

# Introduzione ai Sistemi Operativi Real-Time

Alessandro Fantechi

*Dipartimento di Sistemi e Informatica  
Università degli Studi di Firenze*

AA. 2003-04

## 5. Sistemi Operativi Real Time (RTOS)

- Hard real time
- Tasks: Periodic vs. Aperiodic
- Sistemi embedded, programma in ROM
- No RTOS, solo periodici
- No RTOS, periodici e aperiodici
- Concetto di deadline
- WCET
- Utilizzo dei processi

## Sistemi tempo-reale

- **Soft real-time:**
  - I vincoli sui tempi di risposta non sono stringenti, perché in qualche caso si può non rispettarli
  - Esempio: i programmi di visualizzazione di un DVD devono estrarre i dati e processarli in modo da rispettare i vincoli di un frame ogni 50-esimo di secondo. Ma in casi di computazioni particolarmente complesse (ad es. compensazione di errori, o impegno della CPU in un altro compito, se i vincoli non sono rispettati si produce solamente un peggioramento della qualità dell'immagine.
- **Hard real-time:**
  - I vincoli sui tempi di risposta sono stringenti, perché se non vengono rispettati il sistema di controllo è inutile, o addirittura pericoloso
  - Esempio: programmi di controllo delle superfici di volo di un aereo

3

A. Fantechi

## Sistemi tempo-reale

- **Soft real-time:**
  - I processi critici hanno priorità sugli altri processi e li mantengono fino al completamento dell'esecuzione.
  - Supportato dai SO general purpose nelle applicazioni che richiedono caratteristiche avanzate del SO (multimedia, realtà virtuale).
- **Hard real-time:**
  - Memoria secondaria spesso limitata o totalmente assente, dati memorizzati in memorie volatili o di sola lettura (ROM). → Sistemi embedded
  - Non utilizzano il time-sharing. Le funzionalità hard real-time non sono supportate dai SO general purpose.

4

A. Fantechi

## Task periodici e aperiodici

- I compiti (tasks) di un sistema real-time si distinguono in **periodici** e **aperiodici**
- **Def.:** I task che devono essere eseguiti ogni  $p$  unità di tempo sono detti **periodici** (si periodo  $p$ ).
  - (es. tasks che eseguono un algoritmo di controllo andando a leggere periodicamente dei sensori e comandando degli attuatori)
- Tutti gli altri tasks sono detti **aperiodici** .
- Tasks che richiedono il processore in modo imprevedibile sono detti **sporadici** (se c'è una separazione minima tra due invocazioni)
  - (es. tasks che rispondo a richieste di un utente umano)

5

A. Fantechi

## Concetto di deadline

- Deadline:  
***istante di tempo dopo il quale la computazione non è semplicemente in ritardo, ma è errata.***

Le deadlines sono vincoli di tempo stringenti imposti ai task dall'ambiente esterno.

Ad esempio, se un sistema di controllo di un processo industriale richiede che vengano controllati dei sensori ogni  $p$  unità di tempo, il task relativo dovrà essere in grado di portare a termine il controllo entro  $p$  unità di tempo, indipendentemente da quali sono gli altri task del sistema.

6

A. Fantechi

## Sistemi embedded senza RTOS

1. Caso più semplice: unico task periodico, ripetuto in continuazione
  - All'accensione, il sistema esegue direttamente l'algoritmo di controllo, scritto a partire dalla prima posizione in memoria (ROM), che consiste in un ciclo perpetuo che esegue il task
  - Le deadline sono verificate se l'esecuzione del task è "sufficientemente veloce"
2. Altro caso semplice: unico task periodico, invocato a intervalli regolari da un timer
  - Il timer lancia un'interruzione, il task è realizzato dalla routine di interruzione relativa. La CPU altrimenti è inattiva, cioè cicla su NOP
  - Le deadline sono verificate se l'esecuzione del task è "sufficientemente veloce", l'interruzione non interrompe la normale esecuzione e le deadline sono verificate.

7

A. Fantechi

## Sistemi embedded senza RTOS (o con RTOS primitivo - *Cyclic executive*)

3. Caso più complesso: più task periodici, di ugual periodo
  - Come caso 2, ad ogni interruzione del timer vengono eseguiti in sequenza tutti i tasks
  - Le deadline sono verificate se la somma dei tempi di esecuzione dei task è inferiore alla deadline.
  - Se i periodi non sono uguali, ma multipli tra loro, le interruzioni seguono il periodo minore e ad ogni interruzione si sceglie quali task eseguire
4. Caso ancora più complesso: più task periodici, di periodo diverso, e task sporadici.
  - Le interruzioni da timer arrivano secondo i vari periodi, e vi sono anche le interruzioni relative agli eventi che scatenano i task sporadici. Ogni interruzione è gestita da una propria routine che esegue il task relativo.
  - Le verifica delle deadline diventa difficile

