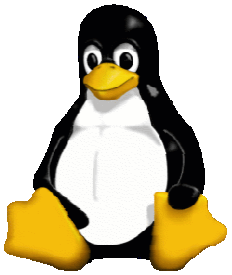
Linux nei sistemi Real-Time

Andrea Sambi

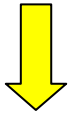


Sistemi Real-Time

Sistema Real-Time (RT) non è sinonimo di "sistema veloce".

Un Processo Real-Time deve terminare rispettando i vincoli temporali (le deadline) stabiliti in modo tale che la sua esecuzione abbia senso.

Un Sistema Real-Time deve essere in grado di prevedere se i processi che andrà ad eseguire siano in grado di rispettare le rispettive deadline.



Un Sistema Real-Time deve essere un sistema deterministico, che sia in grado di garantire a priori il corretto funzionamento di un preciso insieme di processi.

Sistemi Real-Time

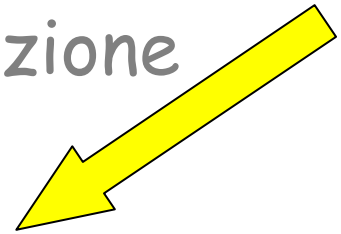
Cosa influenza il determinismo dell'esecuzione dei processi nei moderni sistemi di elaborazione?

- Caratteristiche architetturali dell'elaboratore
- Linguaggio di programmazione
- Sistema Operativo

Sistemi Real-Time

Cosa influenza il determinismo dell'esecuzione dei processi nei moderni sistemi di elaborazione?

- Caratteristiche architetturali dell'elaboratore
- Linguaggio di programmazione
- Sistema Operativo



DSP o microcontrollori DSP-like:

- CPU dedicate con latenza di risposta agli interrupt garantita
- Costo elevato

CPU general purpose:

- Frequenza di clock elevata
- Capacità di implementare schemi di controllo complessi con calcoli floating point
- Costo relativamente basso

Sistemi Real-Time

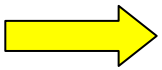
L'architettura degli elaboratori general purpose introduce elementi che possono minare il determinismo di un sistema.

- Arbitrato del bus causato da dispositivi di I/O "intelligenti"
- Cache multi-livello
- Memoria virtuale e unità di gestione della memoria (MMU)
- Pipeline di esecuzione delle istruzioni e predizione dei branch
- Gestione delle interruzioni generate da dispositivi periferici

Sistemi Real-Time

L'architettura degli elaboratori general purpose introduce elementi che possono minare il determinismo di un sistema.

- Arbitrato del bus causato da dispositivi di I/O "intelligenti"
- Cache multi-livello
- Memoria virtuale e unità di gestione della memoria (MMU)
- Pipeline di esecuzione delle istruzioni e predizione dei branch
- Gestione delle interruzioni generate da dispositivi periferici



DMA: utilizzato per trasferire dati dalle periferiche alla memoria senza gravare sulla CPU.

Occorre bloccare la CPU durante i trasferimenti di DMA.

Tecnica di trasferimento DMA a suddivisione di tempo:

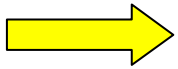
un ciclo di accesso alla memoria viene suddiviso in due intervalli, uno per la CPU uno per il DMA.

Perdita di efficienza, maggiore prevedibilità.

Sistemi Real-Time

L'architettura degli elaboratori general purpose introduce elementi che possono minare il determinismo di un sistema.

- Arbitrato del bus causato da dispositivi di I/O "intelligenti"
- Cache multi-livello
- Memoria virtuale e unità di gestione della memoria (MMU)
- Pipeline di esecuzione delle istruzioni e predizione dei branch
- Gestione delle interruzioni generate da dispositivi periferici



Elementi soggetti a ritardi anche superiori al tempo di esecuzione delle istruzioni. Introducono un fattore di aleatorietà.

Ogni cache fault provoca l'accesso in memoria. Ogni page fault provoca l'accesso al disco. Ogni predizione sbagliata provoca la perdita della pipeline.

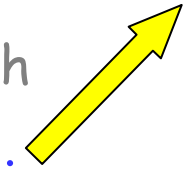
Occorre considerare il caso peggiore (es. cache fault per ogni accesso in memoria) che è improbabile ma possibile.

E' preferibile non avere memoria cache o disabilitarla.

Sistemi Real-Time

L'architettura degli elaboratori general purpose introduce elementi che possono minare il determinismo di un sistema.

