UNIVERSITÄT DORTMUND

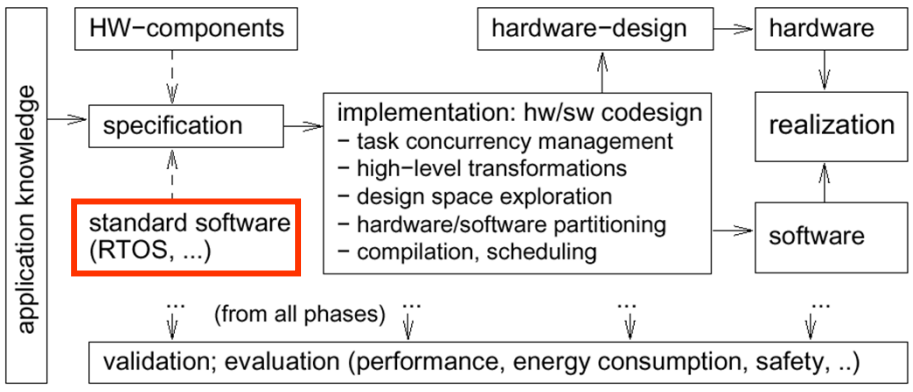
Selected Slides
of Chapter 4, part 1

Embedded Operating Systems, Middleware, and Scheduling

Peter Marwedel
Informatik 12
Univ. Dortmund
Germany

2006/12/05

Simplified design flow for embedded systems



Reuse of standard software components

Knowledge from previous designs to be made available in the form of **intellectual property** (IP, for SW & HW).

- Operating systems
- Middleware
- Real-time data bases
- Standard software (MPEG-x, GSM-kernel, …)

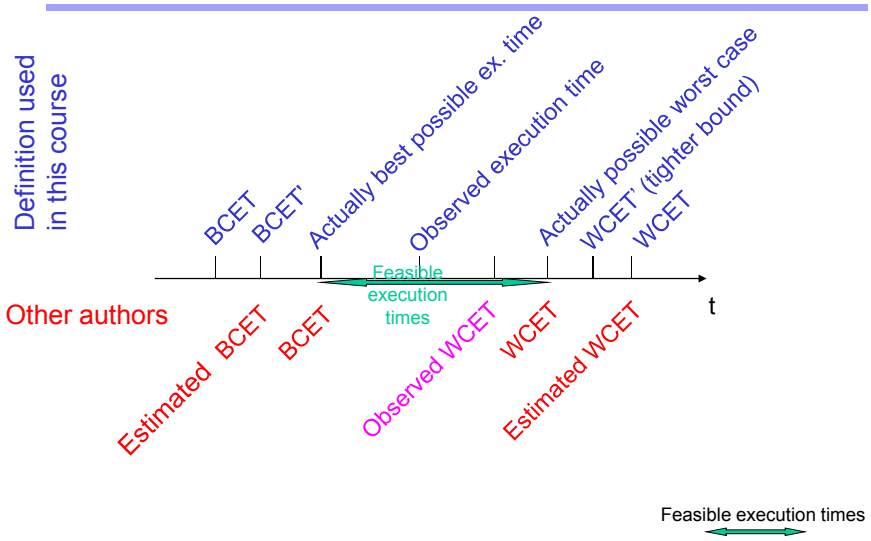
Includes standard approaches for scheduling (requires knowledge about execution times).

Worst/best case execution times (1)

Def.: The **worst case execution time** (WCET) is an **upper bound** on the execution times of tasks.
The term is not ideal, since a program requiring the WCET for its execution does not have to exist (WCET is a **bound**).

Def.: The **best case execution time** (BCET) is a lower **bound** on the execution times of tasks.
The term is not ideal, since a program running at the BCET for its execution does not have to exist (BCET is a **bound**).

Worst/best case execution times (2)



Worst case execution times (2)

Complexity:

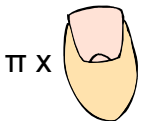
- in the general case: undecidable if a bound exists.
- for restricted programs: simple for „old“ architectures, very complex for new architectures with pipelines, caches, interrupts, virtual memory, etc.

