Programování systémů reálného času
Real-Time Systems Programming
*35PSR

Michal Sojka, Pavel Píša

Czech Technical University in Prague,
FEE and CIIRC

September 25, 2024

What is this course about?

- Real-Time operating systems (RTOS)
  - Demonstrated on VxWorks, similar for others RTOSes as well.
- Real-Time systems principles
  - What is a real-time system? HW, SW, ...
  - The aim is deterministic (predictable) behavior
  - Real-Time scheduling (theory, analysis)
  - Real-Time communication
- Safety critical systems

Literature

- J. Cooling: Real-time Operating Systems: Book 1 – The Theory, 2019
- Giorgio Buttazzo: Hard Real-Time Computing Systems, 2011
- **Jane W.S. Liu, *Real-Time Systems*, Prentice Hall, 2000**.
- Alan Burns and Andy Wellings, *Real-Time Systems and Programming Languages*, Addison Wesley, 2001

Lectures

1. Introduction to Real-Time Systems

2. VxWorks Operating System
3. POSIX 1003.1b – Standard Real-time Systems API
4. Overview of other Real-Time OSes (Linux, RTEMS, ...)
5. Advanced use of C language, GCC compiler

6. Real-Time Scheduling and Analysis (static scheduling, fixed-priority scheduling, deadline scheduling)
7. Static scheduling
8. Fixed priority scheduling
9. Dynamic priority scheduling
10. Real-Time resource management
11. Combining real-time and non-real-time tasks

12. Introduction to safety engineering
13. Safety analysis methods (HAZOP, ...)

Course grading

		Max. points
Tasks 1 – 7		35
within 1 week	5 points	
each other week	-1 point (up to -3)	
Semestral work		25
Code	18 points	
Version control	4 points	
Documentation	3 points	
Exam		40
Exam test	min 13 max 30 pt.	
Oral exam	max 10 pt.	

Authoritative information is on the course web site!

Grade	**Points**
A	90 – 100
B	80 – 89
C	70 – 79
D	60 – 69
E	50 – 59
F	<50 or <13 from exam test

Communication

- Website: `https://rtime.felk.cvut.cz/psr`
- E-mail: *michal.sojka@cvut.cz*
- MS Teams – ask questions that might be of interest for other students (no private emails, please)

Outline

What is a Real-Time System?

- Real-time system is a system whose specification includes both logical and temporal correctness requirements.

  Logical correctness We get correct results, e.g. $1 + 1 = 2$.

  Temporal correctness We get the results at the right time.
- Real-Time system can react in deterministic way to unpredictable events.
- The system is deterministic if the analysis of the worst-case behavior proves that all deadlines (temporal requirements) are met.
  - Techniques to verify the timing requirements are the focus of this course.
  - The question of specifying temporal requirements is important, but it is mostly out of the scope of this course. Typically, it is application specific.

Typical Characteristics of Real-Time Embedded Systems

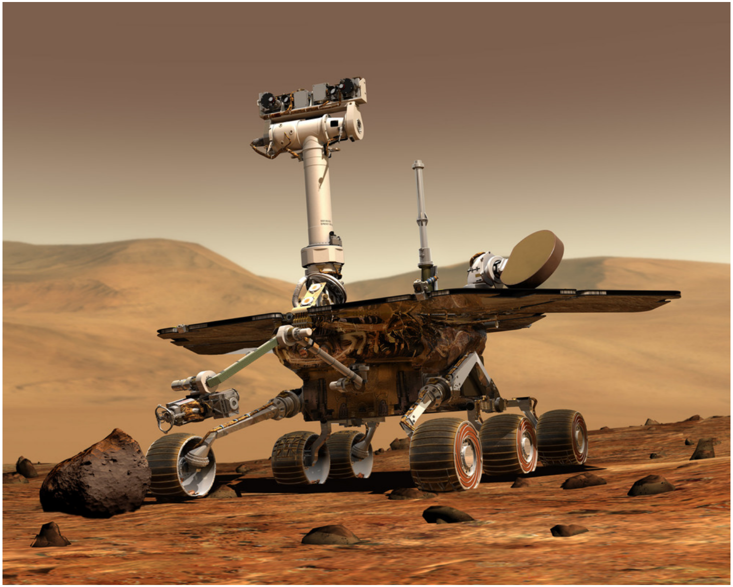
- Event driven, reactive
- Misbehave/failure is often expensive, dangerous, life or environment threatening (safety-critical systems)
- Parallel/multithreaded programming
- Continuous operation without human interaction or supervision
- Strict demands on reliability and fault-tolerance
- Predictable behavior

