SISTEMI OPERATIVI

12.a

Multiprocessing e Multithreading, Real-Time

Real-Time

1

- Real-time computing
- RTOS, real-time scheduling
- Priority inversion
- Real-time nei SO di uso generale
- Tipi di RTOS sul mercato

35

Real-time computing

- La correttezza di un sistema non dipende solo da una gestione ottimale delle risorse e dalla correttezza logica dell'esecuzione ma anche dal tempo in cui il risultato delle computazioni viene prodotto
- Processi (suddivisi in [sub]task) controllano o reagiscono ad eventi real-time provenienti dall'esterno
  (che tipicamente producono interruzioni hardware)
- Tipici sistemi di questa classe sono:
  - Sistemi di controllo di laboratorio
  - Sistemi di controllo di impianti
  - Sistemi di controllo della navigazione
  - Sistemi di telecomunicazione
  - Sistemi robotici
  - Sistemi di controllo/comando militari

Classificazione dei sistemi Real-time

- Ad ogni task real-time viene associato una scadenza (deadline) che è l'istante prima del quale è richiesto che l'esecuzione del task abbia inizio (o termini).
- Hard real-time task sono task con deadline vincolante: il non soddisfacimento della scadenza comporta condizioni inaccettabili o un errore fatale
- Soft real-time task sono task con deadline non vincolante: il non soddisfacimento della scadenza non pregiudica il funzionamento del sistema, anche se può costituire una situazione degradata
- Real-time task aperiodico: il deadline riguarda l'avvio o la terminazione del task o entrambi
- Real-time task periodico: il vincolo del deadline è posto una volta per ogni periodo

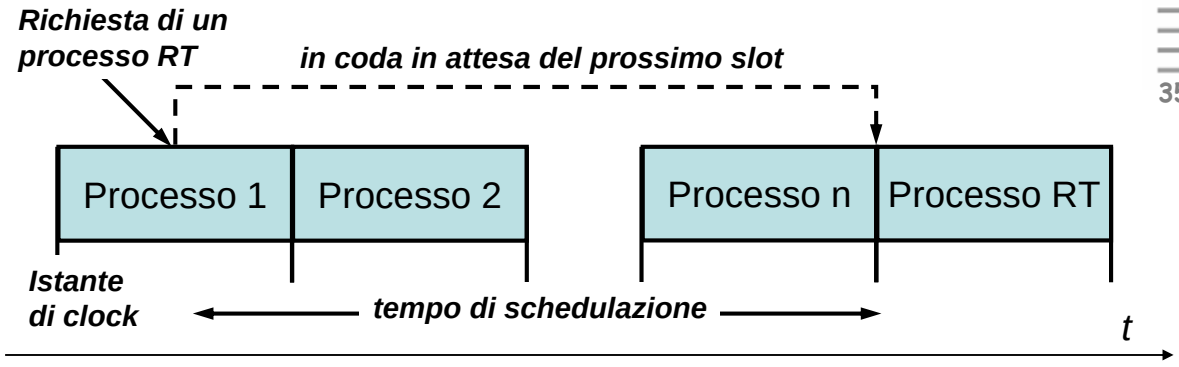
Sistemi Operativi Real-time

- Questi sistemi non gestiscono solo le risorse di sistema ma cercano di garantire i deadline dei task real-time. I requisiti specifici riguardano le seguenti aree:
- Determinismo
  - Le attività hanno luogo in precisi istanti o predeterminati intervalli di tempo
  - Il grado di determinismo dipende dalla velocità di risposta alle interruzioni (rapidità di acknowledge) e dalla capacità complessiva di gestire tutte le richieste entro il tempo previsto
  - Nessun sistema reale è completamente deterministico
- Capacità di reazione (responsiveness)
- Rapidità nel servire un'interruzione
- Dipende dall'overhead dovuto al salvataggio del contesto e riconoscimento dell'interrupt, dalla durata della ISR, dal nesting di interruzioni di diverso livello

Sistemi Operativi Real-time [2]

- Affidabilità
  - Una degradazione per malfunzionamento anche solo temporaneo, tollerata nei sistemi tradizionali, nei sistemi real-time può essere molto dannosa
- Recupero dei malfunzionamenti
  - Il cosiddetto fail-soft operation è l'abilità di un sistema che fallisce di recuperare le operazioni in modo tale da preservare la maggior parte delle capacità e dei dati possibili
  - Un sistema RT è detto stabile, nel caso non sia possibile soddisfare tutti i deadline, se è almeno in grado di garantire il soddisfacimento dei deadline dei task più critici e di quelli a maggior priorità

