

Introduzione ai sistemi operativi real-time

Seminario su Linux-RT

Giuseppe Lipari

<http://feanor.sssup.it/~lipari>

Evidence S.r.l.

15-16 Dicembre 2005 / Corso su Linux-RT per Ericsson Lab
Italia

Sommario

- 1 Cosa vuol dire real-time
 - Definizioni classiche
 - Hard real-time e Soft real-time
- 2 Confronto OS tradizionali e RTOS
 - Differenze principali
 - Memoria
 - Schedulazione
- 3 Classificazione degli RTOS
 - Classificazione secondo la dimensione
 - Standards
- 4 Linux e il Real-Time
 - Perché Linux e Real-Time?
 - Vantaggi dell'Open Source
 - Classificazione di vari Linux con estensioni real-time

Outline

- 1 Cosa vuol dire real-time
 - Definizioni classiche
 - Hard real-time e Soft real-time
- 2 Confronto OS tradizionali e RTOS
 - Differenze principali
 - Memoria
 - Schedulazione
- 3 Classificazione degli RTOS
 - Classificazione secondo la dimensione
 - Standards
- 4 Linux e il Real-Time
 - Perché Linux e Real-Time?
 - Vantaggi dell'Open Source
 - Classificazione di vari Linux con estensioni real-time

Definizione classica

Secondo la definizione classica,

Un sistema si dice real-time se la correttezza dei suoi output dipende non solo dal valore che questi assumono ma anche dal tempo in cui sono prodotti.

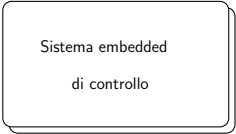
Secondo un'altra definizione più informale

Un sistema real-time è un sistema che produce i suoi output in un tempo predicibile, non necessariamente nella maniera più veloce possibile.

Oltre le definizioni

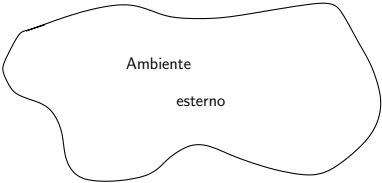
- Che vuol dire **predicibile**?
- Come mai Ericsson Lab Italia usa degli RTOS?
- Quali sono le caratteristiche essenziali di un RTOS?
- Come mai Windows e Linux non sono RTOS?

Sistemi reattivi



Sistema embedded
di controllo

The diagram shows a rounded rectangle with a double-line border, representing a system boundary. Inside, the text 'Sistema embedded di controllo' is centered.

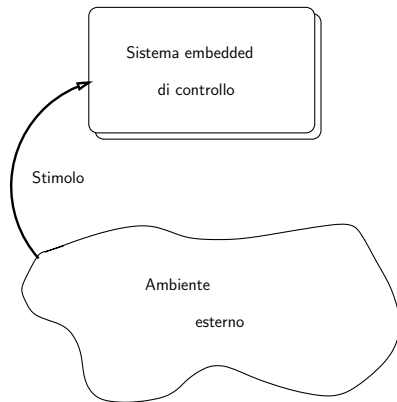


Ambiente
esterno

The diagram shows an irregular, blob-like shape representing an external environment. Inside, the text 'Ambiente esterno' is centered.

Una prima qualità
fondamentale è la **reattività**
agli stimoli esterni.

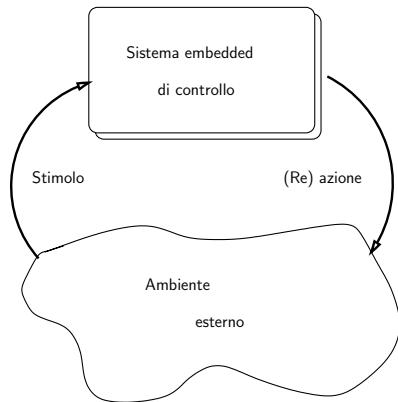
Sistemi reattivi



Una prima qualità fondamentale è la **reattività** agli stimoli esterni.

- Il sistema **reagisce** agli stimoli esterni in tempo utile.

Sistemi reattivi



Una prima qualità fondamentale è la **reattività** agli stimoli esterni.

- Il sistema **reagisce** agli stimoli esterni in tempo utile.
- La *velocità* dipende dalle costanti di tempo dell'ambiente esterno.

Velocità

Comprare un hardware più veloce?

- Un sistema ultra veloce va sempre bene.
- Se avessimo un hardware infinitamente veloce non ci sarebbero problemi di real-time!
- Indipendentemente dal sistema operativo!

Velocità

Comprare un hardware più veloce?