Real-Time System Examples

Real-Time System Examples

Autor: Stanislav Dusík – Vlastní dílo, CC BY-SA 4.0,
https://commons.wikimedia.org/w/index.php?curid=42618707

Real-Time System Examples



Real-Time System Examples

Common Misconceptions about RT Systems
(Stankovic '88)

Advances in computer hardware will solve all real-time requirements for us.

Common Misconceptions about RT Systems
(Stankovic '88)

Advances in computer hardware will solve all real-time requirements for us.

No! Bad design and/or algorithms can cause ***infinite delays****.*

Common Misconceptions about RT Systems
(Stankovic '88)

Real-time computing is equivalent of fast computing.

Common Misconceptions about RT Systems
(Stankovic '88)

Real-time computing is equivalent of fast computing.

May be for PR and advertising agencies. We understand real-time as ***predictable****.*

Common Misconceptions about RT Systems
(Stankovic '88)

“Real-time” systems research is performance engineering.

Common Misconceptions about RT Systems
(Stankovic '88)

“Real-time” systems research is performance engineering.

Deterministic *timing is often more important than system throughput.*

Common Misconceptions about RT Systems
(Stankovic ’88)

The problems of real-time systems design have all been solved in other areas of computer science/engineering.

Common Misconceptions about RT Systems
(Stankovic '88)

The problems of real-time systems design have all been solved in other areas of computer science/engineering.

IT people are mostly concerned about ***average*** *performance.*

Common Misconceptions about RT Systems
(Stankovic '88)

It is not meaningful to talk about guaranteeing real-time performance, because there is no bug free software and 100% reliable HW.

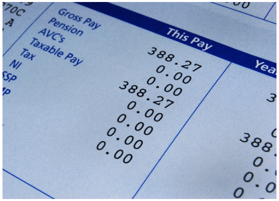
Common Misconceptions about RT Systems
(Stankovic '88)

It is not meaningful to talk about guaranteeing real-time performance, because there is no bug free software and 100% reliable HW.

Even though everything can brake, we do not want the operating system or the application to be the weakest point in the chain.

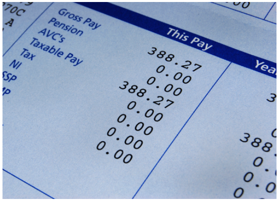
Are All Systems Real-Time Systems?

- *Question*: Is a payroll processing system a real-time system?



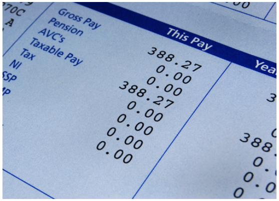
Are All Systems Real-Time Systems?

- *Question*: Is a payroll processing system a real-time system?
  - There is a deadline – print the pay checks once a month.



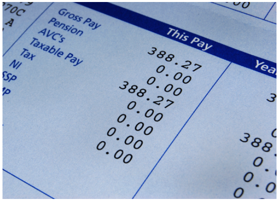
Are All Systems Real-Time Systems?

- *Question*: Is a payroll processing system a real-time system?
  - There is a deadline – print the pay checks once a month.
- According to the definition of real-time systems it is a real-time system. However, we do not consider it as such.



Are All Systems Real-Time Systems?

- *Question*: Is a payroll processing system a real-time system?
  - There is a deadline – print the pay checks once a month.
- According to the definition of real-time systems it is a real-time system. However, we do not consider it as such.
- We are interested in systems for which it is not *a priori* obvious how to meet the timing constraints.