- Arbitrato del bus causato da dispositivi di I/O "intelligenti"
- Cache multi-livello
- Memoria virtuale e unità di gestione della memoria (MMU)
- Pipeline di esecuzione delle istruzioni e predizione dei branch
- Gestione delle interruzioni generate da dispositivi periferici



Tipicamente nei sistemi non real-time le routine di servizio delle interruzioni sono prioritarie rispetto alle applicazioni.

Può non essere vero nei sistemi RT.

- Disabilitare le interruzioni esterne (eccetto quella del timer). Gestione delle periferiche a carico delle applicazioni (polling dei dispositivi esterni), oppure a carico di routine di sistema periodiche.

- Affidare alle routine di servizio solo il compito di attivare un opportuno processo di gestione. Tale processo viene schedulato come un altro processo applicativo.

Sistemi Real-Time

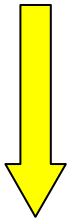
L'architettura degli elaboratori general purpose introduce elementi che possono minare il determinismo di un sistema.

- GPCPU hanno caratteristiche, tecnologie e meccanismi che portano latenza e jitter non deterministici
- Accorgimenti per limitare questi problemi non eliminano il ritardo architetturale: il ritardo per un cambio di contesto (preemption latency) è superiore a 10 µs, fino a 20 µs
- CPU multi-core possono dedicare una CPU al controllo real-time ottenendo valori di preemption latency paragonabili a quelli ottenibili con hw DSP: qualche µs
- In questi casi si possono sviluppare algoritmi con una frequenza massima del controllo fino a 50 KHz (periodo 20 µs)
- Esistono altri problemi legati all'hardware utilizzabile: es. alcune schede video occupano il bus per "lungo" tempo. Anche con processori dual-core non è sempre possibile implementare controllo ed interfaccia utente sulla stessa macchina

Sistemi Real-Time

Cosa influenza il determinismo dell'esecuzione dei processi nei moderni sistemi di elaborazione?

- Caratteristiche architetturali dell'elaboratore
- Linguaggio di programmazione
- Sistema Operativo



Per poter trattare applicazioni real-time il linguaggio di programmazione ad alto livello deve prevedere esplicitamente la gestione dei vincoli temporali.

Sistemi Real-Time

Rispetto ai Sistemi Operativi general purpose nel nucleo dei SO Real-Time hanno particolare rilevanza le funzionalità riguardanti:

- La gestione dei processi
- La gestione delle interruzioni
- La sincronizzazione dei processi
- La mutua esclusione fra le risorse condivise

