Real-Time Systems Programming

Summer-Semester 2002
Lecture 2
5 April 2002

RT System Classification

Overview

1) About this Course
2) Classification of RT Systems

Where are we?

1) ***About this Course***
   - *Aim*
   - *Contents*
   - *Home page*
   - *People*
   - *Homeworks*
   - *Related classes*
   - *Literature*
2) Classification of RT Systems

Aim of this Course

- ***Ultimate objective:***
  you should be prepared to develop real-world RT applications
- ***But don't forget:***
  In theory, theory and practice are the same – but in practice, they aren't ...

- There is only so much a university course can do to prepare you for the real world. In the end, nothing can replace actual experience with real-world problems. But at least you should know where to look for advice when facing a real-world design problem.

Aim of this Course

- After this course, you should:
  - know main ***characteristics*** of real-time (RT) systems
  - understand main ***issues*** involved
  - have an active working knowledge of ***Real-Time POSIX*** and (hopefully) ***Real-Time Java***
  - be aware of alternative language constructs provided by ***Ada 95*** and other languages
  - have ***practical experience*** with small-scale applications
  - (perhaps) have developed a research interest in this area

Course Contents I

1) Classification of RT systems
2) C/POSIX, Real-Time Java, Ada
3) Timing Requirements
4) Dependability requirements
5) Time and Clocks
6) RT entities and RT images
7) Fault prevention and fault tolerance

Course Contents II

8) The pitfalls of C
9) Static program validation
10) Exceptions and exception handling
11) Concurrency
12) Worst-case execution time analysis
13) Scheduling
14) Operating systems for RT applications
15) Low-level programming

The Class Homepage

- www.informatik.uni-kiel.de/inf/von-Hanxleden/teaching/ss02/rt-prog/index.html
- ***Contents:***
  - Lecture slides
  - Homework assignments
  - Current information
  - Further links
  - Questionnaire

- I will try to make the lecture slides available before class – *but may not always succeed* ...
- Further links for example on
  - Papers related to this class
  - Lego Mindstorms – which are the target platform for some of the practical homework assignments
- The questionnaire should be filled out at the end of the semester. It would be particularly helpful if also those students who decide to ***discontinue*** this class for some reason during the semester would submit a questionnaire! ***Thanks in advance!***

People

Reinhard von Hanxleden
rvh@informatik.uni-kiel.de
(Lectures)

Alwin Stengel
ast@...
(Exercises)

Stephan Höhrmann
sho@...
(*E.g.*, Mindstorms)

Kai Witte
kwi@...
(*E.g.*, RT Java)

- *Office hours* for all of us are by appointment – the easiest is to contact us after class

Priorities

- Course certificate (***"Schein"***) depends on
  - Homework submissions
  - Participation in class (in borderline cases)
- If in doubt, ***skip class***, but do ***submit the homeworks,*** and participate in their Friday afternoon discussions!

- It is probably (hopefully!) worthwhile to attend the classes–but to some extent attending class may certainly be substituted by just reading through these slides, and perhaps some of the background literature. However, there is no substitute for doing the homeworks!

Homeworks

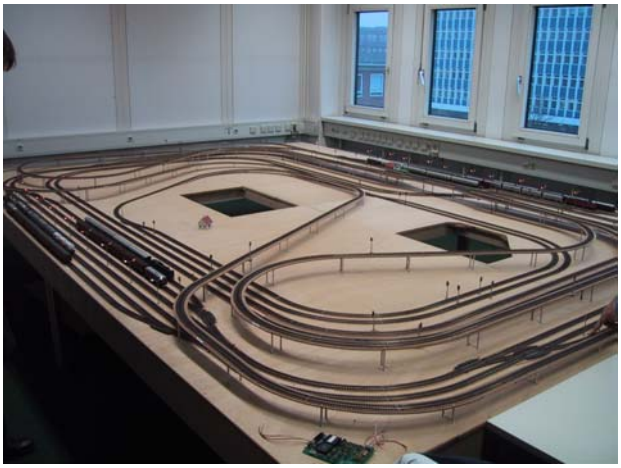
- Homeworks
  - given at end of ***Friday*** lecture,
  - due before following ***Thursday*** lecture,
  - discussed following ***Tuesday*** afternoon
- Homeworks shall be submitted by groups
  - *Ideal group size:* 2 students
  - Each group member should be able to present submissions
- Homeworks should be submitted by e-mail
  - To: ast@...; CC: rvh@...
  - Only one submission per group
  - Submissions should be ASCII-only, no attachments

