technische universität dortmund

fakultät für informatik informatik 12

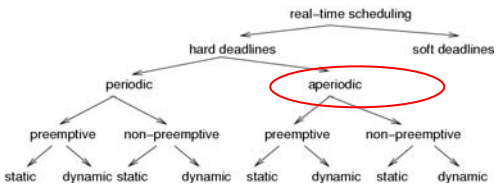
Classical scheduling algorithms for aperiodic systems

Peter Marwedel
TU Dortmund, Informatik 12
Germany

Graphics: © Alexandra Nolte, Gesine Marwedel, 2003

(2010年 12 月 10 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**.

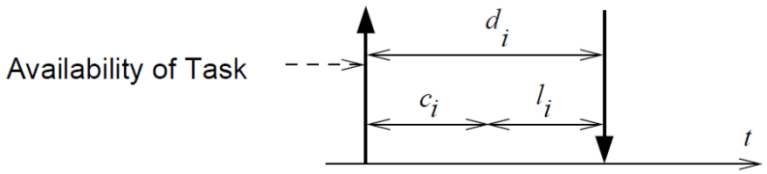
All other tasks are called **aperiodic**.

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.

Aperiodic scheduling; - Scheduling with no precedence constraints -

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

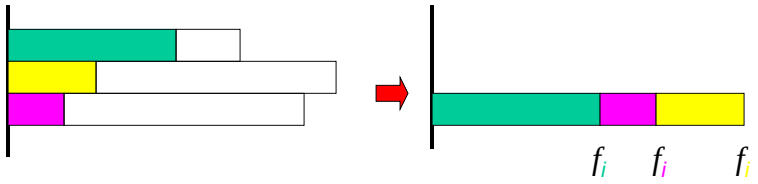
- c_i be the execution time of T_i,
- d_i be the **deadline interval**, that is, the time between T_i becoming available and the time until which T_i has to finish execution.
- l_i be the **laxity** or **slack**, defined as $l_i = d_i - c_i$
- f_i be the finishing time.



Uniprocessor with equal arrival times

Preemption is useless.

Earliest Due Date (EDD): Execute task with earliest due date (deadline) first.



EDD requires all tasks to be sorted by their (absolute) deadlines. Hence, its complexity is *O(n log(n))*.

Optimality of EDD

EDD is optimal, since it follows Jackson's rule:
Given a set of n independent tasks, any algorithm that executes the tasks in order of non-decreasing (absolute) deadlines is optimal with respect to minimizing the maximum lateness.

Proof (See Buttazzo, 2002):

- (omitted in EECS 222C)

Earliest Deadline First (EDF) - Horn's Theorem -

Different arrival times: Preemption potentially reduces lateness.

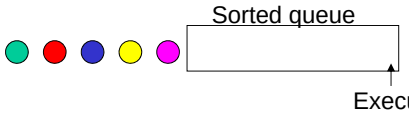
Theorem [Horn74]: Given a set of n independent tasks with arbitrary arrival times, any algorithm that at any instant executes the task with the earliest absolute deadline among all the ready tasks is optimal with respect to minimizing the maximum lateness.

Earliest Deadline First (EDF)
- Algorithm -

Earliest deadline first (EDF) algorithm:

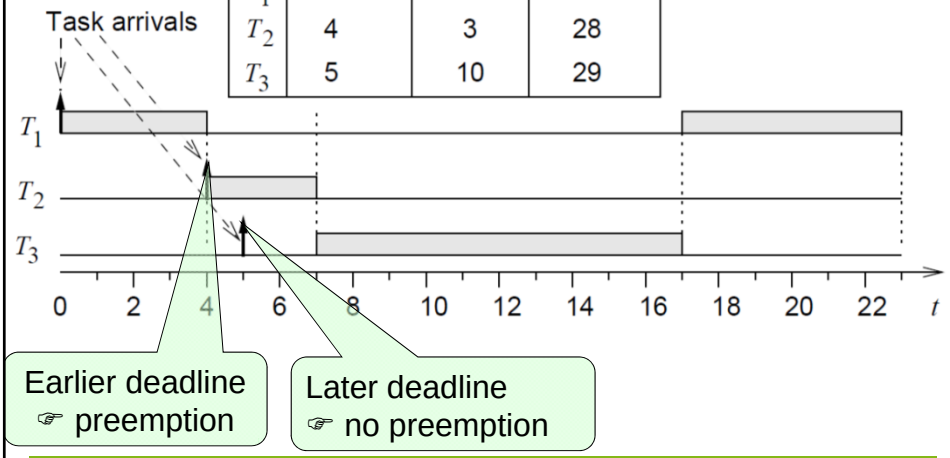
- Each time a new ready task arrives:
- It is inserted into a queue of ready tasks, sorted by their **absolute** deadlines. Task at head of queue is executed.
- If a newly arrived task is inserted at the head of the queue, the currently executing task is preempted.

