SISTEMI DI ELABORAZIONE E CONTROLLO M
Ingegneria dell'Automazione

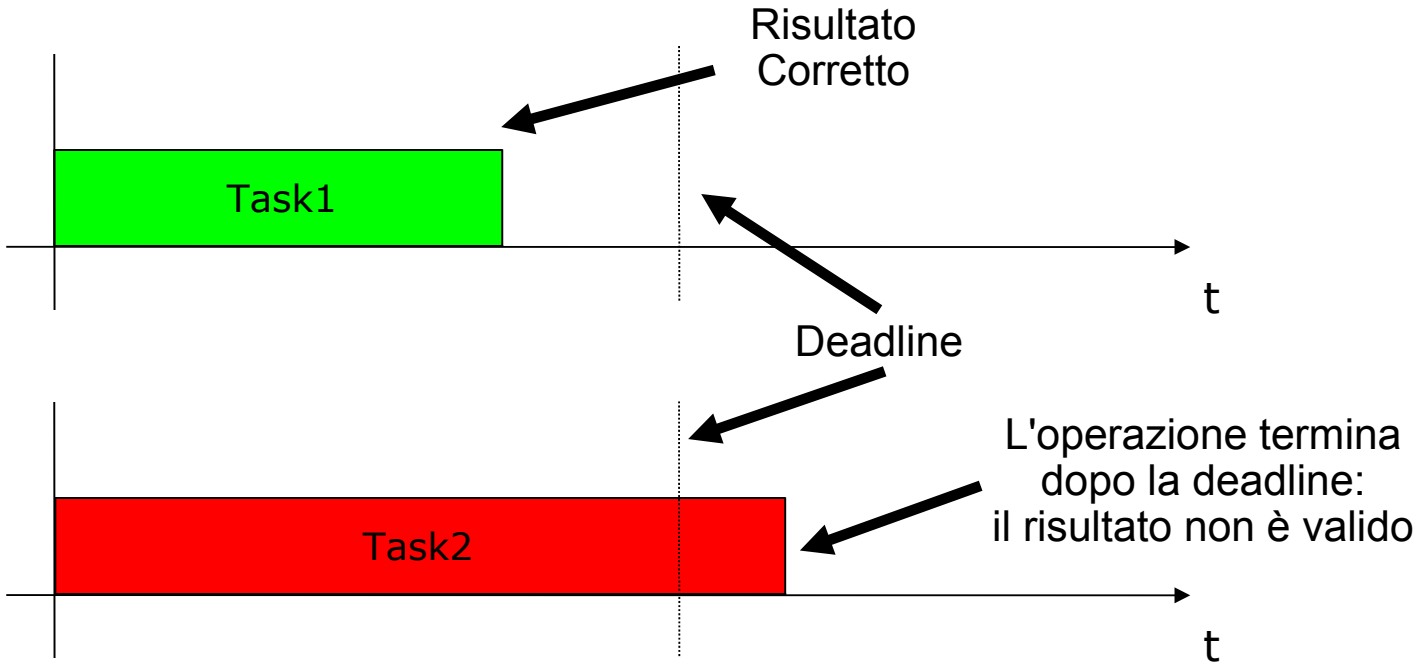
INTRODUZIONE AI SISTEMI REALTIME

Ing. Gianluca Palli
DEIS - Università di Bologna
Tel. 051 2093903
email: gianluca.palli@unibo.it
http://www-lar.deis.unibo.it/~gpalli

Informazioni sul corso

- Struttura del corso
  - Lezioni teoriche in aula
  - Esercitazioni in laboratorio
- Homepage del corso
  - http://www-lar.deis.unibo.it/people/gpalli/SEC.html
- Modalità di esame
  - Discussione di tesine da svolgere in gruppi
  - Date da concordarsi
- Ricevimento
  - Su appuntamento
  - Contattarmi via email gianluca.palli@unibo.it

- Definizione di sistema operativo real time:
  - E' un sistema operativo in cui per valutare la correttezza delle operazioni si considera anche la variabile tempo;
  - Lo scheduler dei processi agisce secondo precise specifiche temporali.



