technische universität dortmund

fakultät für informatik
informatik 12

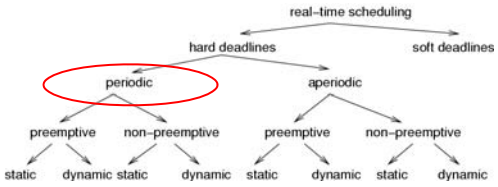
Classical scheduling algorithms for periodic systems

Peter Marwedel
TU Dortmund, Informatik 12
Germany

Graphics: © Alexandra Nolte, Gesine Marwedel, 2003

(2010年 12 月 15 日)
Subset of slides selected for EECS 222C.

Periodic and aperiodic tasks

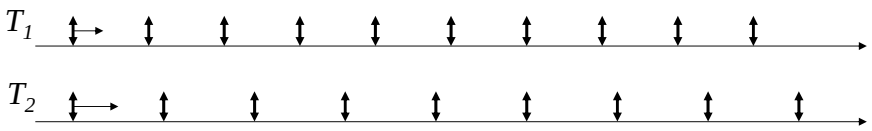


Def.: Tasks which must be executed once every *p* units of time are called **periodic** tasks. *p* is called their period. Each execution of a periodic task is called a **job**.

All other tasks are called **aperiodic**.

Def.: Tasks requesting the processor at unpredictable times are called **sporadic**, if there is a minimum separation between the times at which they request the processor.

Periodic scheduling



Each execution instance of a task is called a **job**.

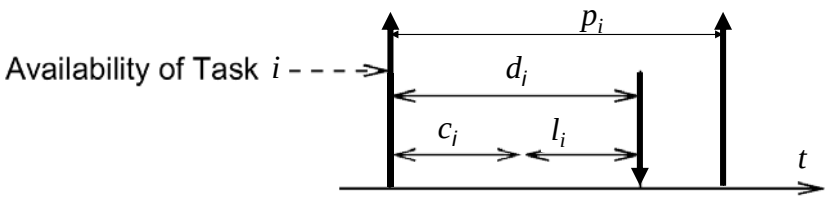
Notion of optimality for aperiodic scheduling does not make sense for periodic scheduling.

For periodic scheduling, the best that we can do is to design an algorithm which will always find a schedule if one exists.
☞ A scheduler is defined to be **optimal** iff it will find a schedule if one exists.

Periodic scheduling
- Scheduling with no precedence constraints -

Let $\{T_i\}$ be a set of tasks. Let:

- p_i be the period of task T_i,
- c_i be the execution time of T_i ,
- d_i be the **deadline interva**l, that is,
  the time between T_i becoming available
  and the time until which T_i has to finish execution.
- l_i be the **laxity** or **slac**k, defined as $l_i = d_i - c_i$
- f_i be the finishing time.



Average utilization, a very important characterization of scheduling problems:

Average utilization:

$$\mu = \sum_{i=1}^{n} \frac{c_i}{p_i}$$

Necessary condition for schedulability (with m=number of processors):

$$\mu \leq m$$

Independent tasks: Rate monotonic (RM) scheduling

Most well-known technique for scheduling independent periodic tasks [Liu, 1973].

Assumptions:

- All tasks that have hard deadlines are periodic.
- All tasks are independent.
- $d_i = p_i$, for all tasks.
- c_i is constant and is known for all tasks.
- The time required for context switching is negligible.
- For a single processor and for n tasks, the following equation holds for the average utilization μ:

$$\mu = \sum_{i=1}^{n} \frac{c_i}{p_i} \leq n(2^{1/n} - 1)$$

Rate monotonic (RM) scheduling
- The policy -

RM policy: **The priority of a task is a monotonically decreasing function of its period.**

At any time, a highest priority task among all those that are ready for execution is allocated.

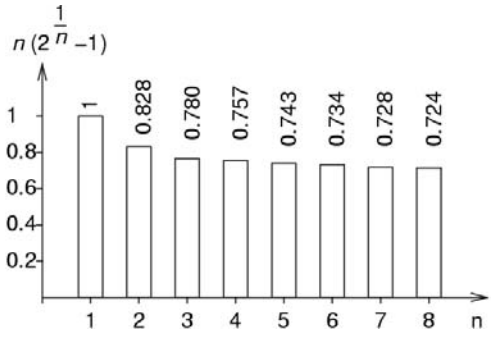
Theorem: If all RM assumptions are met, schedulability is guaranteed.

Maximum utilization for guaranteed schedulability

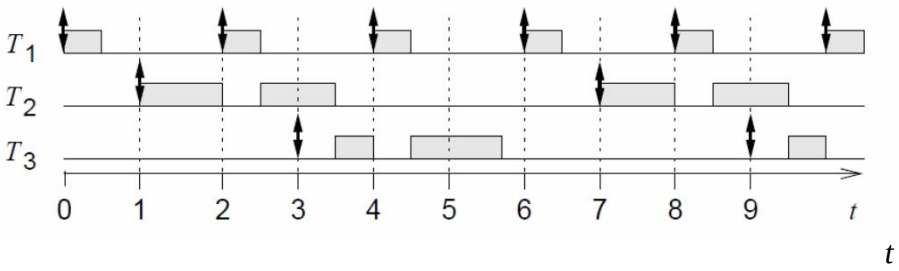
Maximum utilization as a function of the number of tasks:

$$\mu = \sum_{i=1}^{n} \frac{c_i}{p_i} \leq n(2^{1/n} - 1)$$

$$\lim_{n \to \infty} (n(2^{1/n} - 1) = \ln(2)$$



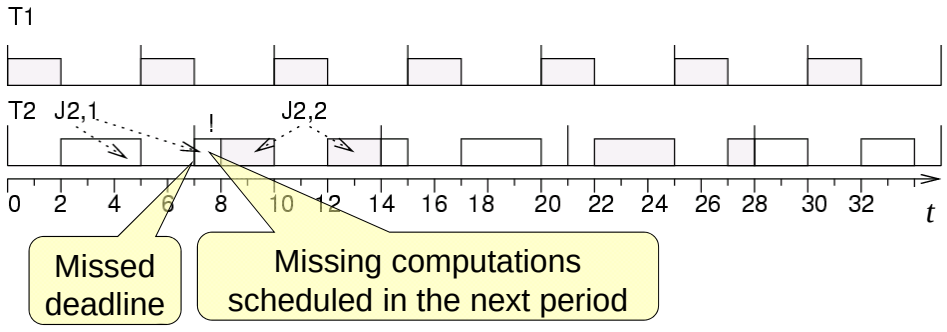
Example of RM-generated schedule



T_1 preempts T_2 and T_3.
T_2 and T_3 do not preempt each other.

Case of failing RM scheduling

Task 1: period 5, execution time 2
Task 2: period 7, execution time 4
μ=2/5+4/7=34/35 ≈ 0.97
$2(2^{1/2}-1) \approx 0.828$



Properties of RM scheduling

- RM scheduling is based on **static** priorities. This allows RM scheduling to be used in an OS with static priorities, such as Windows NT.
- No idle capacity is needed if $\forall i$: $p_{i+1} =_F p_i$:
  i.e. if the **period of each task is a multiple of the period of the next higher priority task**, schedulability is then also guaranteed if $\mu \leq 1$.
- A huge number of variations of RM scheduling exists.
- In the context of RM scheduling, many formal proofs exist.

EDF

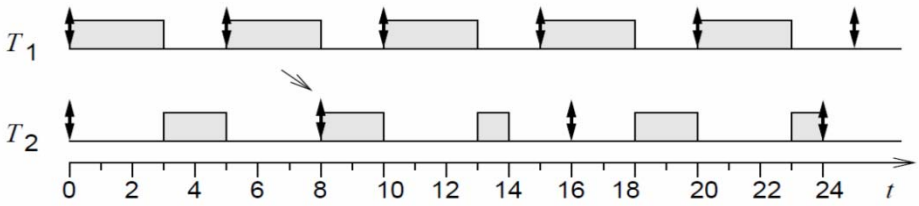
EDF can also be applied to periodic scheduling.

EDF optimal for every **hyper-period**
(= least common multiple of all periods)

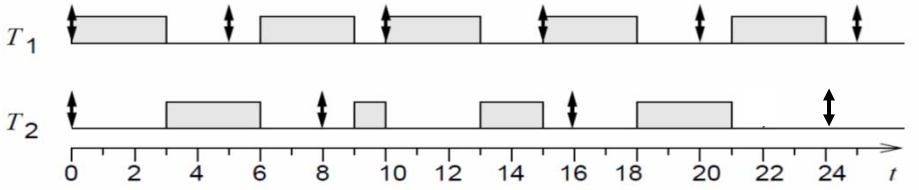
- ☞ Optimal for periodic scheduling
- ☞ EDF must be able to schedule the example in which RMS failed.

Comparison EDF/RMS

RMS:



EDF:



T_2 not preempted, due to its earlier deadline.

EDF: Properties

EDF requires dynamic priorities

☞ EDF cannot be used with an operating system just providing static priorities.

However, a recent paper (by Margull and Slomka) at DATE 2008 demonstrates how an OS with static priorities can be extended with a plug-in providing EDF scheduling (key idea: delay tasks becoming ready if they shouldn't be executed under EDF scheduling.

Comparison RMS/EDF

	RMS	EDF
Priorities	Static	Dynamic
Works with OS with fixed priorities	**Yes**	**No***
Uses full computational power of processor	No, just up till $\mu=n(2^{1/n}-1)$	Yes
Possible to exploit full computational power of processor without provisioning for slack	**No**	**Yes**

* Unless the plug-in by Slomka et al. is added.

Sporadic tasks

If sporadic tasks were connected to interrupts, the execution time of other tasks would become very unpredictable.

☞ Introduction of a sporadic task server, periodically checking for ready sporadic tasks;

☞ Sporadic tasks are essentially turned into periodic tasks.

Dependent tasks

The problem of deciding whether or not a schedule exists for a set of dependent tasks and a given deadline is NP-complete in general [Garey/Johnson].

Strategies:

1. Add resources, so that scheduling becomes easier
2. Split problem into static and dynamic part so that only a minimum of decisions need to be taken at run-time.
3. Use scheduling algorithms from high-level synthesis

Summary

Periodic scheduling

- Rate monotonic scheduling
- EDF
- Dependent and sporadic tasks (briefly)