Scheduling of Real-Time Systems
Fixed Priority and Earliest Deadline First

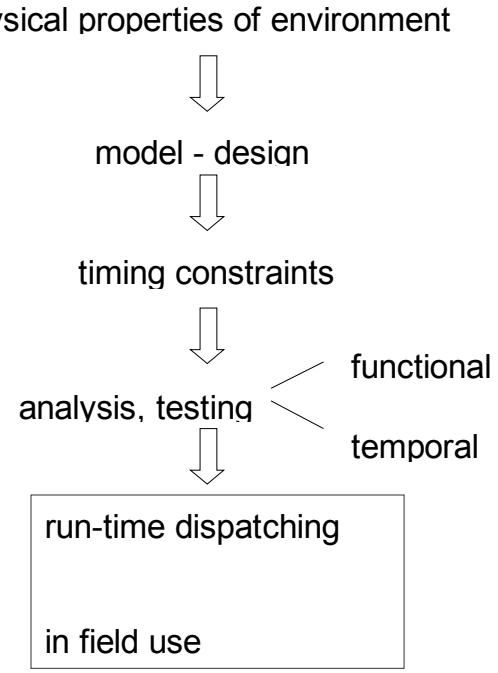
Gerhard Fohler
Mälardalen University, Sweden
gerhard.fohler@mdh.se

Real-time scheduling - making the right decisions to to guarantee time

What do we need?

Guarantees *before* the system is used (*pre run-time, off-line)* that all its activities will meet the specified timing requirements (constraints)

How can we achieve that?



Who is doing the scheduling?

Run-time dispatcher controls which activities are performed at which time. It controls access to the CPU to *tasks*.

The unit performing run-time dispatching is within the *real-time kernel*.

- Keeps track of the system state, e.g., time, resource accesses, book keeping information, e.g., priorities, deadlines.
- Tasks execute until completion or may be interrupted: *non-preemptive* or *preemptive.*
  Non-preemptive dispatching is in general simpler, only one task (and stack etc.) active at a time.
  Run-time dispatching is performed according to a set of rules.

And when?

- System designer selects *scheduling strategy* and *algorithm*
  Constructs a set of rules for the *run-time dispatcher* from specification and timing constraints. These rules range from complete schedules to priorities strategies, etc.
- During *analysis*/*testing,* the designer determines, whether the rules provided will guarantee the temporal behavior, if applied by the run-time dispatcher.
  If no rules can be found or testing gives a negative result, a redesign has to be done.
- Depending on whether these rules determine most scheduling decision before run-time or or leave part of the decisions to the run-time system, the scheduling is called *offline (pre run-time, static)* or *online (run-time, dynamic).*

Pre run-time vs. run-time scheduling

Pre run-time scheduling constructs complete schedules that are feasible before the system is used in-field.
This is a *proof-by-construction* of feasibility.
Run-time dispatching only executes the decision, does not take any by itself.

☺ Very simple for run-time system, e.g., list or table lookup.

☹ Inflexible, can only handle fully specified events and tasks, requires *complete* knowledge.

Run-time scheduling constructs a set of rules for run-time dispatching and a *proof* (*schedulability test) of* feasibility when the rules are kept, before the system is used.
Run-time dispatching can take decisions on its own, as long as rules are kept.

☺ Flexible, can handle only partially known events and tasks.

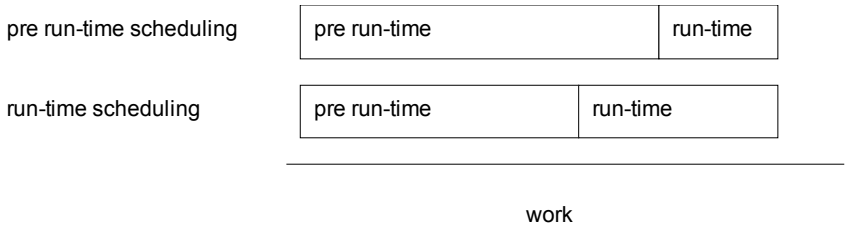
☹ High cost at run-time (book keeping, calculations)
Difficult to predict *exact* behavior at run-time.