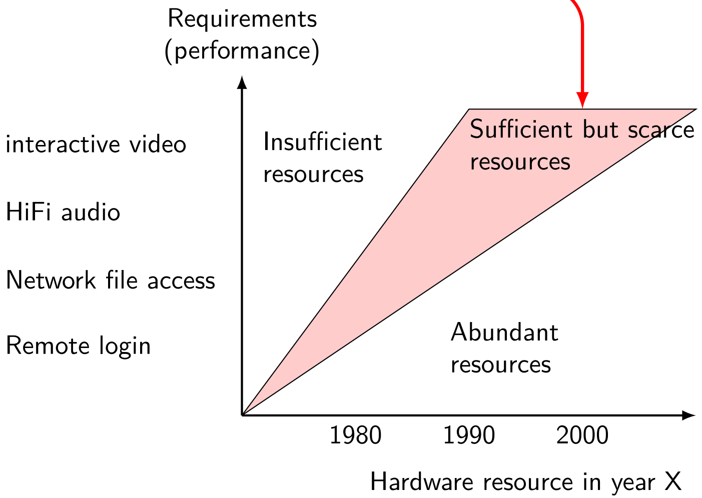
Resource Availability

Resources (CPU, memory, network, ...) may be categorized as:

Abundant	Almost any design methodology can be used to realize timing requirements of the application.
Insufficient	It is not known how to fulfill the application requirements with any known technology.
Sufficient but scarce	It is possible to realize the timing requirements, but careful resource allocation is required.

Example: Interactive/Multimedia Application

- Interesting RT applications are here.

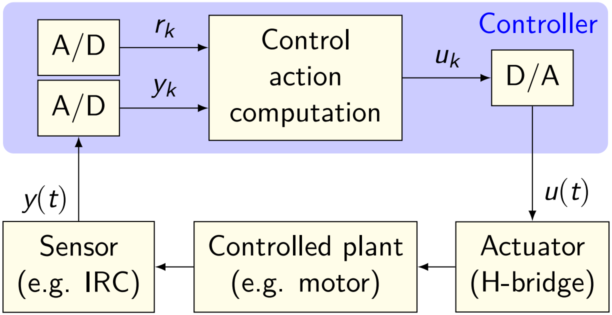


Outline

Example: Real-Time System Application

Almost each **control system** is a real-time system.

Example 1: A simple control system with one sensor and one actuator.



A simple control system (continued)

Pseudocode of a control application

Setup periodic timer to activate interrupt in each period T.
Run the next sequence for each timer interrupt activation:

- run A/D conversion and read value y
- compute controller output u
- write u and start D/A conversion

The parameter T is known as sampling frequency and the section of the proper T value depends on the dynamic properties of the controlled plant. Typical values range from one second to milliseconds or even less. Bad choice makes plant stabilization impossible or setpoint/trajectory tracking less precise than demanded.

Example: Multi-rate system

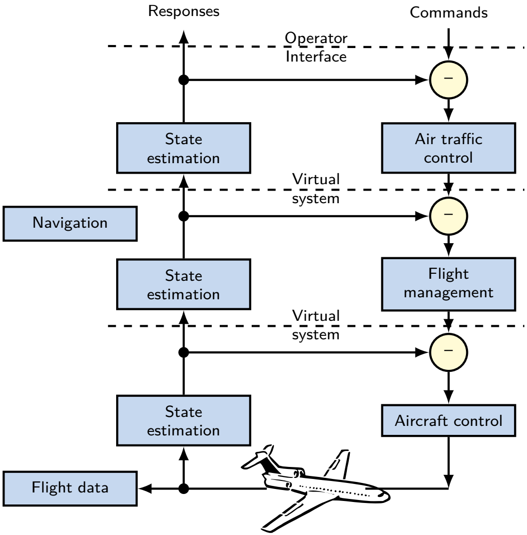
More complex control systems, multiple sensors and actuators. Different system partitions exhibit different dynamics and therefore multiple sampling periods for control loops are used.

