



# Lezione R14

## Linux in ambito real-time

Sistemi operativi open-source, embedded e real-time

12 dicembre 2017

Marco Cesati

Dipartimento di Ingegneria Civile e Ingegneria Informatica  
Università degli Studi di Roma Tor Vergata

Schema della lezione

Linux e real-time

Mono- e dual-kernel

Sistemi partizionati

RTAI e ADEOS

Linux PREEMPT\_RT

# Di cosa parliamo in questa lezione?



In questa lezione si descrivono le soluzioni tecniche che consentono l'impiego di Linux in ambiti hard real-time

- 1 Utilizzo di Linux in ambito real-time
- 2 Approcci mono-kernel e dual-kernel
- 3 Sistemi real-time partizionati
- 4 RTAI
- 5 La patch PREEMPT\_RT

## Schema della lezione

[Linux e real-time](#)

[Mono- e dual-kernel](#)

[Sistemi partizionati](#)

[RTAI e ADEOS](#)

[Linux PREEMPT\\_RT](#)

*È possibile considerare Linux come un sistema operativo real-time? **Dipende...***

Linux **non** è un sistema **hard** real-time: è progettato per massimizzare il **throughput** delle applicazioni e minimizzare i **tempi medi** di risposta

Comunque Linux ha numerose caratteristiche che lo rendono conveniente per un utilizzo in ambienti real-time:

- Buona gestione di tempo e timer (*high resolution timer*)
- Gestione “separata” delle interruzioni
- Schedulazione e gestione delle priorità dei task
- Supporto alle politiche di accesso alle risorse condivise
- Interrompibilità dei task anche in **kernel mode**



Schema della lezione

Linux e real-time

Mono- e dual-kernel

Sistemi partizionati

RTAI e ADEOS

Linux PREEMPT\_RT



Schema della lezione

Linux e real-time

Mono- e dual-kernel

Sistemi partizionati

RTAI e ADEOS

Linux PREEMPT\_RT

- Possibilità di creare facilmente task in **kernel mode**, con cambi di contesto rapidi e gestione della memoria con un unico spazio di indirizzamento
- Supporto alla memoria virtuale e protezione della memoria per i task **user mode** (cambio di contesto lento)
- Possibilità di configurazione a grana fine
- Kernel monolitico, ma codice completamente disponibile, modificabile e modulare per ottenere occupazione della memoria (**footprint**) ridotta
- È il sistema operativo con il maggior numero di architetture supportate
- È il sistema operativo con il maggior numero di driver di periferiche disponibili
- Comunità di sviluppo, supporto e testing molto attive

## Limiti di Linux come hard RTOS

Il limite principale di Linux per il suo uso in ambito **hard real-time** è nella **predicibilità** temporale di kernel e applicazioni:

- Il kernel non è completamente interrompibile
- Le interruzioni sono disabilitate nelle sezioni critiche, sia nelle chiamate di sistema che nella gestione secondaria (ISR) di numerose interruzioni
- Le ISR possono avere una durata **non predicibile**
- La gestione delle interruzioni non fa uso di meccanismi di priorità
- La latenza di schedulazione dei task è elevata, soprattutto a causa del costo dei cambi di contesto

*Si può rimediare a tutti questi limiti?*

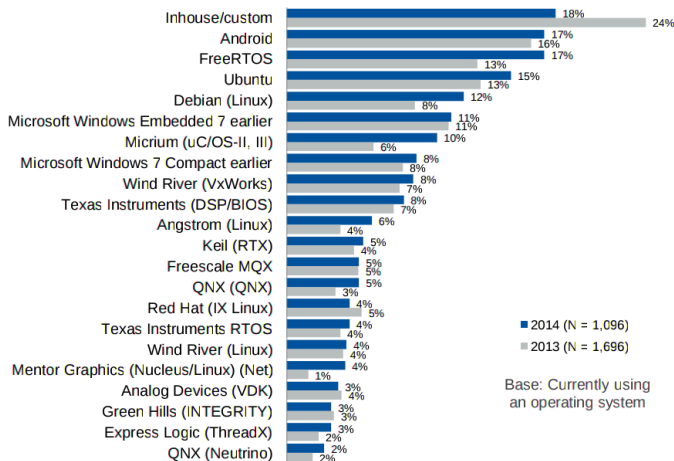
Non è semplice, ma effettivamente è possibile modificare il kernel Linux in modo da renderlo sostanzialmente **predicibile**