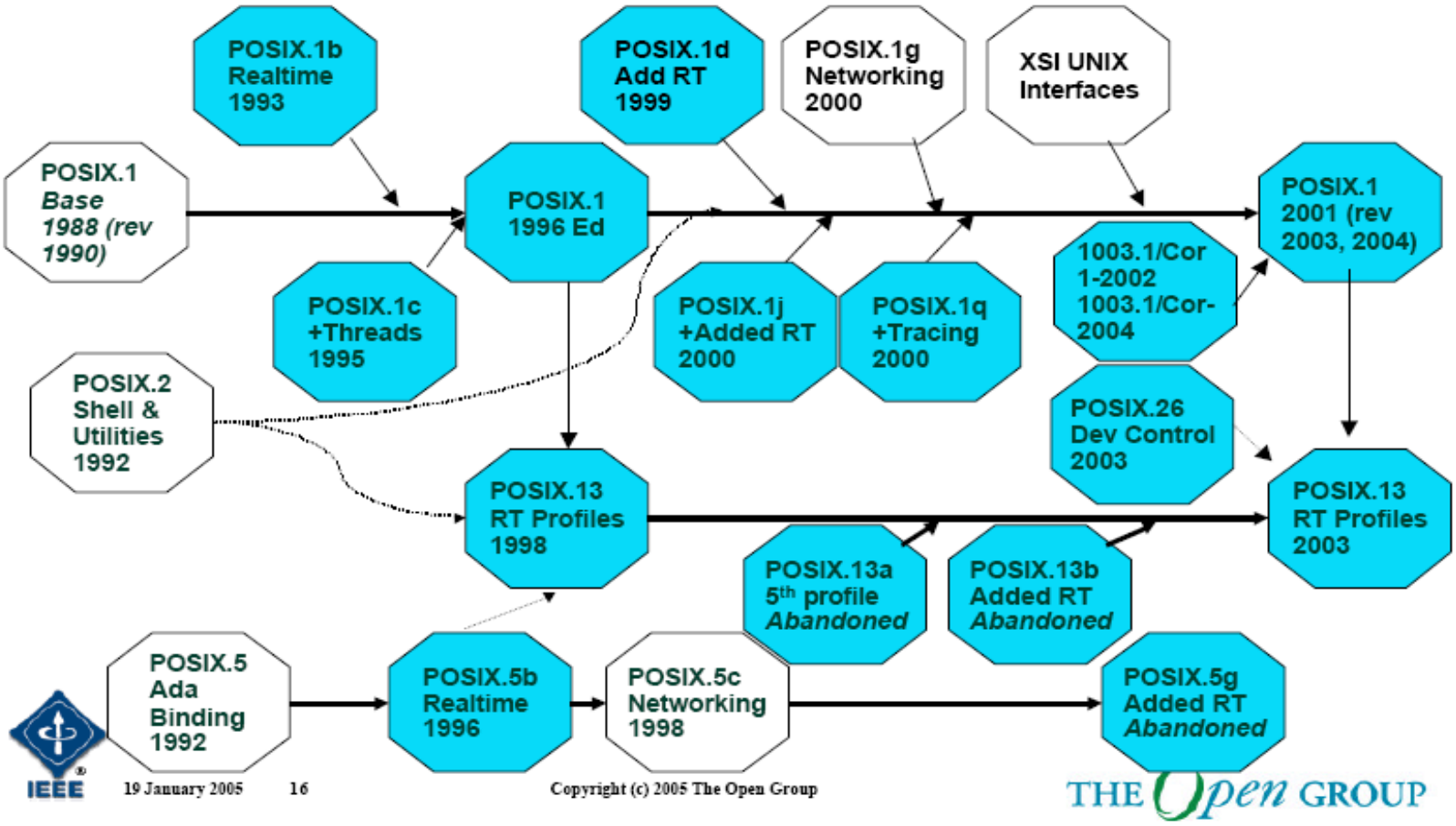
Partendo dallo standard POSIX per sistemi operativi general purpose sono state definite alcune estensioni specifiche per i sistemi RT.

La maggior parte dei sistemi embedded che operano secondo modalità RT usano SO proprietari progettati con l'unico scopo di soddisfare i requisiti di una particolare applicazione, quindi sono poco interessati alla definizione di standard.

Lo standard POSIX (Portable Operating System Interface)

- Standard prodotto dalla IEEE
- Standardizza le funzioni di interfaccia col SO (API), i comandi del SO, i metodi di test
- Non è un sistema operativo e non indica modalità di implementazione
- Permette la massima portabilità del codice
- Può essere implementato anche solo in parte
- POSIX 1003.1a: standard per le interfacce di base
- POSIX 1003.1c: standard per la gestione dei thread
- Sono state definite estensioni per sistemi real-time: POSIX 1003.1b, 1003.1d, 1003.1j (1003.1b è l'unico comunemente implementato)

POSIX Real-Time Roadmap



Sistemi Operativi Real-Time (RTOS)

Quale Sistema Operativo Real-Time scegliere?

POSIX-conformant (completamente conforme allo std) →

- LynuxWorks LynxOS Embedded: la certificazione lo favorisce nell'impiego in ambienti governativi

POSIX-compliance (supporto parziale dello std) →

- Wind River VxWorks: prestazioni elevate. RTOS più diffuso al mondo
- QNX Neutrino: alcune soluzioni innovative basate sulla struttura a micro-kernel. Offre una maggiore protezione del sistema

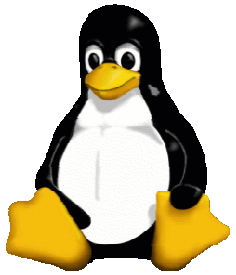
Scarso supporto POSIX →

- Microsoft Windows CE: prestazioni non eccezionali e l'ambiente di sviluppo Visual Studio lo rendono più adatto a sistemi che integrano anche l'interfaccia grafica, piuttosto che a sistemi embedded molto esigenti

Perché non usare Linux?