Example 2: Helicopter control system

- Do 180× per second the following:
    - Check and read sensor data. Reconfigure (mode change) control strategy in case of sensor failure.
    - Run noise filtering and decimation on input data
    - Avionics control every 6th cycle (30 Hz):
        - Check the keyboard for the change of the control mode
        - Normalize sensor data and coordinate transformations
        - Determine reference setpoints for controllers
        - Compute outer pitch control loop control law
        - Compute outer roll control loop control law
        - Compute outer yaw and collective control loop control law.
    - Every 2nd cycle (90 Hz) run inner controller with setpoints computed in 30 Hz avionics computation):
        - Update controller output for inner pitch control loop.
        - Update controller output for inner yaw and collective control loop.
    - Update controller output for inner roll control loop.
    - Write computed control actions into outputs
    - Run internal tests
    - Wait for the next fast cycle release time

Note: Use of harmonic sampling rates simplifies the system. .

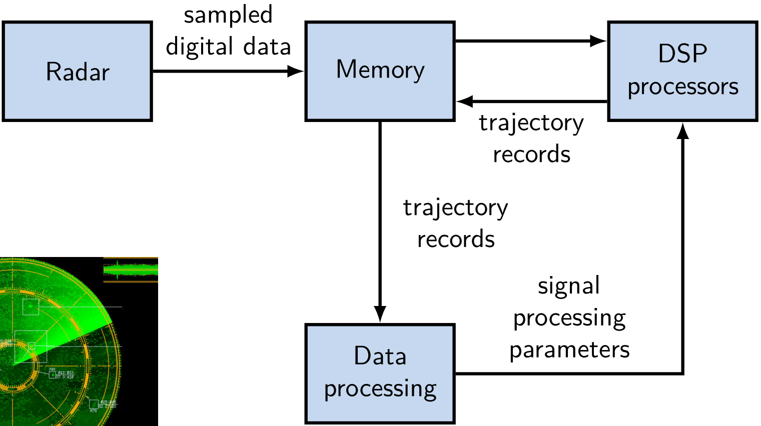
Example: Hierarchical Control System

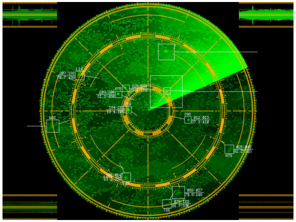


Example: Digital Signal Processing System

- Processing of a signal in different representation and conversion of the signals between these representations.
- Examples:
  - Digital filtering
  - Compression/decompression of video or audio streams
  - Radar signal processing
- Response times range from a few milliseconds to several seconds.

Example: Radar System



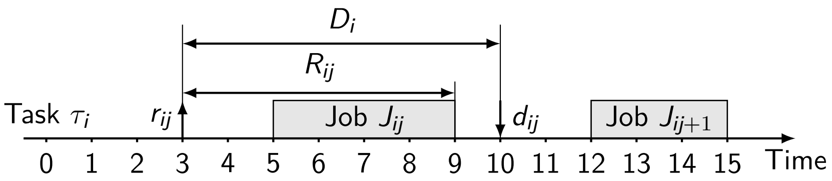


Other examples of real-time system applications I

- Multimedia
  - The typical goal is to process audio and/or video at a constant sampling frequency/framerate
  - Additional limits: Audio and video mutual synchronization, low *jitter* (discrepancy in timing), low latencies for interactive transmission (video phone)
- Real-time (industrial) databases
  - Transactions have to be committed within predefined deadline
  - **Most significant problems:** Classical algorithms for transactions scheduling and optimization aim to achieve high throughput, which is in contrast to the predictability of real-time systems.
  - Requirements for **absolute** or **relative temporal consistency**.
- Virtual reality

Outline

Basic Terminology



- $\uparrow =$ release time (r_{ij}); the job is released at time 3.
- $\downarrow =$ absolute deadline (d_{ij}); the job has to be completed before deadline; equal to 10 for this case.
- Relative deadline (D_i) is 7.
- Response time (R_{ij}) is 6.