# Linux come embedded RTOS

*Il kernel Linux è monolitico, ha senso considerarlo per i sistemi embedded?*

Sì, dato che è possibile configurare il kernel in modo da ridurre drasticamente la sua dimensione



Base: Currently using an operating system



## Limiti di Linux come hard RTOS

Il limite principale di Linux per il suo uso in ambito **hard real-time** è nella **predicibilità** temporale di kernel e applicazioni



Nel meccanismo di gestione delle interruzioni:

Schema della lezione

Linux e real-time

Mono- e dual-kernel

Sistemi partizionati

RTAI e ADEOS

Linux PREEMPT\_RT

- Interruzioni senza priorità
- Molte **ISR** hanno sezioni critiche con interruzioni disabilitate
- Le **ISR** possono avere una durata **non predicibile**

Altri possibili problemi:

- Clock e timer non sufficientemente precisi
- Elevata latenza di schedulazione dei task, soprattutto a causa del costo dei cambi di contesto
- Kernel non completamente interrompibile

## Gestione delle interruzioni in Linux

Un gran numero di problemi legati all'uso di Linux in ambito real-time è dovuto alla gestione delle interruzioni

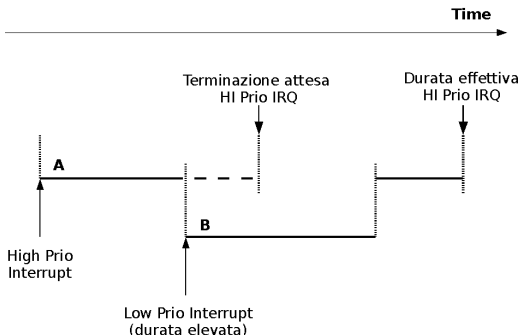
- Le interruzioni hanno priorità più elevata dei processi
- Si utilizza una gestione “separata”, con tre livelli:
- La prima parte della gestione (*top half*) si compone di **interrupt handler** e la parte immediata dell'**ISR**
  - L'**interrupt handler** salva e recupera il contesto interrotto e dà conferma della ricezione dell'interruzione
  - La parte immediata dell'**ISR** interagisce con la periferica H/W ed eventualmente “prenota” la successiva esecuzione del **bottom half**
- La seconda parte della gestione (*bottom half*)
  - svolge operazioni lunghe generalmente interrompibili
  - è costituita da una procedura da eseguire quando nessun altro **top half** è in esecuzione, quindi:
    - immediatamente prima di tornare ad eseguire un processo
    - oppure per mezzo di un **kernel thread**





## Problemi nella gestione delle interruzioni

- Le interruzioni sono in generale **interrompibili** da altre interruzioni
- È un problema perché le interruzioni non hanno priorità



Disabilitare l'interrompibilità delle interruzioni non è una buona soluzione  $\Rightarrow$  **le interruzioni non sono predicibili**





Schema della lezione

Linux e real-time

Mono- e dual-kernel

Sistemi partizionati

RTAI e ADEOS

Linux PREEMPT\_RT

- Linux è in grado di supportare i principali dispositivi hardware (**clock device**) per la gestione del tempo
- In genere ogni architettura offre diversi dispositivi hardware per la gestione del tempo
- In Linux ogni “orologio” hardware è visto come un dispositivo virtuale chiamato **clock source**
- Il tempo interno del sistema è mantenuto in **nanosecondi** (e non più in *jiffies* = numero di occorrenze del clock tick)
- Linux supporta i timer tradizionali di Unix, con bassa precisione e basati sul **clock tick**
- Linux supporta anche **timer ad alta risoluzione**, o **hrtimer**, indipendenti dal **clock tick** e con risoluzione nominale nell'ordine del nanosecondo
- La risoluzione effettiva degli **hrtimer** è comunque dell'ordine del microsecondo