- Un sistema ultra veloce va sempre bene.
- Se avessimo un hardware infinitamente veloce non ci sarebbero problemi di real-time!
- Indipendentemente dal sistema operativo!

Problemi?

Un sistema *troppo veloce* può portare altri problemi . . .

- Costo
- Surriscaldamento
- Fault-robustness

Ottimizzare l'hardware

Il problema del **costo** può essere più o meno importante.

- dipende dal tipo di prodotto

Se si riesce a fare tutto con un sistema *piccolo*, il costo per unità si riduce e l'azienda è più contenta.

A parità di hardware

- Utilizzare un sistema operativo o un altro può fare la differenza in termini di prestazioni.

A che servono le tecniche RT?

Metodologia Real-Time

- Metodologie di progettazione
- Metodologie di programmazione
- Analisi

A che servono le tecniche RT?

Metodologia Real-Time

- Metodologie di progettazione
- Metodologie di programmazione
- Analisi

Opportunità

Tali tecniche:

- consentono soprattutto di risparmiare sul costo di sviluppo,
- perchè ottimizzano l'uso dell'HW per l'applicazione (con i requisiti temporali richiesti)

Sistema reattivo

Ma cosa deve fare di solito un sistema real-time (reattivo)?

- 1 Rispondere a vari stimoli esterni;
- 2 Tali stimoli arrivano con intervalli temporali diversi;
- 3 Ogni stimolo richiede un'azione diversa ...
- 4 ... con tempi di risposta diversi;
- 5 In generale, gli stimoli e le reazioni possono accavallarsi.

Problemi da risolvere

Alcuni dei problemi nella progettazione/programmazione real-time:

1. Trovare una **schedulazione** (cioè a “impacchettare” opportunamente tutte le cose da fare), in modo che ciascuna cosa sia terminata in tempo utile (**analisi di schedulabilità**);

Problemi da risolvere

Alcuni dei problemi nella progettazione/programmazione real-time:

- 1 Trovare una **schedulazione** (cioè a “impacchettare” opportunamente tutte le cose da fare), in modo che ciascuna cosa sia terminata in tempo utile (**analisi di schedulabilità**);
- 2 Assegnare i parametri di schedulazione (tipicamente, le priorità dei thread)

Problemi da risolvere

Alcuni dei problemi nella progettazione/programmazione real-time:

- 1 Trovare una **schedulazione** (cioè a “impacchettare” opportunamente tutte le cose da fare), in modo che ciascuna cosa sia terminata in tempo utile (**analisi di schedulabilità**);
- 2 Assegnare i parametri di schedulazione (tipicamente, le priorità dei thread)
- 3 Quanti thread usare? come strutturare l'applicazione?

Problemi da risolvere

Alcuni dei problemi nella progettazione/programmazione real-time:

- 1 Trovare una **schedulazione** (cioè a “impacchettare” opportunamente tutte le cose da fare), in modo che ciascun cosa sia terminata in tempo utile (**analisi di schedulabilità**);
- 2 Assegnare i parametri di schedulazione (tipicamente, le priorità dei thread)
- 3 Quanti thread usare? come strutturare l'applicazione?
- 4 Qual'è l'hw meno costoso per il quale il problema 1 è risolto? (**ottimizzazione delle risorse**);

Problemi da risolvere

Alcuni dei problemi nella progettazione/programmazione real-time:

- 1 Trovare una **schedulazione** (cioè a “impacchettare” opportunamente tutte le cose da fare), in modo che ciascun cosa sia terminata in tempo utile (**analisi di schedulabilità**);
- 2 Assegnare i parametri di schedulazione (tipicamente, le priorità dei thread)
- 3 Quanti thread usare? come strutturare l'applicazione?
- 4 Qual'è l'hw meno costoso per il quale il problema 1 è risolto? (**ottimizzazione delle risorse**);
- 5 Cosa succede in fase di “overload” temporaneo (cioè quando ho troppe cose da fare per un evento non completamente inatteso); e come posso gestire nella maniera migliore possibile tali situazioni?

