Real-Time Systems Programming

Summer-Semester 2002
Lecture 24
5 July 2002

Priority Inversion

Overview

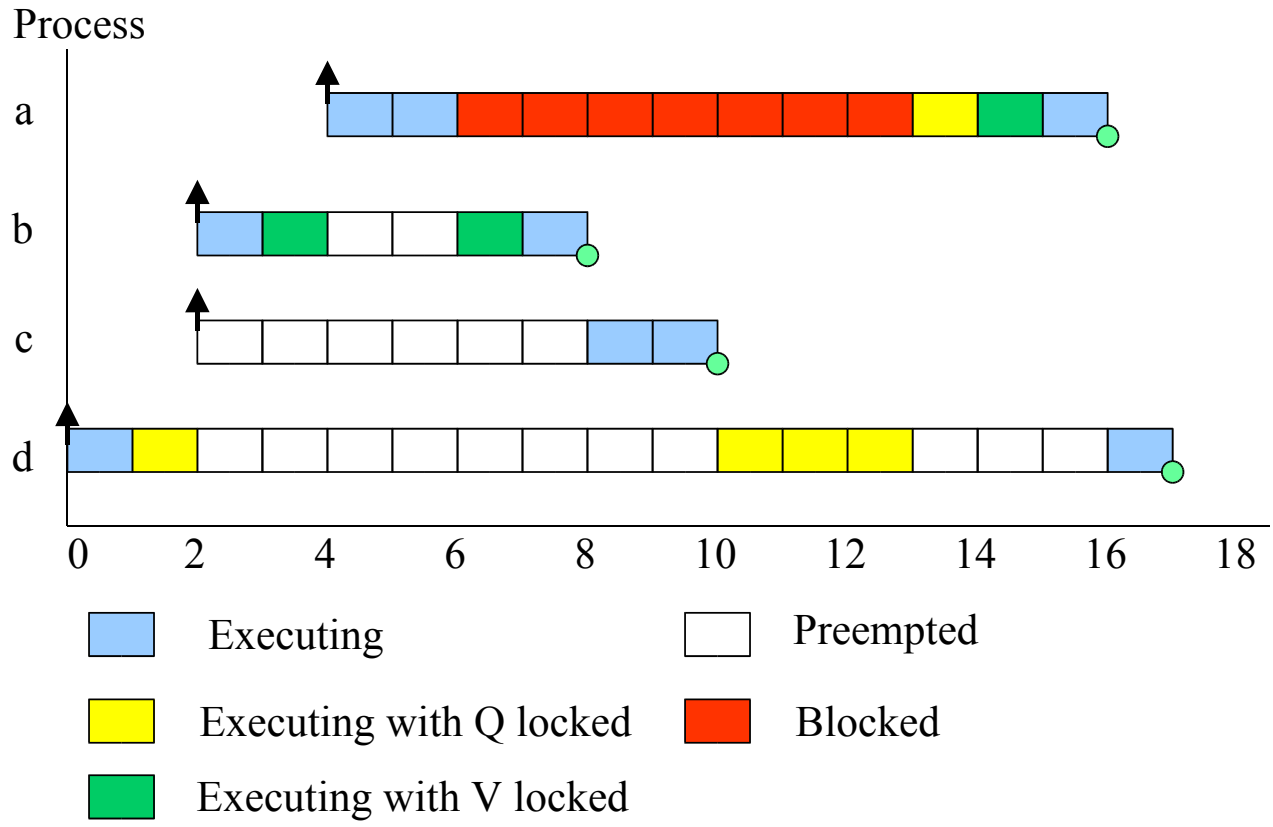
- Process Blocking and Priority Inversion
- Priority Inheritance
- Calculating the blocking time
- Priority ceiling protocols
  - Original Priority Ceiling Protocol (OPCP)
  - Immediate Priority Ceiling Protocol (IPCP)
- Offline vs. Online scheduling

Process Interactions and Blocking

- A process may be ***blocked*** – that is, suspended waiting for a lower-priority process to complete some required computation or to release a resource
- ***Example:*** processes *a*, *b*, *c*, and *d*, accessing resources Q and V as follows