### *Cosa limita la precisione degli hrtimer in Linux?*

- Attivare un **hrtimer** significa programmare un dispositivo hardware in modo che esso generi una interruzione dopo un determinato intervallo di tempo
- Se le interruzioni sono disabilitate per intervalli di tempo troppo lunghi, Linux non può rilevare la scadenza dell'**hrtimer** con sufficiente precisione
- Poiché le interruzioni non hanno priorità, la gestione dell'interruzione associata all'**hrtimer** può essere interrotta da un flusso di altre interruzioni
- Il kernel non è sempre interrompibile, quindi l'applicazione real-time che ha programmato l'**hrtimer** potrebbe non essere posta in esecuzione in tempo utile
- Poiché il costo del cambio di contesto è elevato, la lunghezza minima dell'intervallo effettivamente programmabile in un **hrtimer** è eccessiva



Schema della lezione

Linux e real-time

Mono- e dual-kernel

Sistemi partizionati

RTAI e ADEOS

Linux PREEMPT\_RT

Lo scheduler di Linux è un programma modulare:

- Basato sul concetto di *classi di schedulazione*, ossia una gerarchia di moduli che implementano differenti politiche di gestione dei processi
- Lo scheduler generale si limita ad interrogare le classi di schedulazione in ordine di priorità, chiedendo processi da porre in esecuzione
- Le politiche di schedulazione **real-time** sono definite in classi di schedulazione con priorità massima
- In generale tutte le classi di schedulazione:
  - supportano i sistemi multiprocessore (**SMP**)
  - hanno overhead essenzialmente indipendente dal numero di processi attivi nel sistema



Schema della lezione

Linux e real-time

Mono- e dual-kernel

Sistemi partizionati

RTAI e ADEOS

Linux PREEMPT\_RT

## Politiche di schedulazione real-time

La classe di schedulazione real-time di *default* implementa uno scheduler con processi interrompibili a priorità fissa:

- la schedulazione dei processi real-time è (quasi) completamente distinta rispetto ai processi non real-time
- sono implementate una politica **FIFO** ed una **Round-Robin** per i processi di pari priorità
- lo scheduler è basato su una singola **coda di processi eseguibili** per ciascun processore con 100 livelli di priorità
- consente di implementare algoritmi a priorità fissa come **RM** o **DM**

Esiste un **meccanismo di sicurezza** per evitare che un processo real-time che non rilascia mai la CPU blocchi il sistema

Il **meccanismo di sicurezza** è modificabile o disabilitabile con i file “virtuali” `/proc/sys/kernel/sched_rt_period_us` e `/proc/sys/kernel/sched_rt_runtime_us`





In Linux 3.14 (2014) è stata introdotta la nuova classe di schedulazione real-time **Deadline**:

- **SCHED\_DEADLINE**, con priorità intermedia tra quelle real-time (**SCHED\_RR**, **SCHED\_FIFO**) e **SCHED\_NORMAL**
- sviluppata principalmente da Dario Faggioli e Juri Lelli (Scuola Superiore S. Anna di Pisa) dal 2009 al 2014
- sostanzialmente introduce una schedulazione con priorità dinamica a livello di task: **EDF**

Schema della lezione

Linux e real-time

Mono- e dual-kernel

Sistemi partizionati

RTAI e ADEOS

Linux **PREEMPT\_RT**

## La classe di schedulazione Deadline (2)



Schema della lezione

Linux e real-time

Mono- e dual-kernel

Sistemi partizionati

RTAI e ADEOS

Linux PREEMPT\_RT

- Ad ogni task **Deadline** sono associati i parametri  $(e, p, d)$  (in microsecondi):
  - $e$  è il **tempo di esecuzione** del task
  - $p$  è il **periodo**
  - $d$  è la **deadline relativa**
- I task sono schedulati secondo l'algoritmo **EDF**

*Come vengono calcolate le scadenze assolute?*

- Si utilizza l'algoritmo **CBS**
  - Il tempo di esecuzione è il **budget**
- Lo stato corrente di un task e' descritto da una **deadline assoluta** e dal **tempo di esecuzione residuo**
- Ogni task della classe **SCHED\_DEADLINE** è analogo a un server CBS!

