Real-Time Systems

Characterizing an RTS

SUMMARY

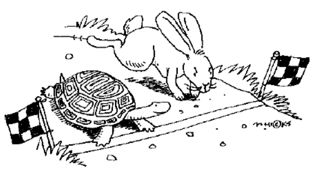
claude.baron@insa-toulouse.fr

Real-time/embedded systems

- **Real-time system**: must provide a service in a time-critical context
  - system evolution (**reactive** system)
  - time constraints (**deadlines**)
- as opposed to interactive or transformational systems
- **Embedded system**:
  - autonomous, characterized by a tight coupling between hardware and software
  - used for a very specific purpose
  - often part of a larger system
  - ~90% of the global processor market

Relevance of response time

- Produce results that are not only accurate (*logical accuracy*)...
- but are also delivered within a specific time (*temporal accuracy*) and are compatible with the system's evolution
  - The time scale depends on the system:
    - a few milliseconds for an air navigation system
    - several minutes/hours for the control of a chemical reactor
- In a real-time system, a mathematically accurate calculation that is delivered after a predefined deadline equals a false result
- Real-time does not mean fast!



Key concepts in real-time systems

Definitions vary depending on the subject matter

Common thread: the importance of the time factor

RTS must react according to the flow of time (timeliness property)

=> differentiates *time-constrained* applications from others

A real-time system is defined as a system whose correctness of the system depends not only on the logical results of computations, but also on the time at which the results are produced [STA 88].

An out-of-time accurate result equals a false result.

Note: Real-time **does not necessarily involve rapidity**, but rather timely execution that takes into account the phenomenon's dynamics

Key concepts in real-time systems

A **minimum performance** is often required (response time, communication deadlines, etc.)

Insufficient:
- correctly adapting them to target time constraints (using appropriate scheduling algorithms)
- oversizing the system

All real-time applications **interact with their *environment*** (industrial processes, aircraft engines, cities, patients, conference call participants, children using their game consoles, etc.).

The nature of the environment directly affects the criticality of the actions performed in a real-time application.

Key concepts in real-time systems

In addition to time constraints, and depending on the field of application, the definitions given to real-time applications include fundamental properties:
<u>predictability of behavior</u> and <u>fault tolerance</u>.

Predictability of behavior: in so-called *time-critical* real-time applications (control of industrial processes or military machinery), **compliance with time constraints is imperative under all circumstances** (including instances of processor and network overload).

The degree of predictability varies from one application to another: while some require absolute predictability, others settle for a fixed threshold below which the quality of the service provided is called into question.

Key concepts in real-time systems

Criticality: a criterion for classifying real-time applications according to severity (cost incurred should time constraints not be respected)

Time fault: should time constraints not be respected
- benign: do not significantly affect the service provided
- catastrophic: loss of human life, financial loss, environmental pollution, etc.

Classification of real-time systems:
− Systems with **critical time constraints** (hard real-time systems): failure to comply with time constraints can entail failures with potentially serious consequences. If the deadline is missed, a catastrophic fault ensues.
− Systems with **strict time constraints** (firm real-time systems): occasional missed deadlines are tolerated. A (benign) fault ensues when a deadline is missed.
− Systems with **flexible time constraints** (soft real-time systems): no fault ensues when a deadline is missed; the result can be used even if delivered after the deadline.

Key concepts: determinism

- Determinism
  - goal to be reached to predict the temporal behavior of the system
    - hard real-time: determine whether all deadlines for all activities will be met
    - soft real-time: determine, for example, what the average delays will be
- Deterministic system
  - Based on the current state of the system, it responds to a given stimulus in a “predictable” way
- Non-deterministic system
  - Based on the current state of the system, it cannot guarantee what action (portion of code) it will execute

Key concepts: accuracy

- Logical correctness
  - Outputs consistent with inputs
  - The system behaves as expected in response to given inputs (data)
- Timeliness
  - Time constraints are met
  - Outputs produced "at the right time"

Key concepts: predictability

- Predictability: ability to detect the future occurrence of a missed deadline and to respond accordingly...
  - either by using preemptive fault-tolerance techniques
  - or by migrating tasks to another site in the distributed system
- This is "THE" feature required of a real-time system, above all else!
  - under a set of assumptions regarding the load (e.g. input frequency) and uncontrollable errors (e.g. bit error rate of the bus), prove that...
  - all constraints (especially deadlines) are respected...
  - at least for critical system tasks

What would you prefer: an on-board computer that triggers the ABS after 1 ms in 99% of cases, or one that triggers the ABS after 10 ms but in 100% of cases?

