RTAI

Andrea Sambi

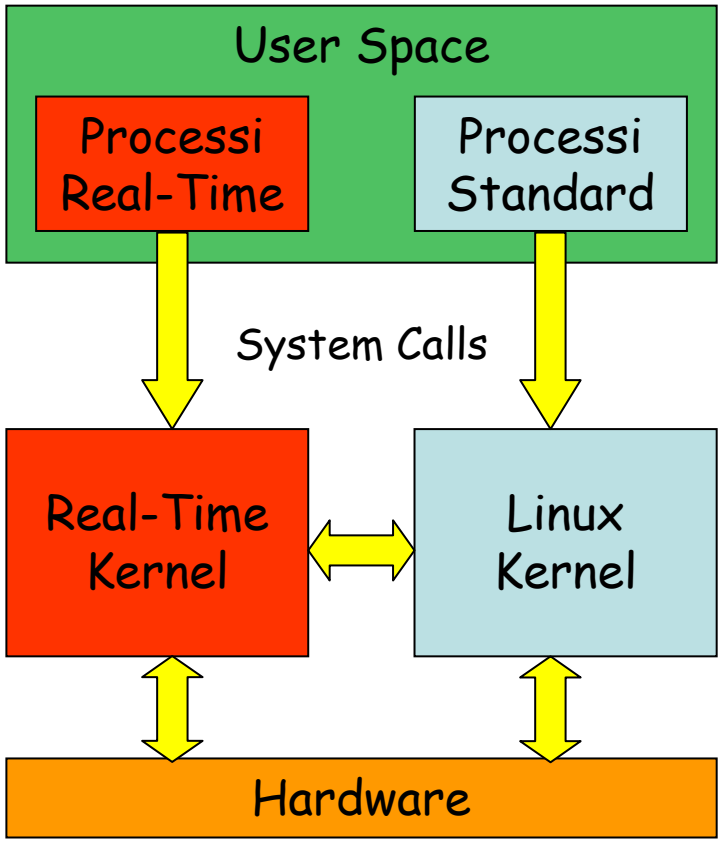
RTAI: un po' di storia

- Real-Time Application Interface
- Sviluppato presso il Politecnico di Milano
- Nato per rispondere alla richiesta di un sistema operativo a basso costo da utilizzarsi per diverse applicazioni di tipo hard real-time
- 1996/1997: abbandono di DOS come SO real-time, studio di fattibilità per estendere il kernel Linux ed utilizzarlo in ambito real-time
- 1999: prima release sotto il nome RTAI, funzionante con Linux kernel versione 2.2

RTAI: panoramica

Real-Time Application Interface: in quale direzione ha evoluto il kernel Linux?

Prevede l'introduzione di un nuovo layer tra hardware e kernel Linux.



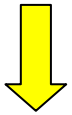
Ha richiesto modifiche limitate al kernel standard.

Le prime versioni di RTAI comportavano la modifica/riscrittura di circa 70 righe di codice.

RTAI: architettura

Si introduce un nuovo layer tra il kernel Linux e il sistema hardware: RTHAL (Real-Time Hardware Abstraction Layer).

- Racchiude tutti i dati e le funzioni temporalmente critiche del kernel in un'unica struttura: permette di catturare facilmente le funzionalità del kernel (interrupts, system calls, timers) per poterle gestire in accordo a politiche real-time
- Sostituisce le operazioni sulle strutture originali con operazioni su puntatori RTHAL
- I puntatori RTHAL sono modificabili dinamicamente. Se RTAI non è attivo puntano alle strutture originali di Linux, se RTAI è attivo puntano alle strutture del kernel real-time

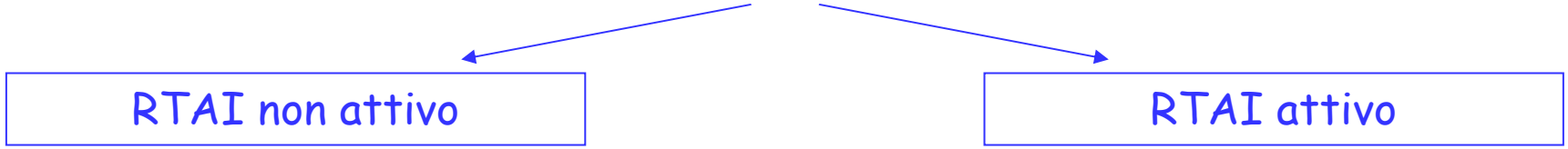


Linux non ha più il controllo sull'abilitazione / disabilitazione delle interruzioni.

