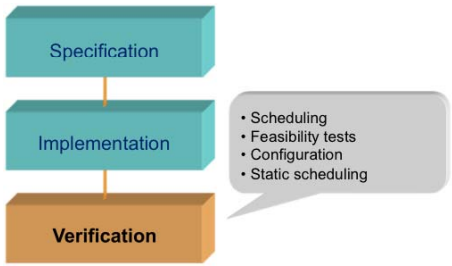
CHALMERS

Real-Time Systems



CHALMERS

Scheduling

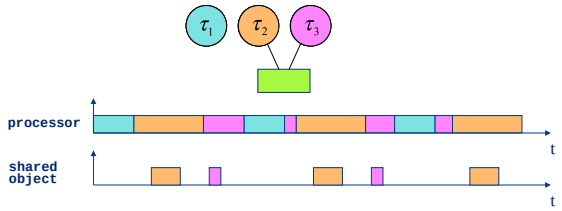
In the general case, the number of tasks is larger than the number of processors available. This raises the following questions:

1. **How** should the processor be shared?
   - Serial execution (cyclic executive)
   - Pseudo-parallel execution
2. **When** should task switches take place?
   - At natural stops (e.g., at *wait* or *delay* operations)
   - At changed system state (e.g., after *signal* operations)
   - At clock or I/O interrupts
3. **Which** task should execute?
   - Scheduling policy

CHALMERS

Scheduling

A schedule is a reservation of spatial (e.g., processor, program objects) and temporal (time) resources for a given set of tasks.



CHALMERS

Scheduling

How is scheduling implemented?

- Static scheduling:
  - Schedule generated "off-line" before the tasks becomes ready, sometimes even before the system is in mission.
  - Schedule consists of a "time table", containing explicit start and completion times for each task instance, that controls the order of execution at run-time.
- Dynamic scheduling:
  - Schedule generated "on-line" as a side effect of tasks being executed, that is, when the system is in mission.
  - Ready tasks are sorted in a queue and receive access to the processor based on priority and/or time quanta ("round-robin").

CHALMERS

Scheduling

How flexible is the schedule?

- Time-triggered system:
  - A task becomes ready according to a known time table.
  - The system becomes very deterministic, but inflexible.
- Event-triggered system:
  - A task becomes ready as a result of an external or internal event in the system.
  - Requires run-time support for dynamic scheduling.
  - The system becomes very flexible, but indeterministic.

CHALMERS

Scheduling

When are scheduling decisions taken?

- Non-preemptive scheduling:
  - New scheduling decision is taken when no task executes.
  - Mutual exclusion is automatically guaranteed.
  - Existing methods for WCET analysis works well.
- Preemptive scheduling:
  - New scheduling decision can be taken as soon as the system state changes, that is, even during an ongoing execution.
  - Mutual exclusion may have to be guaranteed with semaphores (or similar primitives).
  - WCET analysis becomes more complicated, because the state in caches and pipelines will change at a task switch.

CHALMERS

Scheduling

A scheduling algorithm is used for generating a schedule for a given set of tasks for a particular type of run-time system.

- The scheduling algorithm is implemented by a scheduler in the real-time kernel that decides in what order the tasks should be executed.
- Observe that the scheduler decides which task should be executed next, whereas the dispatcher is responsible for starting the selected task.

CHALMERS

Scheduling

How complicated is the scheduler?

- For static scheduling:
  - the implementation of the scheduler is simple because the next task is chosen with a table look-up
  - however, the table must be generated before the system is in mission; this is done by a more advanced algorithm
- For dynamic scheduling:
  - the implementation of the scheduler is more sophisticated because it consists of a decision algorithm that must be activated regularly (at each system event)

CHALMERS

Scheduling

A schedule is said to be feasible if it fulfills all application constraints for a given set of tasks.

A set of tasks is said to be schedulable if there exists at least one scheduling algorithm that can generate a feasible schedule.