Hard Real-Time Systems

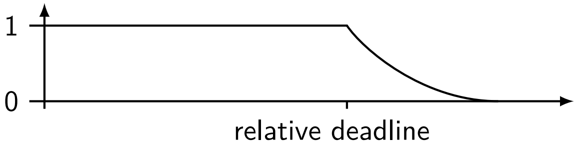
- Hard Deadline is a deadline that has to be met under all circumstances.
  - If a hard deadline is missed, the behavior of the system is wrong and it often has catastrophic consequences.
  - We need mathematical apparatus for verifying that deadlines are met.
  - But: “There is nothing like a hard deadline in the real world.”
- Hard Real-Time System: is a real-time system, where all deadlines are hard.
  - This course is focused on hard real-time systems. They are easier to analyze. Why?
- **Examples:** Nuclear power plant, aircraft control.

Soft Real-Time Systems

- Soft Deadline (required completion time) can be missed occasionally.
  - **Question:** How to define the term “occasionally”?
- Soft Real-Time System: a real-time system where all deadlines are soft.
- **Example:** Multimedia applications, telephone exchanges (but what about emergency calls?).

Try to Define Term "Occasionally"

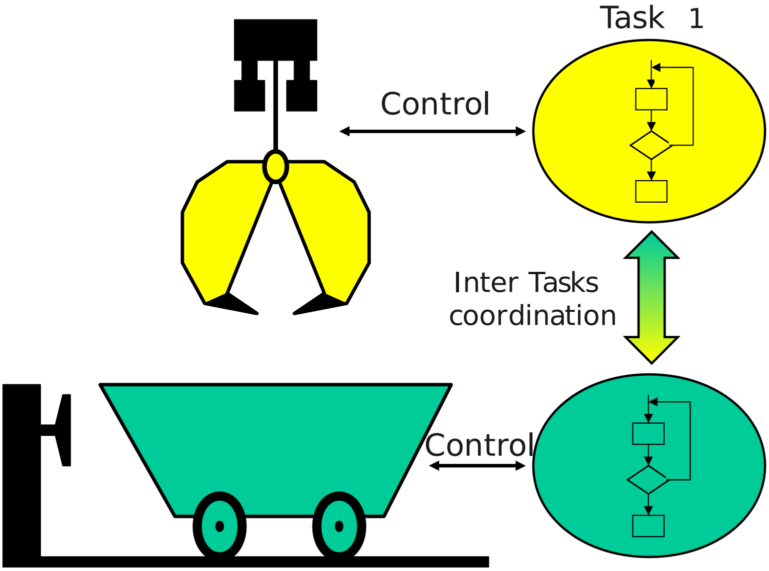
- **One approach:** Use a statistical probability model.
  - For example: 99% deadlines has to be met.
- **Another approach:** Define the function describing the outcome of a job provides as a function of its completion time.



- **Notice:** To analyze system behavior according such model is more complicated.

Outline

Parallelism in the Real World and Control Application



Options for Developing Real-Time Applications

- Bare-metal (microcontroller) application
  - Fast, little overhead, more work (for the programmer)

Options for Developing Real-Time Applications

- Bare-metal (microcontroller) application
  - Fast, little overhead, more work (for the programmer)
- General-purpose OS (GPOS)
  - Parallel data processing reflects the natural parallelism of the real world.
  - Synchronization and data exchange between tasks represent the interaction between the elements in the real world.
  - Allows easy separation of task functions (what tasks do) and temporal characteristics (when it should be done).
  - Provide an interface (API) for application to use the hardware.
  - Allow easier and effective utilization of system resources (hardware) for applications.
  - Higher level of abstraction simplifies application porting to different hardware/software platforms.
  - Address space isolation of processes.
  - No timing guarantees!

Options for Developing Real-Time Applications

- Bare-metal (microcontroller) application
  - Fast, little overhead, more work (for the programmer)
- General-purpose OS (GPOS)
  - Parallel data processing reflects the natural parallelism of the real world.
  - Synchronization and data exchange between tasks represent the interaction between the elements in the real world.
  - Allows easy separation of task functions (what tasks do) and temporal characteristics (when it should be done).
  - Provide an interface (API) for application to use the hardware.
  - Allow easier and effective utilization of system resources (hardware) for applications.
  - Higher level of abstraction simplifies application porting to different hardware/software platforms.
  - Address space isolation of processes.
  - No timing guarantees!
- Real-Time Executive
  - Very simple “OS”, typically just a scheduler and synchronization primitives
  - Microcontrollers, no processes, no networking, no address space isolation, etc.