## La classe di schedulazione Deadline (3)

*Perché non è stato implementato l'algoritmo EDF puro?*

Perché nel mondo reale i task possono avere comportamenti scorretti (ad es., eseguire più del tempo previsto)

Nell'implementazione `SCHED_DEADLINE`:

- Ogni task richiede allo scheduler di “riservare”  $e$   $\mu\text{s}$  ogni  $p$   $\mu\text{s}$  per la sua esecuzione
- Alla sua invocazione, lo scheduler aggiorna le scadenze e sceglie secondo l'algoritmo EDF
- Quando un task termina il proprio budget esso viene sospeso fino alla deadline usata per la schedulazione

Schedulazione EDF con isolamento temporale dei task

**Non** è una implementazione canonica di un server CBS!

Ref: *Documentation/scheduler/sched-deadline.txt*





## Problema # 1

Un processo real-time ad alta priorità può comunque essere interrotto e quindi rallentato dai gestori delle interruzioni



## Problema # 2

Linux implementa meccanismi di *bilanciamento del carico* sui sistemi multiprocessore

Schema della lezione

Linux e real-time

Mono- e dual-kernel

Sistemi partizionati

RTAI e ADEOS

Linux PREEMPT\_RT

## Bilanciamento del carico:

- Lo scheduler cerca di ripartire uniformemente il carico fra i processori effettuando **migrazioni** dei processi
- L'effetto di ciascuna **migrazione** è valutato solo in relazione al carico generato, e **non** considera le priorità
- La **migrazione** diminuisce la predicibilità di esecuzione ed aumenta la latenza a causa dei cambi di contesto aggiuntivi e della invalidazione delle memorie cache

## Il meccanismo di affinità della CPU

*Il problema della non predicibilità dovuta al bilanciamento del carico è facilmente risolvibile: come?*

Questo problema si presenta esclusivamente nei sistemi multiprocessore con **schedulazione globale** dei task

In Linux esiste un meccanismo che permette di definire l'**affinità** dei processi verso le CPU del sistema:

- Ogni processo possiede una “bitmap” che specifica i processori sui quali esso può essere eseguito
- La bitmap è ereditata dai figli del processo
- La bitmap è impostata mediante la chiamata di sistema `sched_setaffinity()`
- Da linea comando si può utilizzare il programma `taskset`

Con il meccanismo di affinità delle CPU si può realizzare un **sistema statico** in cui non si ha mai **bilanciamento del carico**





*Quali soluzioni tecniche possono consentire l'utilizzo di Linux in ambito hard real-time?*

Esistono due strade possibili, radicalmente differenti:

- 1 **Approccio mono-kernel**: modificare radicalmente il kernel Linux per eliminare tutte le cause di imprevedibilità del suo comportamento
- 2 **Approccio dual-kernel**: apportare il minor numero possibile di variazioni al codice di Linux, ma impedire l'accesso diretto ai dispositivi hardware tramite l'inserimento di uno strato di software intermedio tra il kernel e l'hardware

Schema della lezione

Linux e real-time

Mono- e dual-kernel

Sistemi partizionati

RTAI e ADEOS

Linux PREEMPT\_RT

## Approccio mono-kernel

Utilizzando l'approccio mono-kernel il codice sorgente del kernel Linux è modificato in modo molto complesso allo scopo di:

- ottenere la quasi completa **interrompibilità** del kernel
- consentire una gestione **predicibile** delle interruzioni
- proteggere i processi real-time dall'esecuzione dei gestori delle interruzioni
- implementare meccanismi di **priority inheritance**

