1. Embedded Systems Overview

1.2 Introduction and Performance Measures

Roadmap for Section 1.2

- Introduction and Vocabulary
- Performance Measures
- General Structure of a Real-Time System
- Misconceptions about Real-Time Systems
- Incidents

Introduction and Vocabulary

What is a Real-time System?

> "A real-time system is one in which the correctness of the computations not only depends on the logical correctness of the computation, but also on the time at which the result is produced. If the timing constraints of the system are not met, system failure is said to have occurred."

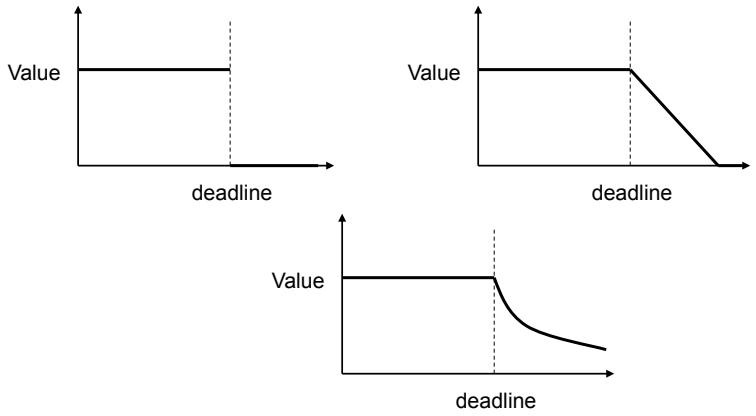
Confusion:

- Not a clear definition!
- What are timing constraints ? (tasks have deadlines)

More confusion

- What is the meaning of a "deadline"?
  - Do all tasks have to be executed before their deadline? (not necessarily)
  - Sometimes "yes": flight control in an aircraft
  - Sometimes "no": Multimedia-App.
- What is the meaning of "executed"?
  - How to decide whether a task has been (completely) executed?
  - Relatively simple: bank transaction
  - Impossible: Computation of π
- How to deal with tasks that missed their deadlines?
  - Terminate or run to completion?
  - Aircraft accident vs. Videoconference

Task Value Functions



Hard vs. Soft Real-time Systems

- Hard real-time systems
  - Embedded systems: aircraft control, nuclear power plants, chemical reactors, jet engines
  - Missing a deadline has life-threatening results.
- Soft real-time systems
  - Multimedia, airline reservation system
  - Missing a deadline is undesirable and impacts system performance but has not destroy lives or equipment.

Vocabulary

- Example: Car & Driver
- Well-known example for human control:
  - Comparable to a real-time computer system in many respects
  - Driver: **real-time controller**
  - Car: **controlled process**
  - Road and additional cars: **operating environment**
- Actuators:
  - Wheels, engine, brakes
- Controls:
  - Steering wheel, brake pedal, switches

Mission Statement

- Drive within the allowed speed range from start A to destination B without collisions with other cars or stationary objects.
- How can driver‘s performance be measured?
  - Departs from A and reaches destination B
  - Total driving time
- But: road conditions have to be taken into account
- What, if driver leaves the road?
  - Success: collision could be avoided
  - Failure: control over vehicle was lost

The Mission – a closer look

- Performance is no absolute measure.
- Performance measures quality of a result in terms of the best possible result under the current environmental conditions.

- A closer look onto the mission:
  - Mission critical: steering, brakes
  - Non-critical: radio, lights
- Deadlines are not constants (rush hour vs. Sunday drive)
  - How to measure the drivers physical condition?

Performance Measures

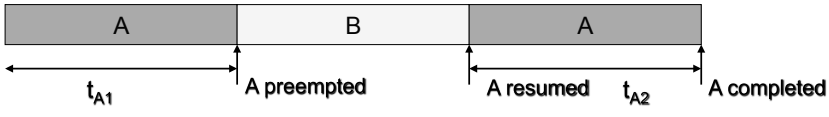
- Average values say very little about the performance of a real-time controller.
- In our scenario:
  - How to value abrupt acceleration/deceleration maneuvers ?
  - How to measure for unnecessarily increased fuel usage?
  - What about extra slow driving?