RTAI: architettura

RTHAL non fornisce servizi real-time: ha la sola funzione di intercettare le chiamate al kernel Linux.

Le chiamate sono redirette alle strutture puntate da RTHAL.



Funzionamento normale di Linux.

Solo funzionalità real-time possono accedere direttamente all'hardware. Linux è gestito come un processo a bassa priorità.

Quindi RTAI può essere attivato o disattivato a piacere?

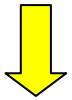
SÌ…perché RTAI è disponibile come modulo kernel di Linux.

Permette di estendere dinamicamente le funzionalità del kernel senza dover essere caricato al boot di sistema.

RTAI: architettura

L'idea di avere uno strato software che intercetta le chiamate al SO è stata ulteriormente sviluppata nel tempo.

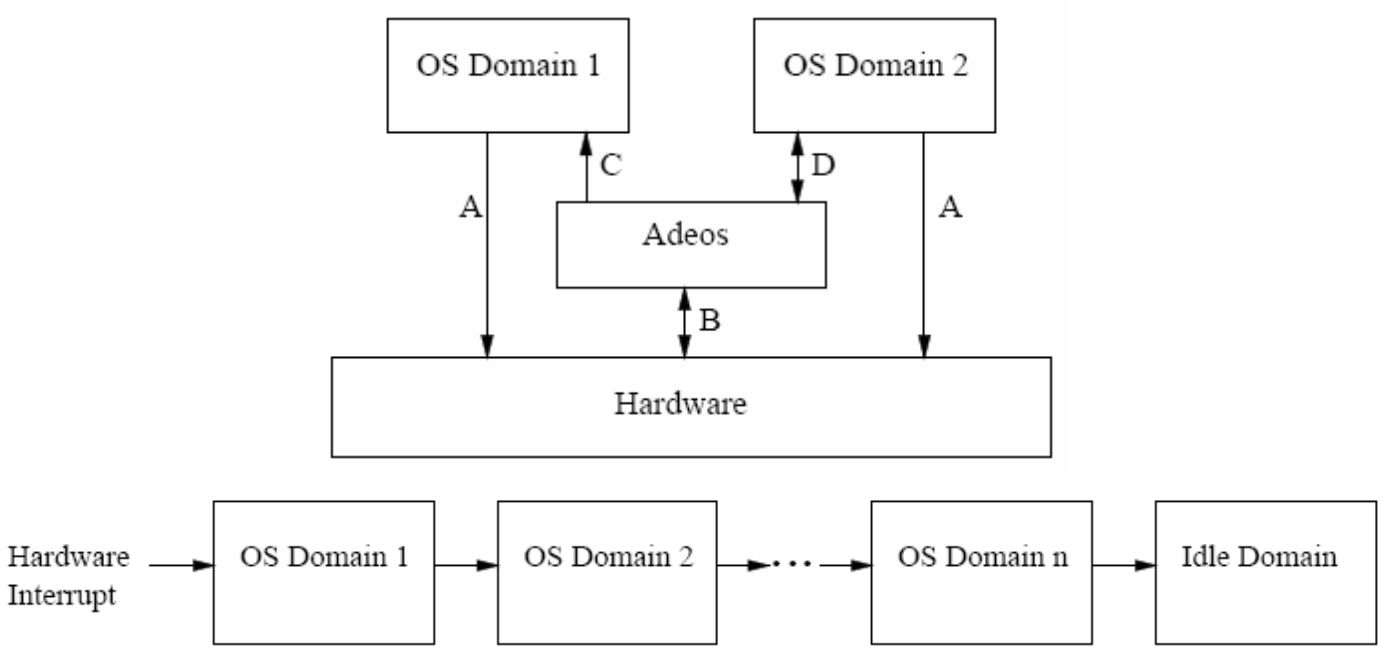
Si è cercato di rendere possibile la strutturazione gerarchica di sistemi operativi componendoli in strati diversi.