Options for Developing Real-Time Applications

- Bare-metal (microcontroller) application
  - Fast, little overhead, more work (for the programmer)
- General-purpose OS (GPOS)
  - Parallel data processing reflects the natural parallelism of the real world.
  - Synchronization and data exchange between tasks represent the interaction between the elements in the real world.
  - Allows easy separation of task functions (what tasks do) and temporal characteristics (when it should be done).
  - Provide an interface (API) for application to use the hardware.
  - Allow easier and effective utilization of system resources (hardware) for applications.
  - Higher level of abstraction simplifies application porting to different hardware/software platforms.
  - Address space isolation of processes.
  - No timing guarantees!
- Real-Time Executive
  - Very simple "OS", typically just a scheduler and synchronization primitives
  - Microcontrollers, no processes, no networking, no address space isolation, etc.
- Real-Time OS (RTOS)
  - Similar to GPOS, but provides algorithms for real-time resource management
  - Gives timing guarantees, prevents (theoretically) unbounded latencies
  - Temporal isolation of processes (avionics)

Requirements of a Real-Time OS

- Multi tasking or multi thread environment

Requirements of a Real-Time OS

- Multi tasking or multi thread environment
- Preemptive

Requirements of a Real-Time OS

- Multi tasking or multi thread environment
- Preemptive
  - RT task should not be blocked by other tasks if it is not necessary (shared resources).

Requirements of a Real-Time OS

- Multi tasking or multi thread environment
- Preemptive
  - RT task should not be blocked by other tasks if it is not necessary (shared resources).
- A mechanism to prevent priority inversion or even better to avoid deadlocks

Requirements of a Real-Time OS

- Multi tasking or multi thread environment
- Preemptive
  - RT task should not be blocked by other tasks if it is not necessary (shared resources).
- A mechanism to prevent priority inversion or even better to avoid deadlocks
- Priority driven (or EDF – earliest deadline first)

Requirements of a Real-Time OS

- Multi tasking or multi thread environment
- Preemptive
  - RT task should not be blocked by other tasks if it is not necessary (shared resources).
- A mechanism to prevent priority inversion or even better to avoid deadlocks
- Priority driven (or EDF – earliest deadline first)
- Sufficient number of priorities

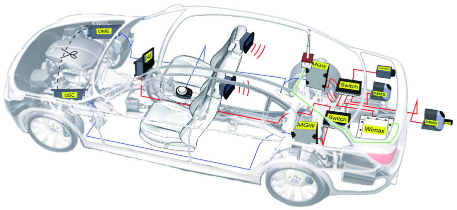
Requirements of a Real-Time OS

- Multi tasking or multi thread environment
- Preemptive
  - RT task should not be blocked by other tasks if it is not necessary (shared resources).
- A mechanism to prevent priority inversion or even better to avoid deadlocks
- Priority driven (or EDF – earliest deadline first)
- Sufficient number of priorities
- Guaranteed response-time for handling of interrupts (interrupt latency)

Requirements of a Real-Time OS

- Multi tasking or multi thread environment
- Preemptive
  - RT task should not be blocked by other tasks if it is not necessary (shared resources).
- A mechanism to prevent priority inversion or even better to avoid deadlocks
- Priority driven (or EDF – earliest deadline first)
- Sufficient number of priorities
- Guaranteed response-time for handling of interrupts (interrupt latency)
- Time resolution for measurement and task activation is fine enough

Distributed Real-Time Systems



- There are many reasons to use distributed systems
  - Increased reliability (redundancy)
  - Distribution of computational power to places where it is needed (fast local servo-control loops)
  - Simpler interconnection of subsystems from different producers (standardized communication protocols)
- Additional problems to solve: **end-to-end properties**
  - End-to-end properties (e.g. response-time) depends on more resources and components (multiple CPUs, network, ...)
  - One of the biggest challenges of today's engineers