Sistemi Operativi
Real-time

Sistemi real-time

- Generalità
- Caratteristiche del sistema
- Caratteristiche dei kernel real-time
- Implementazione dei sistemi operativi real-time
- Scheduling real-time della CPU
- Esempio: VxWorks

Obiettivi

- Illustrazione dei requisiti temporali dei sistemi real-time
- Distinzione tra sistemi real-time “hard’’ e “soft”
- Analisi delle caratteristiche dei sistemi real-time
- Analisi degli algoritmi di scheduling per i sistemi hard real-time

Generalità sui sistemi real-time

- Un **sistema real-time (in tempo reale)** richiede che un'operazione o un risultato venga eseguita/prodotto ***entro un dato intervallo temporale***.

- Un **sistema embedded** è un dispositivo di elaborazione che è inserito in un sistema più grande e più complesso (ad es., un'automobile, un aereo, un elettrodomestico, un robot).

- Un **sistema safety-critical** è un sistema real-time che in caso di guasto/fallimento può causare risultati catrastofici.

- Un **sistema hard real-time** ***guarantisce*** che i task real-time ***siano completati entro un tempo determinato***.

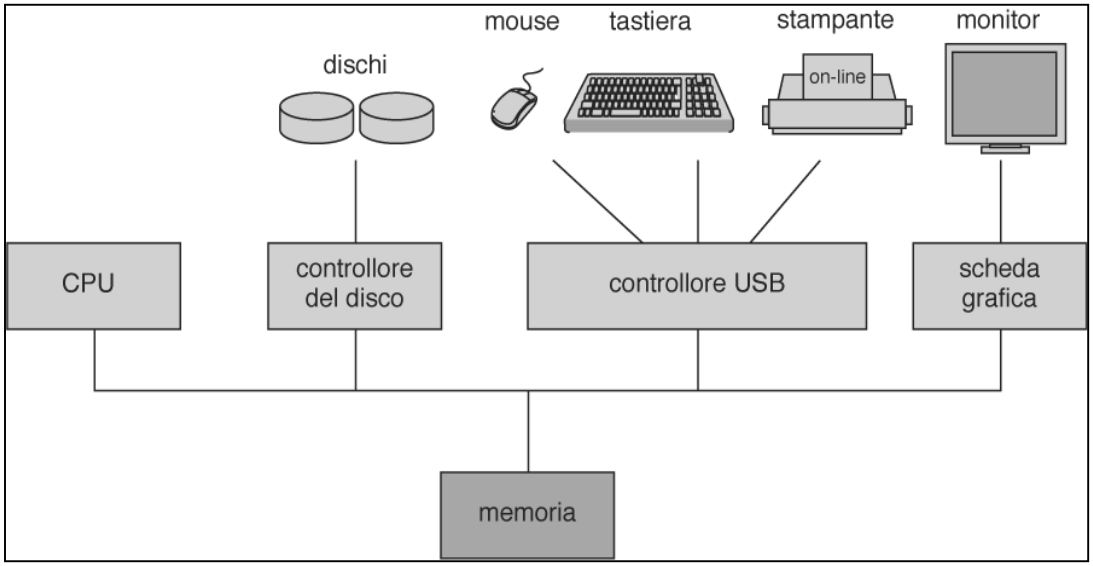
- Un **sistema soft real-time** assegna ai task real-time una priorità maggiore rispetto ai task non real-time.

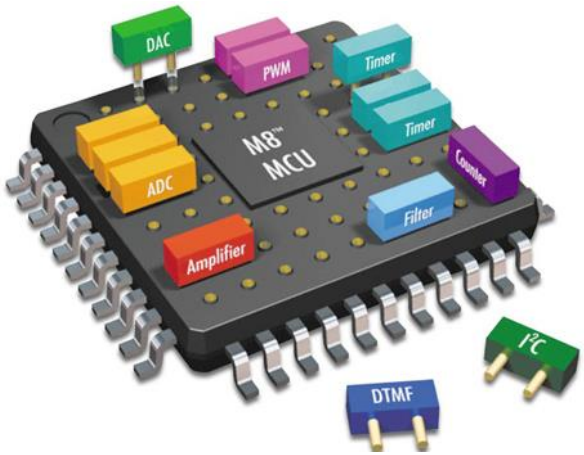
Caratteristiche tipiche dei sistemi real-time

- Single purpose (singolo scopo).
- Small size (piccole dimensioni).
- Sono economici se prodotti in grandi quantità.
- Specificano requisiti di timing stretti.