Key concepts: rapidity and adaptability

- Rapidity: context changes, low-complexity scheduling algorithms, access to data structures, etc.
- Adaptability: deal with online changes (new tasks to execute, loss of network nodes) without compromising the rapidity constraints imposed by the process
- Predictability:
  - assess in advance whether a system will meet its time constraints
  - knowledge of parameters associated with task calculations
    - overall calculation time for each task
    - periodicity and jitter
    - preemption
  - worst-case performance evaluation
  - identify the best scheduling algorithm

Key concepts: Hard/Soft Real-Time

Hard Real-Time

- failure to respect time constraints can result in critical situations
- Hard RTS are often used as control devices in dedicated applications

Soft Real-Time

- the system can "tolerate" certain time-constraint breaches
- certain constraints must still be respected, beyond which the system is rendered unusable (videoconferencing, network gaming)
- Note:
  - the distinction between the two is somewhat vague
  - hard and soft real-time tasks can coexist in a given system, even alongside tasks with no time constraints at all

Key concepts: Hard/Soft Real-Time

- Practical implications
  - Dramatic consequences when the results obtained do not meet certain criteria of logical and—above all—temporal consistency.
  - The appropriate response to a missed deadline depends on the application:
    - The most frequent solution is to opt for a degraded behavior.
    - Obviously, preventing failure by minimizing unpredictable results (particularly from a temporal point of view) is preferable.

Understanding the notion of 'time'

- 'Time' differs from one application to another

- Real-time depends on the point of view
  - Example: "real-time" broadcast of a match

    - from the point of view of the match, stadium lighting must be controlled in real-time (any failure would bring the match to a halt)
    - broadcast failure has no impact on the course of the match itself (although you may miss the goal as it happens!)

- 'Time' can then take many forms:
  - The response time to an event
  - The cycle time of a system
  - The relativity of the concept of time

Response time to an event

- event = time-discontinuous signal

- response time = maximum time the process can accept to obtain a relevant response from the system to any event generated by the process

- Any system must be capable of handling all events generated by the process, while respecting their specific response time.

- Questions will arise regarding the criticality of the event and the notion of relative priority between all events, since a real-time system (no matter how powerful) is incapable of processing all event instances simultaneously.

The system will be real-time if and only if it can process all events in the allotted time.

Relativity of the concept of time

- Designing a real-time system involves:
  - Processing both cyclical and event-driven information, so as to include multimedia data (e.g. the case of the telephone, where a constant data rate must be maintained between remote stations, as well as video with bitrates of several dozen Mbps w/o compression).
  - Accepting all types of data in the same system while respecting temporal and processing criteria for each item of data, knowing that the results of such processing must be accurate and on schedule for the application to work.

-> A rather relative notion!

- If we were to qualify/quantify it, we could say that it is in fact the combination of the system's response time with the workload to perform during said time

Different RTS implementations

- The technologies to implement depend directly on the time performance required

Fonction	Matériel	Système d'exploitation	Temps de réponse
IHM	PC	XP, Linux	>100ms
Contrôle commande global	Station de travail	Lynx OS, HPUX	10ms à 100ms
Fonction TR	Calculateur TR	VxWorks, VRTX	1ms à 10ms (voire 100µs)
Prise en compte Evénement	Calculateur TR	OS + gestion des interruptions	10µs à 100µs
Arcs Réflexes	Composants spécialisés	Machines à états finis	< 10µs

Choices available to the RTS designer

5 axes of freedom

⚠ closely related: modifying one implies adjusting choices made on the others (finding the best compromise)

1. **Architecture**
   – Can cover several levels; we speak of system architecture and hardware architecture *(see 5th year course)*
2. **Communication between processes**
   – Problems such as communication between remote processes running on different processors and using different operating systems
3. **Computer technology**
   – A variety of solutions for computers and/or processors
   – Paradoxically, may pose a number of problems: finding the one(s) best suited to the application's constraints?
4. **Operating systems**
   – A key element
   – Frequent evolutions to keep pace with the life cycle of the various available solutions:
     - Unix and its various versions (LynxOS, HP-UX, etc.)
     - Windows (XP, CE, XP Embedded)
     - Linux and its forks
     - OS9, VxWorks, VRTX, OSE, µCOS, OSEK, etc.
     - Proprietary operating systems
   – Unstable market -> "Should we use libreware or proprietary solutions?"
5. **Languages**
   – Facilitate application development (level of abstraction)
   – Help render real-time applications more accessible
   – Provide code portability

