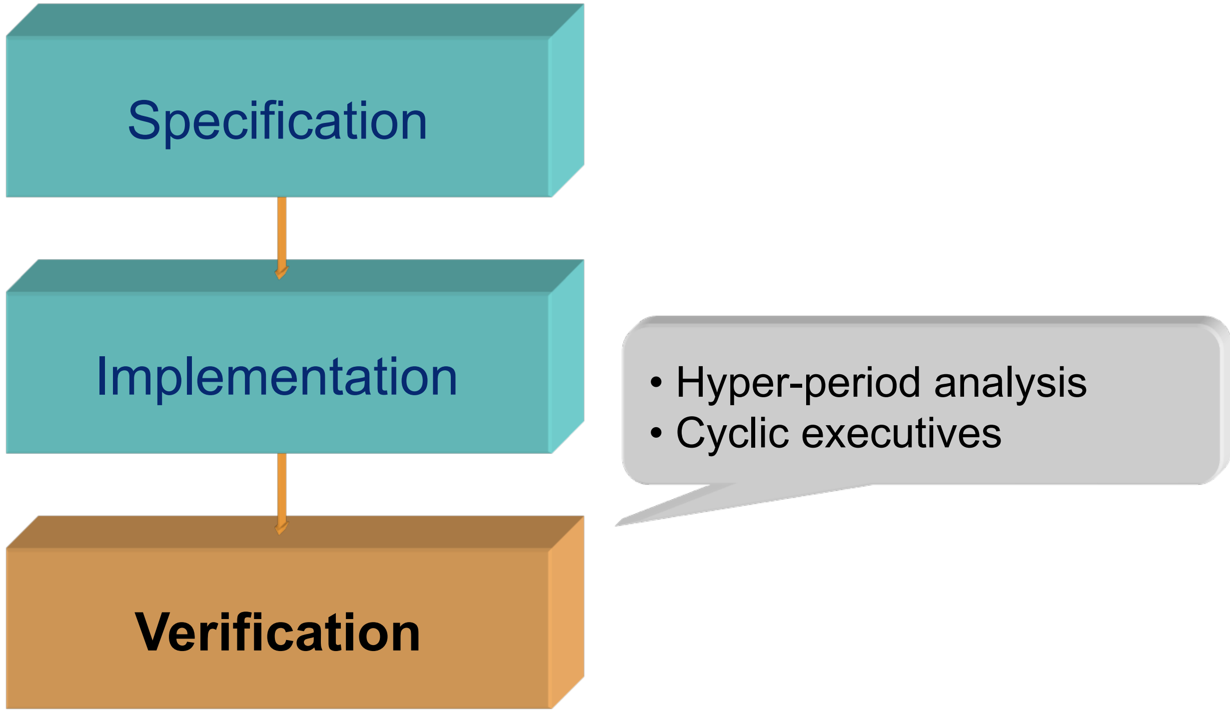
Real-Time Systems

Lecture #10

Professor Jan Jonsson

Department of Computer Science and Engineering
Chalmers University of Technology

Real-Time Systems



Feasibility tests

What types of feasibility tests exist?

- Hyper period analysis (for any type of scheduler)
  - In an existing schedule no task execution may miss its deadline
- Processor utilization analysis (static/dynamic priority scheduling)
  - The fraction of processor time that is used for executing the task set must not exceed a given bound
- Response time analysis (static priority scheduling)
  - The worst-case response time for each task must not exceed the deadline of the task
- Processor demand analysis (dynamic priority scheduling)
  - The accumulated computation demand for the task set under a given time interval must not exceed the length of the interval

Hyper period analysis

Motivation:

- When it is not obvious which feasibility analysis should be used for a given task set and a given scheduler it is always possible to generate a schedule by simulating the execution of the tasks, and then check feasibility for individual tasks.
- The schedule interval that is sufficient to investigate is related to the hyper period of the task set, that is, the least common multiple (LCM) of the task periods.

NOTE: Unless the periods of all tasks are harmonically related (multiples of each other) hyper-period analysis will in general have an exponential time complexity.

Hyper period analysis

Schedule interval to investigate:

- For synchronous task sets: $\forall i, j : O_i = O_j$

  It is sufficient to investigate the interval $[0, P]$ where P is the hyper period of the task set.