Process	Priority (P)	Release Time	Execution Sequence E: Execute Q: Access Resource Q V: Access Resource V
a	4	4	EEQVE
b	3	2	EVVE
c	2	2	EE
d	1	0	EQQQQE

Example of Resource Sharing

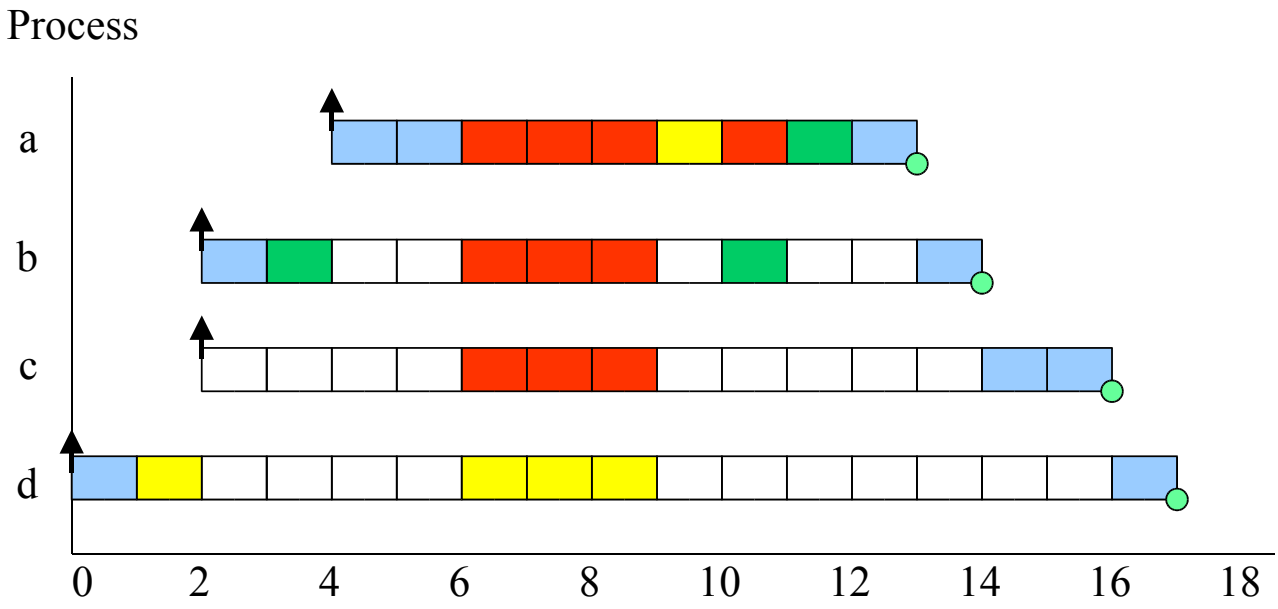


Priority Inversion

- Process (*a*) is ***blocked*** waiting for a lower-priority process (*d*) to complete some required computation or to release a resource:
  - ***Priority inversion***
- Some of this is inevitable if
  - Resources are shared among processes of different priority
  - A lower-priority process that uses a resource cannot be preempted from using this resource by a higher-priority process
- However, the (indirect) blocking of the high-priority process (*a*) by medium-priority processes (*b* and *c*) that prevent the low-priority process (*d*) from freeing the resource *can* (and should) be avoided

A Solution: Priority Inheritance

- If process *p* is blocking process *q*, then *p* runs with *q*'s priority



Calculating Maximal Blocking Time

- If a process has ***m*** critical sections that can lead to it being blocked, then the maximum number of times it can be blocked is *m*
- Let ***K*** be the number of critical sections
- Let ***usage(k, i)*** be 1 if resource *k* is used by at least one process with priority less than P_i, and by at least one process with priority greater or equal to P_i; otherwise 0
- Let ***C(k)*** be the WCET of critical section *k*
- The the maximum blocking time B_i of process *i* is then given by:

$$B_i = \sum_{k=1}^{K} usage(k,i)C(k)$$

Response Time and Blocking

- If we have obtained a blocking time, we can extend formula (5) as follows:

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

- This again can be expressed as a recurrence:

$$w_i^{n+1} = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j$$

Priority Ceiling Protocols

- The standard priority inheritance protocol gives an upper bound on the number of blocks a high-priority process can encounter – however, this may lead to an unacceptably ***pessimistic*** worst case
- Furthermore, ***transitive blocking*** may lead to chains of blocks – process *c* is blocked by *b* which is blocked by *a* etc.
- In addition, we want to eliminate ***deadlock*** when accessing the resource
- All these issues are addressed by the ***Priority Ceiling protocols*** (***PCPs***), of which we will discuss two forms:
  - Original priority ceiling protocol, ***OPCP***
  - Immediate priority ceiling protocol, ***IPCP***

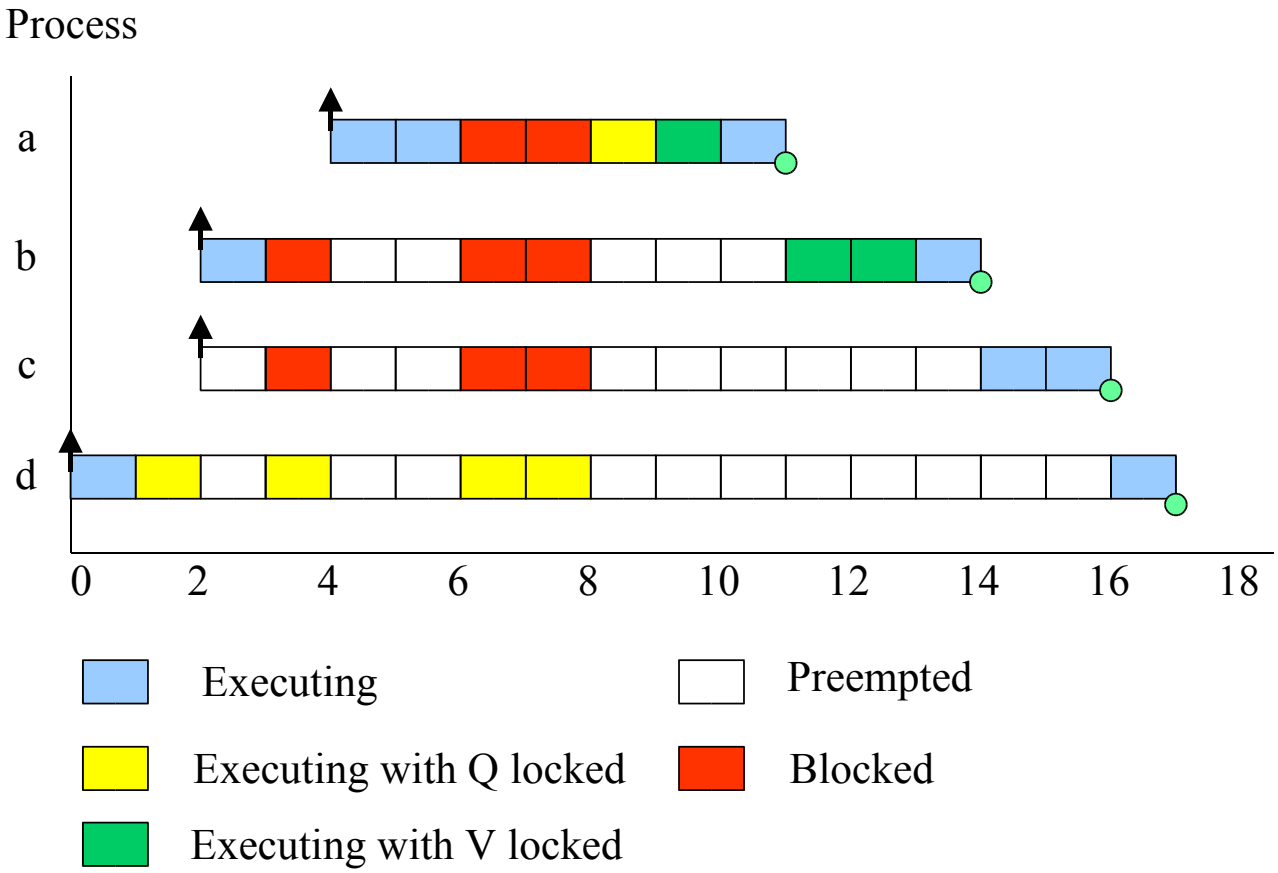
PCP Properties on a Single Processor

- A high-priority process can be blocked ***at most once*** during its execution by lower-priority processes
- Deadlocks are prevented
- Transitive blocking is prevented
- Mutual exclusive access to resources is ensured, by the protocol itself (no semaphores etc. required)