Outline

- 1 Cosa vuol dire real-time
 - Definizioni classiche
 - **Hard real-time e Soft real-time**
- 2 Confronto OS tradizionali e RTOS
 - Differenze principali
 - Memoria
 - Schedulazione
- 3 Classificazione degli RTOS
 - Classificazione secondo la dimensione
 - Standards
- 4 Linux e il Real-Time
 - Perché Linux e Real-Time?
 - Vantaggi dell'Open Source
 - Classificazione di vari Linux con estensioni real-time

Definizione di hard e soft

Partiamo dalla definizione classica:

*un sistema si dice **hard real-time** se il non-rispetto di un vincolo temporale ha conseguenze catastrofiche che in certi casi possono mettere in pericolo vite umane;*

*un sistema si dice **soft real-time** se il non rispetto dei vincoli temporali non provoca conseguenze catastrofiche, ma riduce la qualità del servizio offerta dall'applicazione.*

Hard-Real Time

In realtà, le definizioni precedenti sono un po' imprecise.
Nei sistemi hard real-time

- I vincoli di tempo sono *parte integrante* della funzionalità del sistema
- Se una deadline non è rispettata, il sistema non funziona almeno per un periodo non trascurabile di tempo.
- Un dato in ritardo è un dato scorretto!

Hard-Real Time

In realtà, le definizioni precedenti sono un po' imprecise.

Nei sistemi hard real-time

- I vincoli di tempo sono *parte integrante* della funzionalità del sistema
- Se una deadline non è rispettata, il sistema non funziona almeno per un periodo non trascurabile di tempo.
- Un dato in ritardo è un dato scorretto!
- Il coinvolgimento di vite umane in realtà ha a che fare con l'essere **safety critical** (non con l'essere *hard real-time*)

Hard-Real Time

In realtà, le definizioni precedenti sono un po' imprecise.
Nei sistemi hard real-time

- I vincoli di tempo sono *parte integrante* della funzionalità del sistema
- Se una deadline non è rispettata, il sistema non funziona almeno per un periodo non trascurabile di tempo.
- Un dato in ritardo è un dato scorretto!
- Il coinvolgimento di vite umane in realtà ha a che fare con l'essere **safety critical** (non con l'essere *hard real-time*)
- Pensate per esempio all'automotive. Se il sistema ABS emette un dato scorretto (in ritardo o no), è in pericolo la vita degli occupanti la macchina.

Soft real-time

Una definizione precisa è molto più difficile.

Tipicamente:

- A ogni attività vengono assegnati dei *target temporali* desiderati;
- Il sistema deve, *per quanto possibile*, rispettare tali target;
- Se qualche volta non ci riesce, il sistema è costruito in modo da funzionare lo stesso;
- Il sistema funziona tanto meglio, quanti più target riesce a rispettare.

Esempio 1: sistema home-theater

Un moderno sistema home theater può essere assimilato a un sistema soft real-time;

- Quasi tutti i segnali sono ormai in digitale;
- Processing di più stream audio-video;
- Funzioni sofisticate (es. picture-in-picture);
- Interfaccia utente complessa.

Target temporali:

- Sul frame-rate;
- Sul delay;

Vincoli ulteriori:

- Costo, calore prodotto, rumorosità;

Tipiche applicazioni hard real-time

Fino a qualche anno fa:

- Robotica avanzata
- Sistemi aerospaziali

Tipiche applicazioni hard real-time

Fino a qualche anno fa:

- Robotica avanzata
- Sistemi aerospaziali
- Di solito roba prodotta in pochi pezzi → era possibile usare hardware *dedicato* (e costoso)

Tipiche applicazioni hard real-time

Fino a qualche anno fa:

- Robotica avanzata
- Sistemi aerospaziali
- Di solito roba prodotta in pochi pezzi → era possibile usare hardware *dedicato* (e costoso)

Recentemente:

- Applicazioni automotive;
- Sistemi embedded;

Tipiche applicazioni hard real-time

Fino a qualche anno fa:

- Robotica avanzata
- Sistemi aerospaziali
- Di solito roba prodotta in pochi pezzi → era possibile usare hardware *dedicato* (e costoso)

Recentemente:

- Applicazioni automotive;
- Sistemi embedded;
- Prodotti di larghissimo consumo, milioni di pezzi;
- Obiettivo principale: contenimento del costo
- Ma i requisiti safety critical sono rimasti!

Tipiche applicazioni soft real-time

- Multimedia / Consumer electronics
- Telecomunicazioni

Tipiche applicazioni soft real-time

- Multimedia / Consumer electronics
- Telecomunicazioni
- Non sempre il costo è molto importante (dipende dal numero di pezzi)
- Non sempre un RTOS viene visto come valore aggiunto necessario!

Outline

- 1 Cosa vuol dire real-time
 - Definizioni classiche
 - Hard real-time e Soft real-time
- 2 Confronto OS tradizionali e RTOS
 - Differenze principali
 - Memoria
 - Schedulazione
- 3 Classificazione degli RTOS
 - Classificazione secondo la dimensione
 - Standards
- 4 Linux e il Real-Time
 - Perché Linux e Real-Time?
 - Vantaggi dell'Open Source
 - Classificazione di vari Linux con estensioni real-time

Confronto tra GPOS e RTOS

Che c'è di differente tra un RTOS e un General Purpose Operating System (GPOS)?

