UNIVERSITY OF GOTHENBURG

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DIT162 Real-Time Systems, 7.5 credits

Realtidssystem, 7,5 högskolepoäng

Second Cycle

Confirmation

This course syllabus was confirmed by Department of Computer Science and Engineering on 2017-12-19 to be valid from 2018-08-19, autumn semester of 2018.

Field of education: Technology 50% and Science 50%
Department: Department of Computer Science and Engineering

Position in the educational system

The course is offered within several programmes. It is also a single subject course at the University of Gothenburg.

The course can be part of the following programmes: 1) Computer Science, Master's Programme (N2COS), 2) Applied Data Science Master's Programme (N2ADS), 3) Game Design & Technology Master's Programme (N2GDT), 4) Computer Science, Bachelor´s Programme (N1COS) and 5) Software Engineering and Management Master's Programme (N2SOF)

Main field of studies	*Specialization*
Computer Science	A1F, Second cycle, has second-cycle course/s as entry requirements

Entry requirements

To be eligible to the course, the student should have completed the following courses, or equivalent:

- DIT391 Principles of Concurrent Programming, 7.5 credits
- DIT401 Operating Systems, 7.5 credits
- DIT151 Machine Oriented Programming, 7.5 credits

Learning outcomes

After completion of the course the student should be able to:

Knowledge and understanding

- formulate requirements for embedded systems with strict constraints on computational delay and periodicity
- categorize and describe the different layers in a system architecture for embedded real-time systems
- describe the principles and mechanisms used for designing run-time systems and communication networks for real-time applications
- describe how the general principles in real-time programming are implemented in different high-level programming languages

Competence and skills

- construct concurrently-executing tasks (software units) for real-time applications that interface to hardware devices (sensors/actuators)
- apply the basic analysis methods used for verifying the temporal correctness of a set of executing tasks

Judgement and approach

- reflect and argue in writing on ethical aspects regarding the choice of system implementation

Course content

This course is intended to give basic knowledge about methods for the design and analysis of real-time systems. Examples of real-time systems are control systems for cars, aircraft and space vehicles as well as computer games and multimedia applications.

An embedded system is a computer system designed to perform one or a few dedicated functions. It is embedded in the sense that it is part of a complete device, often including electrical hardware and mechanical parts. For reasons of safety and usability, some embedded systems have strict constraints on non-functional behavior such as computational delay and periodicity. Such systems are referred to as real-time systems. Examples of real-time systems are control systems for cars, aircraft and space vehicles as well as computer games and multimedia applications. This course is intended to give basic knowledge about methods for the design and analysis of real-time systems.

Due to the extremely high costs associated with late discovery of problems in embedded systems, it is important to follow a good design methodology during the development of the software and hardware. One means for that is to use a system architecture that offers good component abstractions and facilitates simple interfacing of components. The system architecture philosophy dictates that the software of an embedded system is organized into multiple concurrently-executing tasks, where each task (or group of tasks) implements a specific functionality in the system. This approach allows for an intuitive way of decomposing a complex system into smaller software units that are simple to comprehend, implement and maintain.

The software environment used in the course is based on the C programming language, enhanced with a software library that provides support for programming of concurrent tasks with timing (delay and periodicity) constraints. To that end, a main objective of the course is to demonstrate how the enhanced C programming language is used for implementing communication/synchronization between tasks, resource management and mutual exclusion. Since other programming languages uses monitors or semaphores to implement these functions, the course also contains a presentation of such techniques. In addition, the course demonstrates how to use low-level programming in C to implement interrupt-driven interaction with hardware devices. To demonstrate the general principles in real-time programming, the course also gives examples of how these techniques are implemented in other programming languages, such as Ada and Java.

In order to execute a program containing concurrent tasks there is a run-time system (real-time kernel) that distributes the available capacity of the microprocessor(s) among the tasks. The course shows how a simple run-time system is organized. The run-time system determines the order of execution for the tasks by means of a scheduling algorithm. To that end, the course presents techniques based on cyclic time-table based scheduling as well as scheduling techniques using static or dynamic task priorities. In addition, protocols for the management of shared hardware and software resources are presented. Since many contemporary real-time applications are distributed over multiple computer nodes, the course also presents topologies and medium access mechanisms for some commonly-used communication networks.

In real-time systems, where tasks have strict timing constraints, it is necessary to make a pre-run-time analysis of the system schedulability. The course presents three different analysis methods for systems that schedule tasks using static or dynamic priorities: utilization-based analysis, response-time analysis, and processor-demand analysis. In conjunction with this, the course also gives an account on how to derive the maximum resource requirement (worst-case execution time) of a task.

Sub-courses

1. **Written examination** *(Skriftlig tentamen),* 4.5 higher education credits
   Grading scale: Pass with Distinction (VG), Pass (G) and Fail (U)
2. **Laboratory work** *(Laboration),* 3 higher education credits
   Grading scale: Pass with Distinction (VG), Pass (G) and Fail (U)

Form of teaching

The course is organized as a series of lectures and a set of exercise sessions where the programming techniques and theories presented at the lectures are put into practice. The course material is examined by means of a final written exam. In addition, there is a compulsory laboratory assignment in which the students should implement the software for an embedded real-time application running on a hardware system consisting of multiple computer nodes interconnected by a bus network. Apart from the programming of cooperating concurrent tasks with strict timing constraints, the laboratory assignment also encompasses low-level programming of hardware devices such as timers and network controllers.

Language of instruction: English

Assessment

The student is evaluated by means of a written exam and a compulsory laboratory assignment.

If a student, who has failed the same examined component twice, wishes to change examiner before the next examination, a written application shall be sent to the department responsible for the course and shall be granted unless there are special reasons to the contrary (Chapter 6, Section 22 of Higher Education Ordinance).

In cases where a course has been discontinued or has undergone major changes, the student shall normally be guaranteed at least three examination occasions (including the ordinary examination) during a period of at least one year from the last time the course was given.

Grades

The grading scale comprises: Pass with Distinction (VG), Pass (G) and Fail (U).
A Pass grade (G) for the entire course requires at least a Pass grade for all sub-courses.

A Pass with Distinction (VG) grade for the entire course requires a VG grade for all sub-courses.

Course evaluation

The course is evaluated through meeting after the course between teachers and student representatives. Further, an anonymous questionnaire is used to ensure written information. The outcome of the evaluations serves to improve the course by indicating which parts could be added, improved, changed or removed.

Additional information

The course is a joint course together with Chalmers.

Course literature to be announced the latest 8 weeks prior to the start of the course.

The course replaces the course DIT161, 7.5 credits. The course cannot be included in a degree which contains DIT161. Neither can the course be included in a degree which is based on another degree in which the course DIT161 is included.