The Original Priority Ceiling Protocol

1) Each process has a ***static default priority*** assigned, perhaps by the deadline monotonic scheme
2) Each resource has a ***static ceiling value*** defined, this is the maximum priority of the processes that may use it
3) A process has a ***dynamic priority*** that is the maximum of its own static priority ***and any it inherits due to it blocking higher-priority processes***
4) A process can ***only*** lock a resource ***if*** its dynamic priority is higher than the ceiling of any currently locked resource (excluding any that it has already locked itself)

OPCP Inheritance

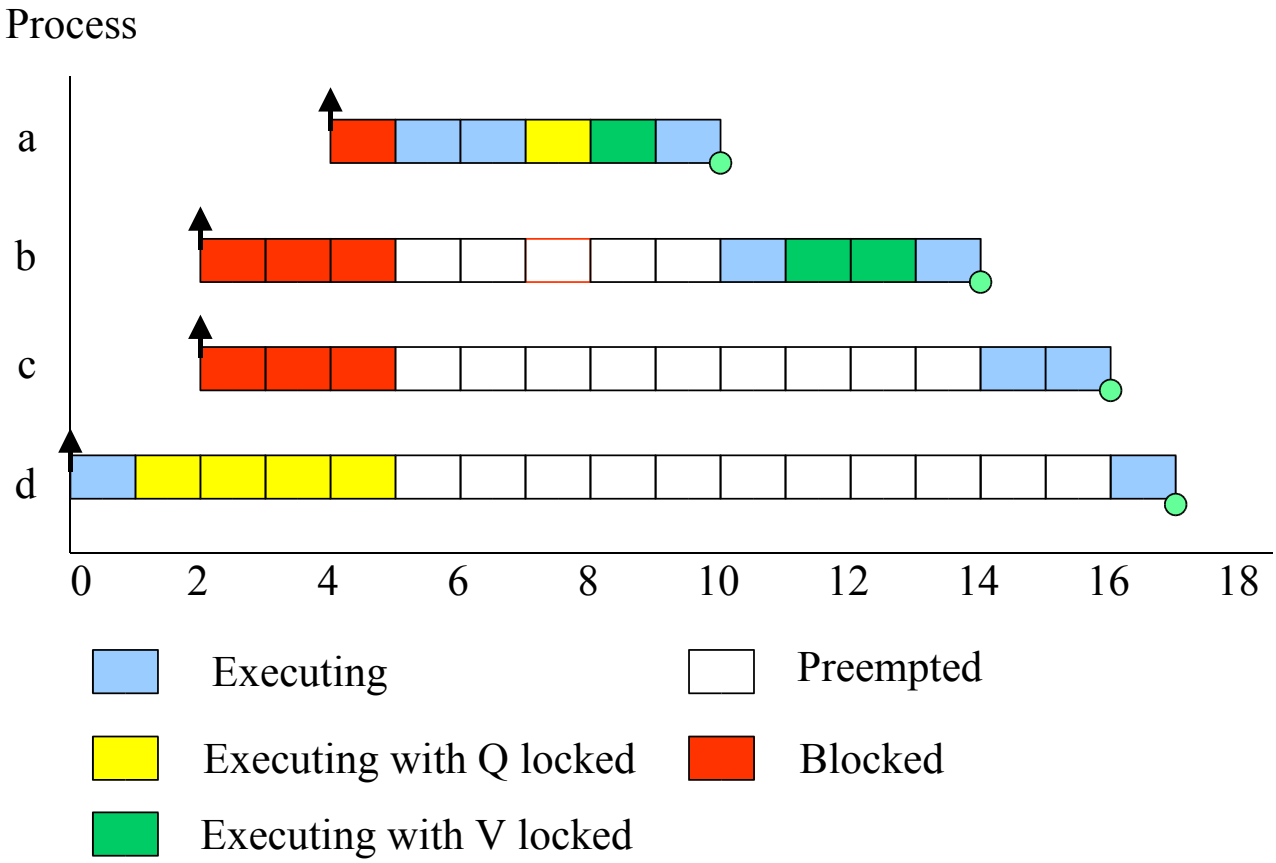


The Immediate Priority Ceiling Protocol

1) Again, each process has a ***static default priority*** assigned, perhaps by the deadline monotonic scheme
2) Also, each resource has a ***static ceiling value*** defined, this is the maximum priority of the processes that may use it.
3) A process has a ***dynamic priority*** that is the maximum of its own static priority ***and the ceiling values of any resources it has locked***

- As a consequence, a process will only suffer a block ***at the very beginning*** of its execution
- Once the process starts actually executing, all the resources it needs must be free; if they were not, then some process would have an equal or higher priority and the process's execution would be postponed

IPCP Inheritance



OPCP versus IPCP

- Although the worst-case behaviour of the two ceiling schemes is identical (from a scheduling view point), there are some points of difference:
  - IPCP is ***easier to implement*** than the original (OPCP) as blocking relationships need not be monitored
  - IPCP leads to ***less context switches*** as blocking is prior to first execution
  - IPCP requires ***more priority movements*** as this happens with all resource usage
  - OPCP changes priority ***only*** if an actual block has occurred
- ***Note:*** IPCP is also called
  - ***Priority Protect Protocol*** (in ***POSIX***)
  - ***Priority Ceiling Emulation*** (in ***Real-Time Java***)

Extensions

- There are numerous extensions to the simple process model discussed so far – for example:
  - Cooperative Scheduling
  - Release Jitter
  - Arbitrary Deadlines
  - Fault Tolerance
  - Offsets
  - Optimal Priority Assignment
- These extensions address real problems – however, today's RT languages and OSs very rarely support any of these extensions
