Dependable Real-Time Systems

7.5 credit points

Professor Jan Jonsson

Department of Computer Science and Engineering
Chalmers University of Technology

Course organization

Lectures (12 of them)

Lectures are offered during study weeks 1–6.

Assuming the background knowledge from the preparatory course EDA223/DIT162 the lectures will introduce new and deeper aspects of scheduling theory (with an increased focus on dependability and multiprocessors).

Consultation sessions (15 of them)

Consultation sessions are offered during study weeks 3–8.

The sessions are used for guidance regarding the homework assignment problems, and also getting your software solutions in the first homework assignment demonstrated and approved.

Course aim

After the course, the student should be able to:

- Formulate requirements for computer systems used in time- and safety critical applications.
- Demonstrate knowledge about the terminology of scheduling, dependability and complexity theory.
- Describe the principles and mechanisms used for scheduling of task execution and data communication in real-time systems.
- Design real-time systems and apply techniques to verify whether the real-time requirements are met or not.

Course aim

After the course, the student should be able to: (cont'd)

- Derive the theoretical performance limitations of a given real-time system.
- Reason about advantages and disadvantages regarding the choice of the optimal design for a real-time systems given certain conditions.

Course examination

Homework assignments

The course contents are examined by means of two homework assignments. The first assignment is handed out in study week 3 and should be done in study week 5. The second assignment is handed out in study week 6 and should be done in study week 8.

Final grade

Each homework assignment is given a score with grade (U, 3, 4, 5).

To pass the course the score of each assignment must have a grade of 3 or higher. The final grade (3, 4, 5) is then based on a weighted average of the scores of the two homework assignments.

Note: GU students use Chalmers grading scale within Canvas, but will get corresponding GU grades in Ladok.

Changes in the course

Since 2018:

- No final written exam (only homework assignments)
- The student must declare percentage of contribution in deriving the solution to each problem in the homework assignments

Since 2019:

- Course web pages are hosted by the Canvas system
- The first homework assignment only has programming problems
- A pass grade in the preparatory course EDA223/DIT162 is required

Online Edition™ 2020:

- New name and course code
  (née Parallel & Distributed Real-Time Systems, EDA422)
- Due to corona virus all teaching should be carried out remotely

Course material

To download: (via Canvas system)

- Lecture notes [Powerpoint hand-outs, guiding material]
- Research articles and book excerpts [very important reading]

- Homework assignment 1 (HWA #1) [available in study week 3]
- Homework assignment 2 (HWA #2) [available in study week 6]

- Template code [for target computer software in HWA #1]
- Handbooks and data sheets [for target computer hardware in HWA #1]
- Development tools [for target computer software in HWA #1]

Course information and support

Teachers and assistants:

- General questions related to the course are primarily answered in conjunction with the lectures.
- Questions related to the homework assignment problems are primarily answered in conjunction with consultation sessions.

Canvas system:

- Get complete information about the course
- Download course material
- Form project groups and submit solutions
- View examination progress and awarded grades

`https://chalmers.instructure.com/courses/9356`

Homework assignments

Homework assignment 1 (HWA #1):

- HWA #1 is handed out in study week 3, and should be solved individually by each student.
- The objective is to (1) implement a dependable CAN message handler, and (2) add some new functionality to the stand-alone melody player from the preparatory course EDA223/DIT162.
- Before the deadline in study week 5 the student should demonstrate the solutions to each of the assignment problems, and also submit the software code to the examiner for review.
- After the demonstration of a solution the student will be awarded points, based on the correctness of the solution and the quality of the software code.

Homework assignments

Homework assignment 1 (HWA #1): (cont'd)

- For HWA #1 each student will be offered loan equipment (MD407 processor card + USB/CAN cables), and work on the assignment at home on their own computer.
- The loan equipment is intended to be used with custom versions of the MD407 monitor and the TinyTimber kernel
  - it is therefore necessary to transfer the old melody player code to the new software environment
  - to that end, it will be possible to pick up the loan equipment already in study week 2 (i.e., in due time before HWA #1 is handed out)

Homework assignments

Homework assignment 2 (HWA #2):

- HWA #2 is handed out in study week 6, and should be solved individually by each student or, if desired, by pairs of students.
- The objective of the assignment is to solve a collection of problems related to real-time scheduling. The problems will encompass a mix of theoretical analysis and paper reading.
- Before the deadline in study week 8 each student should submit a document containing their solutions, and then book a time with the course examiner for an oral examination of the solutions.
- The oral examination of a student's solutions takes place on a day in the two week period following study week 8.