Esempi di approcci mono-kernel applicati a Linux:

- **MontaVista Hard Hat Linux** (commerciale)
- **TimeSys Linux** (commerciale)
- Linux con patch **PREEMPT\_RT** (open-source)



## Approccio dual-kernel

Utilizzando l'[approccio dual-kernel](#) il kernel di Linux deve essere modificato in modo limitato ed in punti ben localizzati:

- L'obiettivo **non** è aumentare la predicibilità di Linux
- L'obiettivo è invece aumentare la predicibilità di un RTOS o applicazione RT in esecuzione [insieme](#) a Linux
- La chiave di volta dell'[approccio dual-kernel](#) è l'introduzione di uno strato intermedio tra l'hardware ed i sistemi operativi

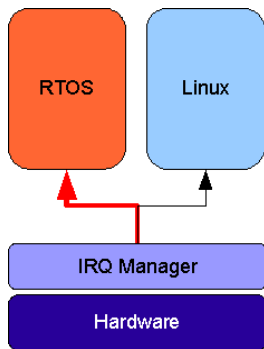
Esempi di approcci dual-kernel applicabili a Linux:

- [RTAI](#) (open-source)
- [Xenomai](#) (open-source)
- [RT-Linux](#) (open-source)
- (fin troppi RTOS commerciali da citare. . . )



## Esempio di sistema con dual-kernel

- Due SO a priorità differente sullo stesso hardware
- Un RTOS a priorità maggiore esegue i processi real-time
- Linux è attivo solo quando non vi sono processi real-time eseguibili
- Lo strato intermedio intercetta e consegna le interruzioni a Linux solo quando questo è in esecuzione
- Lo strato intermedio sostituisce sempre la gestione iniziale delle interruzioni di Linux
- È necessario quindi modificare la gestione iniziale delle interruzioni di Linux
- È possibile attivare il RTOS dopo l'inizializzazione di Linux



*Quali sono i vantaggi dell'approccio mono-kernel in Linux rispetto al dual-kernel?*



Schema della lezione

Linux e real-time

Mono- e dual-kernel

Sistemi partizionati

RTAI e ADEOS

Linux PREEMPT\_RT

- Le applicazioni real-time sono essenzialmente normali applicazioni POSIX o Linux
- L'applicazione real-time può utilizzare direttamente lo stack di rete e le comunicazioni tra processi implementate dal kernel Linux
- L'ambiente di sviluppo è molto flessibile ed efficiente
- Il progettista del sistema real-time può utilizzare il vasto insieme di driver di periferiche di Linux
- Le modifiche per adattare Linux all'ambito real-time sono gradualmente incorporate nel kernel standard  
⇒ le prestazioni del kernel standard tendono a migliorare

*Quali sono i vantaggi dell'approccio dual-kernel in Linux rispetto al mono-kernel?*



Schema della lezione

Linux e real-time

Mono- e dual-kernel

Sistemi partizionati

RTAI e ADEOS

Linux PREEMPT\_RT

- Le prestazioni delle applicazioni real-time non dipendono dal kernel Linux e sono quindi drasticamente migliori rispetto a quelle ottenibili con un mono-kernel
- È possibile utilizzare un RTOS specifico diverso da Linux, ovvero realizzare l'applicazione real-time senza supporto da alcun RTOS
- Il processo di certificazione del sistema real-time è limitato allo strato intermedio di astrazione dell'hardware ed ai componenti software real-time
- Non è necessario effettuare modifiche radicali e pervasive del kernel Linux, quindi è facile aggiornare il progetto del sistema real-time passando a nuove versioni del kernel
- Molti tipi di certificazioni richiedono in ogni caso un "partizionamento" spaziale e temporale del sistema