Homeworks

- *Late* submissions
  - will be accepted
  - however, arbitrary point deductions may result ☺
- *Questions*
  - may be asked at any time, on anything ...
  - ... however, questions on the homework are better asked before the deadline and before submitting the homework!

Related Classes

- *Vorlesung:* Synchrone Sprachen und Modellierungswerkzeuge
- *Seminar:* Programmiersprachen für eingebettete Systeme und Echtzeitsysteme
  - ***Noch Themen erhältlich*** *– bitte Dozenten ansprechen!*
- Modellbahnpraktikum



- *Vorlesung: Synchrone Sprachen und Modellierungswerkzeuge*
  - Do, 14:15 - 15:45, LMS2 - R.Ü1; Fr, 10:00 - 11:30, LMS2 – R.Ü1
  - W.-P. de Roever, R. von Hanxleden, K. Baukus, J. Lukoschus
- *Seminar: Programmiersprachen für eingebettete Systeme und Echtzeitsysteme*
  - S2, Do, 11:45 - 13:15, CAP4 – R.715
  - M. Hanus, K. Höppner, R. von Hanxleden
  - ***Noch Themen erhältlich*** *– bitte Dozenten ansprechen!*
- *Modellbahnpraktikum*
  - P4, Mi, 16:00 – 18:00, CAP4 – R.715
  - J. Lukoschus, A. Stengel

Literature I

[Burns and Wellings 2001] *Real-Time Systems and Programming Languages*, 3rd ed., Burns and Wellings, Addison Wesley, 2001

- Aimed at RT software developers
- Good introduction into ***Ada 95***, ***Real-Time Java***, ***Real-Time POSIX***
- Authors teach at U York
- Good companion website: www.booksites.net/burns
  - Includes code fragments – and slides, gratefully acknowledged here ...
- ***2 copies available in CS library***

Literature II

[Gallmeister 1995] *Programming for the Real World – POSIX.4,* Bill O. Gallmeister, O'Reilly, 1995

- Still probably the best introduction to real-time aspects of POSIX
- Also, a hands-on (and entertaining) introduction to real-time software design in general
- ***2 copies available in CS library***

Literature III

[Laplante 1997] *Real-Time Systems Design and Analysis. An Engineers Handbook*, Phillip A. Laplante, New York, IEEE Press 1997.

- Does not assume strong CS background
- Overview of all topics involved, including hardware
- ***3 copies available in CS library***

Literature IV

[Liu 2000] *Real-Time Systems*, Jane W. S. Liu, Prentice Hall, 2000

- Probably the most systematic treatment of the subject–solid theory
- Good description of exemplary applications
  - Signal processing, video decompression, ...
- Focus on OS aspects of RT computing
- Author teaches at U of Illinois

Literature V

[Kopetz 1997] *Real-Time Systems: Design principles for distributed embedded applications*, Hermann Kopetz, Boston etc., Kluwer 1997.

- Focus on real-time *system* design
- Good introduction into inherent properties of time
- In-depth explanation of time-triggered architectures
- Author teaches at TU Vienna and is associated with TTTech startup
- ***3 copies available in CS library***

Where are we?

