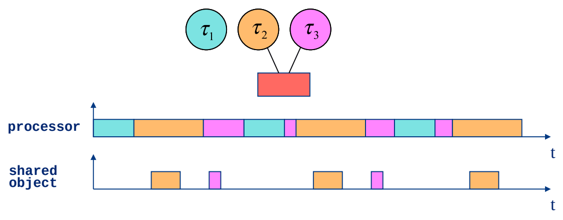
CHALMERS

Scheduling

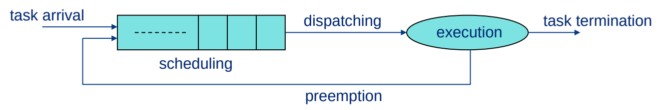
A schedule is a reservation of spatial (processor, RAM) and temporal (time) resources for a given task set.



CHALMERS

Scheduling

- A scheduling algorithm generates a schedule for a given set of tasks and a certain type of run-time system.
- The scheduling algorithm is implemented by a scheduler that decides in which order the tasks should be executed.
- Observe that the scheduler selects which task should be executed next, while the dispatcher starts the execution of the selected task.



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.

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 constraints

Examples of scheduling constraints:

- Non-preemptive scheduling:
  - Once started, a task cannot be preempted by another task
- Greedy scheduling:
  - Once started, a task cannot be preempted by a lower-priority task
- No processor sharing:
  - A processor can only execute one task at a time
- No dynamic task parallelism:
  - A task can only execute on one processor at a time
- No task migration:
  - A task can only execute on one given processor, or cannot change processor during its execution

CHALMERS

Scheduling constraints

Non-preemptive scheduling:

- Advantages:
  - Mutual exclusion is automatically guaranteed
  - Existing methods for WCET analysis works well
- Disadvantages:
  - Negative effect on schedulability
    - Scheduling decision takes effect after a task has executed
    - Once a task starts executing, all other tasks on the same processor will be blocked until execution is complete

CHALMERS

Scheduling constraints

Preemptive scheduling:

- Advantages:
  - Schedulability is not negatively affected
    - Scheduling decisions can take effect as soon as the system state changes (even in the middle of task execution)
    - The capacities of task priorities can be used in full
- Disadvantages:
  - Mutual exclusion has to be guaranteed by e.g. semaphores (or similar constructs)
  - WCET analysis is more complicated since cache and pipeline contents will be affected by a task switch
  - Program security may be compromised (through so-called *covert channels*) if full preemption is allowed

CHALMERS

Scheduling constraints

Greedy scheduling:

- Example: ”traditional” static-priority scheduling (RM, DM)
  - Once a task starts executing, lower-priority tasks cannot grab the processor until execution is complete
- Advantages:
  - Scheduler relatively simple to implement
  - Supported by most real-time operating systems and kernels
- Disadvantages:
  - Schedulability is negatively affected:
    - Lower-priority tasks can starve and hence miss their deadlines

CHALMERS

Scheduling constraints

Fair scheduling:

- Example: p-fair scheduling (Baruah et al. 1995)
  - Although a task has started executing, lower-priority tasks receive a guaranteed time quantum per time unit for execution
  - All tasks hence make some kind of progress per time unit
- Advantages:
  - Schedulability maximized when task switch cost is negligible
- Disadvantages:
  - Scheduler is relatively complicated to implement
  - Poor schedulability when task switch cost is non-negligible
    - Fairness implies significantly more task switches than greediness

CHALMERS

Scheduling algorithm

When are schedules generated?

- 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 at run-time based on priorities and/or time quanta.

CHALMERS

Scheduling algorithm

How much an oracle is the scheduling algorithm?

- Myopic scheduler:
  - Scheduling algorithm only knows about currently ready tasks.
  - Scheduling decisions are only taken whenever a new task instance arrives or a running task instance terminates.
- Clairvoyant scheduler:
  - Scheduling algorithm ”knows the future”; that is, it knows in advance the arrival times of the tasks.
  - On-line clairvoyant scheduling is difficult to realize in practice.