ADEOS (Adaptive Domain Environment for Operating Systems)

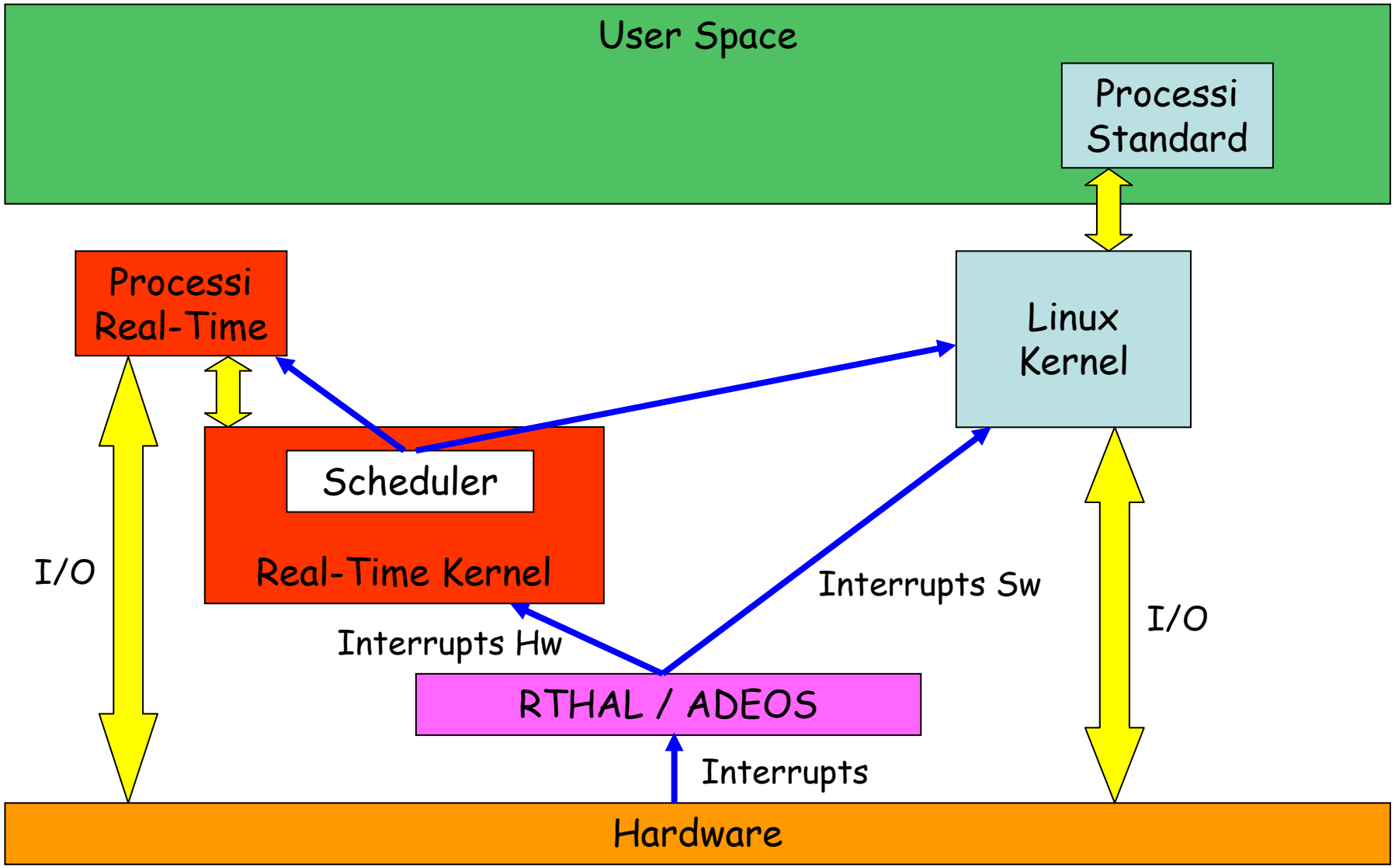
- Scopo: realizzare un ambiente flessibile per condividere risorse hardware tra più sistemi operativi o più istanze di uno stesso SO
- La realizzazione consiste in un microkernel che gestisce la comunicazione con i diversi domini (es. SO) installati
- La gestione delle interruzioni è implementata con uno schema a pipeline in cui ogni stadio rappresenta un dominio
- Ogni interruzione è propagata alla pipeline, ogni stadio può:
  - Accettare l'interrupt, gestirlo, scegliere se propagarlo o terminarlo
  - Ignorare l'interrupt, non accettarlo immediatamente ma gestirlo al momento opportuno, scegliere se propagarlo o terminarlo
  - Scartare l'interrupt e propagarlo
  - Terminare l'interrupt senza propagarlo ulteriormente

RTAI: architettura



- Il meccanismo di ADEOS racchiude tutto ciò che RTAI inizialmente otteneva con RTHAL
- RTAI viene installato come dominio alla massima priorità per ADEOS, Linux è caricato in uno stadio meno prioritario
- RTAI gestisce gli interrupt prima del kernel Linux ed opera in accordo alle esigenze dei processi real-time

RTAI: architettura



RTAI: scheduling

RTAI permette una gestione full-preemptable dei processi, in funzione delle loro priorità.