1) About this Course
2) ***Classification of RT Systems***
   - *The Time-Value Function*
   - *Soft/firm/hard deadlines*
   - *Guaranteed timeliness vs. best effort*
   - *Where do temporal requirements come from?*

Recall:

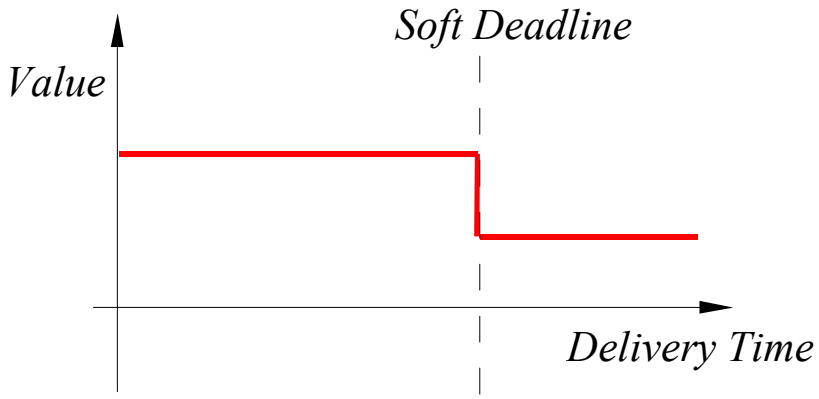
- ***Real-Time Systems:***
  - Focus is predictability – not performance per se
  - Correct behavior = correctness + timeliness of results
  - Must consider dynamics of physical process
- Embedded RT applications are now ubiquitous – making the non-embedded, non-RT system now the *exception* rather than the rule

The Time-Value Function

- Let ***f*(*t*)** be the ***time-value function*** of a real-time computation – that is, the function expressing the value of a result if delivered at time *t*
- Discontinuities of *f* – or of their 1st or 2nd-order derivatives – indicate a ***deadline***
- Another definition of RT-system: a system with a deadline
- RT systems can be classified according to *f*

Soft RT Systems

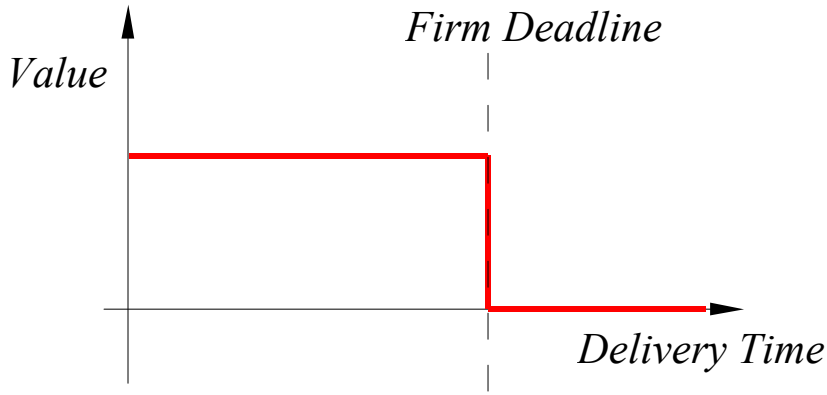
- The result has (some) utility even after deadline



- ***Example:*** Video streaming

Firm RT Systems

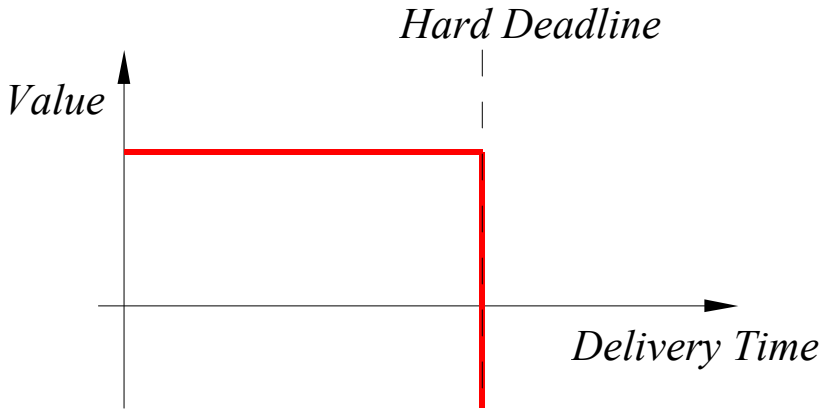
- The result has zero utility after deadline