System-on-a-Chip

- Molti sistemi real-time o sistemi embedded sono realizzati usando un approccio che usa un'architettura hardware di tipo **system-on-a-chip** (**SOC**).
- I SOC permettono che la CPU, la memoria, la memory-management unit, e le periferiche (es., USB) siano contenuti in un unico circuito integrato.
- Un SOC è di solito meno costoso di un sistema tradizionale bus-based.





Caratteristiche dei kernel real-time

- Molti sistemi real-time non hanno le stesse caratteristiche dei sistemi desktop standard.

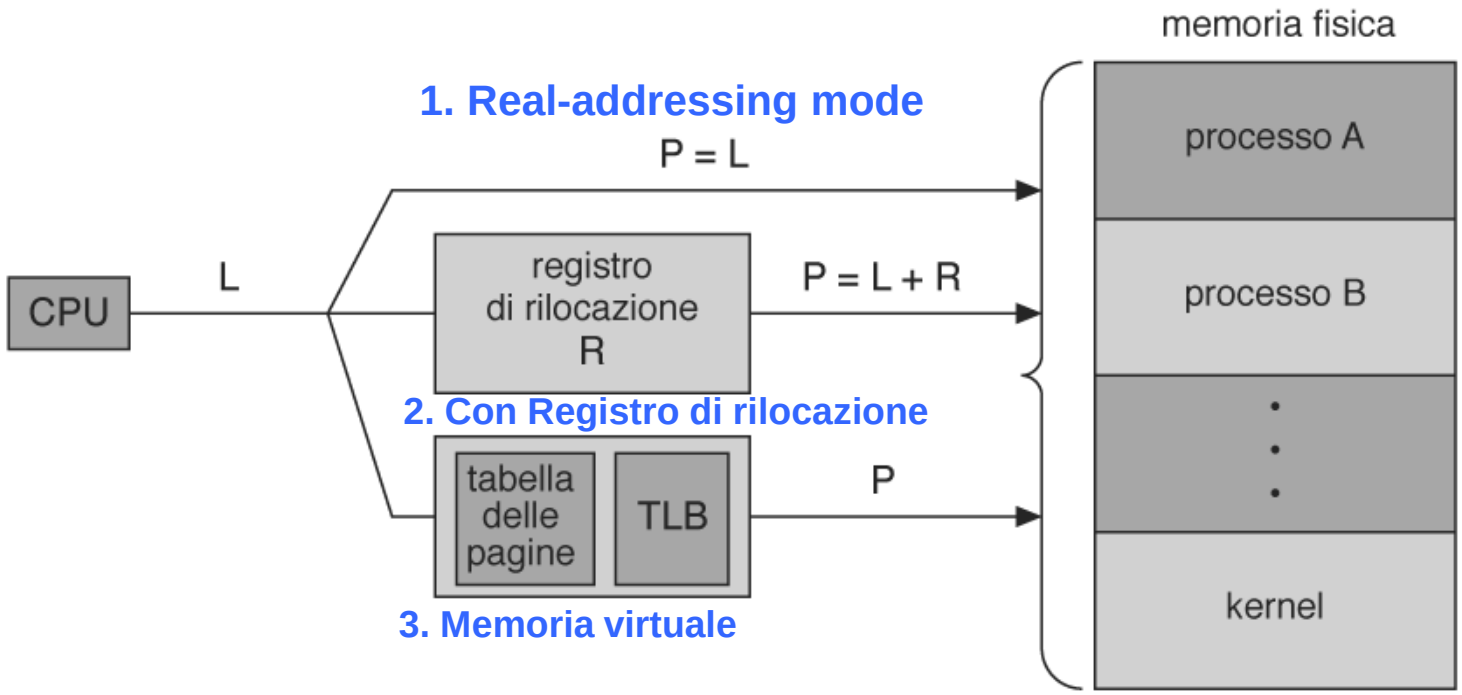
- Alcune moticvazioni:
  - I real-time systems sono in molti casi single-purpose.
  - I real-time systems in molti casi non hanno interfacce per gli utenti.
  - Un desktop PC richiede un insieme di dispositivi hardware che spesso non servono in un sistema real-time.