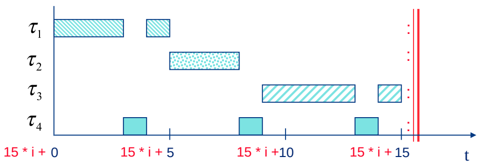
”Predictions are always hard to make. In particular about the future.” (Yogi Berra)

CHALMERS

Static scheduling

General properties:

- Off-line schedule generation:
  - Explicit start and finishing times for each task is derived
  - Cyclic schedule with a meta period equal to the least common multiple (LCM) of the task periods



CHALMERS

Static scheduling

General properties:

- Automatic techniques for schedule generation
  - Simulate a run-time system with dynamic scheduling and record the executions (start and finish times), or
  - Search for a feasible schedule using an intelligent heuristic, such as branch-and-bound (A*) or simulated annealing
- Schedulability test obtained ”for free”
  - Generated schedule can easily be checked for feasibility
- Mutual exclusion and precedence is handled explicitly
  - Heuristic algorithm can be constrained to never perform a task switch in a critical region, and to obey execution order requirements

CHALMERS

Static scheduling

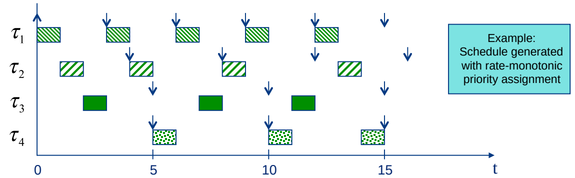
Advantages:
- Predictable execution
  - Monitoring, debugging and schedulability analysis are simplified
- Effective inter-task communication
  - Time for data availability is well known
  - Well suited for interfacing to TDMA networks

Disadvantages:
- Low flexibility (a.k.a. the "Skalman" factor)
  - Schedule cannot adapt itself to changes in the system
- Inefficient for tasks with "bad" periods
  - Tasks with mutually inappropriate periods gives rise to large time tables, which consumes memory

CHALMERS

Dynamic scheduling

General properties:
- On-line schedule generation
  - Schedule determined by on-line behavior controlled by e.g. task priorities or time quanta
  - Schedulability for hard-real-time systems must be tested off-line by making predictions on the on-line behavior



CHALMERS

Dynamic scheduling

General properties:
- Mutual exclusion and precedence must be handled on-line
  - Support for run-time synchronization or task offsets needed
- Large variety of static and dynamic priority schemes
  - Rate-monotonic scheduling [static priority]
  - Deadline-monotonic scheduling [static priority]
  - Weight-monotonic scheduling [static priority]
  - Slack-monotonic scheduling [static priority]
  - Earliest-deadline-first scheduling [dynamic priority]
  - Least-laxity-first scheduling [dynamic priority]

CHALMERS

Dynamic scheduling

Advantages:
- High flexibility
  - Schedule can easily adapt to changes in the system
- Effective for different types of tasks
  - Sporadic tasks easily supported (via suitable priority assignment)
  - Implementation is not affected by task characteristics

Disadvantages:
- Less predictable execution
  - Temporary variations (jitter) in periodicity can occur
- Complicated inter-task communication
  - Task must synchronize to exchange data
  - Difficult to adapt to TDMA networks (but simple for e.g. CAN)

CHALMERS

Dynamic scheduling

Rate-monotonic scheduling (RM):

- Uses static priorities
  - Priority is determined by task frequency (rate)
  - Tasks with higher rates (i.e., shorter periods) are assigned higher priorities
- Theoretically well-established (for the uniprocessor)
  - Sufficient schedulability test can be performed in linear time (under certain simplifying assumptions)
  - Exact schedulability test is an NP-complete problem
  - RM is optimal among all scheduling algorithms that uses static priorities under the assumption that $D_i = T_i$ for all tasks (shown by C. L. Liu & J. W. Layland in 1973)

CHALMERS

Dynamic scheduling

Deadline-monotonic scheduling (DM):

- Uses static priorities
  - Priority is determined by task deadline
  - Tasks with shorter (relative) deadlines are assigned higher priorities
  - Note: RM is a special case of DM, with $D_i = T_i$
- Theoretically well-established (for the uniprocessor)
  - Exact schedulability test is an NP-complete problem
  - DM is optimal among all scheduling algorithms that uses static priorities under the assumption that $D_i \leq T_i$ for all tasks (shown by J. Y.-T. Leung & J. Whitehead in 1982)