- For asynchronous task sets: $\exists i, j : i \neq j, O_i \neq O_j$

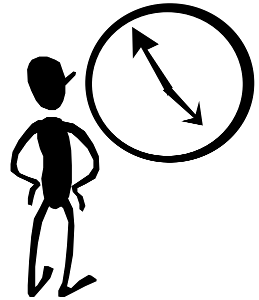
  It is sufficient to investigate the interval $[0, P]$ if no task instance that arrives within the interval executes beyond time P.

  In all other cases it is necessary to investigate more than one hyper period.

Cyclic executives

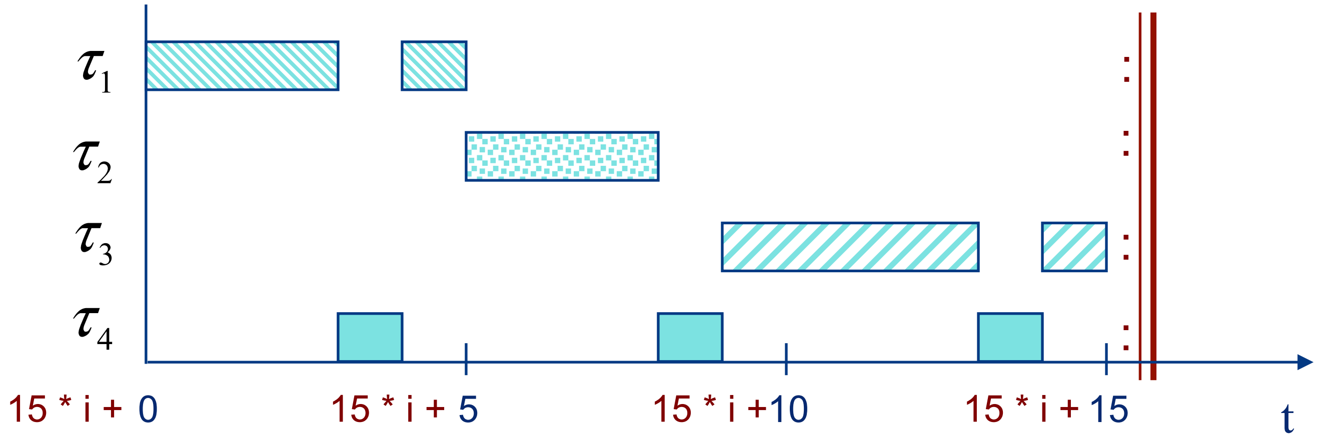
Because of its deterministic properties the cyclic executive is often the choice of scheduler in safety-critical real-time systems, such as automotive and aircraft applications.

Cyclic executives



General properties:

- Table-based schedule
- Feasibility test performed when generating table
- Schedule repeats itself (= “cyclic executive”)



Cyclic executives

General properties:

- Off-line schedule generation
  - Explicit start and finish times for each task are derived off-line, and chosen so that at most one task at a time requests access to the processor during run time.
- 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 (= no need for mutual exclusion support at run-time, e.g. mutex objects)
- Precedence constraints are handled explicitly
  - The schedule must be generated in such a way that specified task execution orderings are respected (= no need for task synchronization at run-time, e.g. semaphores)

Cyclic executives

Advantages:

- Communication between tasks is facilitated
  - The time instant when data becomes available is known
  - Task execution can easily be adapted to any existing time-slot network protocol (e.g., TTCAN, FlexRay).
- Low overhead for scheduling decisions
  - Everything is pre-planned, time table guides the run-time system
  - Feasibility test is done off-line during time table generation
- Task execution becomes very deterministic
  - Simplifies feasibility tests (compare finish time against deadline)
  - Simplifies software debugging (increased observability)
  - Simplifies fault tolerance (natural points in time for self control)

Cyclic executives

Disadvantages:

- Low flexibility (a.k.a. the ”Skalman” factor)
  - The run-time system cannot adapt its schedule to changes in the task set or in the system environment
- External events are not handled efficiently
  - Data from I/O-based events (interrupts) may not be consumed directly by a periodic task due to the pre-planned schedule, which could lead to long response times.
  - An external event with a short deadline must be handled by a task with short period, which may lead to resource waste
- Not so efficient for tasks with ”bad” periods
  - Tasks with mutually inappropriate periods give rise to large time tables, which may require more program code and/or data

Cyclic executives

How is the schedule generated?