- Lo scopo: non esiste un sistema operativo universale!
- Prestazioni temporali;
- Organizzazione della memoria;
- Meccanismi di protezione.

In genere: gli RTOS sono OS più *leggeri*, che garantiscono certe prestazioni.

Prestazioni temporali

RTOS:

- Pensato per essere veloce e predicibile
- garantisce tempo di risposta alle interruzioni limitato e piccolo

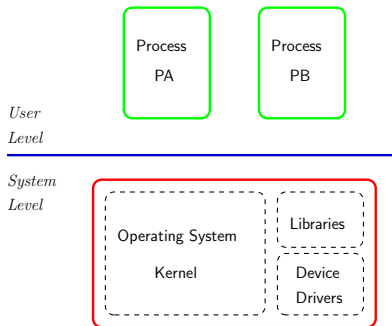
GPOS:

- Pensato per ottimizzare il throughput
- e per fornire un gran numero di servizi
- le varie attività possono interferire l'una con l'altra
- ad esempio, driver che disabilitano le interruzioni per lungo tempo
- sezioni del GPOS non-preemptive

Outline

- 1 Cosa vuol dire real-time
 - Definizioni classiche
 - Hard real-time e Soft real-time
- 2 Confronto OS tradizionali e RTOS
 - Differenze principali
 - **Memoria**
 - Schedulazione
- 3 Classificazione degli RTOS
 - Classificazione secondo la dimensione
 - Standards
- 4 Linux e il Real-Time
 - Perché Linux e Real-Time?
 - Vantaggi dell'Open Source
 - Classificazione di vari Linux con estensioni real-time

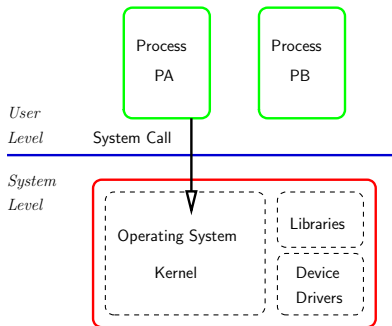
Modello di memoria di un GPOS



I GPOS forniscono la **protezione** della memoria.

- Almeno 2 livelli di privilegio del processore
- le chiamate di SO (syscall) fanno uno switch di livello di privilegio
- il meccanismo di switch su certe architetture (es. i386) è molto costoso

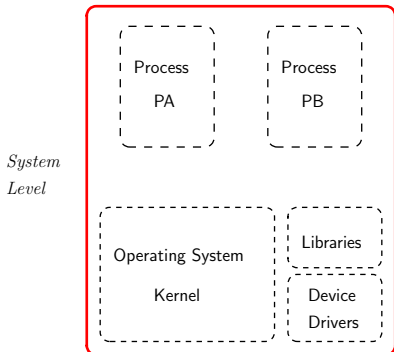
Modello di memoria di un GPOS



I GPOS forniscono la **protezione** della memoria.

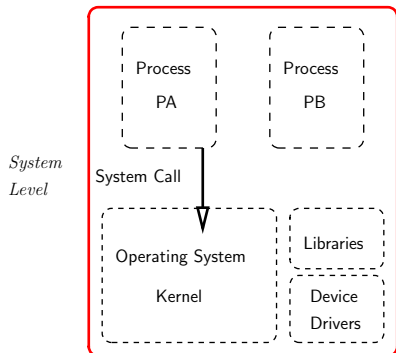
- Almeno 2 livelli di privilegio del processore
- le chiamate di SO (syscall) fanno uno switch di livello di privilegio
- il meccanismo di switch su certe architetture (es. i386) è molto costoso

Modello di memoria di un RTOS



- La maggior parte degli RTOS non forniscono protezione della memoria
- RTOS e applicazioni girano a livello di privilegio massimo
- L'applicazione è compilata e linkata *insieme* al RTOS
- Una syscall è equivalente a una function call → molto veloce

Modello di memoria di un RTOS



- La maggior parte degli RTOS non forniscono protezione della memoria
- RTOS e applicazioni girano a livello di privilegio massimo
- L'applicazione è compilata e linkata *insieme* al RTOS
- Una syscall è equivalente a una function call → molto veloce

Assenza di memory protection negli RTOS

- Le applicazioni embedded sono dedicate
- programmate di solito da un unico team di sviluppo
- tutti i processi cooperano a un certo obiettivo
- la memory protection è un impiccio!