Linux: un po' di storia

- 1991: Linus Torvalds (studente alla University of Helsinki) inizia lo sviluppo di Linux
- Cerca di inglobare le caratteristiche ritenute positive di UNIX
- 1994: Linux 1.0 (sistemi i386 uniprocessore)
- 1995: Linux 1.2 (supporto per altre architetture)
- 1996: Linux 2.0 (supporto per sistemi multiprocessore)
- 1999: Linux 2.2
- 2001: Linux 2.4
- 2003/2004: Linux 2.6



Linux: caratteristiche

- SO general purpose
- SO UNIX-like: kernel monolitico, moduli kernel caricabili a richiesta
- Free, utilizzabile gratuitamente
- Open-source: disponibilità del codice sorgente, possibilità di personalizzazioni
- Portabilità su numerose CPU differenti
- Disponibilità di strumenti di sviluppo gratuiti
- Ampia documentazione
- Meno intuitivo dei sistemi Windows

Linux: capacità

- Stabile, robusto, affidabile
- Buon supporto dei protocolli di rete
- Vicino allo standard POSIX: l'obiettivo è essere POSIX-compliance (non obbligatoriamente, infatti alcune API POSIX sono fornite come funzioni di libreria)
- Buone prestazioni come sistema server/desktop

E il supporto real-time?

Linux ed il Real-Time

Linux kernel fino alla versione 2.4

- Implementazione di alcune estensioni real-time previste dallo standard POSIX
- Insufficienti per garantire la corretta gestione di processi real-time

Linux kernel versione 2.6

- Evoluzione del kernel, caratteristiche più adatte ai sistemi real-time (almeno soft real-time)

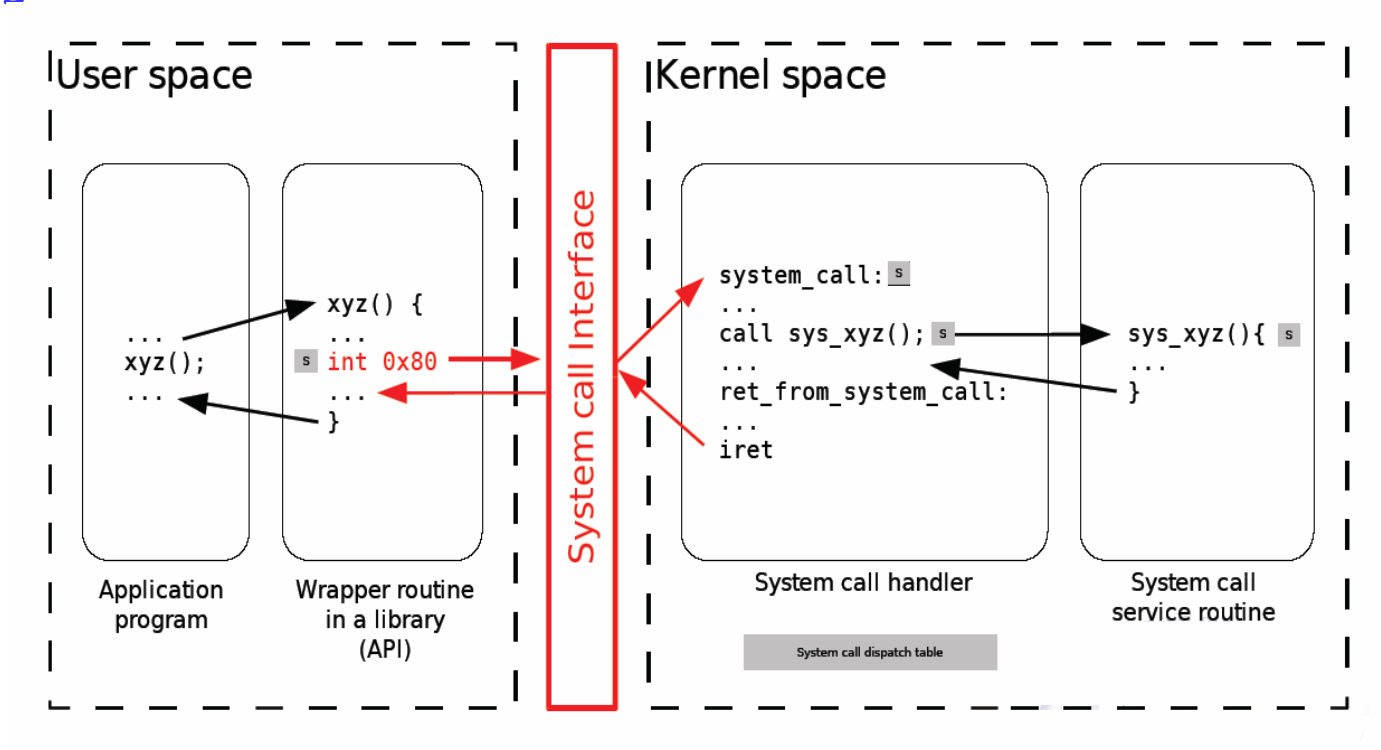
Linux kernel

Le modalità di esecuzione dei processi sono due:

- User mode
  - Modalità di esecuzione dei programmi utente
  - Sono adottate politiche di sicurezza per evitare l'accesso a zone critiche del sistema
- Kernel mode
  - Modalità di esecuzione privilegiata
  - Permette di accedere incondizionatamente a tutte le risorse del sistema
  - Modalità di esecuzione delle chiamate di sistema (System Calls)

System Calls

- Strato software con cui i processi utenti possono accedere alle risorse hardware
- Funzioni con cui il SO (kernel Linux) fornisce servizi all'utente
- La modalità di attivazione di una system call è quella di un interrupt software
- Eseguite in modalità kernel



Linux scheduling

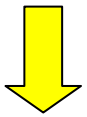
E' possibile effettuare preemption sui processi utente.

Ad ogni processo sono associate:

- una politica di scheduling
- una priorità

Politiche di scheduling (POSIX standard)

- Processi tradizionali: OTHER: politica di default, priorità statica pari a 0
- Processi Real-Time: { ROUND-ROBIN (RR), FIFO } — Priorità statica compresa tra 1 e 99



Processi schedulati con Fifo o RR sono prioritari rispetto ai processi tradizionali.

Scheduling FIFO

- Ogni processo ha solo un valore di priorità statica
- I processi al medesimo livello di priorità sono accodati secondo l'istante di attivazione (FCFS)
- Il processore è assegnato al processo pronto a priorità maggiore
- Il processo in esecuzione perde l'uso della CPU solo se termina, se si sospende o se c'è un processo pronto a priorità maggiore
- Quando un processo perde l'uso della CPU è posto in testa alla lista dei processi pronti al suo livello di priorità (eccetto in caso di terminazione)

Scheduling Round-Robin (RR)

- Per ogni processo è definito un valore di priorità statica e un periodo di tempo massimo durante il quale è assegnato alla CPU (time quantum)
- Il processore è assegnato al processo pronto a priorità maggiore
- Il processo in esecuzione perde l'uso della CPU se termina, se si sospende, se c'è un processo pronto a priorità maggiore o dopo aver esaurito il proprio time quantum
- Quando un processo termina il proprio quanto di tempo viene posto alla fine della lista dei processi pronti al suo livello di priorità
- Quando un processo subisce preemption da parte di un processo più prioritario è posto in testa alla lista dei processi pronti al suo livello di priorità

Scheduling Other

- Ad ogni processo è assegnato un quanto di tempo massimo di uso della CPU ed una priorità dinamica data dalla somma di un valore di base e di un valore che decresce all'aumentare del tempo di CPU utilizzato
- Aggiornamento dinamico della priorità: quando tutti i quanti di tempo dei processi pronti sono esauriti vengono ricalcolate le priorità di tutti i processi
- Ha lo scopo di garantire un'equa ripartizione della CPU tra tutti i processi, e di fornire buoni tempi di risposta per i processi interattivi

Gestione dello scheduler

Alcune primitive (System Calls) per la gestione dello scheduler (sched.h)

- sched_setscheduler(…)
  - Setta l'algoritmo di scheduling (SCHED_FIFO, SCHED_RR, SCHED_OTHER) e la priorità statica per un processo. Richiede i privilegi di root.
- sched_getscheduler(…)
  - Restituisce l'algoritmo di scheduling di un processo
- sched_setparam(…)
  - Setta la priorità statica di un processo
- sched_getparam(…)
  - Ricava la priorità statica di un processo
- sched_rr_get_interval(…)
  - Restituisce il valore del quanto di tempo assegnato ad un processo
- sched_get_priority_min(…)
  - Restituisce il minimo valore di priorità possibile per un algoritmo di scheduling
- sched_get_priority_max(…)
  - Restituisce il massimo valore di priorità possibile per un algoritmo di scheduling
- sched_yield()
  - Fa assumere al processo chiamante lo stato READY

Esempio: scheduling FIFO/RR

Schedulazione di 3 processi che eseguono un ciclo di stampe a video.

Processo padre: crea i 3 processi che saranno schedulati con gli algoritmi "real-time" messi a disposizione da Linux.

```
#include <sched.h>
...
#define ...
...
int main(void) {
  int count;
  int fork_return;

/* Crea i processi figli */
  for (count = 1; count <= NO_OF_CHILDREN; count++) {
    if ((fork_return = fork()) == 0) {
      new_child(count);
    }
    else ...
```

Esempio: scheduling FIFO/RR

Processi figli: cambiano algoritmo di scheduling (OTHER di default) e avviano un ciclo di stampe a video.