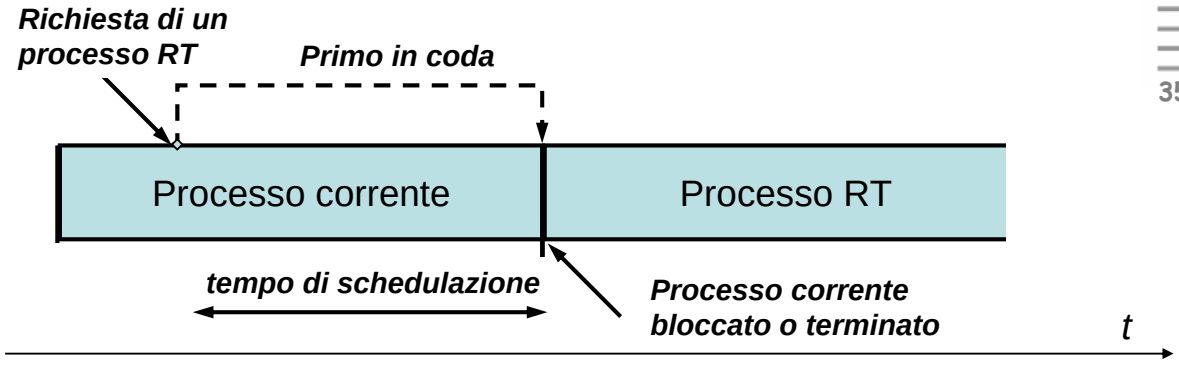
RT – RR preemptive



Scheduler Round Robin preempitive

- Tempo di scheduling 10 s: in genere non accettabile per applicazioni RT

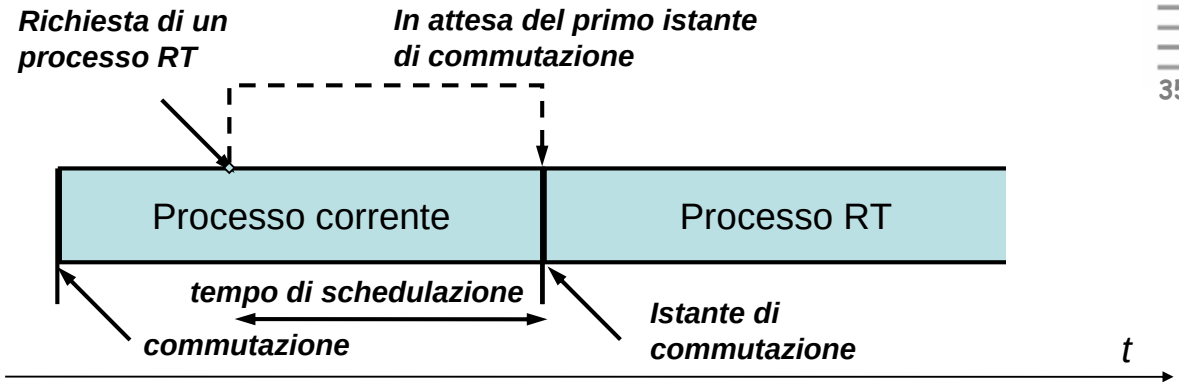
RT – Nonpreemptive a priorità



Scheduler non preempitive a priorità

- Tempo di scheduling più breve, spesso (processi CPU-bound) non essere accettabile per RT

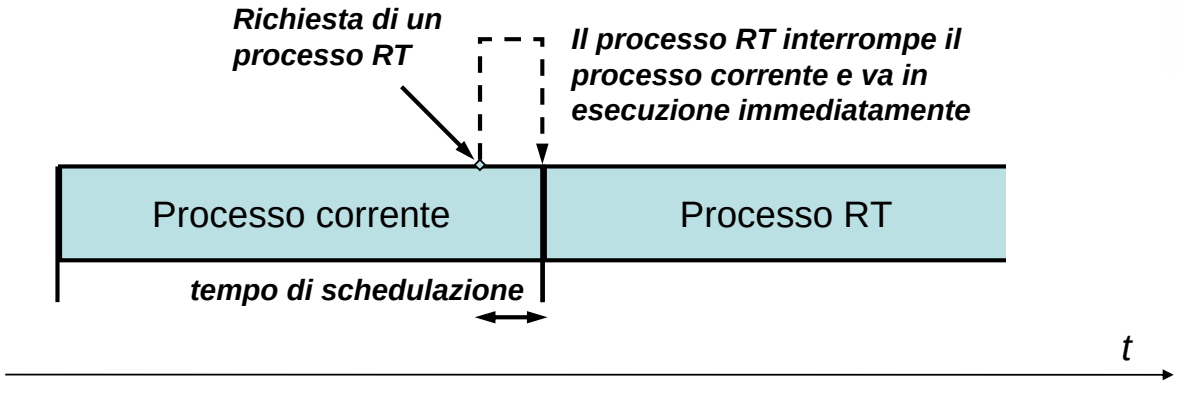
RT – Preemptive a priorità



Scheduler preempitive a priorità con istanti di commutazione

- Il preemption può aver luogo allo scadere di quanti temporali; tempo di scheduling limitato a pochi ms, accettabile per molte applicazioni RT

RT – Preemptive immediato



Scheduler preempitive immediato

- A fronte di una richiesta più prioritaria, il preemption è immediato; tempo di scheduling limitato a 100 μs o meno, accettabile per le applicazioni RT più critiche. (Ma se interrupt disabilitati?)

Trattamento dei deadline

- Molto difficile gestire direttamente i deadline
- Si cerca di garantire che un RT task venga rapidamente schedulato all'approssimarsi del deadline
- Viene solitamente richiesta una risposta deterministica (nel range frazioni di ms - decine di ms) in un ampio spettro di situazioni
- Conviene utilizzare almeno uno scheduling con punti regolari di preemption (basati su RTC) e assegnare priorità più elevate ai RT task
- Meglio ancora uno scheduling con preemption immediato (salvo regioni 'veramente' critiche ove non consentirlo)

Con tempi di risposta alle interruzioni rapidi e preemption immediato si riesce a contenere i ritardi di scheduling sotto i 100 μs

Scheduling real-time

- Statico, basato su tabelle preventive
  - viene eseguita una analisi statica preliminare che fornisce lo schema che determina quando eseguire i task in modo da garantire i deadline
  - applicabile a task periodici di cui si conoscono i parametri significativi (tempo di avvio periodico, tempo di esecuzione, deadline periodico, priorità relativa)
- Statico, basato su priorità precalcolate
  - l'analisi è usata SOLO per assegnare le priorità relative
    (es: l'algoritmo 'rate monotonic')

Scheduling real-time [2]

- Dinamico, basato su planning
  - all'arrivo di un nuovo task si determina run-time se accettarlo o meno a seconda che il suo inserimento consenta di soddisfare i suoi vincoli e mantenga soddisfatti gli altri
  - se sì, viene anche stabilito quando schedularlo
  - in occasione dell'arrivo di un nuovo task, il piano di schedulazione viene ricalcolato in quel momento
- Dinamico, miglior risultato
  - non c'è un'analisi
  - il sistema abortisce tutti i task attivati il cui deadline è scaduto
  - metodo implementato da molti sistemi: all'avvio di un task generalmente non periodico gli viene assegnata una priorità (basata sulle caratteristiche del task); viene applicata una qualche forma di scheduling basato su deadline (es: a deadline più vicino)
  - finché non scade il deadline o il task non termina non è dato sapere se il deadline sarà onorato