Traduzione degli indirizzi nei sistemi real-time

- La traduzione degli indirizzi nei sistemi real-time può avvenire secondo le modalità seguenti:

1. **Real-addressing mode:** I programmi generano gli indirizzi fisici.
2. **Relocation** register mode (con registro di rilocazione).
3. Implementando la **Memoria Virtuale.**

Traduzione degli indirizzi nei sistemi real-time



Implementazione dei sistemi real-time

- In generale, i sistemi operativi real-time usano:

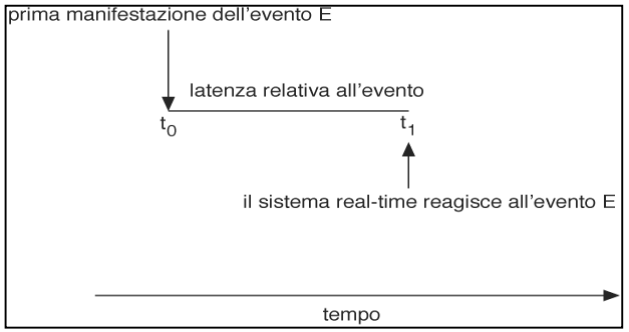
  1. Scheduling **preemptive** e **priority-based**

  2. **Preemptive kernels** (per prelazionare processi in modalità kernel)

  3. **Minimizzazione** della **latenza**.

- **Latenza**: intervallo temporale tra l'occorrenza di un evento ($\mathbf{t_0}$) e l'istante di tempo in cui il sistema gestisce l'evento ($\mathbf{t_1}$).

$$L = t_1 - t_0$$



Latenza relativa all'evento

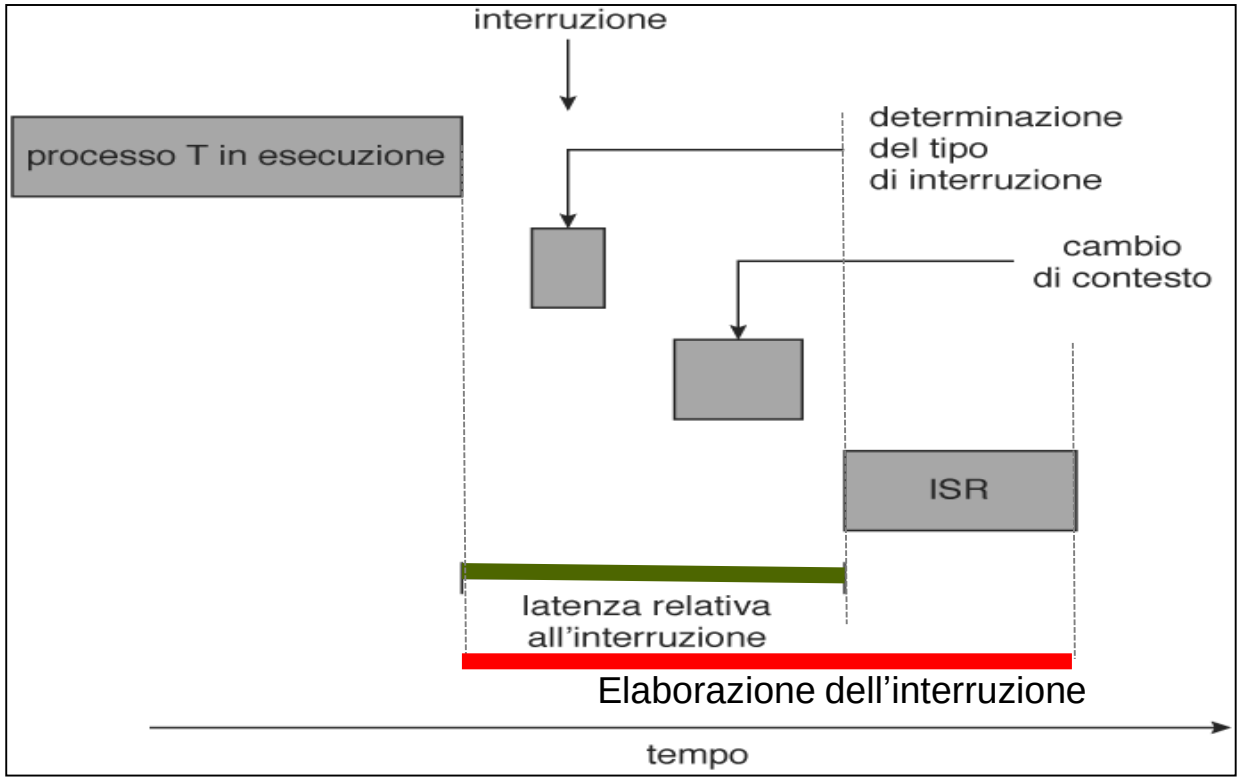
- **Latenza**: intervallo temporale tra l'occorrenza di un evento (t_0) e l'istante di tempo in cui il sistema gestisce l'evento (t_1).