CHALMERS

Dynamic scheduling

Earliest-deadline-first scheduling (EDF):

- Uses dynamic priorities
  - Priority is determined by how critical the process is at a given time instant
  - The task whose absolute deadline is closest in time receives the highest priority
- Theoretically well-established (for the uniprocessor)
  - Exact schedulability test can be performed in linear time (under certain simplifying assumptions)
  - EDF is optimal among all scheduling algorithms that uses dynamic priorities under the assumption that $D_i = T_i$ for all tasks (shown by C. L. Liu & J. W. Layland in 1973)

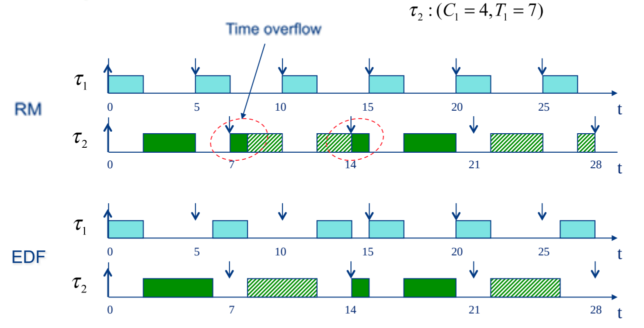
CHALMERS

Dynamic scheduling

Example: RM versus EDF

$\tau_1 : (C_1 = 2, T_1 = 5)$

$\tau_2 : (C_2 = 4, T_1 = 7)$



CHALMERS

Handling shared resources

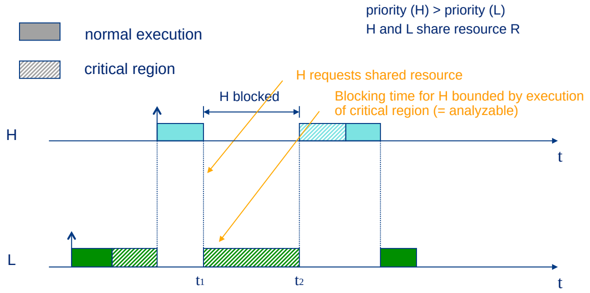
What if tasks are no longer independent?

- Common cause:
  - Multiple tasks access a shared software/hardware object for which mutual exclusion is enforced.
- Solutions:
  - On-line resource access protocols where conflicts are resolved at run-time using dynamic adjustments of task priorities. Examples: PIP, PCP, ICPP, SRP
  - Off-line resource scheduling which produces non-overlapping task execution windows to avoid conflicts at run-time. Examples: Xu & Parnas' algorithm; myopic algorithm

CHALMERS

Handling shared resources

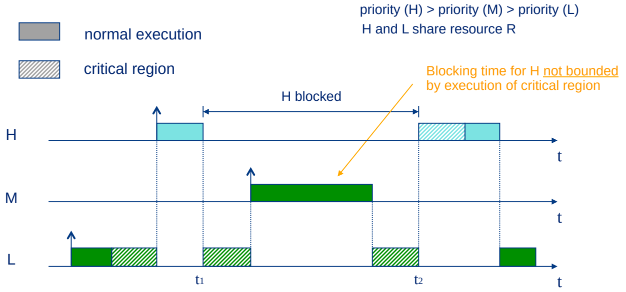
Blocking problem (preemptive scheduling):



CHALMERS

Handling shared resources

Priority inversion phenomenon:



CHALMERS

Handling shared resources

Avoiding priority inversion:

- Non-preemptive critical sections:
  - Creates unnecessary blocking
  - Only recommended for short critical sections
- Access-control protocols for critical sections:
  - Priority Inheritance Protocol (PIP) [static priority]
  - Priority Ceiling Protocol (PCP) [static priority]
  - Immediate Ceiling Priority Protocol (ICPP) [static priority]
  - Stack Resource Policy (SRP) [static and dynamic priority]
  - PIP/PCP for dynamic priorities (EDF)
  - Distributed PCP

CHALMERS

Handling shared resources

Priority Inheritance Protocol: (Sha, Rajkumar & Lehoczky, 1990)

- Basic idea: When a task τ_i blocks one or more higher-priority tasks, it temporarily assumes (inherits) the highest priority of the blocked tasks.
- Advantage:
  - Prevents medium-priority tasks from preempting τ_i and prolonging the blocking duration experienced by higher-priority tasks.
- Disadvantage:
  - Deadlock: priority inheritance can cause deadlock
  - Chained blocking: the highest-priority task may be blocked once by every other task executing on the same processor.