## Partizionamento spaziale di un sistema

- Molti sistemi real-time debbono essere **partizionati** in modo **spaziale**
- Ciò è richiesto da diversi standard di certificazione, ad esempio l'**ARINC 653** utilizzato in ambito avionico
- Il **partizionamento spaziale** del sistema è un meccanismo che consente di condividere le risorse del sistema tra più SO o applicazioni in modo che ciascuno di essi non possa interferire con il funzionamento degli altri
- Ad esempio, ciascuna **partizione** del sistema può essere associata ad una determinata porzione di RAM  
⇒ richieste eccessive di memoria da parte di una applicazione in una partizione non possono influenzare i programmi nelle altre partizioni
- Generalmente il **partizionamento spaziale** è realizzato tramite un programma (*separation kernel*, o **SK**) che crea una astrazione dell'hardware per i programmi in esecuzione nelle **partizioni**



## Partizionamento temporale di un sistema

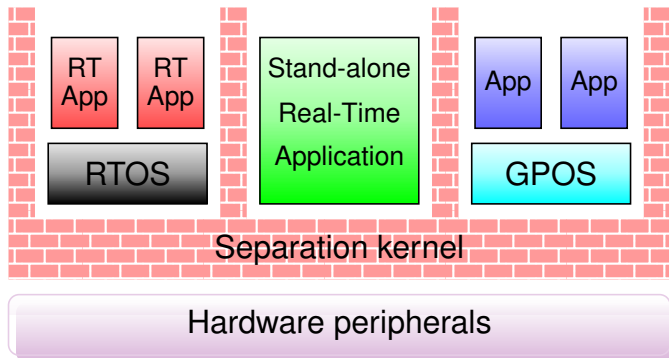
- Anche il **partizionamento temporale** di un sistema è richiesto da diverse certificazioni come **ARINC 653**
- Il **partizionamento temporale** è analogo a quello spaziale, ma la risorsa suddivisa tra le varie partizioni è il **tempo**
- Tipicamente un **separation kernel** definisce una **schedulazione ciclica** delle **partizioni**
- I SO e le applicazioni in esecuzione in ciascuna **partizione** non possono influenzare i tempi d'esecuzione delle altre partizioni, né possono ritardare la loro schedulazione

*In cosa differisce il partizionamento temporale di un RTOS dalla schedulazione dei processi di un sistema operativo generico?*

- La schedulazione dei processi è essenzialmente un meccanismo software influenzabile da politiche di schedulazione, priorità dei processi, interruzioni, ...
- In una **partizione** di un sistema real-time può essere “contenuto” un intero SO e tutte le sue applicazioni



## Esempio di sistema partizionato



Schema della lezione

Linux e real-time

Mono- e dual-kernel

Sistemi partizionati

RTAI e ADEOS

Linux PREEMPT\_RT

RTAI (Real-Time Application Interface) è un RTOS hard real-time con approccio **dual-kernel**:

- Sviluppato nel Dipartimento di Ingegneria Aerospaziale del Politecnico di Milano ([www.rtai.org](http://www.rtai.org))
- Nasce dall'esigenza di un RTOS a basso costo che consenta di utilizzare la FPU in kernel space
- Inizialmente basato su GNU-DOS, oggi su Linux
- L'ultima versione (4.1, marzo 2015) supporta le architetture x86, x86\_64, PowerPC, ARM, M68K (Coldfire)

RTAI è costituito da:

- un *separation kernel* (generalmente una versione modificata di **ADEOS**)
- un modulo di Linux che costituisce il kernel del RTOS



Schema della lezione

Linux e real-time

Mono- e dual-kernel

Sistemi partizionati

RTAI e ADEOS

Linux PREEMPT\_RT

Il **separation kernel** di **RTAI** più utilizzato è in realtà un progetto indipendente ([home.gna.org/adeos](http://home.gna.org/adeos)):

- **ADEOS: Adaptive Domain Environment for Operating Systems**
- è un nano-kernel che introduce uno strato di “virtualizzazione” tra le periferiche hardware e i sistemi operativi
- Le periferiche non sono pilotate direttamente da **ADEOS**, ma dai sistemi operativi
- **ADEOS** intercetta però le interruzioni generate dalle periferiche e le trasmette il più rapidamente possibile ai sistemi operativi
- Ogni sistema operativo è incluso in un proprio *dominio*