RTOS: Definizioni

- Due tipologie di correttezza devono essere garantite:
  - Correttezza logica: i risultati/risposte forniti devono essere quelli previsti (normalmente richiesta a qualunque sistema di elaborazione)
  - Correttezza temporale: i risultati devono essere prodotti entro certi limiti temporali fissati (deadlines) (specifica dei sistemi real time)

- Tipologie di Sistemi real time
  - Sistemi Hard Real Time:
    - Il non rispetto delle deadlines temporali NON e' ammesso
    - porterebbe a danneggiamento del sistema (safety critical)
  - Sistemi Soft Real Time:
    - il non rispetto delle deadlines e' ammissibile
    - le specifiche temporali indicano solo in modo sommario i tempi da rispettare
    - degrado delle performace accettabile

RTOS: Definizioni

- In generale i sistemi real time complessi saranno ibridi hard/soft
  - Vi saranno deadline inderogabili
    - gestione delle condizioni di allarme pericolose
    - controllo digitale in retroazione
    - reazione ad eventi interni
  - Mentre altre non saranno stringenti
    - effettivo avviamento di una macchina dopo il comando ricevuto da un sistema di supervisione
    - salvataggio dei dati su dispositivi di archiviazione
    - interazione con l'uomo

- Attenzione: spesso si confonde il concetto di Hard e Soft Real Time con quello di Real Time "Stretto" e "Largo"
  - Real Time "Stretto": vincoli temporali (deadlines) stretti rispetto ai tempi di calcolo necessari per eseguire le operazioni richieste
  - Real Time "Largo": vincoli temporali (deadlines) larghi rispetto ai tempi di calcolo necessari per eseguire le operazioni richieste

RTOS: Definizioni

- La “larghezza” o “strettezza” di un sistema di elaborazione dell’informazione real time dipende dalla piattaforma Hardware utilizzata
  - tempi di esecuzione dipendono dalla ‘potenza’ dell'unità di elaborazione (velocità + risorse)
- Spesso (purtroppo) i sistemi Hard Real Time sono anche stretti
  - normalmente le deadlines vicine sono anche inderogabili
  - per motivi di costo si sceglie sempre una piattaforma di elaborazione non sovrabbondante rispetto alle esigenze
- Nel caso di sistemi REAL TIME STRETTI e’ fondamentale la corretta organizzazione delle fasi di elaborazione per il rispetto delle deadlines
  - bisogna organizzare le attività di elaborazione al fine di ottimizzare i tempi
  - massimo sfruttamento delle risorse HW
    - forte legame tra il codice e l’HW di elaborazione
    - difficile abbinare astrazione e ottimizzazione

RTOS: Definizioni

- I sistemi real time devono in genere svolgere più attività (task) che possono anche essere completamente indipendenti.
- Le attività sono usualmente innescate a seguito del verificarsi di particolari condizioni (eventi) e devono terminare prima di certe deadlines
  - L'intervallo di tempo tra evento e deadline viene detto "time scope" (finestra temporale) dell'attività
  - I time scope di diverse attività possono essere sovrapposti
    - ESECUZIONE PARALLELA
- La realizzazione parallela delle attività dipende dalla architettura HW utilizzata.
  - Sistemi monoprocessore (parallelismo logico, ma non reale)
  - Sistemi multiprocessore (parallelismo anche reale)
    - ravvicinati (collegamento molto veloce, es.: sulla stessa board)
    - distribuiti (collegamento più lento, es.: via rete ethernet)
- Parallelismo logico gestito con "Programmazione Concorrente"

I Sistemi Operativi Real Time (RTOS)

- 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.

- 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

- FPGA e ASIC:
  - Implementazione totalmente hardware
  - Difficoltà di programmazione
  - Basso costo

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

- 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

- 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

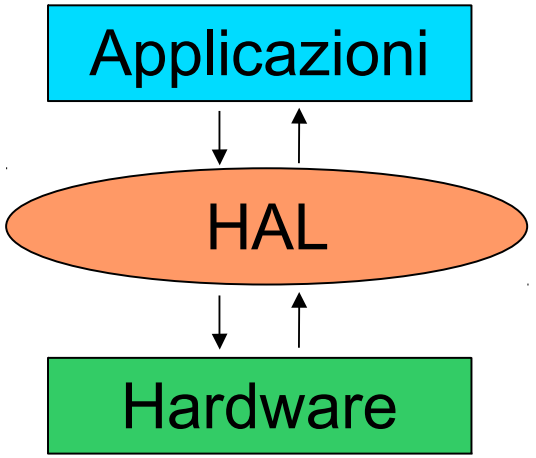
I Sistemi Operativi Real Time (RTOS)