- ***Example***: Speech processor in cell phone

Hard RT Systems

- Missing the deadline may be catastrophic



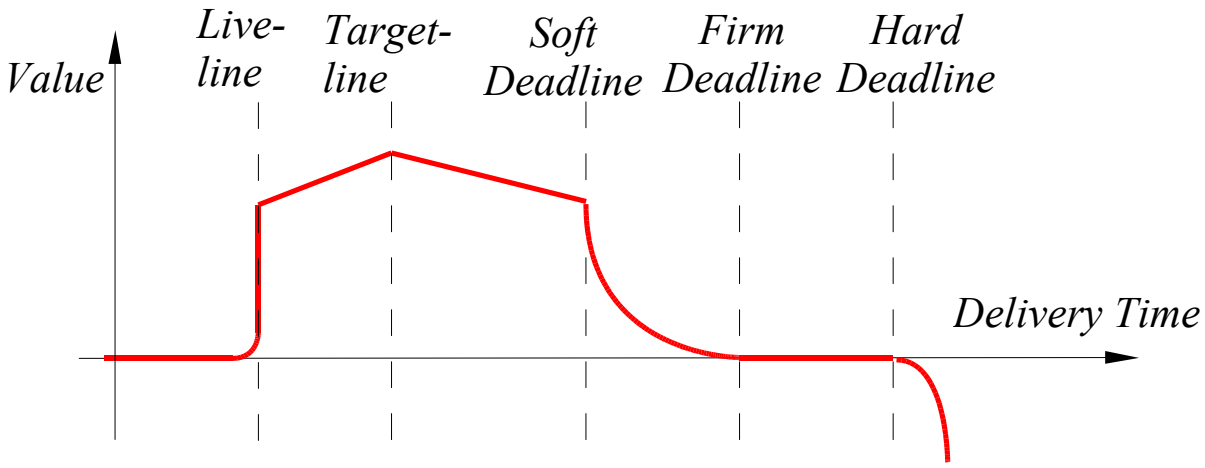
- Hard RT systems ≡ ***Safety-critical systems***
- ***Example***: Air bag controller

Other Types of Deadlines

- ***Recoverable deadline***: miss triggers recovery action
  - ***Example***: time-out at bank teller
- ***Weak deadline***: partial or incomplete results are acceptable if deadline is missed
  - ***Example***: video decompression
- ***Liveline***: result must be delivered after liveline
  - ***Example***: rolling mill
- ***Targetline***: time at which designer aims to deliver result; time of maximum benefit, if known
  - ***Example***: airbag

Notes on the Time-Value Function

- Time-value functions are not always discrete step functions
- Also, a task may have more than a single deadline



- For a further discussion, see A. P. Magalhães, "A Survey on Estimating the Timing Constraints of Hard Real-Time Systems," *Design Automation for Embedded Systems*, vol. 1, no. 3, pp. 213-230, July 1996, Kluwer Academic Publishers, Boston

Guaranteed Timeliness vs. Best Effort

- ***Guaranteed timeliness***:
  - The temporal correctness of a system implementation can be proven – i.e. substantiated by analytical arguments (within the specified load- and fault-hypotheses)
- ***Best effort***:
  - Temporal correctness cannot be proven
  - Temporal verification relies on probabilistic arguments – testing, previous experience, etc.
- Hard RT systems should be based on guaranteed timeliness – this implies ***resource adequacy***

Classification of RT Systems

- ***Soft RT systems***:
  - The result has utility even after deadline
  - ***Example:*** flight reservation system
- ***Firm RT systems***:
  - The result has zero utility after deadline
  - ***Example:*** cell phone
- ***Mission-critical RT systems***:
  - Occasional timing failures are handled as exceptional events
  - ***Example:*** air-traffic control system
- ***Hard RT systems***:
  - Missing the deadline may be catastrophic
  - ***Example:*** air-bag controller; flight control system

