Summary

1. Some definitions.
2. Example of real-time systems.
3. Real-Time systems require specific analysis and programming methods.
4. Summary, further readings, planning of the week.
5. References.
Real-time systems (1)

- "Real-time systems are defined as those systems in which the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced" [STA 88].

- Properties we look for:
  - Functions must be predictable: the same data input will produce the same data output.
  - Timing behavior must be predictable: must meet temporal constraints (e.g. deadline, response time).

⇒ Predictable means ... we can compute the system temporal behavior before execution time.
Real-time systems (2)

• A real time system is NOT a system that runs quickly ... this is a system that has temporal constraints to meet.

• Examples of temporal constraints[1DOR 91, 2DEM 99]:
  
  - Few milliseconds for radar systems.
  - One second for machine-man interfaces (in an aircraft for example).
  - Hours for some chemical reactions.
  - 24 hours for weather forecast.
  - Several months or years for some spacecrafts (Mars Express, Voyager, ...).
Real-time systems (3)

- Different types of real-time systems:
  - **Hard (or critical) real-time systems:** temporal constraints MUST be met, otherwise defects could have a dramatic impact on human life, on the environment, on the system, ...
  - **Soft (or non critical) real-time systems:** temporal constraints cannot (sometimes) be met without any dramatic impact.
  - **Opened or closed real-time systems:** tasks/functions can be launched/created at execution time?
Real-time systems (4)

- Different types of real-time systems:

- **Embedded systems**: An embedded system is a computer system designed for specific control functions within a larger system. Often with real-time computing constraints. It is embedded as part of a complete device often including hardware and mechanical parts (e.g. mobile phone, aircraft, automotive, ...)

- **Distributed systems**: "A distributed system consists of a collection of autonomous computers, connected through a network which enables computers to coordinate their activities and to share resources." Coulouris et al. [COU 94].

- **Distributed systems are required:**
  - For dependability (redundancy).
  - Due to physical constraints.
  - For efficiency (resource sharing).
Real-time systems (5)

- Problems raised by embedded systems:
  - Difficult to update/correct when a software failure is discovered (e.g. mobile phone, spacecraft).

- Problems raised by distributed systems:
  - Heterogeneity.
  - Timing behavior.
  - Have many different resources.
Summary

1. Some definitions.

2. **Example of real-time systems.**

3. Real-Time systems require specific analysis and programming methods.

4. Summary, further readings, planning of the week.

5. References.
Example 1: aircraft

- **Hard and closed real-time systems:**
  - Temporal constraints: worst case response time, deadline, ... Strong requirements to meet temporal constraints.
  - Resource reservation on the worst case: to be sure that resources are available when they are required.
  - Closed systems: all tasks are known at design time and are launched at system’s switch on. Resource requirements are easy to estimate.
  - Use of software and hardware redundancy.
  - Use of dedicated software and hardware components: Real-time operating system (e.g. RTEMS), embedded processor (e.g. Leon Sparc processor).
Example 2: multimedia on the Web

- Soft and opened real-time systems:

  Temporal constraints: worst case response time but also jitter, end to end delay, intra-flow and inter-flow synchronization.

  General purpose execution environment (e.g. PC with a Windows operating system).

  Resource requirements difficult to estimate: number of flow, bandwidth of each flow (e.g. MPEG encoder).

  Can not perform worst case resource reservation.
Others examples

- From a book on real-time systems programming:
  "This book is about real-world programming ... So real-world programs (and real-world programmers) are all around us. What characterizes all of these real-world applications is a critical dependence on time." [GAL 95]

- Transportations (train, aircraft, automotive, underground, ...).
- Satellite TV decoder.
- Mobile phones, MPEG3 players, cameras, ...
- Monitoring services (e.g. health devices).
- Robotic systems
- Nuclear power plant.
- ...
Summary

1. Some definitions.
2. Example of real-time systems.
3. Real-Time systems require specific analysis and programming methods.
4. Summary, further readings, planning of the week.
5. References.
Hard real-time system specific practices

- **Real-time critical (hard) systems:**
  - Are concurrent and synchronized applications. Need to handle time.
  - Have high implementation cost: temporal constraints verification, safety, certification, dedicated software development kits/environments.
  - Do not allow software maintenance: e.g. spacecrafts, mobile phones \[\implies\] we cannot correct erroneous software (bugs).
  - May have dramatic impact on human life, on the environment, on the system, ...