$$L = t_1 - t_0$$

- Due principali tipi di latenza:
  - Latenza delle interruzioni
  - Latenza di dispatch

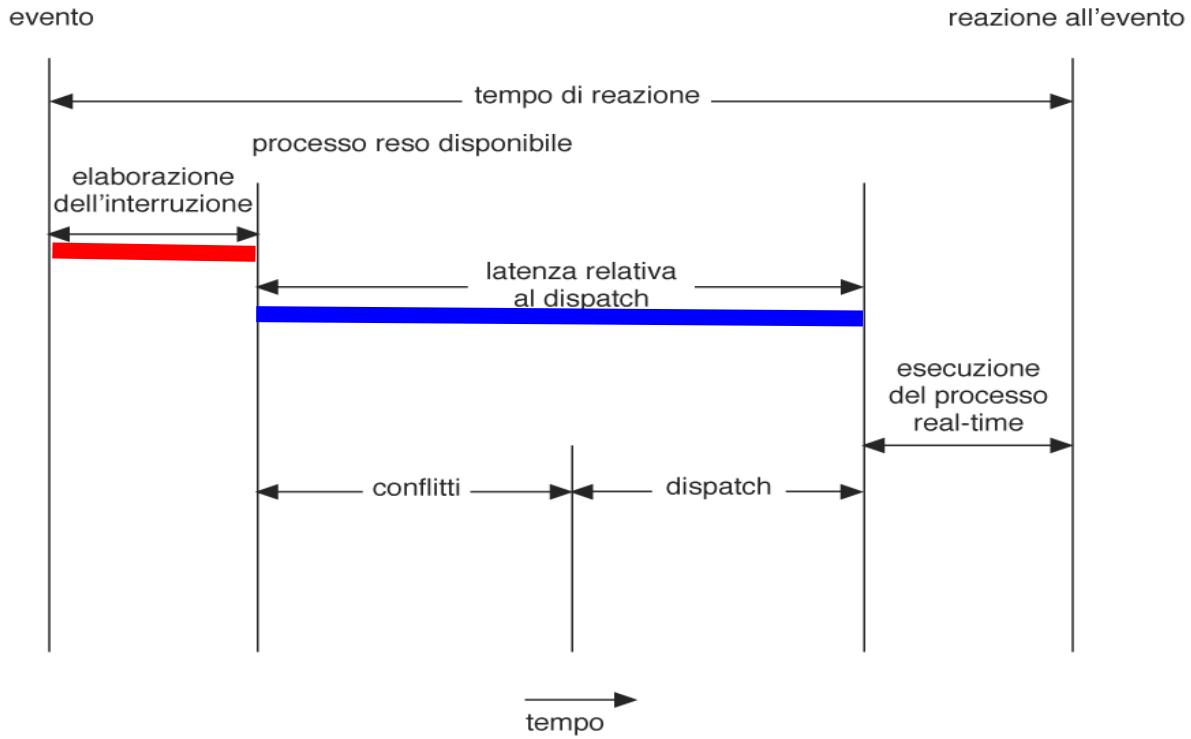
Latenza relativa alle interruzioni

- La **latenza relativa alle interruzioni** considera l'intervallo di ***tempo*** tra la ***notifica dell'interruzione*** alla CPU ***e l'avvio della routine*** di gestione dell'interruzione (ISR).