- See [Burns and Wellings 2001] or [Liu 2000] for a detailed treatment

Offline vs. Online Scheduling Analysis

- For ***hard RT systems***, an ***offline*** scheduling analysis is desirable – ***and often mandatory***
- However, this analysis requires
  - bounded and known ***arrival patterns*** of incoming work
  - bounded and known (!) ***computation times***
  - a predictable ***scheduling scheme***
- However, there are dynamic soft real-time applications in which arrival patterns and computation times are not known *a priori*
- Although some level of offline analysis may still be applicable, this can no longer be complete and hence some form of ***online analysis*** is required

Dynamic Systems and Online Analysis

- The main task of an on-line scheduling scheme is to manage any ***overload*** that is likely to occur due to the dynamics of the system's environment
- EDF is a dynamic scheduling scheme that is optimal
- However, during transient overloads EDF performs very badly. It is possible to get a ***domino effect*** in which each process misses its deadline but uses sufficient resources to result in the next process also missing its deadline.

Admission Schemes

- To counter this detrimental domino effect, many on-line schemes have two mechanisms:
  - an admissions control module that limits the number of processes that are allowed to compete for the processors, and
  - an EDF dispatching routine for the admitted processes
- An ideal admissions algorithm prevents the processors getting overloaded so that the EDF routine works effectively
- If some processes are to be admitted, whilst others rejected, the relative importance of each process must be known
- This is usually achieved by assigning a ***value***

Values

- Values can be classified
  - ***Static***: the process always has the same value whenever it is released.
  - ***Dynamic***: the process's value can only be computed at the time the process is released (because it is dependent on either environmental factors or the current state of the system)
  - ***Adaptive***: here the dynamic nature of the system is such that the value of the process will change during its execution
- To assign static values requires the domain specialists to articulate their understanding of the desirable behaviour of the system – this is not always a trivial task ...