Deadline scheduling

- La maggior parte degli RTOS moderni si concentra sull' avvio il più rapido possibile dei RT task e sulla rapidità di risposta alle interruzioni
- Una misura ragionevole della qualità di un RTOS è il grado di precisione di attivazione (e terminazione) dei task al tempo giusto, a prescindere da altre questioni in gioco (richieste di risorse, conflitti, ecc.)
- Il semplice concetto di priorità è in tal senso insufficiente
- Per uno scheduling RT efficace servono informazioni aggiuntive sui task:
  - ready time, tempo di esecuzione (opzionale)
  - deadline di avvio, deadline di completamento
  - risorse necessarie, priorità relativa
  - struttura di sottotask

Earliest deadline scheduling

Table 10.2 Execution Profile of Two Periodic Tasks

Process	Arrival Time	Execution Time	Ending Deadline
A(1)	0	10	20
A(2)	20	10	40
A(3)	40	10	60
A(4)	60	10	80
A(5)	80	10	100
•	•	•	•
•	•	•	•
•	•	•	•
B(1)	0	25	50
B(2)	50	25	100
•	•	•	•
•	•	•	•
•	•	•	•

Scheduling ogni 10 ms

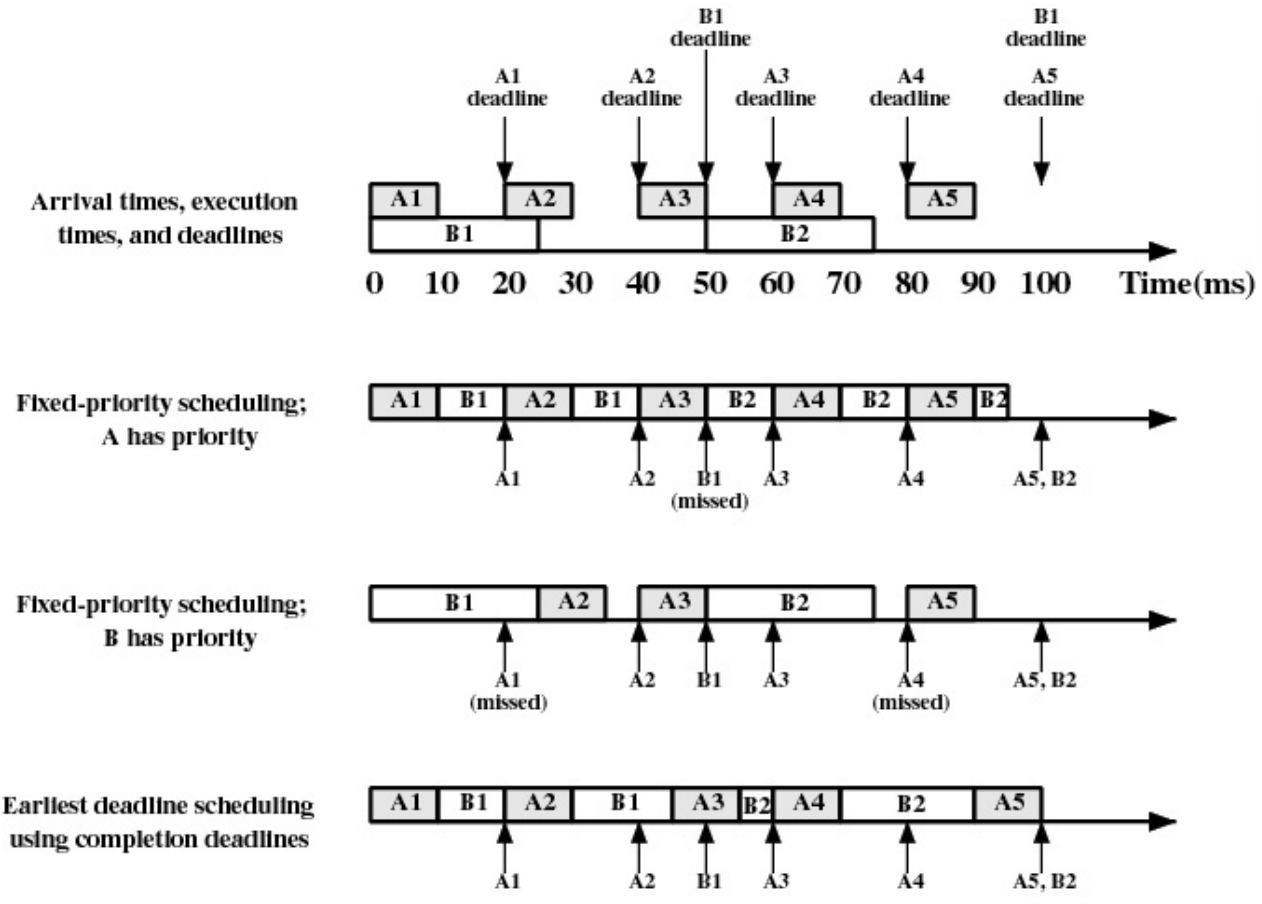


Figure 10.5 Scheduling of Periodic Real-time Tasks with Completion Deadlines

Earliest deadline scheduling [3]

Table 10.3 Execution Profile of Five Aperiodic Tasks

Process	Arrival Time	Execution Time	Starting Deadline
A	10	20	110
B	20	20	20
C	40	20	50
D	50	20	90
E	60	20	70