Run-time scheduling can provide more flexibility, but

no magic:

What is not exactly known before run-time cannot be guaranteed then, independent of the used scheduling strategy.

Only events for which a task has been specified, i.e., code is available, can be handled.



Recently, algorithms have been presented to *integrate pre run-time* and *run-time scheduling.*

Benefits from pre run-time, but more flexibility.

Rate Monotonic

presented by Liu and Layland in 1973

Assumptions

- Tasks are periodic with deadlines equal to periods. Release time of tasks is the period start.
- Tasks do not suspend themselves
- Tasks have bounded execution time
- Tasks are independent
- Scheduling overhead negligible

Priorities

Tasks priorities are assigned according to their periods;
shorter period means higher priority

Run-time

The ready task with the highest priority is executed.

Schedulability test

If following condition holds, taskset is schedulable

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \qquad n(2^{1/n} \qquad 1)$$

Very simple test, easy to implement.

- we know (thanks to the prove of Liu and Layland), that if the test says “yes”, and the run-time dispatcher executes according to rules, all tasks will meet their deadline

Liu and Layland have been a bit *pessimistic:*There are, tasksets with higher utilization that can be scheduled (sufficient condition only).

Example :

taskset: t1,t2,t3,t4; t = (T,C)

t1 = (3, 1)

t2 = (6, 1)

t3 = (5, 1)

t4 = (10, 2)

- we know (thanks to the prove of Liu and Layland), that if the test says “yes”, and the run-time dispatcher executes according to rules, all tasks will meet their deadline

Liu and Layland have been a bit *pessimistic:*There are, tasksets with higher utilization that can be scheduled (sufficient condition only).

The taskset is feasible, all deadlines are met.

The schedulability test gives

$1/3 + 1/6 + 1/5 + 2/10 \leq 4(2^{(1/4)} - 1)$

$0.9 < 0.75$? ... “no!”

not schedulable.

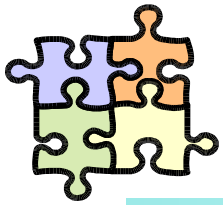
The given schedulability condition is only sufficient, but not necessary.

E.g., when periods are harmonic, i.e., multiples of each other, utilization can be 1.

A schedulability test is

- **sufficient**, if *all* tasksets that *pass* are (definitely) *schedulable*; but there *may* exist tasksets that *fail* the test, which are *schedulable*
- **necessary**, if *all* tasksets that *fail* are (definitely) *not schedulable*; but there *may* exist tasksets that *pass* the test which are *not schedulable*

An example for a necessary condition is that the utilization of the taskset is <= 1: When the utilization is higher, we do not have enough CPU time, and no scheduling algorithm can schedule the task set. When the utilization is <= 1, it doesn't mean necessarily that the taskset is schedulable.



The “0.7 rule”

- “I do not like real-time, because I can only have 70% utilization”
- “I like real-time, because if the utilization is less than 70%, my system is real-time”

- all wrong!
- mixing many concepts specific to one single algorithm (and its wrong even for that one)

Proof of schedulability test

worst case

- all tasks start at same time:
- low priority tasks will be maximally delayed by higher priority tasks
- *critical instance*

proof has to show that

- critical instance is really worst case
- all tasks meet deadline even in worst case
  i.e., response time of lowest priority task smaller than deadline

- many schedulability tests are based on critical instance argument

Exact Analysis

Rate monotonic simple but pessimistic, can we do more precise testing?

Yes, but things get more tricky - exact analysis by Joseph and Pandya.

Based on critical instance analysis as well.

(longest response time of task, when it is released at same time as all higher priority tasks)

What is happening at the critical instance?

- Let T_1 be the highest priority task. Its response time R_1 is

  $R_1 = C_1$ since it cannot be preempted

- What about T_2 ?

  $R_2 = C_2$ + all the time it is interrupted by T_1. Since T_1 has higher priority, it has shorter period. That means it will interrupt T_2 at least one time, probably more often. Assume T_1 has half the period of T_2,

  $R_2 = C_2 + 2 \times C_1$

- In general:

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

- $hp(i)$ is the set of tasks with higher priority than task i.
- $\lceil x \rceil$ is *ceiling* of x, e.g., $\lceil 3.5 \rceil = 4$ $\lceil 2 \rceil = 2$

so far so good, but ...