Programming Priority-Based Systems

- Traditionally, priority-based scheduling has been more an issue with OSs rather than with programming languages
- In the introduction to scheduling (Lecture 19), we discussed the scheduling-related facilities provided by standard ***UNIX***
- Will further discuss the scheduling interfaces provided by
  - ***POSIX***
  - ***Real-Time Java***

Summary

- Processes may be blocked by lower-priority processes – this is referred to as ***priority inversion***
- A solution for this is ***priority inheritance*** – however, this may still lead to more blocking than necessary
- ***Priority ceiling protocols*** are an improvement over priority inheritance protocols – we discussed the ***Original PCP*** and the ***Immediate PCP***

Problem Set 12 – Due Mon, 15 July 2002

1) Experiment with loading up a machine with processes (better use your own machine!), and how different `nice` values influence the responsiveness of a process. Can you establish a quantitative relationship? **(3 pts)**

2) Complete the optimality proof for the Deadline Monotonic Priority Ordering **(4 pts)**

3) Describe the scheduling mechanism implemented in legOS. **(2 pts)**

4) Give a summary of the technical problems encountered by the Mars Pathfinder mission, as described in http://research.microsoft.com/~mbj/Mars_Pathfinder/ **(3 pts)**

5) *See next page*

Problem Set 12 – Due Mon, 15 July 2002

5) Modify the bar code reader you built last week such that:
After reading the integer x from the bar code, the robot turns x degrees to the right; that is, if $x<0$, the robot turns $-x$ degrees to the left. The effective turning axis of the robot should be at the end of the bar code. If the robot cannot interpret the bar code as valid code, it should give an acoustic signal and continue driving straight.

A primary quantitative success criterion is the number X of successfully followed bar code directions for a given parcours.

A secondary criterion is the driving speed. **(5 + min(X, 5) pts)**

Announcement

No class July 11 – next Thursday

(Excursion to Philips)

Eine wesentliche Eigenschaft eines Computerprogramms ist dessen Ausführungszeit. In vielen Anwendungen reicht es nicht, diese asymptotisch abzuschätzen ("dieser Sortieralgorithmus hat $O(n \log n)$ Laufzeit"), sondern es muss eine Worst Case Execution Time (WCET) möglichst konkret ermittelt werden ("dieser Airbagcontroller hat eine maximale Zykluszeit von 15,8 msec").
Eine Abschätzung der WCET erfordert Analyse des Kontrollflusses (Halteproblem!), der Zielarchitektur, und gegebenenfalls des Compilers. Das Problem ist also nicht trivial, aber gerade für Echtzeitanwendungen oft von zentraler Bedeutung. Der in der Praxis noch immer am weitesten verbreitete Ansatz ist extensives Testen, welches aber sehr aufwändig ist und meistens ein gewisses Restrisiko einer Unterschätzung der WCET birgt. In diesem Seminar werden wir uns mit aktuellen Arbeiten zur WCET-Analyse beschäftigen.

Announcement

Seminar (2h) im WS 2002/03:

Analyse von Programmausführungszeiten

(Or: "*Where do these computation times come from, anyway?*")

http://www.informatik.uni-kiel.de/inf/von-Hanxleden/teaching/ws02-03/s-wcet/index.html

Vorbesprechung:
Fr, 12.07.2002, 13:15 Uhr,
LMS2 - R.Ü1

Eine wesentliche Eigenschaft eines Computerprogramms ist dessen Ausführungszeit. In vielen Anwendungen reicht es nicht, diese asymptotisch abzuschätzen ("dieser Sortieralgorithmus hat $O(n \log n)$ Laufzeit"), sondern es muss eine Worst Case Execution Time (WCET) möglichst konkret ermittelt werden ("dieser Airbagcontroller hat eine maximale Zykluszeit von 15,8 msec").
Eine Abschätzung der WCET erfordert Analyse des Kontrollflusses (Halteproblem!), der Zielarchitektur, und gegebenenfalls des Compilers. Das Problem ist also nicht trivial, aber gerade für Echtzeitanwendungen oft von zentraler Bedeutung. Der in der Praxis noch immer am weitesten verbreitete Ansatz ist extensives Testen, welches aber sehr aufwändig ist und meistens ein gewisses Restrisiko einer Unterschätzung der WCET birgt. In diesem Seminar werden wir uns mit aktuellen Arbeiten zur WCET-Analyse beschäftigen.