- **Specific software engineering:** methods, models and tools to master software quality and development cost.
- **Specific programming tools:** languages, cross-compilers, operating systems.
Typical software engineering process:

- Requirements = What? Are they consistent?
- Design software architecture = How? Verification of the timing/logical constraints. Does the architecture provide the right answer?
- Implementation = write software components.
- Tests: check correctness of the software.
Specific software Engineering (2)

- Software engineering specific practices for real-time software:
  - To reduce costs: do early verification each time we can => design step.
  - Use of Models and tools that automatically handle those models.
  - Analysis methods:
    - Simulation.
    - Model checking. Formal methods for real-time systems: Petri nets, timed automata, synchronous languages, ...
    - Analytical approaches: real-time scheduling analysis.
Specific programming tools (1)

- Use of monitors:
  - Small software size. Very deterministic systems.
    Small memory footprint.
  - No system calls and no separate address spaces: set of functions built as an unique executable.
  - Sometimes no scheduler (high critical).
  - Cross compilers.
Specific programming tools (2)

- Cross-compiling:

  Why cross-compiling: because target has a limited amount of resource, is composed of specific hardware/software (timing behavior).

  **Host**: where we compile the program.

  **Target**: where we run the program.
Specific programming tools (3)

- High level versus low level programming languages:
  - Low level languages: C or assembly languages.
  - High level languages: Esterel or Ada languages.
Specific programming tools (4)

- An example of low level language: C
  - Largely known.
  - Direct access to hardware components.
  - But must be used with libraries for concurrency/synchronization/timing/scheduling.
  - Use of language subsets (e.g. dynamic allocation).
  - Low portability.
  - Well suited for small real-time applications.
Specific programming tools (5)

- An example of high level language: Ada
  - Standard: semantics is well defined $\Rightarrow$ portability.
  - Separate compilation.
  - Safety programming (strong typing, static analysis).
  - Complex language which is difficult to use. Few programmers.
  - Well suited for large real-time applications.
Summary

1. Some definitions.
2. Example of real-time systems.
3. Real-Time systems require specific analysis and programming methods.
4. Summary, further readings, planning of the week.
5. References.
Summary and further readings (1)

- **Real-time systems**: systems with timing constraints to meet. Are usually concurrent systems (tasks and synchronization).

- **Embedded systems and distributed systems**.

- **Specific development process and programming technologies**: modeling and early verifications (e.g. real-time scheduling analysis), specific operating systems and programming languages (cross-compiler, real-time features).
Summary and further readings (2)

- About real-time scheduling theory:

- Real-time scheduling facilities with POSIX 1003:

- Real-time scheduling facilities with Ada:
Summary and further readings (3)

- Other books on real-time systems:
Planning of the week

1. Monday:
   - Lecture: introduction, real-time scheduling (fixed priority scheduling, EDF, shared resources).
   - Exercise: real-time scheduling analysis.

2. Tuesday:
   - Lecture: real-time programming; C programming with POSIX/RTEMS.
   - Exercise: real-time scheduling (cont).
   - Lab: real-time programming; C with POSIX/RTEMS.

3. Wednesday:
   - Lecture: real-time programming; Ada programming with RTEMS.
   - Lab: real-time programming; C with POSIX/RTEMS (cont); Ada RTEMS.

4. Thursday:
   - Lecture: real-time programming; Ada programming with RTEMS (cont).
   - Lab: real-time programming; Ada with RTEMS (cont).

5. Friday:
   - Lecture: real-time scheduling (architecture description languages, multi-core/distributed systems).
   - Lab: real-time programming; Ada with RTEMS (cont).
References