Homework assignments

Homework assignment 2 (HWA #2): (cont'd)

- After the oral examination of a solution to an assignment problem the student will be awarded points for the solution, based on:
  - the correctness of the solution
  - the ability of the student to defend and explain the solution

  and, in case two students work together,
  - the amount of contribution made by that student in deriving the solution (self declared; see Rules of Conduct for details)

Real-Time Systems

REVISITED

Why real-time systems?

Computer systems with special properties:

- Strict timing constraints
  - Responsiveness (= deadlines) and periodicity
  - Failure to meet timing constraints will cause system failure or will negatively affect quality of the user-perceived utility
- Application-specific design
  - Embedded systems (e.g., computer is part of a larger mechanical system)
  - Strict safety requirements (e.g., ISO 26262, IEC 62304, and DO-178C standards)
  - High reliability (e.g., fault tolerance)

Why real-time systems?

Computer systems with special properties:

- Control systems
  - Industrial robots, cars
  - Aircraft, satellites, medical equipment
  - Failure to meet timing constraints may cause major physical/economical damage or even loss of life

- Multimedia systems
  - Portable music players, streaming music
  - Computer games; video-on-demand, virtual reality
  - Failure to meet timing constraints will degrade user-perceived quality

Now, add ‘parallel and distributed’

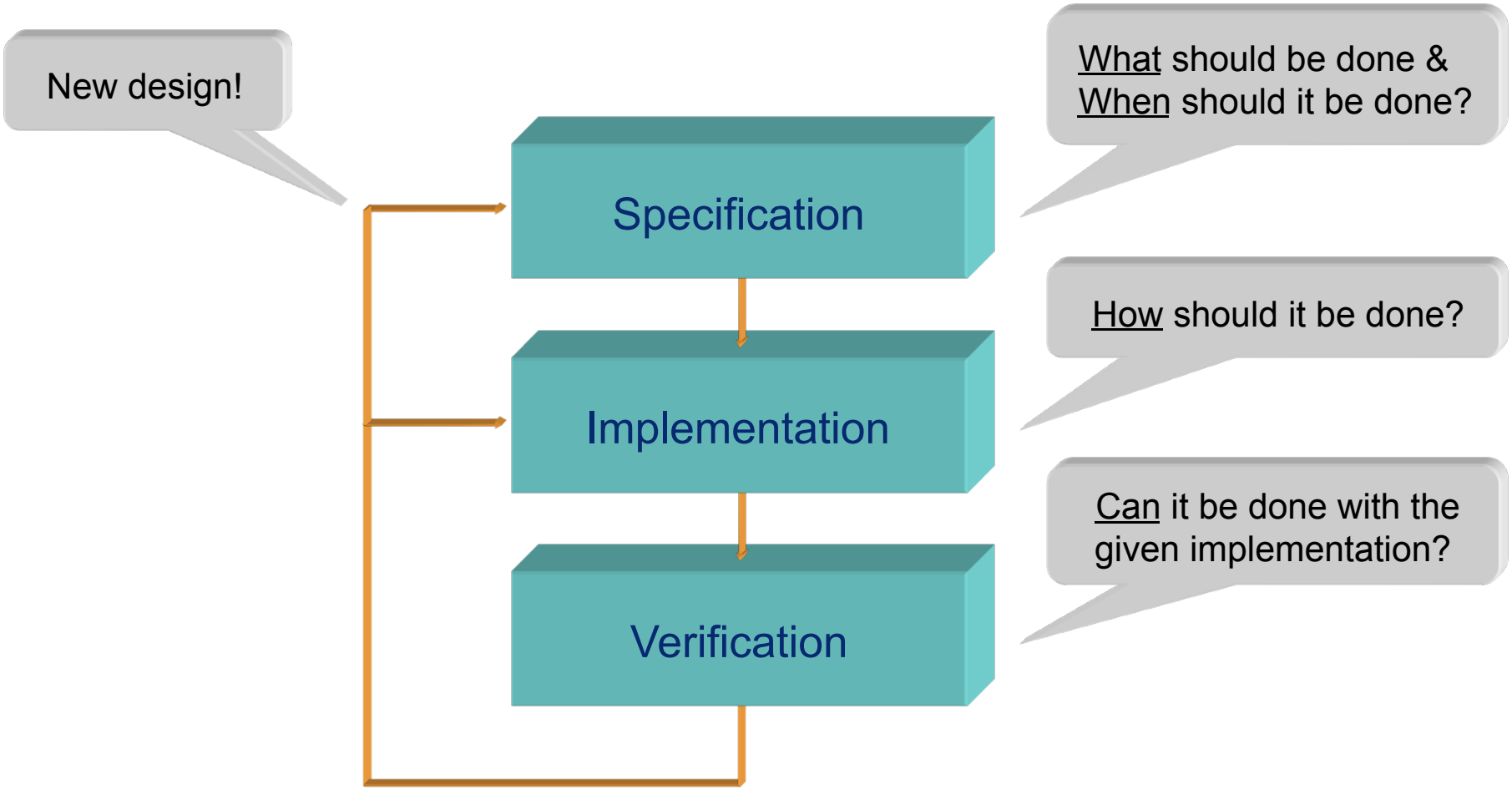
Enables distributed data processing:

- Locality constraints
  - Data processing must take place close to a resource (e.g., sensor or actuator)
- Replication constraints
  - Resources must be replicated to provide reliability (e.g., processors or data buses)

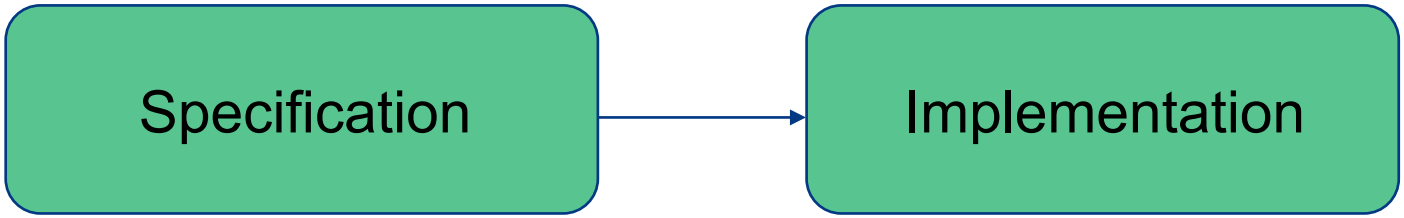
Enables higher performance:

- Improved instruction and data throughput
  - Truly parallel execution of tasks (e.g. multicore processor) and instructions (e.g. superscalar processor w/ multiple pipelines)

Designing a real-time system



Specification



Requirements:		Constraints:
Sampling rate	→	Periodicity
Response time	→	Deadline
Reliability	→	Replication
Resources	→	Locality

Specification

Examples of application constraints:

- Timing constraints
  - A task must complete its execution within given time frames (e.g., task periodicity or deadline)
- Exclusion constraints
  - A task must execute a code region without being interrupted (e.g., a task needs exclusive access to a shared resource)
- Precedence constraints
  - A task must complete its execution before another task can start (e.g., a data exchange must take place between the tasks)

Specification

Examples of application constraints:

- Locality constraints
  - A task must execute on a specific processor because of the vicinity to some resource (e.g., DSP chip, sensor, actuator)
- Anti-clustering constraints
  - Identical copies of a task must execute on different processors for reliability reasons ("spatial replication constraint")
    (e.g., to implement fault tolerance)
  - A group of tasks must execute on different processors for performance reasons
    (e.g., to exploit task parallelism)

Specification

Examples of application constraints:

- Clustering constraints
  - A group of tasks must execute on the same processor for functional reasons
    (e.g., only one processor is used in low-power mode)
  - A group of tasks must execute on the same processor for performance reasons
    (e.g., intensive communication within the group)
  - A group of tasks must execute on the same processor for security reasons
    (e.g., risk for eavesdropping of network bus)

Specification

How critical are timing constraints?

Hard constraints:

If the system fails to fulfill a timing constraint, the computational results is useless.

Correctness must be verified before system is put in mission!

Soft constraints:

Single failures to fulfill a timing constraint are acceptable, but the usefulness of the result decreases the more failures there are.

Statistical guarantees often suffice for these systems!