- we extend the response time of a task by the computation time of "hits" from higher priority tasks
- because the response time of that task became longer, it might be hit even more by shorter period tasks
  Assume two tasks, T_1 = (3,2) and T_2 = (8,2).
  R_1 = 2
  first calculation of R_2 = 2; including "hits"

$$R_2 \quad 2 \quad \frac{2}{3} \quad 2 \quad 2 \quad 2 \quad 4$$

period of $T_1 = 2$, so the new R_2 gets even more hits:

$$R_2 \quad 2 \quad \frac{4}{3} \quad 2 \quad 5$$

$$R_2 \quad 2 \quad \frac{5}{3} \quad 2 \quad 5$$

stable

- we need iterations for each task!

In general:

$$R_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j$$

R^n_i denotes the n^{th} iteration of the response time of task I

hp(i)… tasks with higher priority than i

Not so nice to calculate

Example - Exact Analysis

Let us look at our example, that failed the pure rate monotonic test, although we could schedule it. What does exact analysis say?

- R_1 = 1; easy
- R_3, second highest priority task
  $hp(t_3) = T_1$

$$R_{t3}^1 \quad C_{t3} \quad \frac{1}{3} \quad C_{t1} \quad 1 \quad 1 \quad 2$$

$$R_{t3}^2 \quad C_{t3} \quad \frac{2}{3} \quad C_{t1} \quad 1 \quad 1 \quad 2$$

$$R_{t3}^3 \quad R_{t3}^2$$

R_3 = 2

Is that correct? Check schedule.

Earliest Deadline First

Rate monotonic assumes that deadlines of tasks are equal to their periods; not very realistic.

Assumptions

Same as rate monotonic

Priorities

Tasks with closer deadlines get higher priority.

Run-time

The ready task with the highest priority is executed, i.e., the one with the closest deadline.

Sort priorities such that tasks with closer deadlines get higher priority at run-time as well, *dynamic priorities*.

Schedulability test

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

Sufficient and necessary condition.

EDF is *optimal* if utilization is beyond 1; performs badly under overload.

Optimal means, that if any algorithm can find a schedule, EDF will find as well.

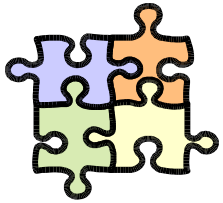
Example EDF

t1 = (3, 1)

t2 = (6, 1)

t3 = (5, 1)

t4 = (10, 3) (where t4 misses deadline with RM)



Fixed vs. dynamic priorities Comments

- EDF has higher run-time overhead (deadline updates) but fewer preemptions

- overload
  - FPS provides better, i.e., less delayed, execution for highest priority tasks under overload
  - EDF might end up without any tasks completing in time

  but:

  - EDF distributes overload more fairly, i.e., all tasks get delayed by same factor

State of analysis methods – both FPS and EDF

dependent tasks

- mutual exclusion
  - e.g., priority ceiling protocol (Mars Pathfinder)
  - blocking time can be bounded
  - included in schedulability analysis

- precedence can be “emulated” via modifying priorities, deadlines

- other constraints difficult to include
  (offline scheduling can easily)

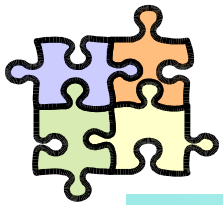
aperiodic tasks

- aperiodic tasks, i.e., no information about actual arrival times
  - both with and without deadlines – acceptance tests

distributed systems

- run-time scheduling algorithms still premature
- tricky synchronization between computers over network at run-time

- offline scheduling can handle distributed systems



Comments on both FPS, EDF

- priorities and deadlines are only to steer scheduler (rules)
  do not have inherent semantic
  - e.g., highest priority task may not be most important one
  - source of confusion for designer

- translating application temporal constraints into scheduler parameters
  process of its own

- schedulability analysis after
- mixing semantic and rules can become a mess
  e.g., “priority” – importance