Assenza di memory protection negli RTOS

- Le applicazioni embedded sono dedicate
- programmate di solito da un unico team di sviluppo
- tutti i processi cooperano a un certo obiettivo
- la memory protection è un impiccio!

Con l'avvento di sistemi sempre più complessi...

- Si integrano pezzi di SW prodotti da aziende diverse;
- Per questioni di fault-tolerance e fault-confinement, si sta re-introducendo la protezione della memoria!

Memoria Virtuale

Il meccanismo di memoria virtuale è usato nei GPOS per

- Virtualizzare la memoria di ogni processo
- Estendere la memoria RAM del sistema con un memoria di massa
- Richiede meccanismi hardware di supporto (i.e. la MMU)

Negli RTOS non è implementato perché

- Altamente imprevedibile
- Molti processori embedded non hanno la MMU
- Molto spesso non c'è memoria di massa.

Occupazione di memoria

Un GPOS è molto complesso:

- Occupa molta memoria, perché fornisce tanti servizi;
- La struttura è monolitica (ovvero un unico pezzo di codice)
- Anche se molti servizi sono inutili in un sistema embedded, non è possibile eliminarli in maniera semplice (per dipendenze interne)
- Ad esempio, in Windows è impossibile liberarsi dell'interfaccia grafica!

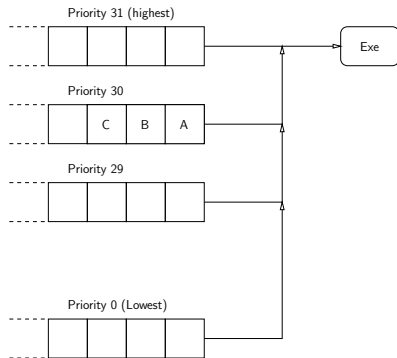
Un RTOS:

- E' molto più semplice
- E' stato scritto dal principio per essere piccolo e ottimizzato.

Outline

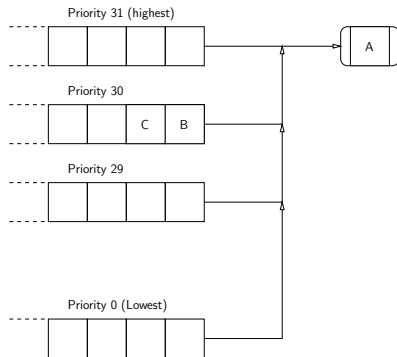
- 1 Cosa vuol dire real-time
 - Definizioni classiche
 - Hard real-time e Soft real-time
- 2 Confronto OS tradizionali e RTOS
 - Differenze principali
 - Memoria
 - **Schedulazione**
- 3 Classificazione degli RTOS
 - Classificazione secondo la dimensione
 - Standards
- 4 Linux e il Real-Time
 - Perché Linux e Real-Time?
 - Vantaggi dell'Open Source
 - Classificazione di vari Linux con estensioni real-time

Meccanismi di schedulazione



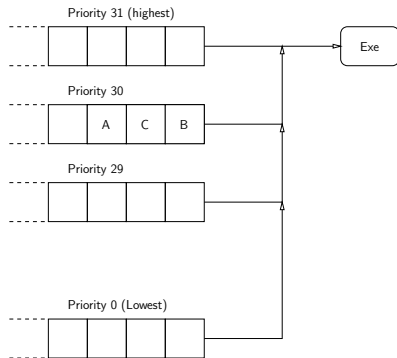
- Tutti (o quasi) gli OS in commercio supportano la schedulazione Fixed Priority con Round Robin (SCHED_RR in POSIX).
- Quindi questo non è un elemento di distinzione!
- (Nonostante quello che c'è scritto sui libri di RT).

Meccanismi di schedulazione



- Tutti (o quasi) gli OS in commercio supportano la schedulazione Fixed Priority con Round Robin (SCHED_RR in POSIX).
- Quindi questo non è un elemento di distinzione!
- (Nonostante quello che c'è scritto sui libri di RT).

Meccanismi di schedulazione



- Tutti (o quasi) gli OS in commercio supportano la schedulazione Fixed Priority con Round Robin (SCHED_RR in POSIX).
- Quindi questo non è un elemento di distinzione!
- (Nonostante quello che c'è scritto sui libri di RT).