Approaches:

- for hardware: requires detailed timing behavior
- for software: requires availability of machine programs; complex analysis (see, e.g., www.absint.de)

Average execution times

- **Estimated cost and performance values:**
  Difficult to generate sufficiently precise estimates;
  Balance between run-time and precision



- **Accurate cost and performance values:**
  Can be done with normal tools (such as compilers).
  As precise as the input data is.

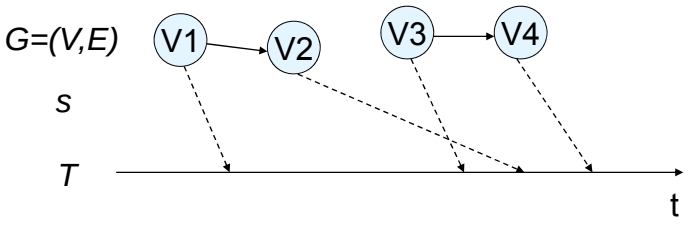
Real-time scheduling (1)

Assume that we are given a task graph *G=(V,E)*.

Def.: A **schedule** *s* of *G* is a mapping

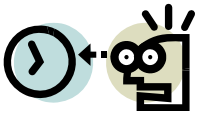
$$V \rightarrow T$$

of a set of tasks *V* to start times from domain *T*.



Real-time scheduling (2)

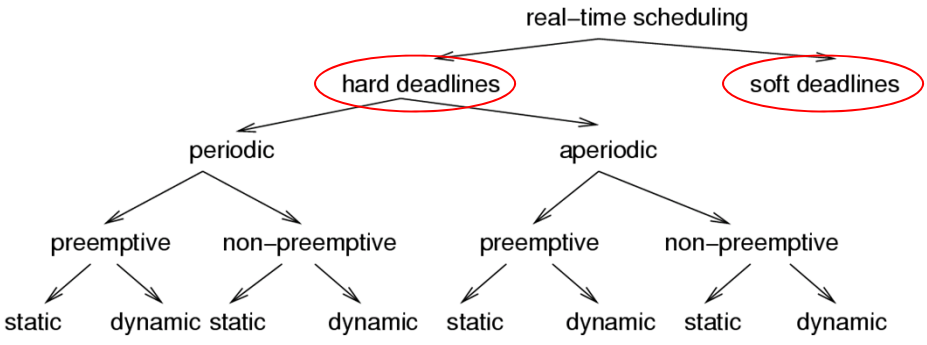
Typically, schedules have to respect a number of constraints, incl. resource constraints, dependency constraints, deadlines.



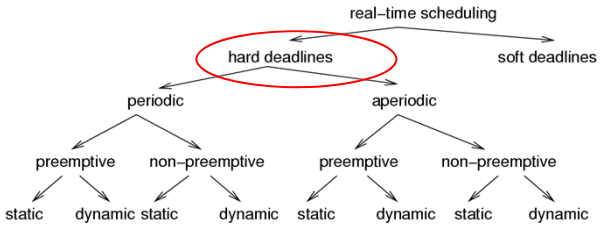
Scheduling = finding such a mapping.

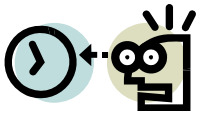
Scheduling to be performed several times during ES design (early rough scheduling as well as late precise scheduling).

Classification of scheduling algorithms



Hard and soft deadlines



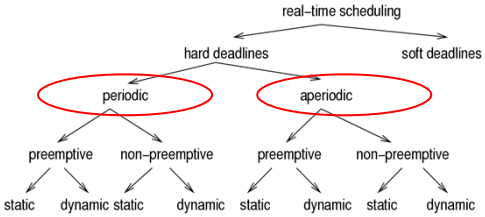


Def.: A time-constraint (deadline) is called **hard** if not meeting that constraint could result in a catastrophe [Kopetz, 1997].

All other time constraints are called **soft**.

We will focus on hard deadlines.