Latenza relativa al dispatch

- **La latenza di dispatch** è l'intervallo di tempo necessario per fermare un processo e per avviarne un altro (a causa di un'evento).

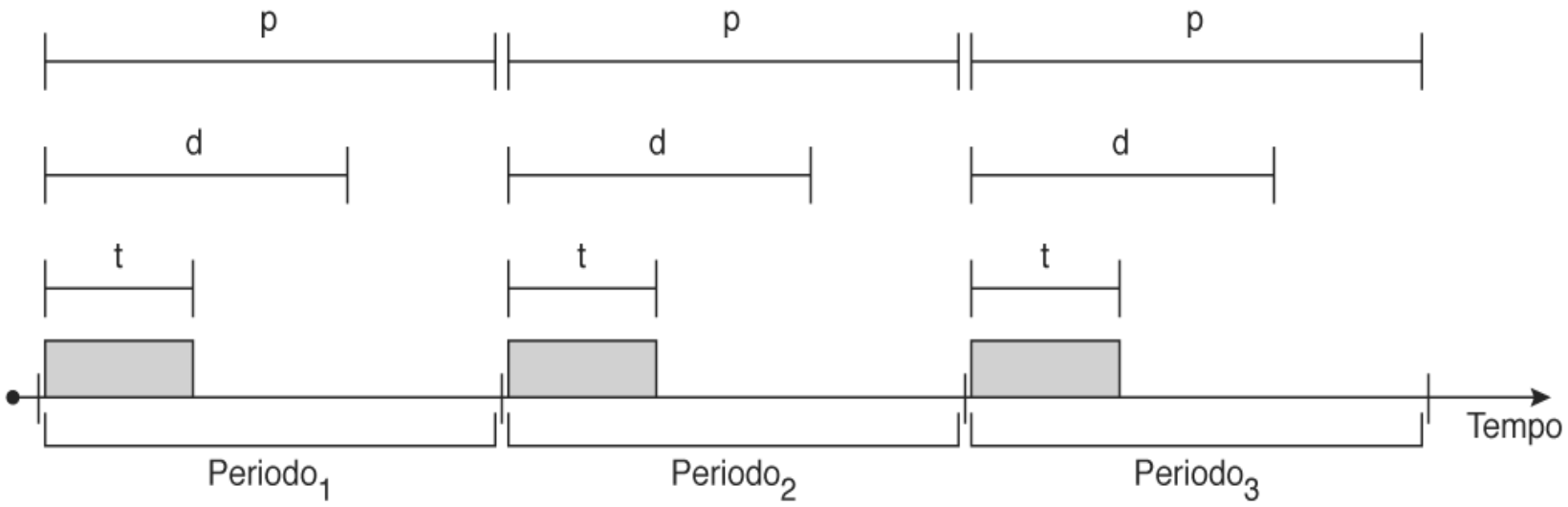


- La gestione dei conflitti deve effettuare la **prelazione dei processi in esecuzione nel kernel** e la cessione delle risorse al processo da eseguire.