CHALMERS

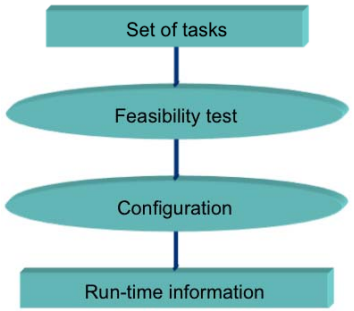
Scheduling

A scheduling problem is said to be NP-complete if it (most probably) can only be solved with an exponential time complexity in the general case.

A scheduling algorithm is said to be optimal with respect to schedulability if it can always find a feasible schedule whenever any other scheduling algorithm can do so.

CHALMERS

Scheduling



CHALMERS

Feasibility tests

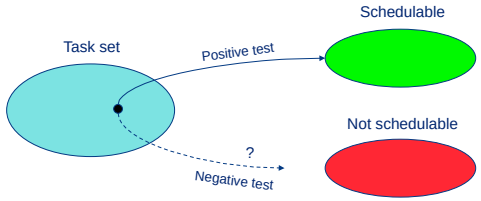
A feasibility test is used for deciding whether a set of tasks is feasible or not for a given scheduler.

- For some schedulers the test can be done in linear time.
  - These schedulers are typically special cases with very simplified assumptions as regards the task properties.
- For most schedulers there exists (so far) no test that can be done in polynomially bounded time.
  - All possible schedules must be considered.
  - These feasibility tests are NP-complete problems.

CHALMERS

Feasibility tests

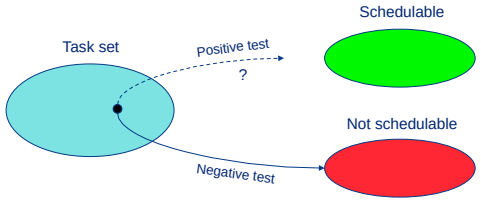
- A feasibility test is sufficient if it with a positive answer shows that a set of tasks is definitely schedulable.
  - A negative answer says nothing! A set of tasks can still be schedulable despite a negative answer.



CHALMERS

Feasibility tests

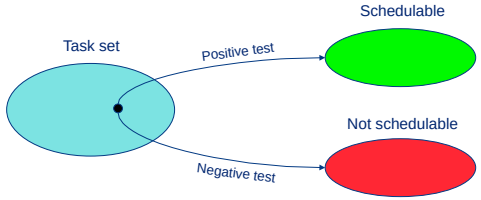
- A feasibility test is necessary if it with a negative answer shows that a set of tasks is definitely not schedulable.
  - A positive answer says nothing! A set of tasks can still be impossible to schedule despite a positive answer.



CHALMERS

Feasibility tests

- An exact feasibility test is both sufficient and necessary. If the answer is positive the task set is definitely schedulable, and if the answer is negative the task set is definitely not schedulable.



CHALMERS

Feasibility tests

How is a feasibility test done?

- For static scheduling:
  - the schedule is verified at the same time as it is generated
- For dynamic scheduling:
  - Processor utilization analysis
    (can be done in linear time)
  - Response time analysis
    (general NP-complete problem)
  - Processor demand analysis
    (general NP-complete problem)

CHALMERS

Configuration

A configuration consists of static information regarding system tasks used by the scheduler at run time. Besides information about the tasks' timing properties and WCET, the configuration contains:

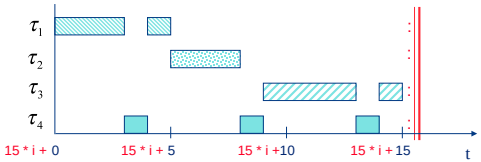
- For static scheduling:
  - Start and finish times for each task.
- For dynamic scheduling:
  - For static priorities, the tasks' (base-) priorities are used together with the ceiling priorities of any shared objects.
  - For dynamic priorities, the configuration does not contain any further information (because all necessary information is already available via the tasks' timing properties and WCET).