Regarding the choice of OS

- Classic OSs limited for real-time applications
- Mechanisms ill-suited to real-time
  – scheduling policies:
    - aimed at balancing execution time
    - not adapted to tasks that are more critical than others
  – mechanisms for accessing shared resources must be adapted to remove temporal uncertainties (I/O management entails long delays, sometimes unbounded)
  – unoptimized interrupt management
  – need to rethink virtual memory management mechanisms (e.g. swapping)
  – timer control is not fine enough (for many real-time applications)

Why use an RTOS?

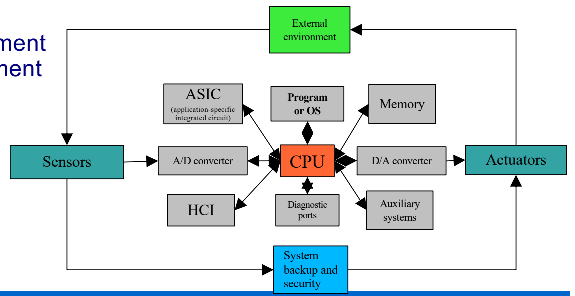
Real-time computer systems involve a combination of software and hardware

Key role of software (among other things): to manage hardware resources efficiently, so as to perform specific tasks within specific time limits

=> RTOS: providing services to application software

These services depend on available hardware resources

Examples:
- Task management
- Synchronization management
- Communication management
- Memory management
- Time management
- Management of peripherals



RTOS main features

To guarantee compliance with time constraints, it is essential that:

- its different services and algorithms used run in bounded time
  - => the RTOS must be deterministic and preemptive, so that the highest priority task can take over during the next tick
  - in this case, the evolution of the RTS can be predicted (in theory)
- the different potential sequences of processing ensure that none of them exceeds its time limit

=> Key feature of an RTOS = predictable response time to an external stimulus

- If a peripheral generates an interrupt, the RTOS must respond by launching the service within a known period of time and regardless of processor load

An OS can be deemed an RTOS when context switching and response time to an interrupt are guaranteed to occur within a 10 μs-period

Real-Time Operating System

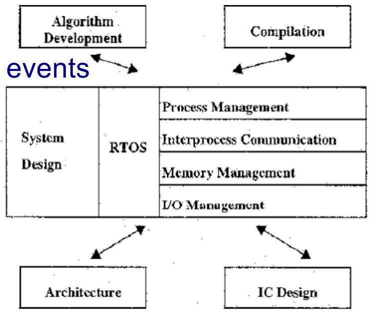
Real-Time Operating System:

- Multitasking operating system for real-time applications
  - limited set of functions determined by the hardware design
  - reacts to input within a specific time period
  - guarantees a certain capability within a specified time constraint
- Characteristics:
  - while facilitating the creation of an RTS, it does not guarantee that the end result will respect real-time constraints
  - doesn't necessarily target performance and speed but providing services and primitives which—if used correctly—can ensure meeting the required deadlines
  - uses specialized schedulers to provide RTS developers with tools and primitives required to achieve the intended real-time behavior in the final system
- Requirements:
  - multitasking
  - priority assignment to processes
  - adequate number of interrupt levels

Real-Time Operating System

4 main roles of an RTOS

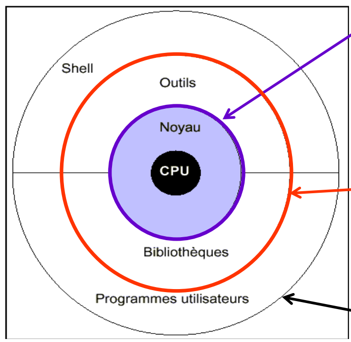
- Process Management
  - Process creation
  - Process loading
  - Process execution control
  - Interaction of the process with signal events
  - Process monitoring
  - CPU allocation
  - Process termination
- Interprocess Communication
  - Synchronization and coordination
  - Deadlock and Livelock detection
  - Process Protection
  - Data Exchange Mechanisms
- Memory Management
  - Services for file creation, deletion, reposition and protection
- Input/Output Management
  - Handles requests and release subroutines for a variety of peripherals and read, write and reposition programs



RTOS structure

Built around:

- a kernel that provides the essential algorithms for scheduling and resource management
- services
  - (file system management, network access, etc.)