Problems of RT Computing

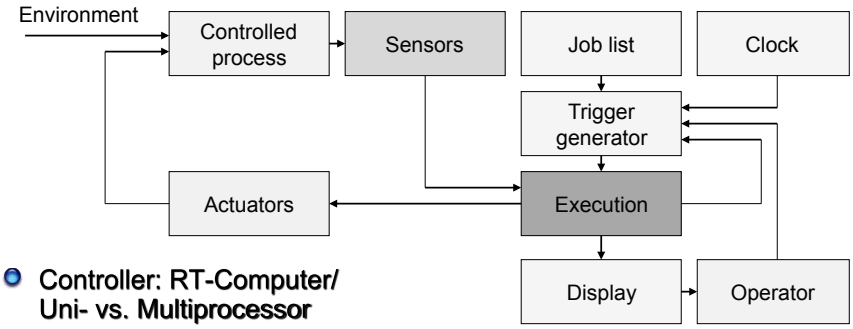
- Reliability, Fault-tolerance
  - Harsh environments, electromagnetic noise, rapidly changing computation loads
- Task Scheduling
  - Traditional Approach: fairness / round robin scheduling / time slicing
  - RT System: fixed priority scheduling / generalized rate monotonic scheduling / earliest deadline first
- Memory Management
  - Swapping / paging
  - Static pre-allocation (mpin(), vm_wire())

Problems of RT Computing (contd.)

- Cache Allocation Policy
  - Preemption may cause cache invalidation -> missed deadline
  - Does $t_A = t_{A1} + t_{A2}$ hold?



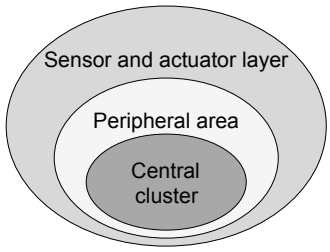
Structure of a Real-time System



- Controller: RT-Computer/ Uni- vs. Multiprocessor
- Input data rates: typically < 1 KB/sec
- Fixed set of processes; software is "pre-loaded"
- Scheduler (offline vs. online schedules)
- Output data rates: typically < 1 command/25 ms

Data Rates

- Sensors/Actuators/Display/Input Panels: low
- Data conversion/formatting: medium (peripheral area)
- Control algorithm: high (central cluster)
- Controlled process often moves through different phases
  - Varying sets of priorities, control tasks, deadlines



Task Classes

- Periodic, sporadic and aperiodic tasks
- Critical and non-critical tasks
- Non-critical real-time (soft real-time tasks):
  - Objective: maximize percentage of jobs successfully executed

Areas of Interest

- Architecture
  - Processor Architecture
  - Network Architecture
  - Architectures for Clock Synchronization
  - Fault-tolerance and Reliability Evaluation
- Operating System
  - Task Assignment and Scheduling
  - Communication Protocols
  - Failure Management and Recovery
  - Clock Synchronization Algorithms
- Others
  - Programming Languages
  - Databases
  - Performance Measures

Misconceptions

(John A. Stankovic, Krithi Ramamritham; "Hard Real-Time Systems"; IEEE Computer Society Press, ISBN 0-8186-0819-6)

- There is no science in real-time system design
  - state-of-the-art is mostly ad hoc
  - scientific approach badly needed: example Space Shuttle, timing bug, transient overload during initialization
  - formal description techniques; verification needed

Real-time vs. Parallel Computing

- Advances in supercomputer hardware will take care of real-time requirements
  - parallel processing to improve throughput: timing constraints not automatically met
  - the more hardware, the harder task and communication scheduling problems
  - no substitute for intelligent deployment of finite resources

Misleading Performance Measures

- Real-time computing is equivalent to fast computing
  - fast computing: minimize average response time
  - real-time computing: meet individual timing requirement for each of the tasks
  - real-time systems need predictability
  - Given a set of demanding real-time requirements and an implementation using the fastest hardware and software possible: how can one show that the specified timing behavior is indeed being achieved? testing is not an answer!