- 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

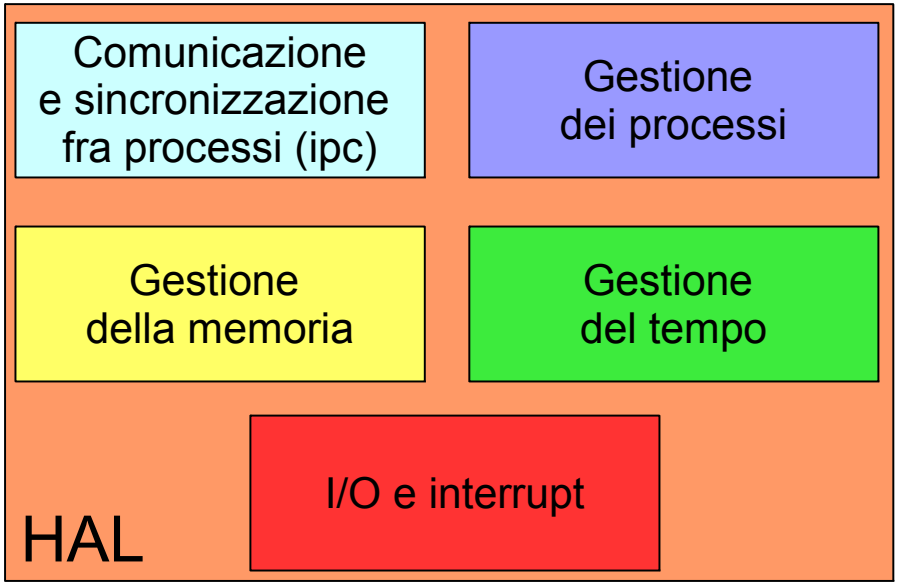
- Partendo dallo standard POSIX per sistemi operativi general purpose sono state definite alcune estensioni specifiche per i sistemi RT.

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

- I sistemi operativi real time fornisco alle applicazioni un'astrazione dell'hardware (Hardware Abstraction Layer, HAL).



- L'Hardware Abstraction Layer mette a disposizione delle applicazioni una serie di servizi (primitive real time).



RTOS: Applicazioni

- Campi di applicazione dei sistemi operativi real time:
  - Macchine automatiche
  - Automobili
  - Computers
  - Sistemi di telecomunicazione
  - Centrali Elettriche
  - Avionica
  - ecc...

- I sistemi operativi real time sono “nascosti”:
  - La loro principale applicazione è in sistemi embedded;
  - Non sono studiati per interagire con l'utente (uomo);
  - La programmazione di questi ambienti avviene solitamente con l'ausilio di altri sistemi operativi non real time.

RTOS: Soluzioni

- La maggior parte dei sistemi operativi real time consiste in soluzioni proprietarie studiate appositamente per l'hardware su cui deve funzionare
- Alcuni dei sistemi operativi real time “generici”:
  - Soluzioni proprietarie:
    - VxWorks (POSIX-compliant)
    - QNX (POSIX-compliant)
    - LynuxWorks LynxOS (full POSIX-conformant)
    - Microsoft Windows CE (non POSIX)
    - ...
  - Soluzioni opensource:
    - Linux Preemptive (POSIX-compliant)
    - **RTAI** **(non POSIX)**
    - ...

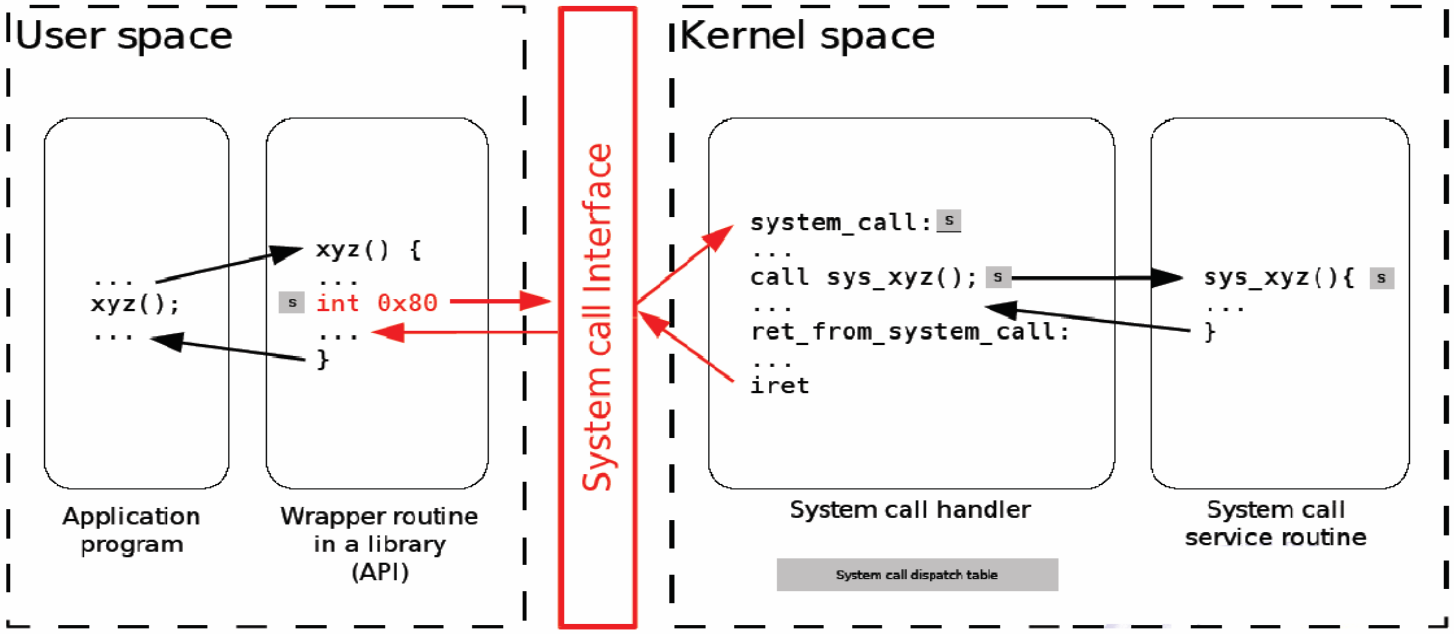
- RTAI è una soluzione basata sul kernel di Linux
  - Può sfruttare tutte le applicazioni e l'ambiente del sistema operativo di partenza.