- Simulation of pseudo-parallel execution:
  - Simulate a run-time system with a (myopic) priority-based scheduler and then ”execute” the tasks on that simulator.
  - Example: find a schedule by simulating a run-time system with the (dynamic priority) earliest-deadline-first scheduler.
- Exhaustive search:
  - Use an algorithm that searches for a feasible schedule by considering all possible execution orders for the tasks.
  - Example: use the well-known A* search algorithm to find a feasible (optimal or non-optimal) schedule.

If the simulated scheduler or search algorithm is optimal for the given system model a feasible schedule will be found whenever one exists.

Cyclic executives

How is the size of the time table restricted?

- Only cyclic schedules are considered:
  - Schedule is repeated with a cycle time ("hyper period") that is equal to the LCM ("least common multiple") of the task periods.
  - Tasks that are not periodic, or that have very long periods, can be handled by reserving time slots in the schedule for a "server" that can handle such special tasks when they arrive.
- Suitable task periods are chosen:
  - To obtain reasonably long cycle times, the task periods should (if application allows) 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

Cyclic executives

How is the scheduler implemented?

- Use 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 start time of the next (the one whose start time is closest in time) element in the queue.
  - When the interrupt occurs, the next element in the circular queue is fetched and the procedure is repeated.

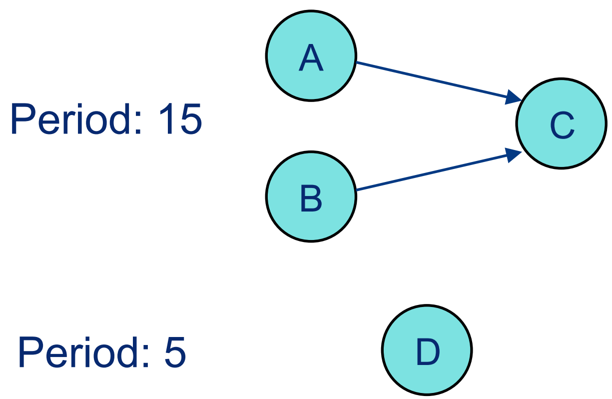
Cyclic executives

Remarks:

- Emulating a cyclic executive
  - By assigning offsets to tasks in a priority-based run-time system it is possible to mimic the behavior of a cyclic executive.
  - Example: assigning offsets to AFTER() operations in TinyTimber
- Emulating other priority-based schedulers
  - By generating a table-based schedule by simulation it possible to mimic the behavior of tasks with static priorities on a run-time system with dynamic priorities (and vice versa).
  - Example: simulating the rate-monotonic (static priority) scheduler to generate a time table that is used to emulate a cyclic executive on the (dynamic priority) TinyTimber run-time system.

Example: simulating EDF

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 cyclic schedule for these tasks by simulating preemptive earliest-deadline-first (EDF) scheduling.



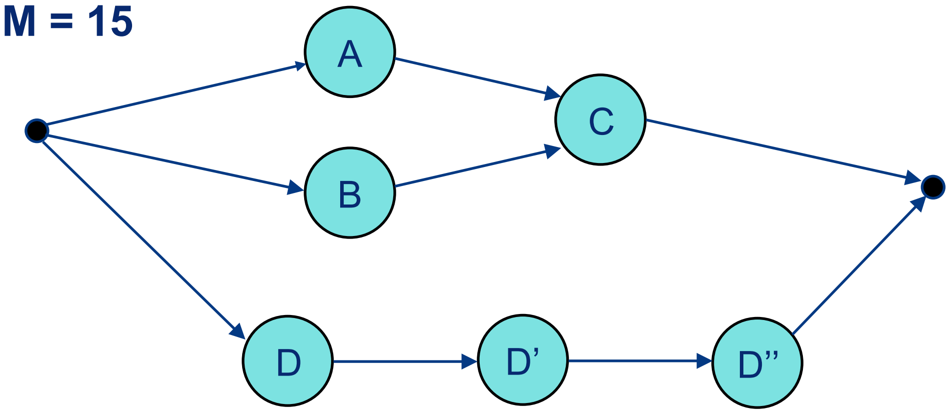
Task	C_i	O_i	D_i	T_i
A	4	0	7	15
B	3	0	12	15
C	5	0	15	15
D	1	3	1	5

Example: simulating EDF

Begin by calculating the LCM of the tasks: $\text{LCM}\{15,5\}=15$

Then generate a new version of the task graph with cycle time 15.