Straightforward approach with sorted lists (full comparison with existing tasks for each arriving task) requires run-time $O(n^2)$; (less with binary search or bucket arrays).



Earliest Deadline First (EDF)
- Example -

	arrival	duration	deadline
T_1	0	10	33
T_2	4	3	28
T_3	5	10	29



Optimality of EDF

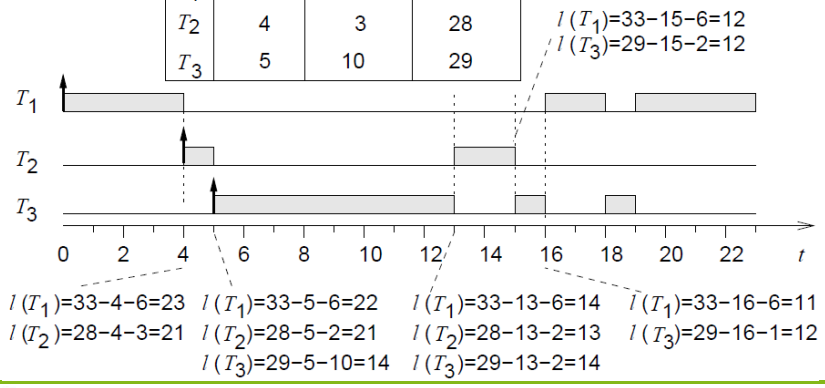
To be shown: EDF minimizes maximum lateness.
Proof (Buttazzo, 2002):

- (omitted in EECS 222C)

Least laxity (LL), Least Slack Time First (LST)

Priorities = decreasing function of the laxity
(lower laxity ☞ higher priority); changing priority; preemptive.

	arrival	duration	deadline
T_1	0	10	33
T_2	4	3	28
T_3	5	10	29



Properties

- LL is also an optimal scheduling for mono-processor systems.
- Not sufficient to call scheduler & re-compute laxity just at task arrival times.
- Overhead for calls of the scheduler.
- Many context switches.
- Dynamic priorities ☞ cannot be used with a fixed prio OS.
- LL scheduling requires the knowledge of the execution time.
- **Detects missed deadlines early.**

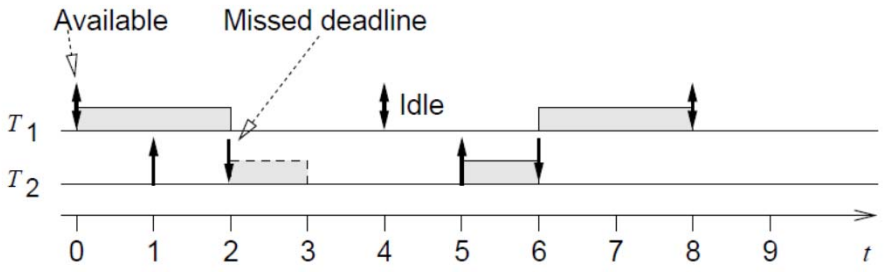
Scheduling without preemption (1)

Lemma: If preemption is not allowed, optimal schedules may have to leave the processor idle at certain times.

Proof: Suppose: optimal schedulers never leave processor idle.

Scheduling without preemption (2)

T_1: periodic, $c_1 = 2$, $p_1 = 4$, $d_1 = 4$
T_2: occasionally available at times $4*n+1$, $c_2 = 1$, $d_2 = 1$
T_1 has to start at $t = 0$
☞ deadline missed, but schedule is possible (start T_2 first)
☞ scheduler is not optimal ☞ contradiction! q.e.d.



Scheduling without preemption

Preemption not allowed: ☞ optimal schedules may leave processor idle to finish tasks with early deadlines arriving late.

☞Knowledge about the future is needed for optimal scheduling algorithms
☞No online algorithm can decide whether or not to keep idle.

EDF is optimal among all scheduling algorithms not keeping the processor idle at certain times.

If arrival times are known a priori, the scheduling problem becomes NP-hard in general. B&B typically used.