Configurazione dello scheduler

- Funzione del numero di processori presenti
- Scelta della modalità di funzionamento dello scheduler
- Scelta della politica di scheduling

Ogni possibile configurazione dello scheduler è adatta ad una specifica combinazione di hardware e requisiti delle applicazioni.

RTAI: scheduling

La scelta dello scheduler in funzione dell'hardware in uso impatta su quale modulo kernel viene caricato con RTAI.

- Uniprocessor Scheduler (UP)
  - Utilizzabile nei sistemi monoprocessore
- Symmetric Multiprocessor Scheduler (SMP)
  - In un sistema multiprocessore permette una distribuzione di carico simmetrica.
  - Ogni processo di default può essere assegnato a qualsiasi processore; per ottimizzare l'esecuzione è possibile imporre l'esecuzione di un processo su una CPU o su un insieme ristretto di CPU
- Multi Uniprocessor Scheduler (MUP)
  - In un sistema multiprocessore impone ad ogni processo l'esecuzione su una CPU stabilita al momento della sua creazione. Meno flessibile ma più efficiente di SMP

RTAI: scheduling

Occorre scegliere la temporizzazione con cui lo scheduler viene eseguito.

- Periodic mode
  - Esegue lo scheduler periodicamente al termine di un periodo prefissato: il timer viene settato una sola volta all'inizio dell'esecuzione
  - Il periodo dei processi deve essere multiplo esatto del periodo dello scheduler, in caso contrario il periodo dei processi è approssimato al multiplo del periodo dello scheduler più vicino (introduce un jitter di attivazione)
- One-shot mode
  - Lo scheduler viene eseguito in maniera non periodica. Il timer deve essere settato ogni volta in base al processo più prioritario che andrà in esecuzione
  - Permette una gestione più flessibile delle temporizzazioni di tutti i processi a costo di un maggiore overhead dovuto alla necessità di riprogrammare il timer al termine di ogni periodo

RTAI: scheduling

Politiche di scheduling

- FIFO
  - Il processo attivo a priorità più alta ottiene il controllo della CPU fino a quando la rilascia volontariamente oppure diventa attivo un processo a priorità maggiore
- Round-Robin (RR)
  - Il processo attivo a priorità più alta ottiene il controllo della CPU per un determinato intervallo di tempo, al termine del quale il controllo passa ad un altro processo allo stesso livello di priorità (se presente). Un processo può subire preemption da parte di un processo a priorità maggiore

RTAI integra primitive che associate alla politica FIFO permettono una semplice implementazione degli algoritmi di scheduling Rate Monotonic Priority Order ed Earliest Deadline First.

RTAI: Inter-Process Communication (IPC)

- Real-time fifos
  - Meccanismo di base per scambiare dati in modo asincrono tra processi real-time e processi Linux non real-time
- Shared Memory
  - Condivide aree di memoria tra processi RT e processi Linux
- Messages
  - Possibilità di inviare messaggi sia in maniera asincrona che sincrona (RPC) tra processi RT
- Mailboxes
  - Permettono di inviare/ricevere messaggi di qualsiasi dimensione, ordinati per priorità o per istante di arrivo, tra processi RT e processi Linux
- Semaphores
  - Permettono di sincronizzare i processi nell'accesso a risorse condivise evitando inversioni di priorità incontrollate

RTAI: ulteriori funzionalità

- Scarso supporto alle estensioni real-time dello standard POSIX
  - Parzialmente supportati gli standard 1003.c (pthread, mutex) e 1003.b (pqueue)
- LXRT
  - I processi RTAI nativi eseguono in kernel mode: le API originarie di RTAI sono utilizzabili solo da moduli kernel
  - LXRT permette di sviluppare processi real-time usando le API RTAI da spazio utente (si è cercato di mantenere simmetria tra le API in spazio kernel e quelle in spazio utente)
  - Permette un passaggio più graduale dai processi Linux ai processi real-time
  - Permette una maggiore protezione dell'hardware (in spazio kernel si può accedere liberamente alla memoria)
  - Utile in fase di test e prototipazione, meno efficiente rispetto alla programmazione in spazio kernel

Esempio: un modulo kernel RTAI

Parte dichiarativa

```
#include <rtai.h>
#include <rtai_sched.h>
...
RT_TASK task;
...
```