Outline

- 1 Cosa vuol dire real-time
 - Definizioni classiche
 - Hard real-time e Soft real-time
- 2 Confronto OS tradizionali e RTOS
 - Differenze principali
 - Memoria
 - Schedulazione
- 3 **Classificazione degli RTOS**
 - **Classificazione secondo la dimensione**
 - Standards
- 4 Linux e il Real-Time
 - Perché Linux e Real-Time?
 - Vantaggi dell'Open Source
 - Classificazione di vari Linux con estensioni real-time

Classificazione degli RTOS

Gli RTOS non sono tutti uguali!

- Sistemi minimali (nanokernel)
- Sistemi intermedi (RTOS classici)
- Sistemi grandi

Sistemi minimali

Caratteristiche:

- Occupano pochissimi byte di memoria (< 5 kb)
- offrono poco più che uno scheduling a priorità fisse
- Task, variabili, risorse del sistema, sono dichiarati staticamente
- pochissimi meccanismi di sincronizzazione (es. niente semafori!)
- Niente librerie

Ambiti applicativi:

- Automotive, sensor networks, piccoli controllori (8 - 16 bit)

Standard:

- OSEK

Sistemi intermedi

Caratteristiche:

- Media occupazione di memoria (50 - 300 Kb)
- Supporto per i thread
- Librerie standard
- Stack di protocolli
- Meccanismi di sincronizzazione elaborati

Ambiti applicativi

- Tantissimi, (telecom, robotica, controllo industriale, aerospaziale)

Standard

- POSIX (profilo minimale), μ -Itron

Sistemi grandi

Caratteristiche:

- Occupano un bel po di memoria (> 500 Kb)
- Supporto per processi e thread
- Protezione della memoria
- Vari tipi di scheduling
- Partizionamento temporale, per supportare diverse
- Supporto per fault-tolerance

Ambiti applicativi

- Sistemi di controllo safety critical (aerospaziale, nucleare, etc.)

Esempi:

- VxWorks AE

Outline

- 1 Cosa vuol dire real-time
 - Definizioni classiche
 - Hard real-time e Soft real-time
- 2 Confronto OS tradizionali e RTOS
 - Differenze principali
 - Memoria
 - Schedulazione
- 3 **Classificazione degli RTOS**
 - Classificazione secondo la dimensione
 - **Standards**
- 4 Linux e il Real-Time
 - Perché Linux e Real-Time?
 - Vantaggi dell'Open Source
 - Classificazione di vari Linux con estensioni real-time

RTOS standards

Alcuni standard di RTOS

- RT-POSIX
- OSEK – automotive industry
- APEX – large avionics systems
- μ -ITRON – embedded systems, mainly Japan

POSIX

POSIX = Portable Operating System Interfaces based on the UNIX operating system

- **goal**: application portability at the source code level
- Contains:
 - Base standard – C-like interfaces to operating system services
 - Profiles – Subset of the Base standard adapted to different operating system services
 - Bindings – Interfaces to other languages (ADA, Fortran 77)

POSIX Base Standard

- standard C library
- process and thread primitives, scheduling
- file and I/O primitives
- synchronization (semaphores, mutex, condition variables, message queues)
- shared memory
- signals and timers

POSIX Real-Time Extensions

- Fixed Priority Scheduling
- FIFO or Round Robin priorities
- Sporadic Server
- High resolution timers
- Virtual Memory Management

POSIX.13 Real-Time profiles

- PSE51 minimal realtime system profile (target: small embedded systems)
 - no file system
 - no memory protection
 - monprocess multithread kernel
- PSE52 realtime controller system profile (target: robot controllers, ...)
 - PSE51 + file system + asynchronous I/O
- PSE53 dedicated realtime system profile (target: avionics)
 - PSE52 + process support and memory protection
- PSE54 multi-purpose realtime system profile (basically the entire POSIX standard)
 - PSE53 + file system + asynchronous I/O

OSEK/VDX

It is a standard for an open-ended architecture for distributed control units in vehicles

The name:

- **OSEK**: Offene Systeme und deren Schnittstellen für die Elektronik im Kraft-fahrzeug (Open systems and the corresponding interfaces for automotive electronics)
- **VDX**: Vehicle Distributed eXecutive (another french proposal of API similar to OSEK) OSEK/VDX is the interface resulted from the merge of the two projects

<http://www.osek-vdx.org>

OSEK – Objectives

- Portability and reusability of the application software
- Specification of abstract interfaces for RTOS and network management
- Specification independent from the HW/network details
- Scalability between different requirements to adapt to particular application needs
- Verification of functionality and implementation using a standardized certification process