L'algoritmo di scheduling e la priorità dei processi sono definiti tramite #define

```
void new_child(int child_number) {
  int i;
  struct sched_param priority;

  priority.sched_priority = PRIORITA;
  if (sched_setscheduler(0, ALGORITMO_SCHEDULING, &priority) < 0) {
    printf("Errore nell'impostazione dello scheduler (devi essere root)\n");
    exit(-1);
  }
  usleep(SLEEP_TIME);
  printf("Processo %d iniziato\n", child_number);
  for (i = 1; i <= NO_OF_ITERATIONS; i++) {
    routine_perdi_tempo();
    printf("Processo %d: iterazione %d\n", child_number, i);
  }
  printf("Processo %d terminato\n",child_number);
  exit(0);
}
```

Esempio: scheduling FIFO/RR

```
#define ALGORITMO_SCHEDULING      SCHED_FIFO | SCHED_RR
#define PRIORITA sched_get_priority_max(ALGORITMO_SCHEDULING) | child_number
```

Scegliendo child_number i processi hanno priorità diverse, pari al loro numero identificativo.

FIFO – priorità uguali

```
Processo 1 iniziato
Processo 1: iterazione 1
Processo 1: iterazione 2
Processo 1: iterazione 3
Processo 1: iterazione 4
...
Processo 1 terminato
Processo 2 iniziato
Processo 2: iterazione 1
Processo 2: iterazione 2
Processo 2: iterazione 3
...
Processo 2 terminato
Processo 3 iniziato
Processo 3: iterazione 1
Processo 3: iterazione 2
...
Processo 3 terminato
```

FIFO – prio. diverse

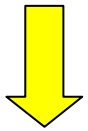
```
Processo 3 iniziato
Processo 3: iterazione 1
Processo 3: iterazione 2
Processo 3: iterazione 3
Processo 3: iterazione 4
...
Processo 3 terminato
Processo 2 iniziato
Processo 2: iterazione 1
Processo 2: iterazione 2
Processo 2: iterazione 3
...
Processo 2 terminato
Processo 1 iniziato
Processo 1: iterazione 1
Processo 1: iterazione 2
...
Processo 1 terminato
```

RR – priorità uguali

```
Processo 1 iniziato
Processo 2 iniziato
Processo 3 iniziato
Processo 1: iterazione 1
Processo 1: iterazione 2
Processo 1: iterazione 3
Processo 2: iterazione 1
Processo 2: iterazione 2
Processo 3: iterazione 1
Processo 3: iterazione 2
Processo 1: iterazione 4
Processo 1: iterazione 5
...
Processo 1 terminato
Processo 2: iterazione 30
Processo 2 terminato
Processo 3: iterazione 30
Processo 3 terminato
```

Problemi nei sistemi Real-Time

- Lo scheduler non ha nessuna conoscenza dei requisiti temporali dei processi (deadline, periodo, …)
- L'algoritmo di scheduling OTHER ha durata imprevedibile (kernel 2.4): dipende dal numero N di processi attivi nel sistema (O(N))
- Gli algoritmi di scheduling "real-time" offrono politiche insufficienti alla gestione di un sistema RT



Cambiando lo scheduler si possono risolvere i problemi?

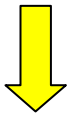
Linux kernel 2.4

Cambiando lo scheduler si possono risolvere i problemi?

NO…rimane un problema molto importante…

Quando può essere invocato lo scheduler?

- Linux non è preemptable quando sta eseguendo in spazio kernel
  - Un processo funzionante in kernel mode (per es. system call) non può subire preemption
  - Il cambio di contesto può avvenire solo al termine dell'esecuzione in kernel mode
- System calls, interrupt ed exception (eseguiti tutti in spazio kernel) sono annidabili

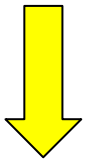


Il tempo di attesa per l'assegnazione del processore ad un processo più prioritario di quello in esecuzione è indeterminabile.

Linux kernel 2.4

E il supporto real-time? …scadente!

- Algoritmi di scheduling insufficienti
- Alta latenza di scheduling
  - Kernel non preemptable
  - Il periodo minimo di attivazione dell'algoritmo di scheduling è 10 ms (frequenza del timer interno di Linux 100 Hz)
- Indeterminismo della latenza di scheduling



Miglioramenti con la versione 2.6 del kernel Linux.