Linux Kernel

- In ambiente Linux, 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

- Cos'è una System Call?
  - 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



- Spesso è necessario creare nuovi processi a partire dal “programma principale”
  - Praticamente sempre nei sistemi realtime

- Uso della fork();

```
...
if ((fork_return = fork()) == 0) {
     new_child(count); /* esecuzione figlio */
} else ... /* esecuzione padre */
```

- Uso della pthread_create();

```
pthread_t mythread;

if (pthread_create(&mythread, &mythread_attr, new_thread,
&count) != 0) /* chiamata a new_thread() nel nuovo thread */
{
     printf("error creating thread\n");
     exit(-1);
}... /* continua esecuzione thread principale */
```

- Che differenza c'è fra l'uso di fork() e l'uso di pthread_create()?

Processi vs Threads

- fork()
  - Crea un nuovo **processo** che diviene figlio (cioè generato da) del processo che lo ha creato (il padre)
  - Entrambi i processi eseguono l'istruzione successiva lla chiamata della fork() (system call)
  - I processi padre e figlio hanno due copie distinte ma identiche del proprio spazio di indirizzamento, codice e stack
  - Al figlio viene assegnato un nuovo pid (process ID)
  - La fork() crea un vero è proprio clone indipendente del programma (processo) chiamante che esegue lo stesso codice
- pthread_create()
  - Cread un nuovo **thread** appartenente allo stesso processo (o thread) che lo ha creato
  - Threads appartenenti allo stesso processo possono comunicare attraverso shared memory
  - I thread di un processo (e il processo principale) condividono le stesse variabili globali, spazio di indirizzamento, file aperti, signal handlers, working directory, user e group ID.
  - Il nuovo thread avrà un suo stack, thread ID e registri
  - Un thread è entità che può agire indipendentemente dal resto del programma ma condivide con esso le stesse informazioni

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 subisce preemption da parte di un processo più prioritario è posto in testa alla lista dei processi pronti al suo livello di priorità

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

Gestione dello Scheduler

- Alcune primitive (System Calls) per la gestione dello scheduler (sched.h)
  - 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()
    - Sospende il processo chiamante e gli fa assumere lo stato READY. Il processo viene posto alla fine della coda dei processi pronti alla sua priorità.

