(r)$ is the worst case computation time in critical section r
a task can only be blocked once for each employed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

Priority inheritance



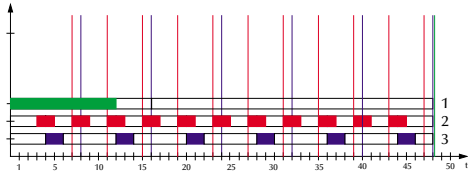
(1 inherits priority of 2, when 1 is in lock and 2 is dispatched)

Without priority inheritance



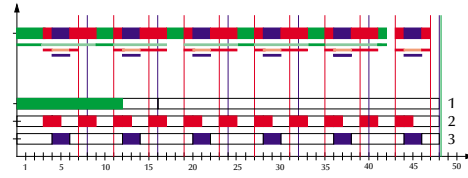
\Rightarrow Priority inversion
(interdependent task set \Rightarrow lock shared between 1 and 2)

A more complex example



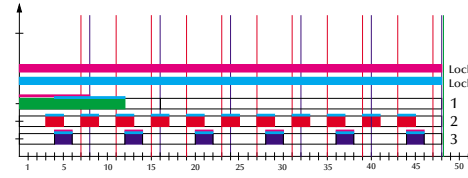
(independent task set)

A more complex example

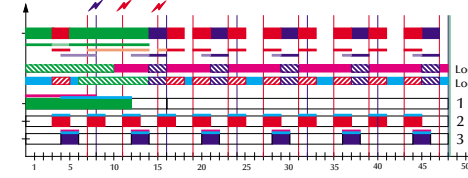


(independent task set)

Interdependencies

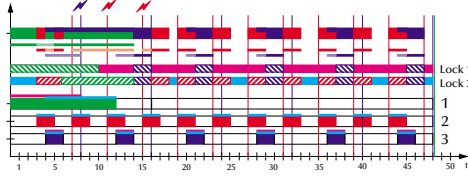


Interdependencies



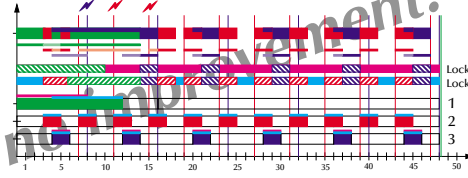
Priority inversion

Priority inheritance



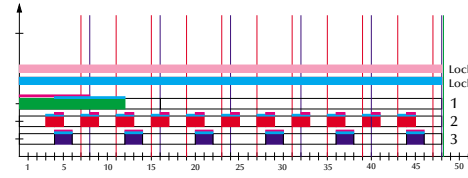
(and inherit priority of , when in lock and is dispatched)

Priority inheritance

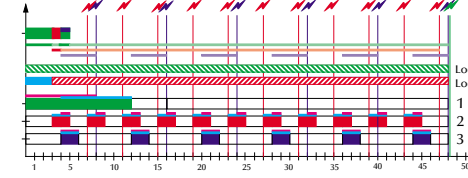


(and inherit priority of , when in lock and is dispatched)

One additional lock request



One additional lock request



Deadlock

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

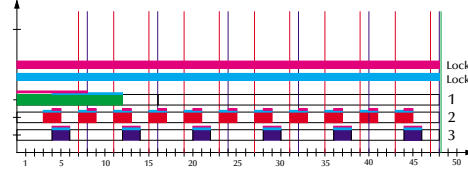
- Each task t_i has static default priority P_i .
- Each resource (lock, monitor) R_k has a static ceiling priority C_k , which is the maximum of priorities of the tasks t_i which employ this resource.

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

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

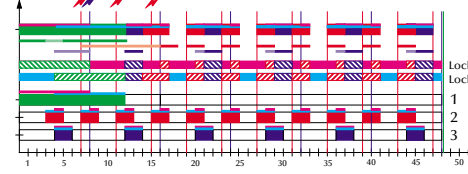
$$P_j^D = \max \{ P_j, \max_k \{ \text{locked}(j, k) \cdot C_k \} \}$$

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



(, and inherit the ceiling priority of or when entering the lock)

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



(, and inherit the ceiling priority of or when entering the lock)

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 (POSIX, Ada, RT-Java)

$$\text{Maximal blocking time: } B_j = \max_r \{ \text{usage}(r, j) \cdot C(r) \}$$

- with R the number of critical sections
- $\text{usage}(r, j)$ 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 $P_k \geq P_j$
- $C(r)$ is the worst case computation time in critical section r

a task can only be blocked once by any lower priority task!

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 be reduced by:

Deferred pre-emption – Cooperative Scheduling

- Each task t_j is divided in k non-pre-emptive blocks of $C_{jk} \leq B_{max}$
- 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

Deferred pre-emption – Cooperative Scheduling

Response times:

$$R_i = R_i^n + F_{ij}, \text{ with } R_i^{k+1} = B_{max} + C_i - F_{ij} + \sum_{j>i} \left\lceil \frac{R_j^k}{T_j} \right\rceil C_j$$

and F_{ij} the execution time of the final code-block

... in the simplified case $C = C_i = F_{ij} = B_{max}$:

$$R_i = R_i^n, \text{ with } R_i^{k+1} = C + \sum_{j>i} \left\lceil \frac{R_j^k}{T_j} \right\rceil C$$

... and with even $T = T_j \forall i$:

$$R_i = C + \sum_{j>i} C$$

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-empts** 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_j 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:

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

with q : number of releases

$$R_i = \max_{q \in \{1 \dots q_{max}\}} \{R_i(q) - (q-1)T_i\} \text{ and } q_{max} = \left\lfloor \frac{R_i(q)}{q} \leq T_i \right\rfloor$$

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 :

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

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

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

Scheduling — real-world considerations

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

Language Support for Scheduling : Ada95

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

Language Support for Scheduling : Ada95

```
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 : in Ada.Task_Identification.Task_ID
    := Ada.Task_Identification.Current_Task)
    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

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)

Language Support for Scheduling : Real-Time JAVA

Real-time Java

```
public abstract class Scheduler
{
  protected Scheduler ();
  protected abstract boolean addToFeasibility (Schedulable s);
  protected 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!

Language Support for Scheduling : Real-Time JAVA

Real-time Java

```
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);
} ...
```

This PriorityScheduler is the only requested instantiation

Real-time 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 scheduling parameter

Real-time 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!

Real-time Java

```
public class PeriodicParameters extends ReleaseParameters
{
    public PeriodicParameters
    (HighResolutionTime start,
     HighResolutionTime 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

Real-time Java

```
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

Real-time Java

```
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);
}
```

Sporadic events are not allowed to come in bursts!

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 tasks (priority inheritance, priority ceiling protocols)
- Language support
 - Ada95, POSIX #= static, off-line analysis mostly – RT-Java #= on-line, dynamic scheduling