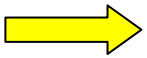
Linux kernel 2.6

- Algoritmo di scheduling ottimizzato: algoritmo O(1)
  - L'overhead dovuto alle operazioni di scheduling è determinabile a priori e non è dipendente dal numero di processi attivi
- Frequenza interna del clock 1000 Hz
  - Periodo minimo di esecuzione dello scheduler: 1 ms
- Kernel preemptable
  - Introduzione di punti di preemption nel kernel in cui è possibile eseguire lo scheduling

Linux ed il Real-Time

Con la vesione 2.6 del kernel Linux il supporto ai processi real-time è migliorato, ma rimane insufficiente per processi che hanno vincoli di tempo stringenti ed insuperabili.

Versioni future del SO potranno essere orientate alle applicazioni Hard Real-Time?

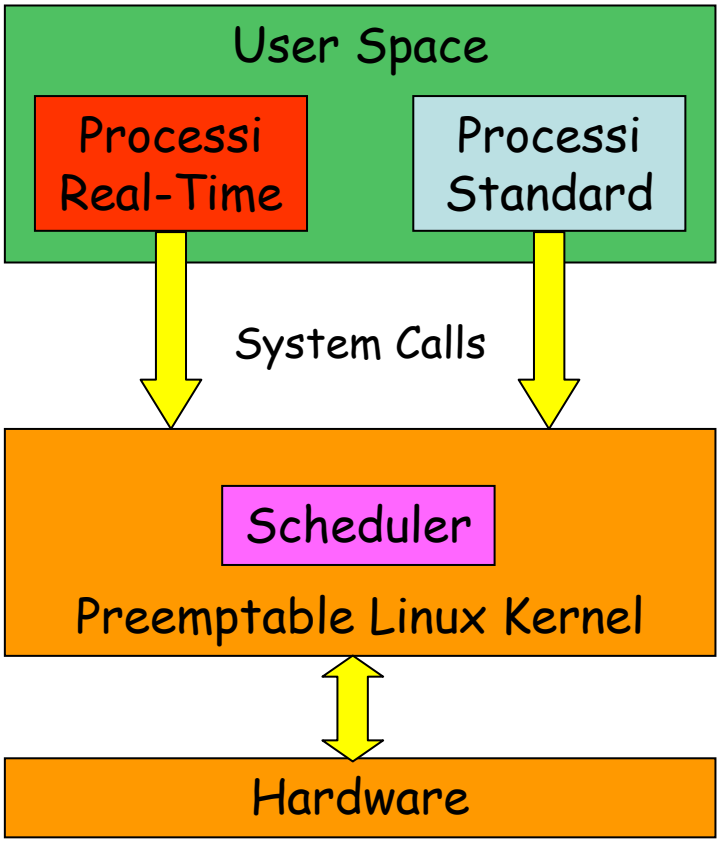


Apparentemente sì…la pesantezza intrinseca di alcuni meccanismi di Linux (es. gestione degli interrupt) viene mascherata dalla velocità del microprocessore e dalla sua latenza architetturale.

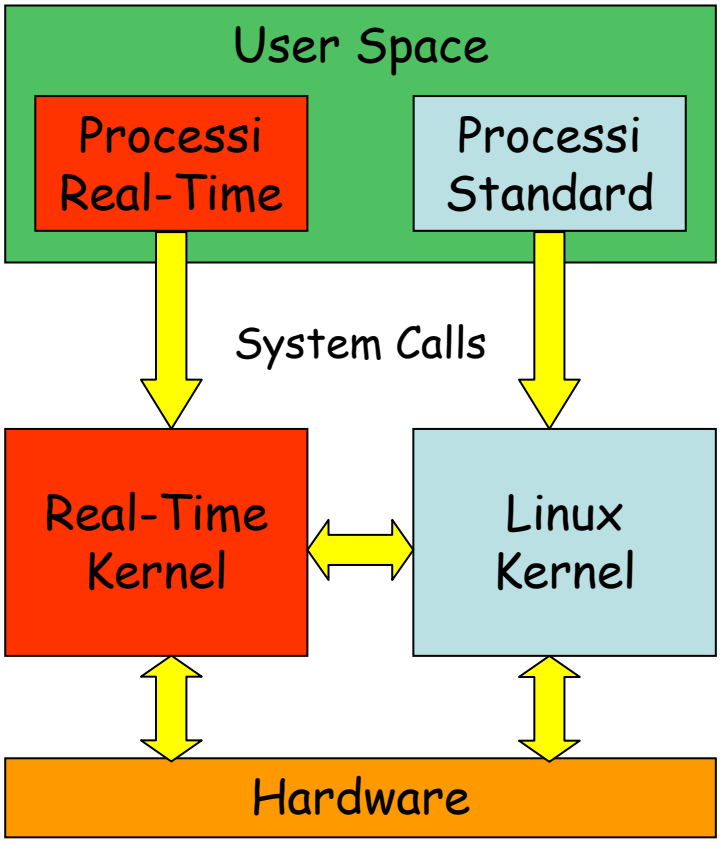
- Per ora, in attesa di sviluppi e verifiche, Linux è utilizzabile solo per applicazioni Soft Real-Time
- Ottimo punto di partenza per realizzare un sistema Hard Real-Time