Scheduling di task aperiodici: starting deadline

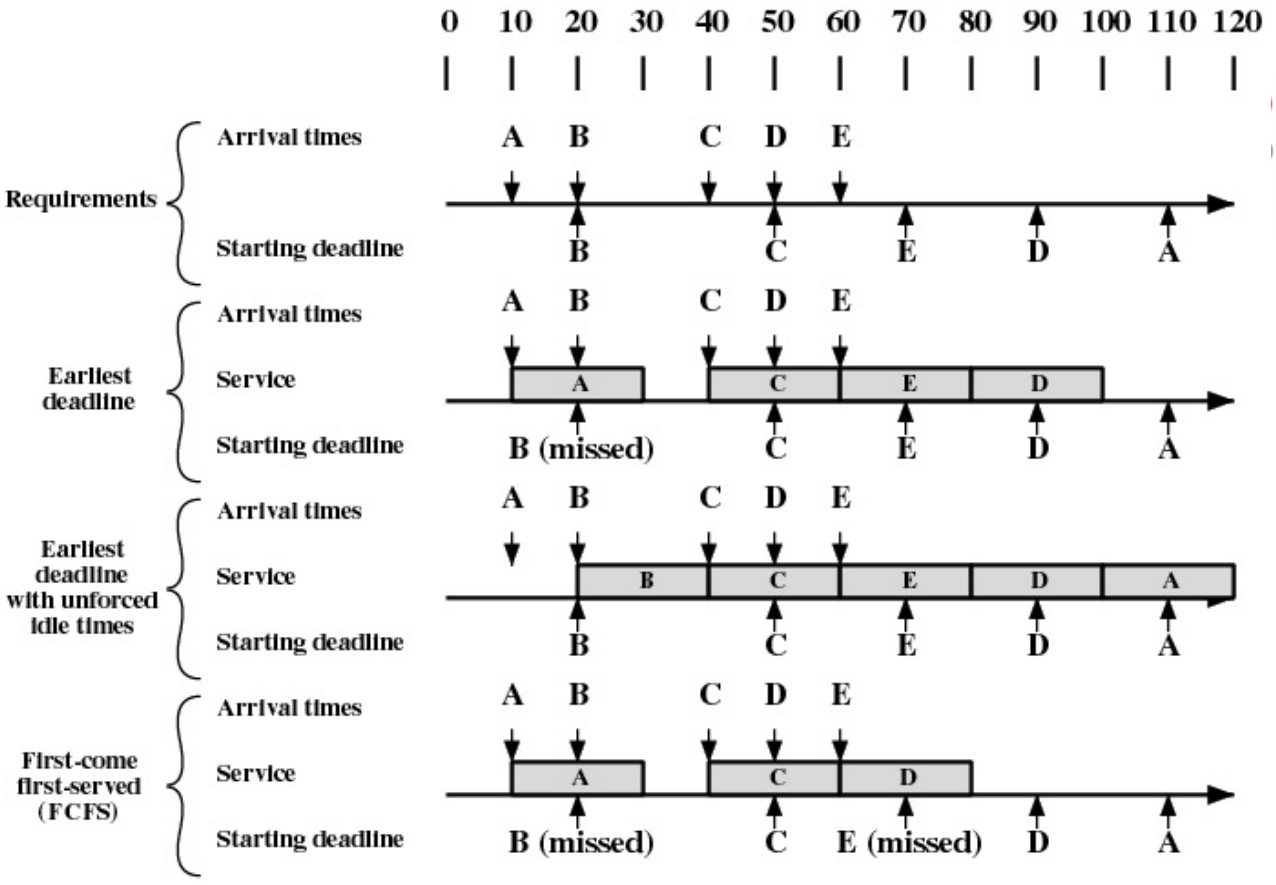
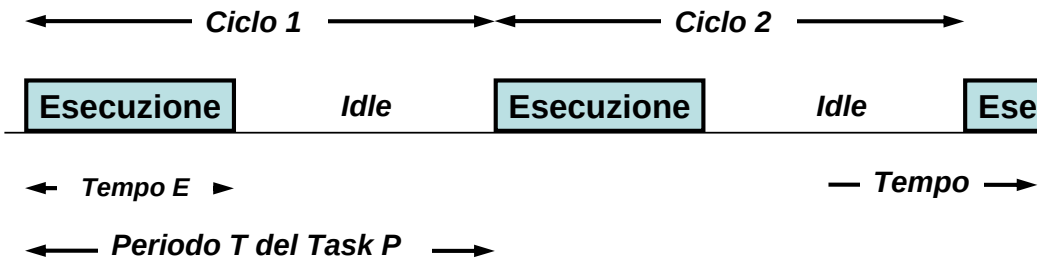


Figure 10.6 Scheduling of Aperiodic Real-time Tasks with Starting Deadlines

Rate Monotonic scheduling

- Alcune applicazioni Real-Time sono costituite da processi periodici dei quali si conoscono le caratteristiche:
  - R - Ready Time (istante in cui è pronto ad iniziare),
  - S - Starting Deadline (istante entro cui deve iniziare),
  - C - Completion Deadline (istante entro cui deve completare),
  - E - Tempo di esecuzione,
  - T - Periodo di attivazione.



Andamento temporale di un task periodico

Rate Monotonic scheduling [2]