LCM = 15



Task	C_i	O_i	D_i	T_i
A	4	0	7	15
B	3	0	12	15
C	5	0	15	15
D	1	3	1	15
D'	1	8	1	15
D''	1	13	1	15

Observe that D must execute 15/5 = 3 times within the cycle, hence instances D' and D'' in the new graph.

Example: simulating EDF

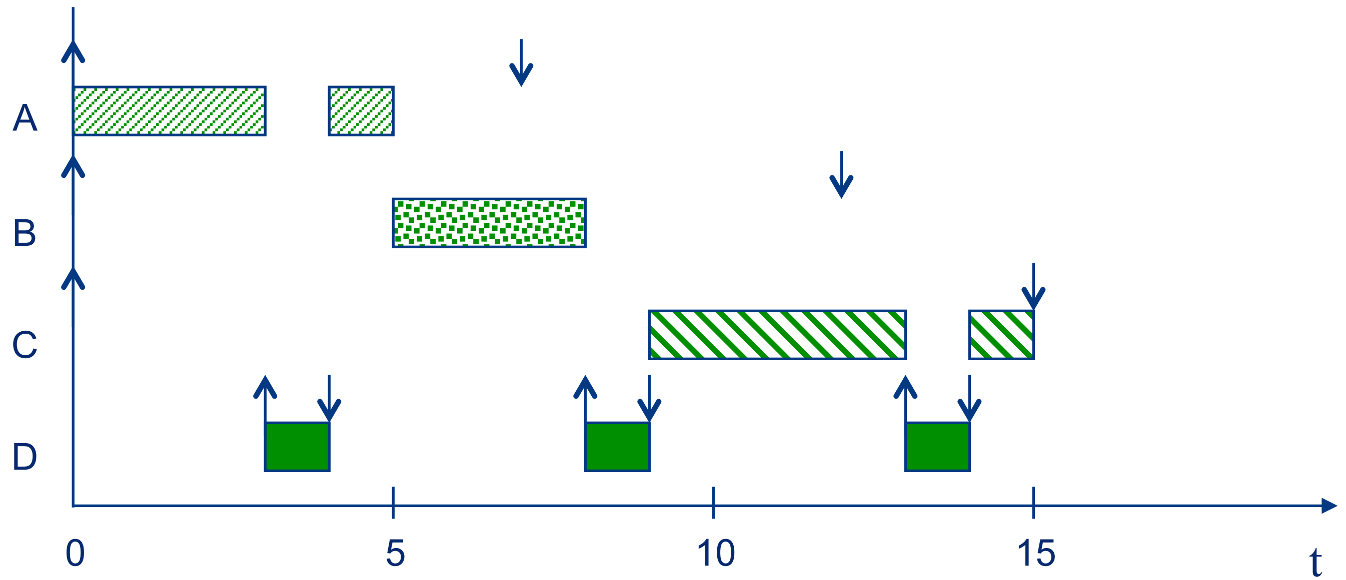
Now generate a schedule by assuming preemptive, earliest-deadline-first scheduling and simulate execution of the tasks:

1. A is scheduled first since it has the earliest deadline among the tasks (A and B) that are ready at t = 0.
2. D becomes ready at t = 3 and preempts A since D's deadline is closer in time than A's and B's deadlines.
3. A resumes its execution at t = 4 and finishes at t = 5.
4. B is scheduled at t = 5 and finishes at t = 8. C becomes ready.
5. D' becomes ready and is scheduled at t = 8 since the deadline of D' is closer in time than C's deadline.
6. C is scheduled at t = 9.
7. D'' becomes ready at t = 13 and preempts C since the deadline of D'' is closer in time than C's deadline.
8. C resumes its execution at t = 14 and finishes at t = 15.

Example: simulating EDF

Static schedule:

NOTE: Since no task executes beyond t = 15, it is sufficient to generate, and check feasibility of, a schedule that spans one hyper period ([0,15])



Cyclic time table:

(A,0,3) (D,3,4) (A,4,5) (B,5,8) (D',8,9) (C,9,13) (D",13,14) (C,14,15)