OSEK – Advantages

- clear savings in costs and development time.
- enhanced quality of the software
- creation of a market of uniform competitors
- independence from the implementation and standardised interfacing features for control units with different architectural designs
- intelligent usage of the hardware present on the vehicle
- for example, using a vehicle network the ABS controller could give a speed feedback to the powertrain microcontroller

Static is Better

Everything is specified before the system runs

-
- Static approach to system configuration
- No dynamic allocation on memory
- No dynamic creation of tasks
- No flexibility in the specification of the constraints

Custom languages that helps off-line configuration of the system

- **OIL**: parameters specification (tasks, resources, stacks)
- **KOIL** kernel aware debugging

OSEK – documents

The OSEK/VDX consortium packs its standards in different documents

- OSEK OS operating system
- OSEK Time time triggered operating system
- OSEK COM communication services
- OSEK FTCOM fault tolerant communication
- OSEK NM network management
- OSEK OIL kernel configuration
- OSEK ORTI kernel awareness for debuggers

OSEK – Conformance Classes

- OSEK OS should be scalable with the application needs
 - different applications require different services
 - the system services are mapped in Conformance Classes
 - a conformance class is a subset of the OSEK OS standard
- objectives of the conformance classes
 - allow partial implementation of the standard
 - allow an upgrade path between classes
- services that discriminates the different conformance classes
 - multiple requests of task activations
 - task types
 - number of tasks per priority
- There are four conformance classes: BCC1, BCC2, ECC1, ECC2

Typical OSEK/VDX services

- Fixed Priority scheduling with Immediate Priority Ceiling and Preemption Groups
- Task handling (Activation, self-termination)
- Events (blocking wait for synchronization)
- ISR and IRQ disabling handled in the API
- Counters and Alarms to implement periodic wakeups and timeouts
- Application modes
- Centralized Error Handling and OS Hooks

- the firmware is mainly (90%) composed by device drivers
 - devices change for each microcontroller
 - difficult to standardize (only the RTOS is simple to standardize)
 - recently, some car makers agreed on a generic interface for device drivers
- HIS working group (<http://www.automotive-his.de>)
 - it is NOT a standard, but a working group
 - design a configurable interface for device drivers
 - device drivers are templates
 - a customized version of the device drivers is produced for each application, using a configuration language called DIL

AUTOSAR

Next generation automotive standard

(<http://www.autosar.de>).

Technical goals

- modularity
- scalability
- transferability
- re-usability of functions

AUTOSAR:

- provides a common software infrastructure for automotive systems of all vehicle domains
- system-wide and configuration process optimization to meet the runtime requirements of specific devices and hardware constraints
- Applicative domains: Powertrain, Chassis, Safety (active and passive), Multimedia/telematics, Body/comfort, Man-Machine Interface

APEX (ARNIC)

- Standard for an operating system interface for Avionic Systems.
- Idea: old systems made by a set of communicating "black box" subsystems
- The ARNIC consortium then proposed the IMA (Integrated Modular Avionics) Project to integrate different avionics software to save physical resources.
- APEX is the Operating System Interface for IMA Applications

Typical example:

- distributed multiprocessor architecture with shared memory and network communications
- APEX has been used in critical systems within the Boeing 777

APEX – Firewalling facilities

- Physical Memory is subdivided into partitions
- Each subsystem in a different partition
- APEX schedules partitions using a cyclic schedule
- partitions are temporally isolated from each other
- processes in a partition cannot use more time than expected
- processes inside a partitions are scheduled under a fixed priority basis
- a process deadline miss that happens when the process' partition is not in execution are detected at the next partition time slice

APEX – Message passing

- Processes in a partition can communicate with processes in other partitions using message passing
- Physical channels established at initialization time
- Processes communicate using Ports, that are connected to channels
- Two forms of message-passing over ports:
 - Sampling messages – single slot message buffer, write overwrites, read non destructive
 - Queuing Messages – FIFO-queued message passing with wait
- Other communication mechanisms available as well
 - conventional buffers, semaphores, events

The ITRON (Industrial TRON: "The Real-Time Operating system Nucleus") started in 1984 in Japan, together with the TRON Consortium head by Prof. Ken Sakamura.