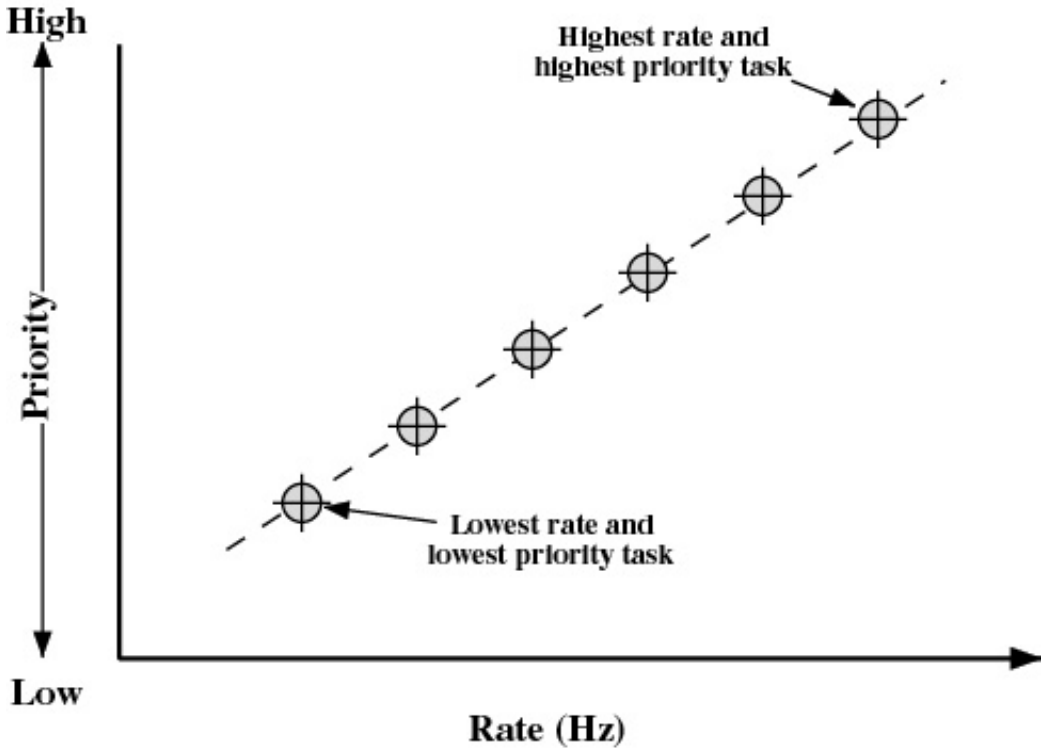
- Per task periodici, assegna la priorità sulla base del valore del periodo, le priorità più elevate sono assegnate ai task di periodo inferiore (la priorità cresce con il crescere della frequenza)
- Occorre verificare se e con quale margine tutti i deadline sono soddisfatti
- In generale, per qualsiasi algoritmo di schedulazione di n task periodici, condizione necessaria perché i deadline siano soddisfatti è la seguente:

  $$\Sigma i\ Ci / Ti \leq 1$$

  Ci = tempo di esecuzione in ciascun periodo del task i
  Ti = periodo del task i
  Ci / Ti = percentuale di utilizzazione della CPU
- Il valore 1 corrisponde alla piena utilizzazione della CPU nel caso di un algoritmo di schedulazione idealmente perfetto (si trascurano gli overhead di sistema); la relazione limita il numero di task schedulabili
- Nel caso RMS la condizione da soddisfare è più stringente all'aumentare di n
- $$\Sigma i\ Ci / Ti \leq n\ (2^{1/n} - 1)$$

  Il termine a destra tende a ln 2=0.693 per $n \to \infty$

Rate Monotonic scheduling [3]

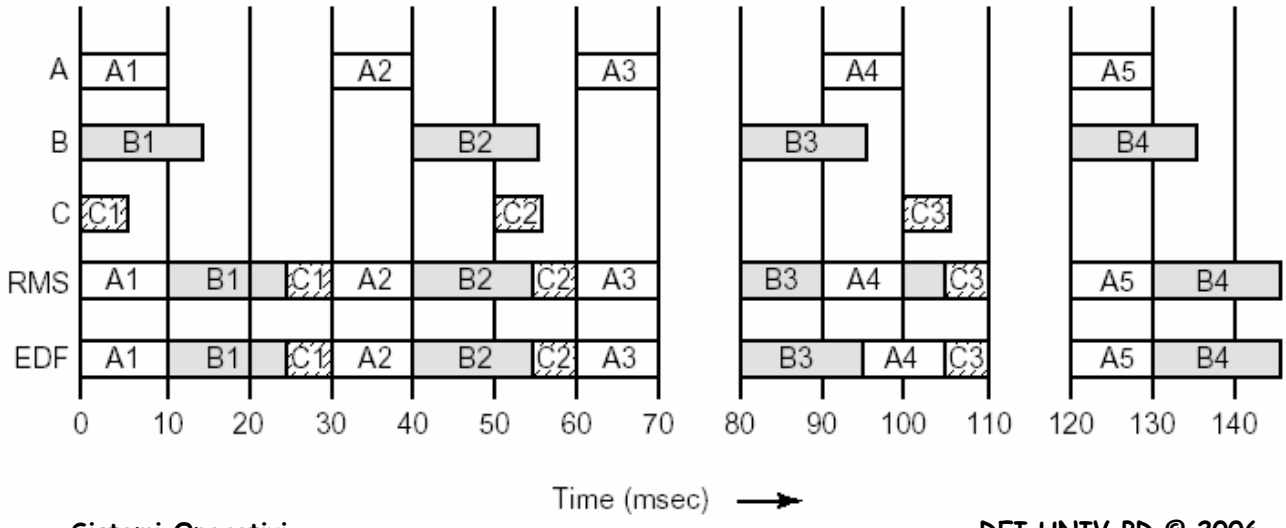


Confronto tra RMS e EDS

TA=30ms TB=40ms TC=50ms → prioA=33 prioB=25 prioC=20
CA=10ms CB=15ms CC=5ms (execution time)