The Art of Real-Time Programming

- Real-time programming is assembly coding, priority interrupt programming and writing device drivers
  - current practice relies heavily on machine-level optimization techniques
  - reliance on clever hand-coding and difficult-to-trace timing assumptions is major source of bugs
  - research objective: synthesis of highly efficient code and schedulers from timing constraint specifications

RT Systems Engineering

- Real-time systems research is performance engineering
  - important: resource allocation strategies; but also:
  - specification/verification of timing behavior
  - programming language semantics
  - which role does time play as a synchronization mechanisms?
  - is there a least restrictive set of timing constraints that is sufficient for the purpose?

Nothing new?

- The problems in real-time system design have all been solved in other areas of computer science of operations research
  - there are unique problems which are not investigated in other areas of computer science
  - average performance parameters versus meeting stringent deadlines
  - queuing models usually make assumptions based on stable operating conditions/large populations which may be unrealistic for real-time systems
  - scheduling: one shot versus periodic tasks; communication, synchronization

Guarantees are incomplete?

- **It is not meaningful to talk about guaranteeing real-time performance because we cannot guarantee that the hardware will not fail and the software is bug-free or that the actual operating conditions will not violate the specified design limits**
  - It's true: one can only hope to minimize the probability of failure in the system one builds
  - how is a system to be build in a way that we can have as much confidence as possible that it will meet specifications at acceptable costs
  - odditities of external world do not give designer a license to INCREASE odds of failure

The Environment

- Real-time systems function in a static environment
  - different sets of timing constraints at different times
  - design of hierarchical schedulers important
  - objective: re-configuration and minimal disruption to ongoing operation
  - long-lived systems

The Ariane 5 Failure
(4 June 1996)

On 4 June 1996 the maiden flight of the Ariane 5 launcher ended in a failure, about 40 seconds after initiation of the flight sequence. At an altitude of about 3700 m, the launcher veered off its flight path, broke up and exploded. The failure was caused by **"complete loss of guidance and attitude information"** 30 seconds after liftoff.

To quote the synopsis of the official report:

- "This loss of information was due to ***specification and design errors in the software*** of the inertial reference system. The extensive reviews and tests carried out during the Ariane 5 development programme did not include adequate analysis and testing of the inertial reference system or of the complete flight control system, which could have detected the potential failure." Because of this conclusion, the accident has generated considerable public and private discussion amongst experts and lay persons. Code was reused from the Ariane 4 guidance system. The Ariane 4 has different flight characteristics in the first 30 seconds of flight and exception conditions were generated on both IGS channels of the Ariane 5.
- Failures often came not from the first, careful, conservative implementation of a design, but from its extension.

A Space Shuttle Control Incident
(1981)

- After a delay in a space shuttle mission in 1981, the crew put in some time in the simulator in Houston. They tested a **"Transatlantic abort"** sequence, which dumps fuel and leads to a landing in Spain. The flight control computers **"locked up and went catatonic".**
- It turns out that an exception condition was generated by a **"computed GOTO"** (but remember this is assembly, not FORTRAN).
- The incident was recounted by Tony Macina and Jack Clemons to Alfred Spector and David Gifford for the *Case Study: The Space Shuttle Primary Computer System*, in *Communications of the ACM* 27(9), September 1984, pp874-900.

Further Reading

- Jane W.S. Liu, *Real Time Systems*, Prentice Hall, ISBN 0-13-099651-3, 2000.
- C.M. Krishna and Kang G. Shin, *Real-Time Systems*, McGraw-Hill, ISBN: 0-07-057043-4, 1997.
- Hermann Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers, ISBN 0-79-239894-7, 1997.
- John A. Stankovic, Krithi Ramamritham; "Hard Real-Time Systems"; IEEE Computer Society Press, ISBN 0-8186-0819-6.