The **kernel** controls
- ✓ peripherals
- ✓ program execution (processes)
- ✓ memory allocated to each process
- ✓ process scheduling
- ✓ inter-process communication and synchronization
- ✓ network protocol management
- ✓ small, compact software (less than 10 Kb)
- ⇒essential software for real-time processing:
  - scheduler
  - task management
  - inter-task communication
  - resource allocation, hardware interface, security
  - ⇒ somewhat limited but powerful system

The **executive**
- ✓ built on the kernel
- ✓ includes features such as
  - I/O management
  - enhanced cooperation mechanisms (based on those of the kernel)
  - file system management
  - accessibility (e.g. remote connection)

The operating system
- includes all application programming and fine-tuning tools (compiler, debugger, simulator, analyzer, etc.)

Role of a real-time executive

A real-time executive can be:

event driven: system switches tasks when an event of higher priority needs service

time sharing: system switches tasks on regular clock interrupts and on events

=> Calls to the executive ensue from:

event triggered by the process

the process of receiving and handling the hardware interrupt associated with these events triggers the call to the executive

time

the interrupt regularly generated by the real-time clock on the computer triggers the call to the executive

the tasks themselves

Role of a real-time executive

Roles

schedule task execution

protect access to shared resources

centralizing role; an interface that:

- redirects process events to the corresponding tasks
- triggers task wake-up when waiting for a deadline or start time
- receives and retransmits synchronization signals or data between asynchronous tasks

Provides user-accessible services for diverse tasks

task management: activation, suspension, resumption, forced termination, etc.

hardware event management (interrupts)

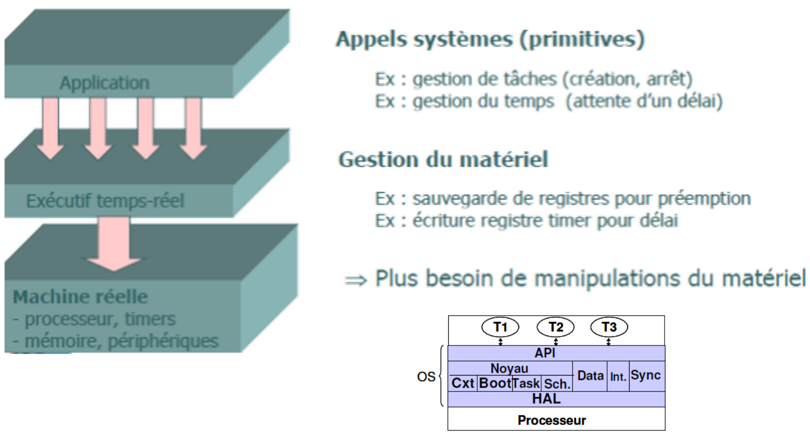
synchronization: sending/receiving of “internal” signals by tasks

=> The executive provides services for generating and monitoring these “software events”

- inter-task message communication (mailboxes, data exchange ports or client-server calls)
- time management: enabling deferred, scheduled or periodic task wake-ups
- management of shared resources, memory, exceptions, etc.

Why use an RTOS?

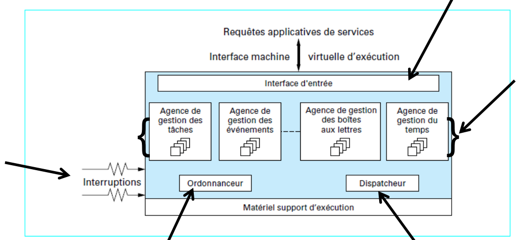
Other uses: portability, application reuse



Source: Thèse de doctorat de Gabriela Nicolescu, Laboratoire TIMA, France, 2002

Structure of a generalized executive

Input interface
- receives application requests
- performs context backups
- redirects requests



Interrupts
- executive services can be called upon either by application tasks or by external requests (clock interrupt, sensor signaling an alarm, network card receiving a message, etc.)
- requests are relayed by a short code sequence (interrupt service routine) which calls upon an executive service

Handlers
- managers of basic executive objects
- each features data structures (tables, lists), in the form of descriptors, which are an abstract representation of the objects used by the application
- cooperation between handlers

Scheduler
- module in charge of managing the critical resource of the processor
- implements a scheduling algorithm based on the notion of static priority (in industrial applications)
- is only called upon by the handlers when necessary

Dispatcher
- acts as a mirror of the input interface
- in charge of reactivating the current task (or one newly chosen by the scheduler) by restoring its execution context, thus terminating the execution of a service