Schema della lezione

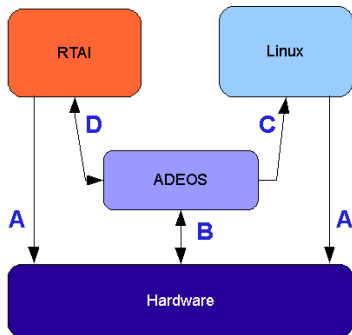
Linux e real-time

Mono- e dual-kernel

Sistemi partizionati

RTAI e ADEOS

Linux PREEMPT\_RT



Schema della lezione

Linux e real-time

Mono- e dual-kernel

Sistemi partizionati

RTAI e ADEOS

Linux PREEMPT\_RT

A : normale uso delle periferiche hardware

B : gestione iniziale delle interruzioni

C : dominio “parzialmente consapevole” di ADEOS (Linux)

D : dominio “consapevole” di ADEOS (RTAI)

## Gestione delle interruzioni con ADEOS



Schema della lezione

Linux e real-time

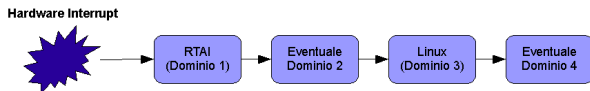
Mono- e dual-kernel

Sistemi partizionati

RTAI e ADEOS

Linux PREEMPT\_RT

- In ADEOS ogni **dominio** è associato ad una priorità
- Ogni **dominio** è inserito in una catena di consegna delle interruzioni in relazione alla sua priorità



- Ogni singola interruzione è propagata da ADEOS lungo tutta la catena
- Ciascun **dominio** può accettare l'interruzione (normale gestione)
- oppure può **bloccare** l'interruzione (ad esempio disabilitando le interruzioni)
- oppure se è consapevole della presenza di ADEOS può **ignorare** l'interruzione e/o **terminare** la sua propagazione



Schema della lezione

Linux e real-time

Mono- e dual-kernel

Sistemi partizionati

RTAI e ADEOS

Linux PREEMPT\_RT

- Sia **ADEOS** che **RTAI** sono implementati come moduli del kernel Linux
- **ADEOS** e **RTAI** possono essere attivati dopo l'inizializzazione di Linux
- **RTAI** è installato come dominio di **massima priorità** in **ADEOS**
- Linux diventa un dominio a priorità inferiore ad **RTAI**
- Al kernel Linux viene aggiunto il supporto per la gestione della catena di interruzioni di **ADEOS**

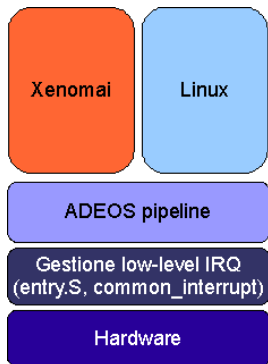
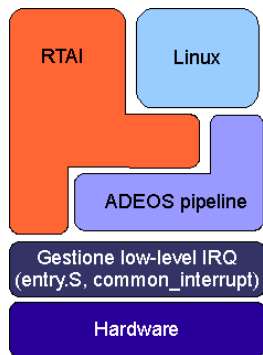
In realtà con **RTAI** si utilizza una versione modificata di **ADEOS** che consente una gestione più veloce delle interruzioni





Gestione veloce di RTAI

ADEOS standard (es. Xenomai)



Schema della lezione

Linux e real-time

Mono- e dual-kernel

Sistemi partizionati

RTAI e ADEOS

Linux PREEMPT\_RT

## Schedulazione in RTAI

Lo scheduler di **RTAI** è sia *clock-driven* che *event-driven*:

Il meccanismo di schedulazione viene attivato:

- alla ricezione di una interruzione di timer
- all'uscita di una **ISR**
- alla sospensione spontanea di un task

In **RTAI** è possibile schedulare processi di Linux (**LXRT**):

- sono stati “promossi” hard real-time e sottratti allo scheduler di Linux
- **non** possono utilizzare chiamate di sistema di Linux

È “facile” realizzare una applicazione real-time che dialoga con il sistema Linux creando un programma multi-thread in cui alcuni thread sono assegnati ad **RTAI** ed altri a Linux!