CHALMERS

Static scheduling

General properties:

- Time-triggered system (cyclic executive)



CHALMERS

Static scheduling

General properties:

- Off-line schedule generation
  - Explicit start and finish times for each task is derived
  - Feasibility test can be done as soon as these times are available
  - Configuration phase encompasses generating a time table indicating which task is started at which time instant
- Mutual exclusion is handled explicitly
  - The schedule must be generated in such a way that a task switch is not made within a critical region
  - Support for mutual exclusion is not needed in real-time kernel
- Precedence constraints are handled explicitly
  - The schedule must be generated in such a way that enforced task execution orders are respected

CHALMERS

Static scheduling

Advantages:

- Simplifies the communication between tasks
  - The time instant when data becomes available is known
  - Task execution can easily be adapted to any existing time triggered (TDMA) network protocol.
- Minimal overhead at task switches
  - Only requires a time table lookup
- Task execution becomes very deterministic
  - Simplifies feasibility tests (compare finish time to constraint)
  - Simplifies software debugging (increased observability)
  - Simplifies implementation of fault tolerance (natural points in time for self control)

CHALMERS

Static scheduling

Disadvantages:

- Low flexibility (a.k.a. the "Skalman" factor)
  - Schedule cannot adapt itself to changes in the task set or in the system environment
- External events are not handled efficiently
  - I/O units are handled by "polling"
- Only efficient for periodic tasks
  - Sporadic events with short deadline must either be handled by a task with short period (= resource waste) or by a task with longer period (= long response time)
- Inefficient for tasks with "bad" periods
  - Tasks with mutually inappropriate periods give rise to large time tables, which consumes memory in the real-time kernel

CHALMERS

Static scheduling

How is the schedule generated?

- Simulation of dynamic scheduling:
  - Simulate a run-time system in a real-time kernel and then "execute" the system tasks on that simulator, e.g., according to earliest-deadline-first scheduling.
- Exhaustive search:
  - Use an algorithm that searches for a feasible static schedule by considering all possible execution orders for the system tasks.
  - To maintain a low average time complexity of the search, intelligent heuristic search algorithms are used, for example, "Branch-and-Bound" or "Simulated Annealing" (more about this in the advanced course EDA421).

CHALMERS

Static scheduling

How is the size of the time table restricted?

- Only cyclic schedules are considered:
  - The schedule is repeated with a cycle time that is equal to the LCM ("least common multiple") of the task periods.
  - Tasks that are not periodic, or that have a very long period can be handled by a periodic server task (more about this in the advanced course EDA421) .
- Suitable task periods are chosen:
  - To obtain reasonably long cycle times, the task periods should be adjusted to be multiples of each other.
  - Example:
    periods 7, 13, 23 ms ⇒ cycle time 2093 ms, but
    periods 5, 10, 20 ms ⇒ cycle time 20 ms

CHALMERS

Static scheduling

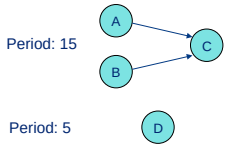
How is the scheduler implemented?

- Create a circular queue that corresponds to the time table
  - Each element in the queue contains start and finish times for a certain task (or task segment in case of preemptive scheduling)
  - The elements in the queue are sorted by the start time
- Use clock interrupts
  - When a task starts executing, a real-time clock is programmed to generate an interrupt at the task's expected finish time.
  - When the interrupt occurs, the next task (i.e., the one whose start time is closest in time) in the circular queue is fetched and the system waits until that task's given start time is due.

CHALMERS

Example: static scheduling

Problem: Assume a system with tasks and precedence constraints according to the figure below. Timing constraints for the tasks are given in the table. Generate a static schedule for these tasks by simulating preemptive earliest-deadline-first scheduling.



Task	C_i	O_i	D_i
A	4	0	7
B	3	0	12
C	5	0	15
D	1	3	1

We solve this on the blackboard!