Further RT System Classifications

- ***Fail-Safe*** vs. ***Fail-Operational***
  - Error detection coverage critical
  - Often use watch dog, heart-beat signal
- ***Guaranteed Response*** vs. ***Best-Effort***
  - *GR:* Assumption coverage critical
- ***Resource-Adequate*** vs. ***Resource-Inadequate***
- ***Event-Triggered*** vs. ***Time-Triggered***
  - Dynamic vs. static scheduling
  - Presence of global time base

- ***Fail-safeness*** is characteristic of the controlled object, not the computer system
  - ***Example:*** railway signalling system; if a failure is detected, all signals can be set to red (prompting trains to stop)
- System with ***guaranteed response***:
  - ***Requires***
    - a rigorous specification of the underlying assumptions (e.g., regarding the characteristics of the "worst case"/peak load)
    - a precise argument why the system does not fail if the assumptions hold
  - Then the probability of failure is reduced to the probability that the assumptions hold

What is not a Real-Time System ?

- A ***fast*** system is ***not*** necessarily real-time
- RT is ***not*** about ***performance***
- RT ***is*** about ***predictability***
- RT does ***not*** imply ad-hoc, low-level design
- RT design should be a ***systematic process***
  - Architecture
  - Programming Languages
  - Algorithms

Temporal Requirements

Where do temporal requirements come from ?

- Determining these requirements mostly done in ad-hoc fashion – no unified, objective method yet
- Different views:
  - RT controller design
  - Controlled system design
  - Synergistic approach

The RT Controller Designer's View

- Start from pre-existing functional and timing specification, provided by the customer
- Focus on
  - Task scheduling
  - Operating systems
  - Communication protocols etc.
- Prevalently a binary view of timeliness:
  - Time-value function assumed to be a step function
  - Most scheduling algorithms based on this assumption
  - No concept of timing tolerance or graceful degradation

The System Designer's View

- Tend to use (mathematical) model to describe behavior of controlled system
- Focus on
  - ***Stability analysis*** of controlled processes
  - ***Grace time***: tolerance against controller malfunctioning
  - ***Reversibility***: ability to recover from erroneous commands
  - ***Safe state***: non-dangerous system attitude, reachable by passive means

The Synergetic Approach

- Models the controlled system and the controller as single unit
- Concept of ***accomplishment levels*** exhibited by total system
- Obtaining hard deadlines by considering
  - Allowed state-space boundaries
  - Admissible inputs
  - Actual state-space placement as function of time

Announcement

- Next week's lectures (April 11/12) are ***cancelled***
- As a substitute, course participants are advised to read the following papers:
  - Niklaus Wirth, Toward a Discipline of Real-Time Programming, Communications of te ACM, Vol. 20, No. 8, pp. 577-583, Aug. 1977 http://www.informatik.uni-kiel.de/inf/von-Hanxleden/teaching/ss02/emb-pl/papers/p577-wirth.pdf
  - Niklaus Wirth, Embedded Systems and Real-Time Programming, T.A. Henzinger, C.M. Kirsch (Eds.): Proceedings of the First International Workshop on Embedded Software (EMSOFT 2001), Tahoe City, CA, USA, October, 8-10, 2001, Lecture Notes in Computer Science 2211, pp. 486-492 http://www.informatik.uni-kiel.de/inf/von-Hanxleden/teaching/ss02/emb-pl/papers/p486-wirth.pdf
- These references are also the basis for the first homework (see next page)

Problem Set 1 – Due: 18 April 2002

1) What does the time-value functions look like for "showing up for class" – for students and for the lecturer?
   (2 + 2 pts)
2) Give one example each for soft, firm, and hard RT systems
   (3x1 pts)
3) Give a brief summary of both Wirth papers (see previous page), about 2000 chars (+/- 500) each
   (4 + 4 pts)
4) Based on the papers, summarize
   a) What has changed in the 24a between these two papers in the field of real-time programming **(3 pts)**, and
   b) Which issues have remained the same **(3 pts)**