## Schedulazione in RTAI (2)

In RTAI è anche possibile avere task “nativi”:

- Sono simili a **kernel thread** schedulati esclusivamente da **RTAI**
- Offrono un cambio di contesto molto veloce, ma non hanno protezione della memoria e dello spazio di indirizzamento
- Non è facile integrarli con le funzionalità del kernel Linux
- Generalmente vengono affiancati ad un task **LXRT** per la comunicazione con il kernel Linux

Nei sistemi multiprocessore:

- Ogni task è forzato ad eseguire su una sola CPU
- La CPU può essere determinata dall'utente o automaticamente da **RTAI**

**RTAI** supporta gli algoritmi di schedulazione **FIFO**, **Round Robin**, **Rate Monotonic (RM)** e **Earliest Deadline First (EDF)** e alcuni meccanismi di **priority inheritance**



Il principale progetto open-source per realizzare un RTOS mono-kernel basato su Linux è la [patch PREEMPT\\_RT](#) ([rt.wiki.kernel.org](http://rt.wiki.kernel.org))



- È la continuazione della patch MontaVista di Ingo Molnar per rendere il kernel pienamente interrompibile
- Il suo sviluppo procede di pari passo con quello del kernel Linux standard
- I suoi meccanismi vengono gradualmente inseriti all'interno del kernel standard
- Ha due obiettivi principali:
  - ridurre la dimensione delle sezioni non interrompibili del kernel
  - permettere l'esecuzione dei gestori delle interruzioni in un contesto schedulabile con priorità confrontabili con quelle dei processi

Schema della lezione

Linux e real-time

Mono- e dual-kernel

Sistemi partizionati

RTAI e ADEOS

Linux PREEMPT\_RT

[Schema della lezione](#)[Linux e real-time](#)[Mono- e dual-kernel](#)[Sistemi partizionati](#)[RTAI e ADEOS](#)[Linux PREEMPT\\_RT](#)

## I meccanismi di PREEMPT\_RT

L'idea fondamentale della patch `PREEMPT_RT` consiste nella sostituzione di alcuni meccanismi di sincronizzazione:

- Molte primitive del kernel standard disabilitano l'interrompibilità e iterano in un ciclo attendendo che una variabile di condizione cambi stato
- Queste primitive sono sostituite da `rt-mutex` che possono sospendere l'esecuzione del processo corrente
- Gli `rt-mutex` implementano un sofisticato protocollo di `priority inheritance`

Inoltre:

- l'operazione di acquisizione di uno spinlock può essere interrotta: disabilitare le interruzioni non basta!
- i gestori di interruzioni sono eseguiti all'interno di `kernel thread`

Alcuni tipi di interruzioni dalla durata breve e predicibile continuano ad essere eseguiti in modo "tradizionale"

## PREEMPT\_RT e sistemi multiprocessori

Abbiamo visto che il modo migliore per garantire la predicibilità di un sistema multiprocessore consiste nel vincolare l'esecuzione di ciascun task ad uno specifico processore

Tuttavia la patch **PREEMPT\_RT** cerca di mitigare gli effetti del bilanciamento del carico sulle CPU modificandone il funzionamento

- Il bilanciamento non può migrare un processo mentre esso è in esecuzione
- Il bilanciamento viene effettuato solo prima di mettere in esecuzione un processo o subito dopo la terminazione della sua esecuzione
- Il kernel cerca di non sovraccaricare un processore con processi di priorità elevata (i processi di priorità elevata sono distribuiti su più processori)
- Al risveglio di un processo ad alta priorità che dovrebbe interrompere il processo corrente si valuta euristicamente se è meglio migrarlo su un altro processore