ITRON is

- an architecture for real-time operating systems used to build embedded systems
- designed to work with 8, 16, and 32 bit embedded systems, 10–20Kb footprint
- used in many environments (audio/visual equipment, home appliances, personal information appliances, entertainment, PC peripherals, office equipments, communication equipment, transportation, industrial control, and others)

Outline

- 1 Cosa vuol dire real-time
 - Definizioni classiche
 - Hard real-time e Soft real-time
- 2 Confronto OS tradizionali e RTOS
 - Differenze principali
 - Memoria
 - Schedulazione
- 3 Classificazione degli RTOS
 - Classificazione secondo la dimensione
 - Standards
- 4 Linux e il Real-Time
 - **Perché Linux e Real-Time?**
 - Vantaggi dell'Open Source
 - Classificazione di vari Linux con estensioni real-time

Linux e il Real-Time

- Linux nasce come GPOS
- E' un sistema **monolitico**, quindi poco modulare
- E' però un sistema **aperto** e **Open Source**

Linux e il Real-Time

- Linux nasce come GPOS
- E' un sistema **monolitico**, quindi poco modulare
- E' però un sistema **aperto** e **Open Source**
- Quindi, tutti possono modificarlo per i propri scopi
- La base di sviluppatori è enorme!

Outline

- 1 Cosa vuol dire real-time
 - Definizioni classiche
 - Hard real-time e Soft real-time
- 2 Confronto OS tradizionali e RTOS
 - Differenze principali
 - Memoria
 - Schedulazione
- 3 Classificazione degli RTOS
 - Classificazione secondo la dimensione
 - Standards
- 4 Linux e il Real-Time
 - Perché Linux e Real-Time?
 - Vantaggi dell'Open Source
 - Classificazione di vari Linux con estensioni real-time

Perché Linux per le aziende

Linux è interessante per le aziende perché:

Linux è un sistema non proprietario

Difetti dei sistemi proprietari:

- Se fallisce la ditta di software fornitrice, non c'è più nessuno in grado di supportarvi.
- Siete legati a vita alla ditta fornitrice!
- La quale può cercare di tirare la corda (alzando i prezzi al massimo consentibile)
- Il software proprietario *non favorisce la concorrenza*, a meno che non ci siano standard forti.

Difficoltà nel fare il porting

Se anche avete un sistema proprietario, e volete cambiare:

- Fare il porting della base di programmi non è affatto semplice
- Oltre la API del RTOS (che potrebbe anche essere std), ci sono i device driver!
- E se avete un sistema proprietario, è **oggettivamente** difficile trovare supporto al di fuori della ditta fornitrice.

Vantaggi dell'Open Source

Se invece avete un sistema *non proprietario*, come Linux

- Ci sono milioni di sviluppatori nel mondo
- Avete in mano tutto il codice sorgente
- E' molto più facile trovare chi vi supporta vicino a casa vostra

Domanda fondamentale: e la licenza ???

Più avanti nel corso:

Tutto quello che avreste voluto sapere sull'open source e le licenze e non avete mai osato chiedere.

Perché Linux

Ci sono tanti RTOS open source (eCos, RTEMS, Shark OS, ...), ma

- Linux è il più diffuso, lo conoscono tutti
- qualcuno è riuscito a ridurre Linux abbastanza da farlo diventare “embedded” (vedi μ -cLinux), e anche in un orologio!
- Linux è standard Unix, perfettamente compliant con POSIX
- Sono state proposte estensioni real-time di Linux (le vedremo dopo)
- Linux ha un parco programmi invidiabile.
- Linux è disponibile su un numero di processori veramente incredibile

Outline

- 1 Cosa vuol dire real-time
 - Definizioni classiche
 - Hard real-time e Soft real-time
- 2 Confronto OS tradizionali e RTOS
 - Differenze principali
 - Memoria
 - Schedulazione
- 3 Classificazione degli RTOS
 - Classificazione secondo la dimensione
 - Standards
- 4 Linux e il Real-Time
 - Perché Linux e Real-Time?
 - Vantaggi dell'Open Source
 - Classificazione di vari Linux con estensioni real-time

Primo tentativo di classificazione dei Linux-RT

Innanzitutto, distinguiamo Linux-Embedded e Linux-RT.

Linux-Embedded

- Gli sforzi sono indirizzati a ridurre l'occupazione di memoria di un Linux *minimale*.
- Linux gira anche su macchine senza MMU (μ cLinux).
TBD: ci vorrebbero degli esempi di processori! Guardare il sito ucLinux.

Linux-RT

- Gli sforzi sono indirizzati a ridurre la latenza delle attività *Real-Time*.
- Ne esistono vari specie, che andremo a classificare.

Approcci a Linux-RT

- RT-Linux (Yodaiken and Barabanov, FSMLabs), e RTAI prima maniera
- RTAI + Adeos
- Xenomai (già RTAI + fusion)
- Real-Time in user space (Linux/RK, Linux+reservations, RT-patch)