CHALMERS

Handling shared resources

Priority Ceiling Protocol: (Sha, Rajkumar & Lehoczky, 1990)

- Basic idea: Each resource is assigned a priority ceiling equal to the priority of the highest-priority task that can lock it. Then, a task τ_i is allowed to enter a critical section only if its priority is higher than all priority ceilings of the resources currently locked by tasks other than τ_i.
  When the task τ_i blocks one or more higher-priority tasks, it temporarily inherits the highest priority of the blocked tasks.
- Advantage:
  - No deadlock: priority ceilings prevent deadlocks
  - No chained blocking: a task can be blocked at most the duration of one critical section.

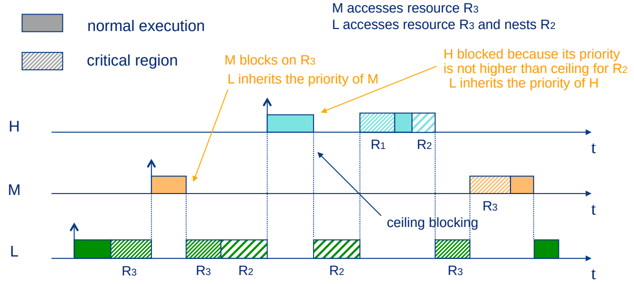
CHALMERS

Handling shared resources

Priority Ceiling Protocol:

priority (H) > priority (M) > priority (L)

H sequentially accesses resources R_1 and R_2
M accesses resource R_3
L accesses resource R_3 and nests R_2



CHALMERS

Handling shared resources

Distributed PCP: (Rajkumar, Sha & Lehoczky, 1988)

- All critical sections associated with the same global resource are bound to a specified synchronization processor.
- A task "migrates" to the synchronization processor to execute the critical section (using remote-procedure calls)
  - deadlock-free algorithm
  - large overhead for message-passing protocol
- All critical sections associated with the same global resource are executed at a priority equal to the semaphore's priority ceiling
  - short blocking times

CHALMERS

Handling shared resources

Alternative approach:

Lock-free and wait-free object sharing

If several tasks attempt to access a lock-free (wait-free) object concurrently, and if some proper subset of these tasks stop taking steps, then one (each) of the remaining tasks completes its access in a finite number of its own steps.

CHALMERS

Handling shared resources

Lock-Free Object Sharing: (Anderson et al., 1996)

- Basic idea: The lock-free object sharing scheme is implemented using "retry loops". Object accesses are implemented using compare-and-swap instructions typically found in modern RISC processors.
- Advantage:
  - Resource accesses are non-blocking
  - Deadlock-free
  - Avoids priority inversion
  - Requires no kernel-level support
- Disadvantage:
  - Potentially unbounded retry loops

CHALMERS

Handling shared resources

Wait-Free Object Sharing: (Anderson et al., 1997)

- Basic idea: The wait-free object sharing scheme is implemented using a "helping" strategy where one task "helps" one or more other tasks to complete an operation. Before beginning an operation, a task must announce its intentions in an "announce variable". While attempting to perform its own operations, a task must also help any previously-announced operation (on its processor) to complete execution.
- Advantage:
  - Non-blocking, deadlock-free, and priority-inversion-free
  - Requires no kernel-level support
  - Precludes waiting dependencies among tasks

CHALMERS

Handling shared resources

Non-existence of optimal on-line shared-resource scheduler: (Mok, 1983)

When there are mutual exclusion constraints in a system, it is impossible to find an optimal on-line scheduling algorithm (unless it is clairvoyant).

Complexity of shared-resource feasibility test: (Mok, 1983)

The problem of deciding feasibility for a set of periodic tasks which use semaphores to enforce mutual exclusion is NP-hard.