Periodic and aperiodic tasks

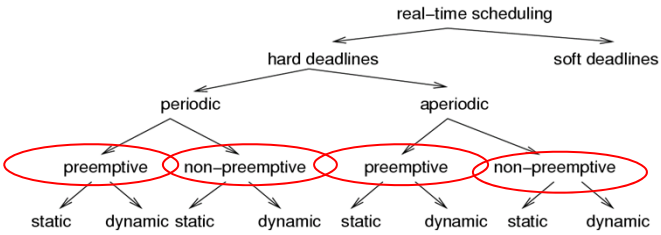


Def.: Tasks which must be executed once every *p* units of time are called **periodic** tasks. *p* is called their period. Each execution of a periodic task is called a **job**.

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.

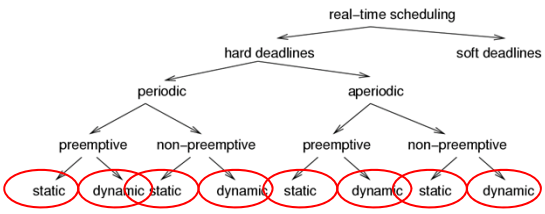
Preemptive and non-preemptive scheduling



- **Non-preemptive schedulers:**
  Tasks are executed until they are done.
  Response time for external events may be quite long.
- **Preemptive schedulers:** To be used if
  - some tasks have long execution times or
  - if the response time for external events to be short.

Dynamic/online scheduling

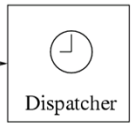
- **Dynamic/online scheduling:**
  Processor allocation decisions (scheduling) at run-time; based on the information about the tasks arrived so far.



Static/offline scheduling

- **Static/offline scheduling:**
  Scheduling taking a priori knowledge about arrival times, execution times, and deadlines into account. Dispatcher allocates processor when interrupted by timer. Timer controlled by a table generated at design time.

Time	Action	WCET
10	start T1	12
17	send M5	
22	stop T1	
38	start T2	20
47	send M3	



Time-triggered systems (1)

In an entirely time-triggered system, the temporal control structure of all tasks is established ***a priori*** *by off-line support-tools. This temporal control structure is encoded in a* ***Task-Descriptor List (TDL)*** *that contains the cyclic schedule for all activities of the node. This schedule considers the required precedence and mutual exclusion relationships among the tasks such that an explicit coordination of the tasks by the operating system at run time is not necessary. ..*

The dispatcher is activated by the synchronized clock tick. It looks at the TDL, and then performs the action that has been planned for this instant [Kopetz].

Time	Action	WCET
10	start T1	12
17	send M5	
22	stop T1	
38	start T2	20
47	send M3	

Time-triggered systems (2)

… pre-run-time scheduling is often the only practical means of providing predictability in a complex system. [Xu, Parnas].

It can be easily checked if timing constraints are met.
The disadvantage is that the response to sporadic events may be poor.

Centralized and distributed scheduling

- **Centralized and distributed scheduling:**
  Multiprocessor scheduling either locally on 1 or on several processors.

- **Mono- and multi-processor scheduling:**
  - Simple scheduling algorithms handle single processors,
  - more complex algorithms handle multiple processors.
    - algorithms for homogeneous multi-processor systems
    - algorithms for heterogeneous multi-processor systems (includes HW accelerators as special case).

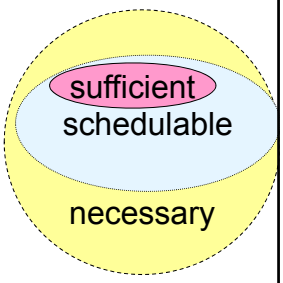
Schedulability

Set of tasks is **schedulable** under a set of constraints, if a schedule exists for that set of tasks & constraints.

Exact tests are NP-hard in many situations.

Sufficient tests: sufficient conditions for schedule checked. (Hopefully) small probability of indicating that no schedule exists even though one exists.

Necessary tests: checking necessary conditions. Used to show no schedule exists. There may be cases in which no schedule exists & we cannot prove it.



Cost functions

Cost function: Different algorithms aim at minimizing different functions.

Def.: Maximum lateness =
$\max_{\text{all tasks}}$ (completion time – deadline)
Is <0 if all tasks complete before deadline.