8

A. Fantechi

## Predicibilità

- Finora abbiamo considerato tasks indipendenti, ma possono esistere delle relazioni di dipendenza tra task che modificano la situazione (ad es. un task deve precedere un altro)
- Problema della *predicibilità* del tempo di esecuzione dei task: l'algoritmo può contenere salti o cicli che rendono non facile il calcolo del tempo di esecuzione.
- Per calcolare il tempo di esecuzione occorre conoscere i tempi di esecuzione delle istruzioni macchina

9

A. Fantechi

## Worst case execution time

**Def .: Worst Case Execution Time (WCET):**

**Limite superiore al tempo di esecuzione del task**

- Il calcolo del WCET è in generale indecidibile
- per programmi ristretti: semplice per architetture “vecchie” ,
- molto complesso per architetture moderne con pipeline, cache, memoria virtuale, etc.
- richiede l'analisi del codice macchina.
- strumenti automatici per il calcolo del WCET

10

A. Fantechi

## Time- triggered systems

- In un sistema **time-triggered**, il sequenziamento temporale dei vari tasks viene stabilito **a priori** da strumenti off-line e memorizzato in una **Task - Descriptor List (TDL)** che contiene lo scheduling ciclico per tutti i task, considerando anche le relazioni di dipendenza
- Un dispatcher viene attivato da un clock sincrono, e invoca le azioni che sono pianificate per l'istante corrente nella TDL
- Per quanto riguarda i task sporadici, si stabilisce di controllare ogni tanto (e quindi di inserire nella TDL tale controllo) se l'evento scatenante è accaduto: in pratica, si trattano i task sporadici come periodici (non sempre questo trattamento è soddisfacente)

11

A. Fantechi

## RTOS

- Finora abbiamo considerato casi in cui non vi è un vero e proprio Sistema Operativo, ma solo un programma sequenziatore (cyclic executive) che, basandosi sul timer, invoca i vari task secondo un ordine prefissato a priori.
- L'alternativa è quella di avere un Sistema Operativo, e di realizzare ogni **task** come un **processo**, che viene messo nella coda dei processi pronti non appena sono valide le condizioni necessarie per la sua esecuzione (partenza periodo per i task periodici, evento scatenante per quelli sporadici, etc...)
- E' quindi lo scheduler che si deve occupare di scegliere quali task mandare in esecuzione, usando particolari algoritmi che mirano alla predicibilità (Earliest Deadline First, Rate Monotonic..)
- Scheduling dinamico vs. scheduling statico

12

A. Fantechi

## Inadeguatezza degli OS convenzionali

- I sistemi operativi convenzionali sono inadeguati a soddisfare deadline hard
- • le interruzioni sono servite con uno schema di priorità fisso
- • le interruzioni hanno priorità più alta dei processi
- **problema** in real-time systems:
  - i task elaborativi possono essere più importanti di quelli I/O-bound!
- Nessun concetto di clock Real-Time (corrispondenza con tempo fisico)

13

A. Fantechi

## RT Scheduling

- Nei Sistemi Operativi convenzionali gli algoritmi di scheduling dei processi mirano ad **ottimizzare la performance** globale del sistema (ad esempio considerando un opportuno compromesso tra la necessità di minimizzare l'overhead introdotto e di fornire la risorsa processore a tutti i processi).
- Nei Sistemi Operativi soft-real time gli algoritmi di scheduling dei processi mirano ad **ottimizzare i tempi di risposta**.
- Nei Sistemi Operativi hard real time gli algoritmi di scheduling dei processi mirano a **garantire il rispetto delle deadlines**.

14

A. Fantechi

## Real-time operating systems

### Requisiti di un RTOS:

- **1. Il comportamento temporale dell'OS deve essere predicibile**
  - Tutti i servizi del sistema operativo devono avere un limite superiore al tempo di esecuzione
  - RTOSs devono essere deterministici
- **2. L'OS deve gestire il tempo e lo scheduling RT**
  - L'OS deve essere in grado di garantire le deadlines dei task; (a meno che lo scheduling non venga fatto off-line).
  - L'OS deve fornire servizi temporali precisi e ad alta risoluzione
    - *problema clock synchronization in presenza di varie unità distribuite*
- **3. L'OS deve essere veloce!**
- **4. Configurabilità**