Announcement

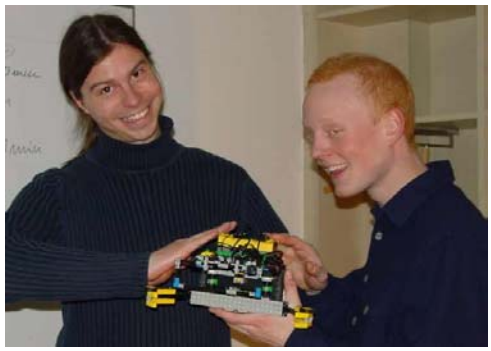
Praktikum (4h) im WS 2002/03:

Entwurf eingebetteter Echtzeitsysteme

(Or: "*How to build a real reactive system*")

http://www.informatik.uni-kiel.de/inf/von-Hanxleden/teaching/ws02-03/p-mindstorms/index.html

Last year's contest ...



... and the winners:

Das Praktikum führt in den Entwurf eingebetteter Systeme ein und behandelt den gesamten Entwurfszyklus von Modellierung, Design, Programmierung bis zum Testen des Systems. Als Plattform verwenden wir das Lego-Mindstorms-System. Neben einer programmierbaren Steuerung und den Konstruktionselementen eines Legobaukastens stellt es Aktoren (Motoren) und verschiedene Sensoren bereit, mit deren Hilfe frei und relativ einfach zum Beispiel Roboter oder andere aktive Systeme konstruiert und programmiert werden können.
Im Rahmen des Praktikums wird ein funktionales Modell eines eingebetteten Systems erstellt und auf Basis der Lego-Mindstorms-Hardware implementiert. Die Programmierung der Mindstorms-Controller wird nicht in der originalen graphischen Software von Lego ausgeführt, sondern mit dem erheblich leistungsfähigeren LegOS und der Sprache C. Für die Modellierung und Design der Controller werden wir darüberhinaus moderne und leistungsfähige Werkzeuge einsetzten, unter anderem die kommerzielle Estudio/Esterel Entwicklungsumgebung. Ausgehend von einfacheren Aufgaben zum Kennenlernen der Werkzeuge wird dann ein größeres Projekt durchgeführt.

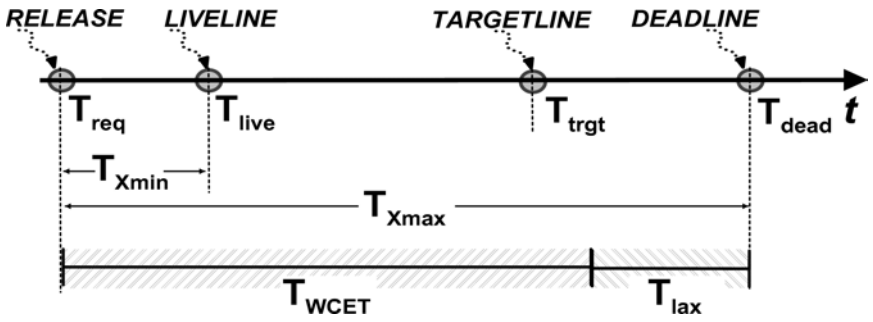
Announcement

Vorlesung (4h + 2h) im WS 2002/03:

Echtzeitsysteme I

(Or: "*RT programming languages – and beyond*")

http://www.informatik.uni-kiel.de/inf/von-Hanxleden/teaching/ws02-03/v-rt1/index.html



Verissimo and Rodriguez 2001

Ein Handy, welches Sprachsignale richtig, aber zu stark verzögert überträgt, ist unbefriedigend. Ein Airbagcontroller, der die richtigen Airbags zündet, aber 100 msec zu spät, ist lebensgefährlich - ebenso wie die zu frühe Zündung der Airbags. Es handelt sich damit bei beiden Anwendungen um *Echtzeitsysteme*, bei denen die Reaktionszeit ebenso entscheidend ist wie das Ausgabeergebnis selbst.

Diese Vorlesung bietet eine Einführung in die verschiedenen Aspekte des Entwurfs von Echtzeitsystemen. Schwerpunkte sind

- Grundlagen zum Zeitbegriff als solchen
- Zuverlässigkeit eingebetteter Systeme
- Programmiertechnische Umsetzung zeitlicher Bedingungen
- Scheduling
- Nebenläufigkeit

Die praktischen Übungsaufgaben sollen zum Teil auf Lego Mindstorms Robotern implementiert werden, in C, Java, oder Real-Time Java.