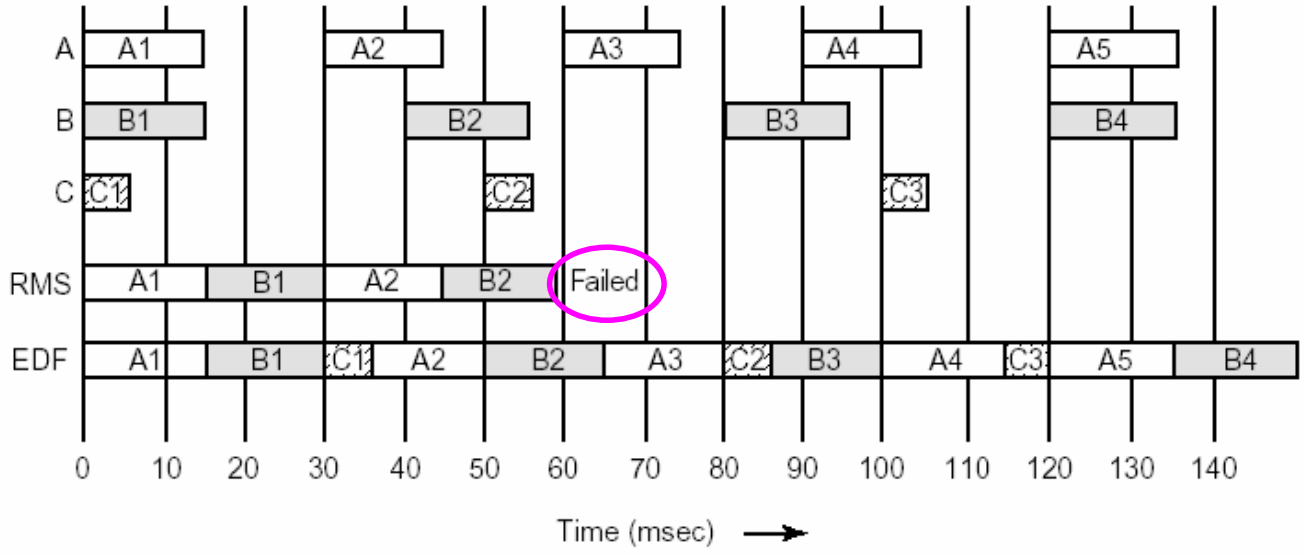
$\Sigma i\ Ci / Ti = 1/3 + 3/8 + 1/10 = 0.808$

(Rate Monotonic) vs early deadline



Confronto tra RMS e EDS [2]

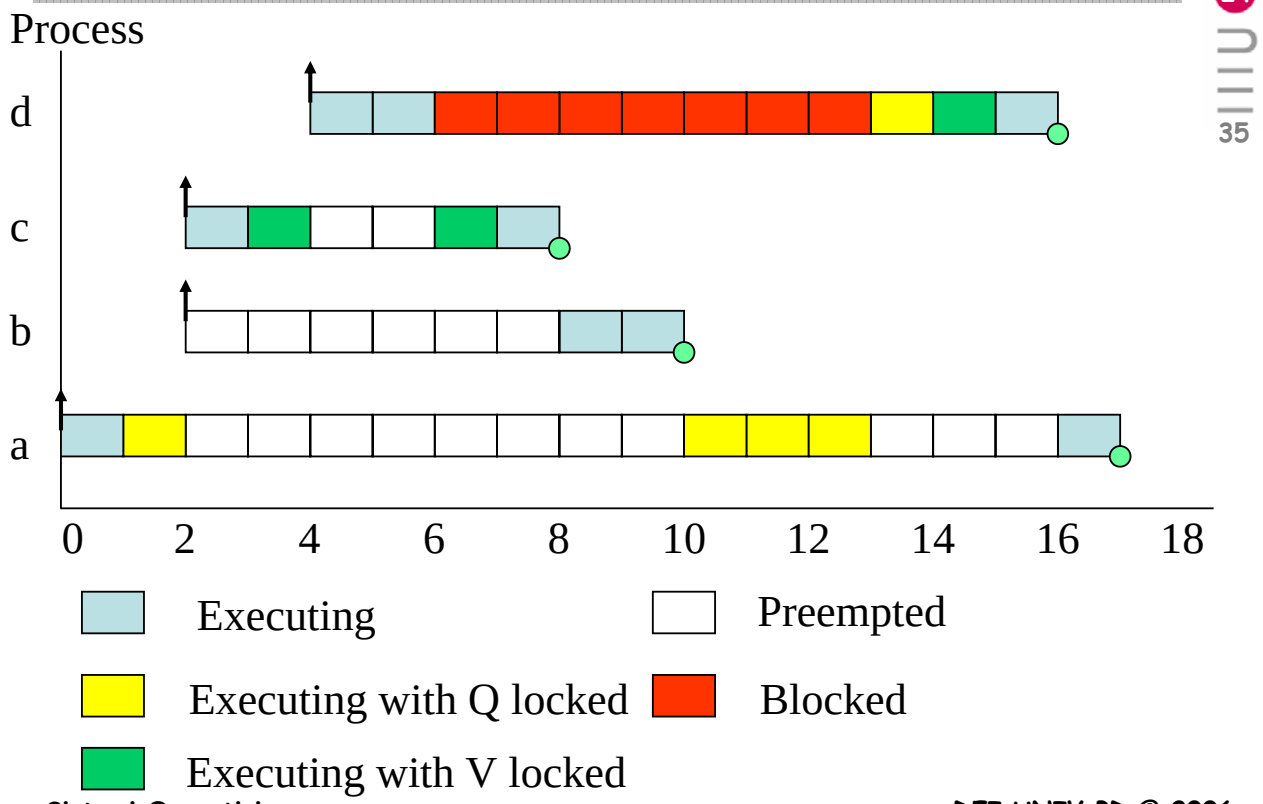
TA=30ms TB=40ms TC=50ms → prioA=33 prioB=25 prioC=20
CA=15ms CB=15ms CC=5ms (execution time)
$\Sigma i\ Ci / Ti = 1/2 + 3/8 + 1/10 = 0.975 \quad 3(2^{1/3} - 1) = 0.779$