15

A. Fantechi

## Embedded operating systems Configurabilità -

- Nessun RTOS potrà soddisfare tutti i bisogni, e d'altra parte non possono essere tollerati overhead dovuti a funzioni non usate (sia per quanto riguarda il tempo, che lo spazio di memoria)
  - **configurabilità**
  - – rimozione delle funzioni inutilizzate (da parte del linker ?).
  - – compilazione condizionale (usando #if e #ifdef in C).
  - – in caso di uso di linguaggi object-oriented: uso di sottoclassi derivate

16

A. Fantechi

## RTOS - Kernels

### Includono

- gestione del processore,
- gestione della memoria,
- gestione del time,
- gestione dei task,
- comunicazione e sincronizzazione inter-task

### Distinzione tra

- real-time kernels e kernels di OS standard modificati .
- RTOS generici e RTOS per domini specifici,
- standard APIs (e. g. POSIX RT- Extension of Unix, ITRON, OSEK) o APIs proprietarie .

17

A. Fantechi

## RTOS diffusi sul mercato

- Proprietari:
  - VxWorks by WindRiver
  - LynxOS by Lynx
- Free/Academical/Open-source:
  - QNX
  - RTLinux
  - Spring
  - RTX
  - ...

18

A. Fantechi

## 6. Sistemi palmari

(sistemi *general purpose* di ridotte dimensioni)

- **Applicazioni Real-time embedded: i dispositivi diventano sempre più piccoli....**
  - Palmari
  - Lavatrici
  - VCRs, fotocamere digitali
  - Sistemi di allarme
- **....con molte più funzionalità**
  - Maggiore elaborazione dati
  - Comunicazione con altri dispositivi
  - Comunicazione con il PC
  - Accesso ad Internet
- **Ciò che è richiesto è un Sistema Operativo General Purpose adatto per il maggior numero possibile di applicazioni**



19

A. Fantechi

## Motivazioni per nuovi Windows OS

- **Limitazioni di Windows 2000**
  - Funziona solo su architetture x86
  - I dispositivi embedded non sono come i PC
    - Hardware eterogeneo
  - Vincoli di memoria, di consumo energetico, di utilizzo della CPU
  - Windows 2000 occupa molta memoria
  - Nessuna gestione del risparmio energetico
  - Nessun supporto real-time
- **Requisiti richiesti**
  - Sistema operativo configurabile dal costruttore del dispositivo (Original Equipment Manufacturer, OEM) per eseguire funzionalità mirate su hardware specifico
  - Ampio supporto hardware
  - Funzionamento con risorse limitate
  - Portabile e progettato per sistemi a batteria

20

A. Fantechi

## Windows CE

- **Supporto Real-Time**
- **Gestione aggressiva del consumo energetico**
- **Gestione ottimizzata delle risorse e bassa occupazione di memoria**
  - I sistemi CE generalmente non prevedono l'uso di dischi
  - Ogni applicazione risiede in memoria in formato compresso (deve essere scompattata prima di essere eseguita)
  - La memoria fisica è suddivisa in RAM (volatile) e ROM
  - La ROM contiene gli applicativi stabiliti dal costruttore OEM nonché le applicazioni built-in del Sistema Operativo (es: pocket Word)

21

A. Fantechi

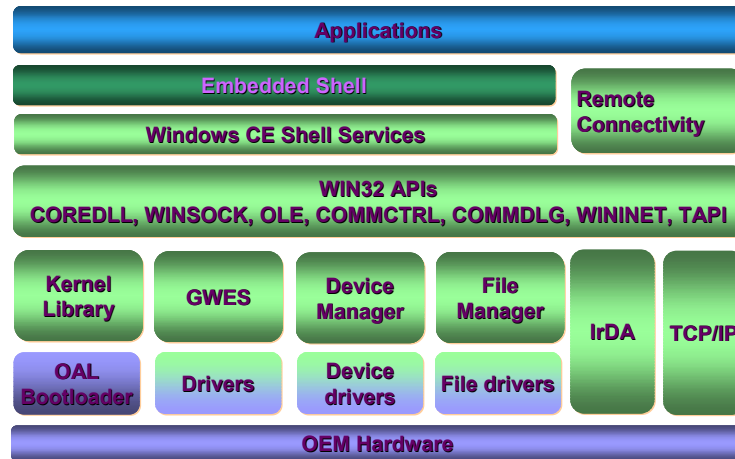
## Windows CE

- **Architettura modulare**
  - OS suddiviso in 202 moduli (EXE/DLL)
  - Ogni modulo può essere scomposto in componenti (file .LIB)
  - Tutti i componenti possono essere eseguiti dalla ROM o compressi sulla flash
- **Il costruttore OEM può decidere quali componenti utilizzare, configurarne la dimensione nonché le caratteristiche e la performance**
  - Il modulo più piccolo occupa circa 400K
  - Networking (no graphics) circa 800K
  - GWE, Shell e applicazioni circa 4MB
  - Internet Explorer for Windows CE circa 3MB