Parte Real-Time

```
void task_routine() {
  while(1) {
/* Codice real-time */
    rt_task_wait_period();
  }
}
...
```

Setup del modulo

```
int init_module(void) {
  ...
  rt_task_init(&task, task_routine, 0, stack_size, priority, 0, 0);
  timer_period = start_rt_timer(nano2count(1e9));
  task_period = 2*timer_period;
  rt_task_make_periodic(&task, now, task_period);
}
void cleanup_module(void) {
  rt_task_delete(&task);
  stop_rt_timer();
}
```

Esempio: un processo RTAI LXRT

Parte dichiarativa

Inizializzazione del processo

Parte Real-Time

Terminazione del processo

```
#include <rtai_lxrt.h>
...
int main(void) {
  RT_TASK *task;
  ...
  sched_setscheduler(0, SCHED_FIFO, &priorita);
  mlockall(MCL_CURRENT | MCL_FUTURE);
  task = rt_task_init(nam2num("Name"), priority, stack_size, 0);
  timer_period = start_rt_timer(nano2count(1e9));
  task_period = 2*timer_period;
  ...
  rt_make_hard_real_time();
  rt_task_make_periodic(task, now, task_period);
  while(1) {
/* Codice real-time */
    rt_task_wait_period();
  }
  rt_make_soft_real_time();
  rt_task_delete(task);
  return 0;
}
```

RTAI: installazione

Passi principali per l'installazione di RTAI

- Installare Linux
  - Installare una distribuzione di Linux (Fedora, Mandrake, Slackware, Gentoo, …)
- Scaricare i file sorgenti del kernel e una versione di RTAI compatibile con essi
  - È preferibile scaricare i sorgenti "vanilla" del kernel (www.kernel.org)
  - Scaricare una versione di RTAI (www.rtai.org) che contenga la patch adatta al kernel scaricato
  - Esempio: la patch di nome hal-linux-2.6.14-i386-1.1-02.patch che si trova nella cartella /rtai-3.3/base/arch/i386/patches/ permette di installare RTAI 3.3 su un kernel versione 2.6.14
- Patch del kernel
  - Dalla cartella dei sorgenti del kernel:
    patch –p1 < ../rtai-3.3/base/arch/i386/patches/hal-linux-2.6.14-i386-1.1-02.patch

RTAI: installazione

- Configurare il kernel
  - Utilizzare il file .config della distribuzione in uso come base per configurare il kernel:
    cp /boot/config-versione_in_uso .config
  - make menuconfig
  - Code maturity level → Prompt for development and/or incomplete drivers → YES
    Loadable module support → Enable loadable module support → YES
    Module versioning support → NO
    Processor type and features → Preemptible kernel → NO
    Processor type and features → Use register arguments → NO
    Processor type and features → Interrupt pipeline (IPIPE) → YES
    File systems → Pseudo filesystems → /proc file system support → YES

RTAI: installazione

- Compilare ed installare il kernel
  - make
  - make modules_install
  - mkinitrd /boot/initrd-versione_kernel.img vesione_kernel
  - cp arch/i386/boot/bzImage /boot/vmlinuz-versione_kernel
  - cp System.map /boot/System.map-versione_kernel
  - ln –s /boot/System.map-versione_kernel /boot/System.map
  - Editare il file di configurazione del boot-loader (es. /boot/grub/menu.lst) aggiungendo una entry per caricare il nuovo kernel
  - Riavviare caricando il nuovo kernel

RTAI: installazione

- Configurazione ed installazione di RTAI
  - Dalla cartella di RTAI:
    make menuconfig
  - Selezionare la cartella dei sorgenti del kernel preparato con la opportuna patch
  - make
  - make install
  - Riavviare
- Test
  - cd cartella_di_installazione_RTAI/testsuite/user/latency
  - ./run

Documentazione

- DIAPM RTAI for Linux, P. Mantegazza
- Linux e real-time, D. Ciminaghi
- Linux Real Time Application Interface (RTAI) in low cost high performance motion control, L. Dozio – P. Mantegazza
- Real-Time and Embedded Guide, H. Bruyninckx
- RTAI: Real-Time Application Interface, P.S. Hughes - D. Beal
- www.rtai.org, www.aero.polimi.it/~rtai, www.fdn.fr/~brouchou/rtai/rtai-doc-prj, www.rtai.dk, www.rts.uni-hannover.de/rtai/lxr/source, www.realtimelinuxfoundation.org