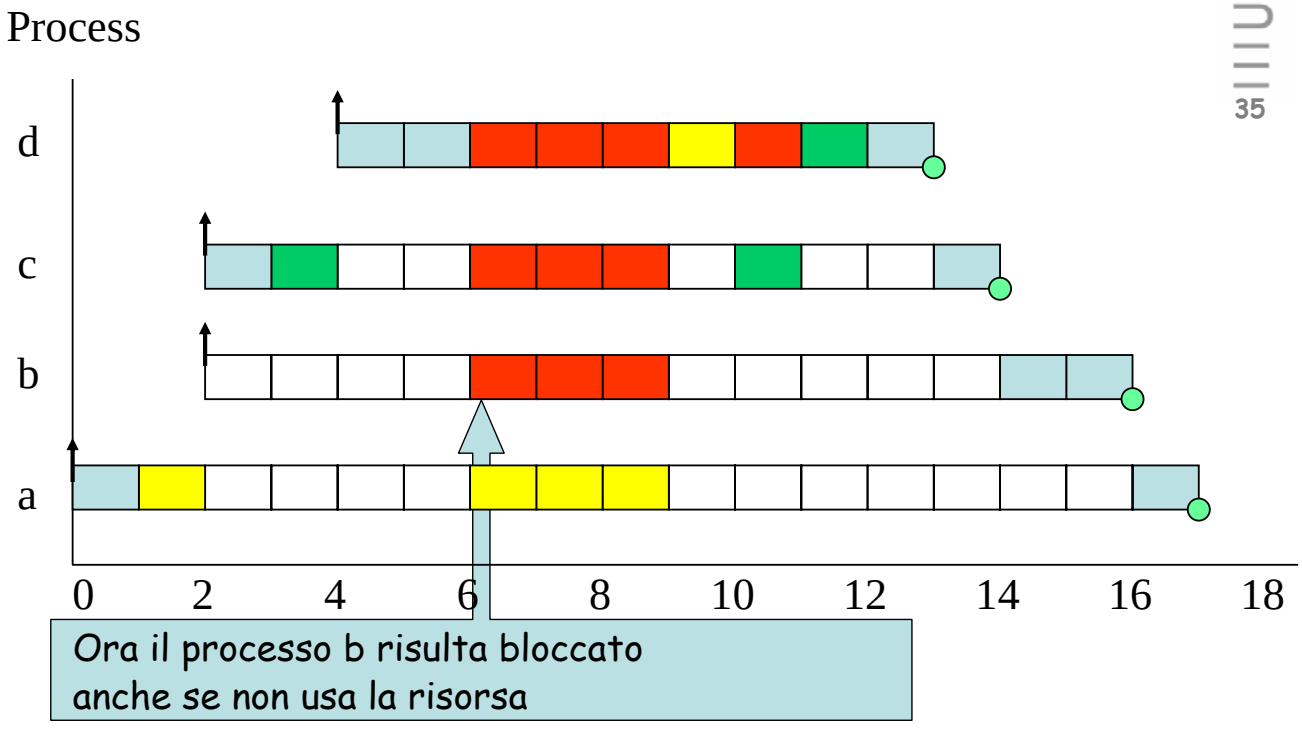
Priority Inversion

- In un sistema con scheduling preemptive (preemptive priority scheduling) c'è il rischio di priority inversion:
  - un processo a bassa priorità esegue positivamente una wait(s) su un semaforo di mutua esclusione, successivamente un processo ad alta priorità vi si accoda
  - altri processi a priorità intermedia per preemption impediscono al processo a bassa priorità di avanzare ed eseguire signal(s) per liberare il semaforo,
  - di conseguenza i processi con priorità inferiore bloccano quello con priorità elevata (inversione della priorità);
- Una soluzione: priority inheritance (il processo che impegna un semaforo eredita, temporaneamente, la priorità più elevata dei processi in coda sullo stesso semaforo);
- l'adozione di questo meccanismo aumenta l'overhead.

Esempio di priority inversion



Esempio di priority inversion [2]



Proprietà tipiche dei RTOS

- Switch molto veloce del contesto (di processo o thread)
- Dimensioni ridotte
- Risposta rapida agli interrupt esterni
- Multitasking con IPC (semafori, segnali, eventi)
- File contigui per accessi veloci ai dati memorizzati
- Riduzione al minimo dei periodi ad interrupt disabilitati
- Primitive per ritardare i task a tempo
- Primitive di sospensione/ripresa di un task
- Allarmi speciali e timeout
  - Molti RTOS commerciali implementano tutte/alcune queste caratteristiche

Standard

- OSEK/VDX (Siemens AG, www.osek-vdx.org)
  - specifica per RTOS in ambito automotive
- ITRON-μITRON
  (TRON association, www.assoc.tron.org)
  - standard nell'ambito di RTOS per ES
- Real-time POSIX
  - Si tratta di una serie di estensioni real-time allo standard POSIX (per sistemi Unix-like)
  - 1003.1b realtime extensions
  - 1003.1c threads
  - 1003.1d additional realtime extensions
  - 1003.1j advanced realtime extensions
  - 1003.1q tracing
  - 1003.13 real-time profile
- Esiste un sottospecifica (PSE) dedicata ai test sui sistemi embedded

Sistemi RTOS tipici

- A livello di kernel
  - AMX (Kadak Products Ltd., www.kadak.com)
  - embOS (Segger Microcontroller Systeme GmbH, www.segger.de)
  - QNX (Qnx Software System, www.qnx.com)
  - ThreadX (Green Hills Software, www.ghs.com)
  - Integrity (Green Hills Software, www.ghs.com)
  - OSE (Enea Embedded Technology, www.ose.com)
  - OS/9 (Microware / RadiSys, www.radisys.com)
  - eCos (RedHat sources.redhat.com/ecos/) - open-source RT system supportato dall'ambiente GNU

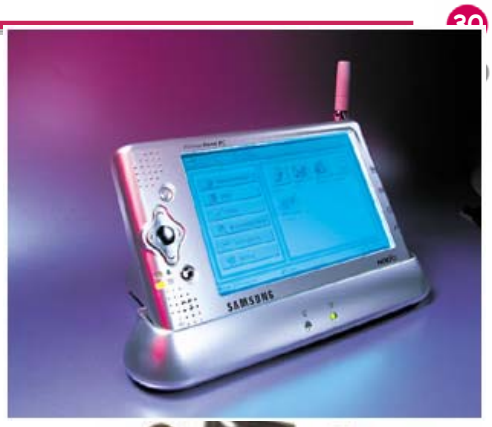
Sistemi RTOS tipici [2]