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.

```
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

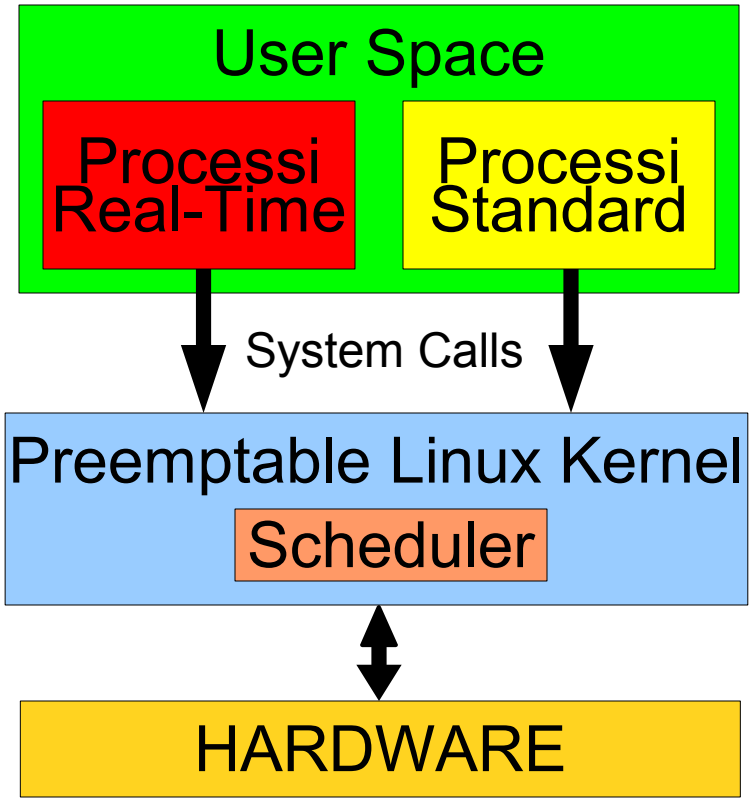
- Lo scheduler non ha nessuna conoscenza dei requisiti temporali dei processi (deadline, periodo, ...)
- Gli algoritmi di scheduling “real-time” offrono politiche insufficienti alla gestione di un sistema RT
- Cambiando lo scheduler si possono risolvere i problemi?
  - NO...rimane un problema molto importante...
  - Quando può essere invocato lo scheduler?

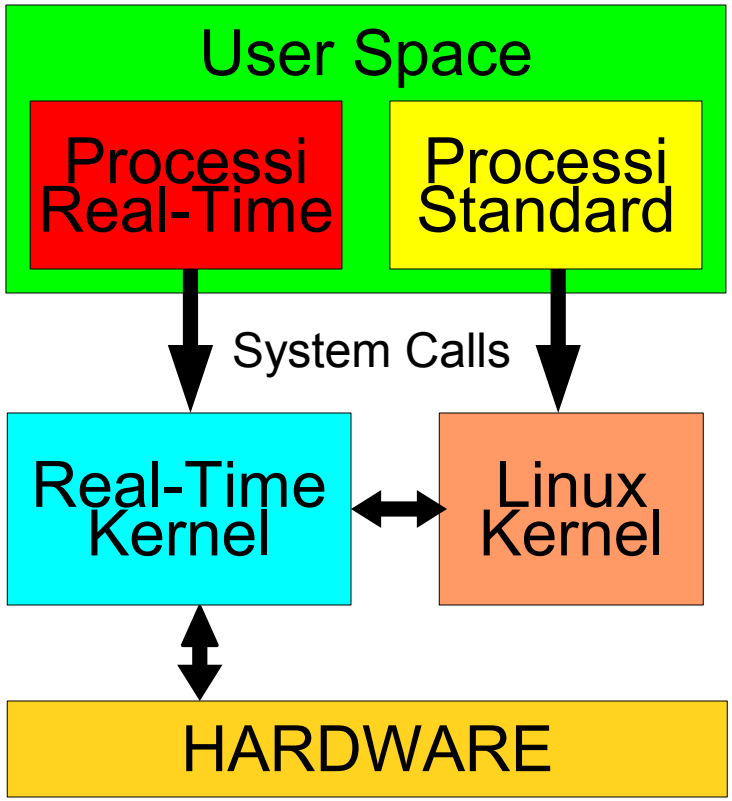
Problemi nei sistemi Real-Time

- Linux nelle sue vecchie versioni (kernel 2.4) non è preemptable quando è 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.

- 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 massima del clock 1000 Hz
  - Periodo minimo di esecuzione dello scheduler: 1 ms
  - Frequenza tipica 250 Hz
- Kernel preemptable
  - Introduzione di punti di preemption nel kernel in cui è possibile eseguire lo scheduling

Estensioni Real-Time





- Evoluzione del kernel Linux per renderlo full preemptable e deterministico
- Il kernel Real-Time gestisce il kernel Linux come un processo a bassa priorità

SISTEMI DI ELABORAZIONE E CONTROLLO M
Ingegneria dell'Automazione

INTRODUZIONE AI SISTEMI REALTIME
FINE

Ing. Gianluca Palli
DEIS - Università di Bologna
Tel. 051 2093903
email: gianluca.palli@unibo.it
http://www-lar.deis.unibo.it/~gpalli