22

A. Fantechi

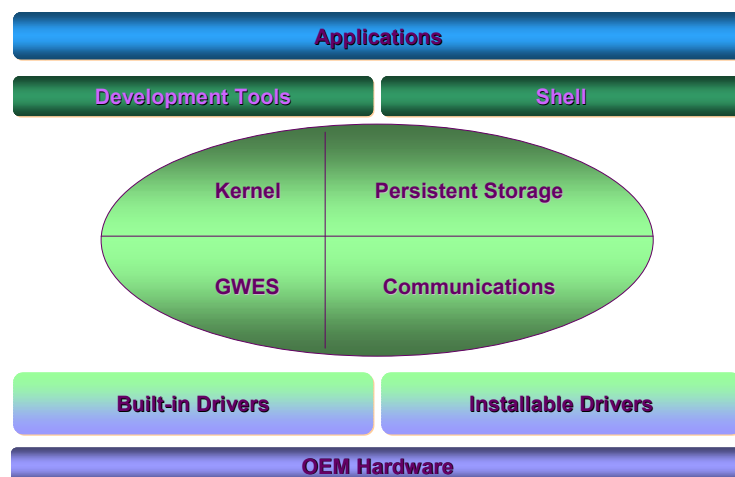
## Architettura di Windows CE



23

A. Fantechi

## Windows CE – Architettura Semplificata



24

A. Fantechi

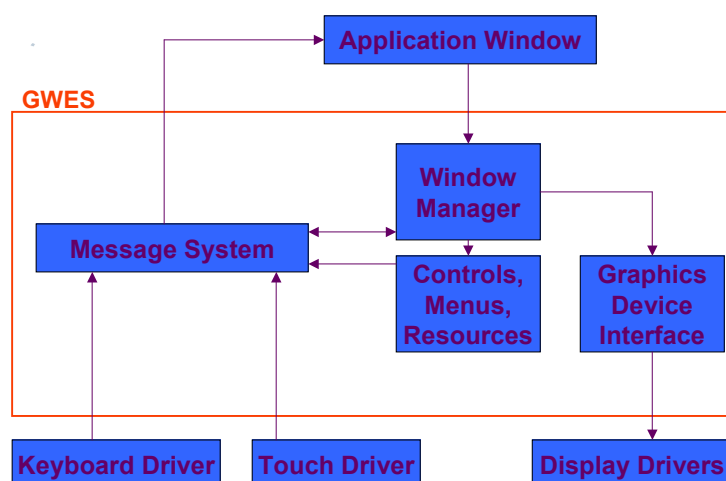
## Graphics, Windowing and Event Subsystem (GWES)

- **Interfaccia Grafica tra utente, applicazioni e Sistema Operativo**
- **La GWES gestisce**
  - Input
    - Traduce in messaggi le immissioni da tastiera (keystrokes), i movimenti dello stilo del palmare, e le selezioni sul display
  - Output
    - Crea e gestisce finestre, grafica e testo
  - Message Event Passing
  - Gestione del consumo energetico
- **Supporta tutte le finestre, dialoghi, controlli e menu che costituiscono l'interfaccia utente di Windows CE**
  - Win CE non supporta Ingrandisci e Riduci a Icona!
- **Le piattaforme che non hanno una GUI utilizzano GWES per la gestione del messaging**

25

A. Fantechi

## Struttura della GWES



26

A. Fantechi

## GWES – Windows Messaging

- Tutte le applicazioni hanno bisogno di una finestra per comunicare con il Sistema Operativo (anche nel caso di dispositivi senza display grafico)
- Ad ogni nuova finestra creata, CE associa una coda dei messaggi
- Il Sistema Operativo traduce gli input utente in messaggi che vengono inseriti nella coda della finestra attiva
- Le applicazioni processano la maggior parte di questi messaggi, lasciando il resto al Sistema Operativo

27

A. Fantechi

## GWES – Power Management

- **Stati della gestione consumi**
  - No Power
    - No power from any source (AC, battery, backup battery)
    - Registry initialised from scratch
  - On
    - Utilizzo del dispositivo a regime
  - Idle
    - Il dispositivo è attivo, ma non utilizzato per un determinato periodo di tempo
    - PCMCIA cards, modems, NICs sono disattivati
  - Suspend
    - La CPU e le periferiche sono spente
    - L'energia erogata è comunque sufficiente per mantenere la RAM
  - Critical Off
    - Simile alla sospensione, ma si raggiunge quando la batteria è al limite
    - Si può uscire da questo stato solo tramite ricarica

28

A. Fantechi