- Sistemi scalabili, dotati di molte risorse (file system, networking ecc.)
  - VxWorks (Wind River, www.windriver.com)
  - VRTX (Mentor Graphics, www.mentor,com)
  - RTOS-UH (Univ. Of Hannover, www.irt.uni-hannover.de)
  - RTMX O/S estensione di OpenBSD, www.rtmx.com

Sistemi RTOS tipici [3]

- Mondo Microsoft
  - Windows CE 5.0 (luglio 2004)
    - Windows Mobile per architettura Pocket PC

Fonte: Samsung



Fonte: Exertris

  - XP Embedded
    - Sistemi embedded basati su processori Intel

Sistemi RTOS tipici [4]

- Embedded Linux (Soft real-time)
  - AMIRIX (Debian), per sistemi diskless
  - Coollinux, Internet access
  - Xlinux, mobile/handheld
  - RedBlue Linux, wireless
  - Embedix, hard RT, multiprocessor
  - Embedded Linux Solutions (Metrowerks)
  - Linu@, smartphone, PDA
  - NeoLinux, chioschi, thin client
  - Tynux, player MP3, Internet TV/phone, PDA/cell
  - Red Hat Embedded Linux, diverse tecnologie (Embedded Linux, EL/IX, RedBoot, uClinux, e GNUPro)

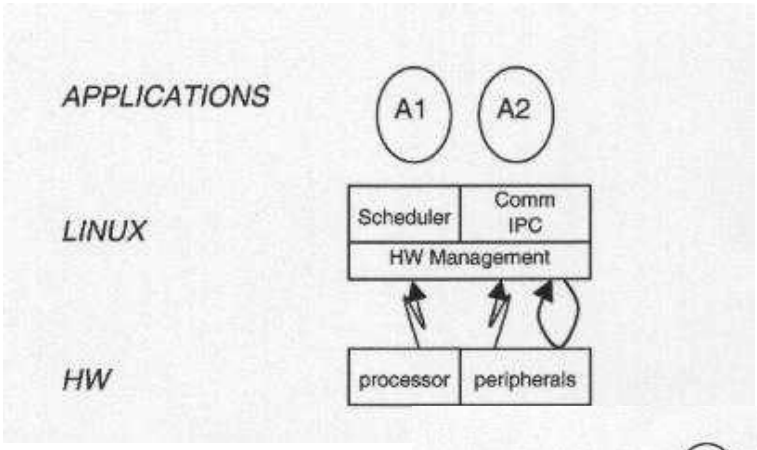
Sistemi RTOS tipici [5]

- Real-time Linux (hard real time)
  - RTLinuxPro (FSMLabs, www.fsmlabs.com)
    - Si tratta di un kernel RT POSIX 1003.13 PSE51 (minimal) che fa girare un Linux POSIX 1003.13 PE54 (full) come suo thread a minima priorità
  - RTAI (DIA PoliMi, www.rtai.org)
    - Approccio simile a RTLinux
  - LynxOS (LinuxWorks www.linuxworks.com)
    - Hard RT POSIX e Linux ABI compatibile

RTAI – Real Time Application Interface

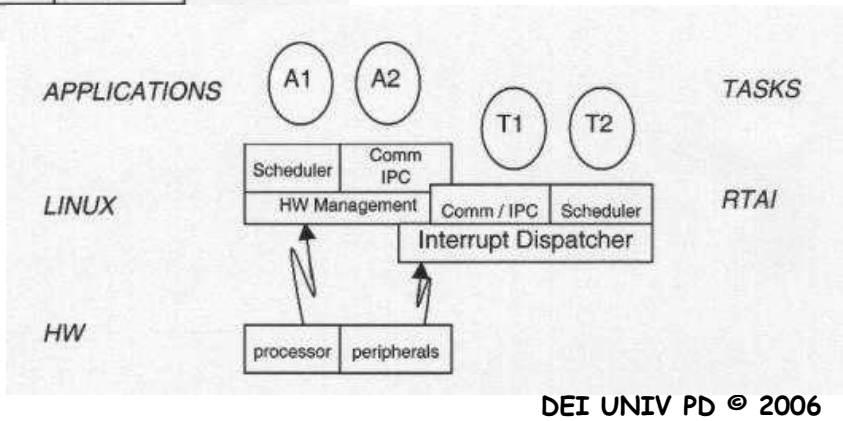
- Sviluppato originariamente al Dip. di Ing. Aerospazione – POLIMI, ora un progetto di community internazionale (www.rtai.org) ed è Open Source
- Non è un RTOS completo, piuttosto una tecnologia per rendere Linux fully-preemptible e integrare nel kernel uno scheduler hard-realtime
- Con pochi interventi, in forma di patch al kernel, soprattutto finalizzati a catturare gli interrupt, è possibile far gestire allo scheduler RT i task RT ad alta priorità e far coesistere i task Linux e i suoi servizi come se fossero un task a più bassa priorità
- Poiché utilizza un approccio di tipo HAL (*Hardware Abstraction Layer*) la dipendenza dal Kernel di Linux è ridotta al minimo, facilitando il porting alle successive versioni
- Compatibilità POSIX 1003.1b,c,d

RTAI [2]



Linux

Linux + RTAI



RTAI [3]

- Una limitazione è data dal fatto che i task RT operano all'interno del kernel e non sono soggetti alle protezioni di memoria
- LXRT (Linux Real Time) è un'estensione di RTAI volta a fornire le funzionalità RT a task Linux che operano in modalità utente
- In questo modo è possibile sviluppare applicazioni real-time disponendo delle protezioni di memoria e degli strumenti di debug di Linux
- È anche possibile far condividere le risorse (shared memory, messaggi, semafori e timimg) in modo simmetrico: Linux↔Linux, Linux↔RTAI, RTAI↔RTAI

Fine

12.a

Multiprocessing e Multithreading, Real-Time