Estensioni Real-Time

Estensioni real-time al kernel Linux sono possibili in due direzioni.



Evoluzione del kernel Linux per renderlo full preemptable e deterministico.



Il kernel Real-Time gestisce il kernel Linux come un processo a bassa priorità.

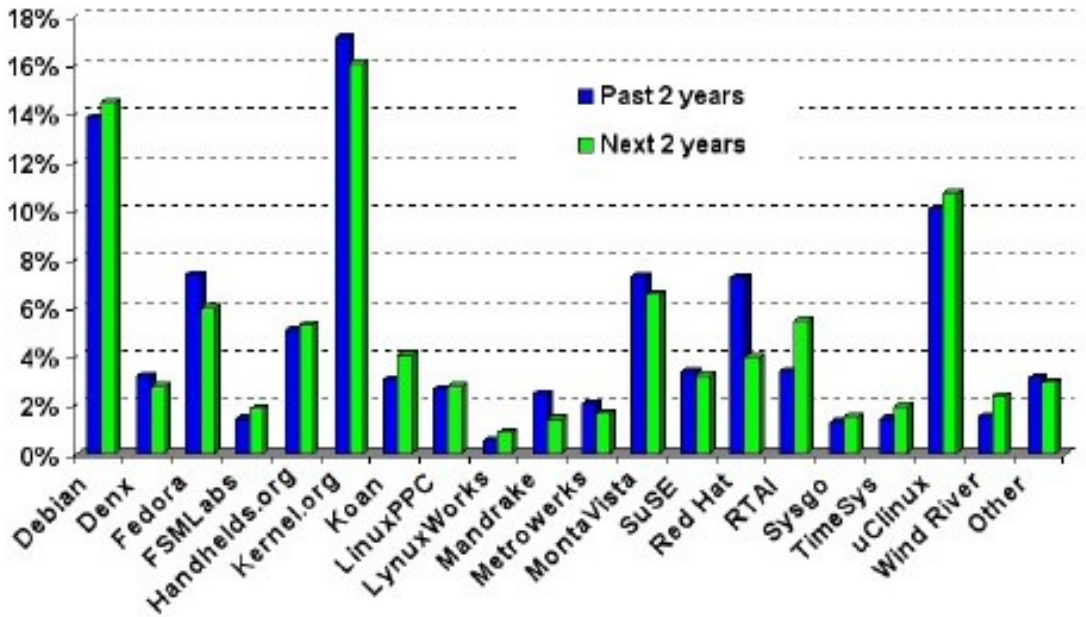
Estensioni Real-Time

RTOS Linux-based commerciali

- Montavista Linux
- LynuxWorks BlueCat Embedded Linux
- Wind River Linux
- Koan KaeilOS
- TimeSys Linux

RTOS Linux-based free

- RTAI Linux
- µClinux



Distribuzioni Linux per applicazioni in dispositivi embedded: trends di impiego. Da "Embedded Linux Market Survey 2006", Linuxdevices.

Documentazione

- RTAI-Linux: un sistema real-time distribuito per applicationi di Motion Control, L. Dozio – P. Mantegazza
- Sistemi in Tempo Reale, G. C. Buttazzo
- Linux Device Drivers, J. Corbet - A. Rubini - G. Kroah-Hartman
- Linux e real-time, D. Ciminaghi
- Linux e le sue applicazioni industriali, nei sistemi embedded e real-time, S. Pozzi
- Linux Kernel 2.6, G.P. Ghilardi
- www.linux.org, www.kernel.org, www.tldp.org, yolinux.com, www.linux.it