Verification

Since timeliness is such an important characteristic of a real-time system: how do we verify that the timing constraints are met for a given system implementation?

… so we don't miss that hard deadline …

… so we don't miss too many soft deadlines …

… while we at the same time avoid analyzing all possible software execution scenarios

Verification

Approaches for checking correctness:

- Verification

  Capability: prove the absence of faults!

  Requires formal methods based on mathematical models and theories (e.g., schedulability analysis)

- Testing and validation

  Capability: detect the presence of faults!

  Testing: checking whether the system works correctly in a simplified environment (e.g. in a laboratory room)

  Validation: checking whether the system works in its real target environment (e.g., in a vehicle or a satellite)

Verification

What is needed for formal verification?

- A solid timing model

  Enables expressing the timing properties of the application in a syntactically unambiguous way

  Enables timing constraints to be reflected at all design levels: from specification level (end-to-end constraints) to implementation level

- A solid schedulability analysis

  Enables prediction of required processing capacity, e.g. # and speed of processors, of the hardware (when software is known)

  Enables prediction of required resource usage from the software (when hardware implementation is known)

Verification

How do we simplify schedulability analysis?

- Concurrent, reactive, timing-aware programming paradigm
  - Suitable schedulable unit of concurrency (e.g., task, thread, …)
  - Language constructs for expressing application constraints for schedulable unit (e.g., priorities, delays, …)
  - WCET must be derivable for schedulable unit
- Deterministic task execution
  - Time tables (cyclic executive)
  - Static or dynamic task priorities (+ time quanta for fairness)
  - Run-time protocols for access to shared resources (e.g., priority adjustments and non-preemptable code sections)

Verification

How do we perform schedulability analysis?

- Introduce abstract models of system components:
  - Task model (computation requirements, timing constraints)
  - Processor model (resource capacities)
  - Run-time model (task states, dispatcher)
- Predict whether task executions will meet constraints
  - Use abstract system models
  - Make sure that computation requirements never exceed resource capacities
  - Generate a (partial or complete) run-time schedule resulting from task executions, and detect worst-case scenarios

Task model

Task parameters (static):

C_i Worst-case execution time (WCET)

Longest undisturbed execution time for one arrival of the task

T_i Periodicity

Guaranteed time between each subsequent task arrival

D_i Deadline (relative time)

Maximum allowed time for a task to complete after arrival

O_i Offset (absolute time)

The time of the first arrival of the task

Task model

Classification of task sets:

- Periodic tasks
  - Subsequent task arrivals are separated by a time interval T_i
- Sporadic tasks
  - Subsequent task arrivals are separated by a time interval ≥ T_i
- Aperiodic tasks
  - Subsequent task arrivals have no guaranteed time separation

What we know:

- For preemptive uniprocessor scheduling the periodic and sporadic cases have the same worst-case schedulability properties

Task model

Classification of task sets:

- Implicit-deadline tasks
  - For each task it applies that $D_i = T_i$
- Constrained-deadline tasks
  - For each task it applies that $D_i \leq T_i$
- Arbitrary-deadline tasks
  - No restrictions placed on the relation between D_i and T_i

What we know:

- For preemptive uniprocessor scheduling the implicit-deadline and constrained-deadline cases have schedulability analysis methods with polynomial or pseudo-polynomial time complexity

Task model

Classification of task sets:

- Synchronous tasks
  - There exists a point in time where all tasks arrive simultaneously
- Asynchronous tasks
  - There does not exist a point in time where all tasks arrive simultaneously

What we know:

- For preemptive uniprocessor scheduling the synchronous case constitutes the worst-case scenario (the critical instant) from a schedulability point-of-view

Processor model

Homogeneous processors:

- Identical processors
  - WCET is a constant (the same for all processors)

Heterogeneous processors:

- Uniform processors
  - WCET is the product of a basic execution time (same for all processors) and a scaling factor (may differ for each processor)
- Unrelated processors
  - WCET is not necessarily related for different processors

Run-time model

Task states:

- Waiting
  - Task has not yet arrived for the first time, or has finished executing but not re-arrived
- Ready
  - Task has arrived and can potentially execute on the processor (kept waiting in a ready queue)
- Running
  - Task is currently executing on the processor

Dispatcher:

- A run-time mechanism that takes the first element (task) in the ready queue and executes it on the processor.