Scheduling di processi periodici hard real-time

- I **processi periodici** richiedono la CPU a intervalli costanti di tempo (**periodi**).
- Un processo periodico una volta avuto l'accesso alla CPU, esegue per un tempo fissato, ha una scadenza per completare l'esecuzione e un periodo.
- ***p*** è il periodo
- ***d*** è la scadenza
- ***t*** è il tempo di elaborazione

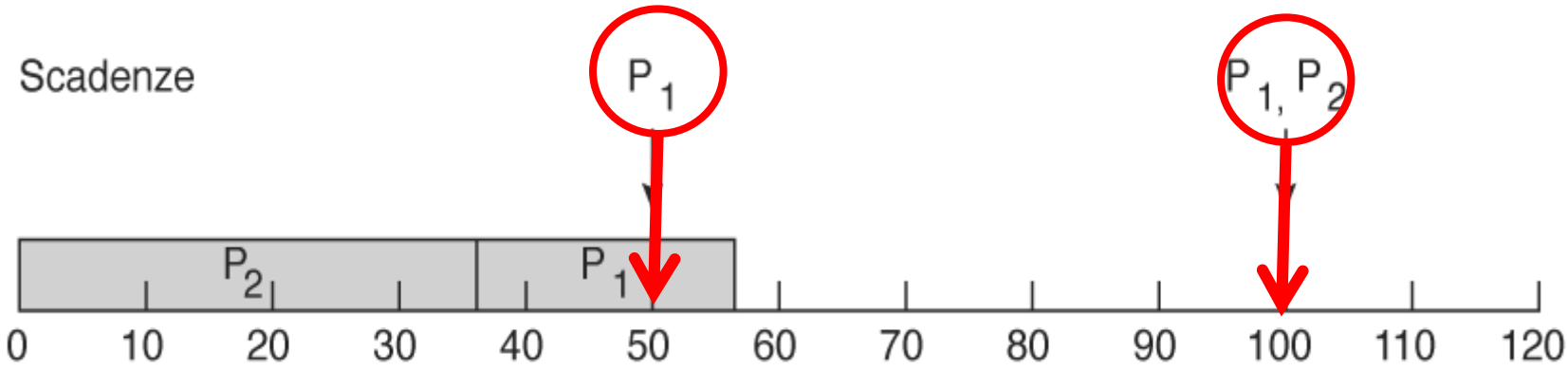
$$0 \leq t \leq d \leq p$$



Scheduling dei processi: caso P_2 con priorità maggiore di P_1

- Due processi real-time: P_1 e P_2 con stesso tempo di partenza.
- Periodi : P_1 = 50 e P_2 = 100
- Scadenze = periodi
- Tempi di elaborazione : t_1 = 20 e t_2 = 35

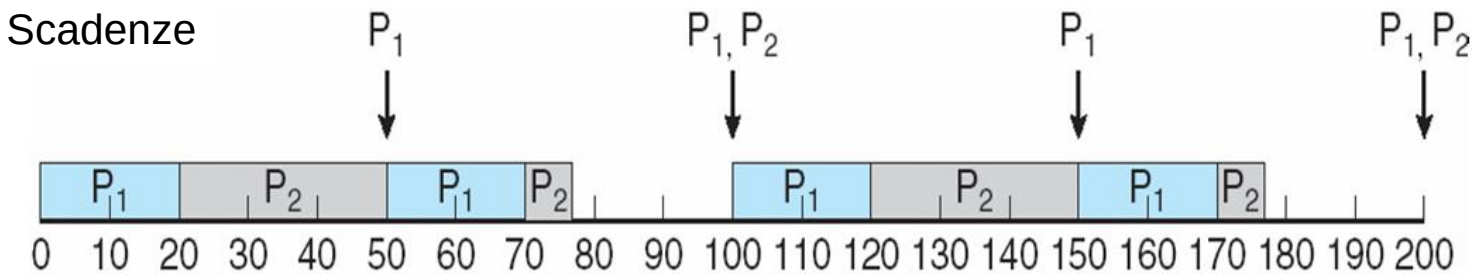
- Supponiamo che la priorità di P_2 sia maggiore della priorità di P_1:



- P_1 **non rispetta la scadenza!**

Scheduling con priorità proporzionale alla frequenza

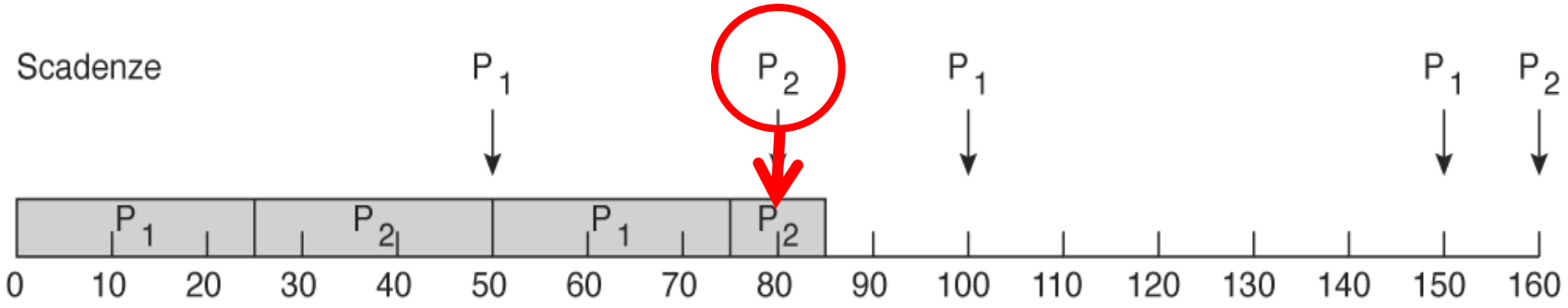
- Se la **priorità viene assegnata usando l'inverso del periodo** il problema si può risolvere (***Rate monotonic scheduling***).
- Principio: **Assegnare priorità maggiori ai processi che usano più frequentemente la CPU**.
- Periodi più brevi = Priorità più alta;
- Periodi più lunghi = Priorità più bassa;
- A P_1 viene assegnata una priorità maggiore di P_2



Scadenze non rispettate con lo scheduling con priorità proporzionale alla frequenza

- Due processi: P_1 e P_2
- Periodi : P_1 = 50 e P_2 = 80
- Scadenze = periodi
- Tempi di elaborazione : t_1 = 25 e t_2 = 35

- ***Rate monotonic scheduling***: priorità di P_1 > priorità di P_2:



- **Nonostante lo rate monotonic scheduling , P_2 non rispetta la scadenza!**

Scadenze non rispettate con lo scheduling con priorità proporzionale alla frequenza

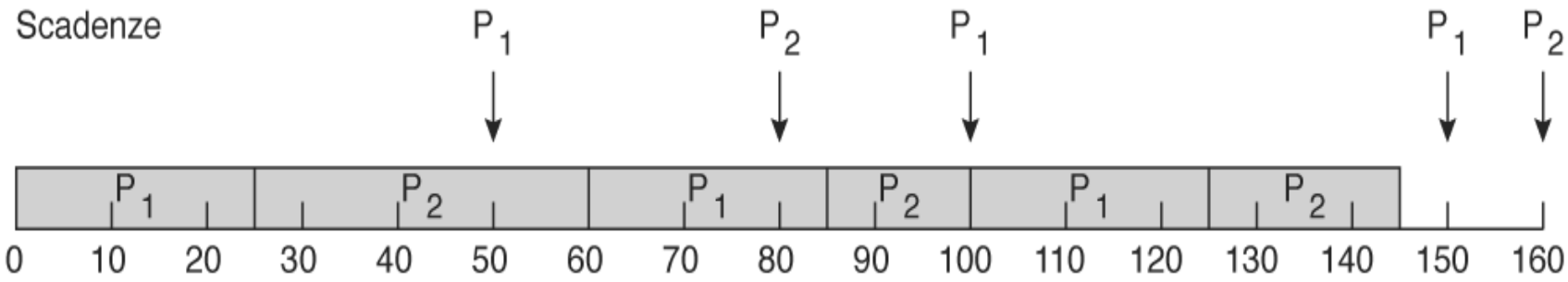
- Nel caso di eventi periodici, se esistono m eventi e se l'evento i arriva con periodo P_i e richiede T_i unità di tempo di CPU, il carico di elaborazione può essere gestito solo se:

$$U = \sum_{i=1,m} T_i / P_i \leq 1$$

- Un **sistema real-time** che rispetta questo vincolo si dice **schedulabile**.

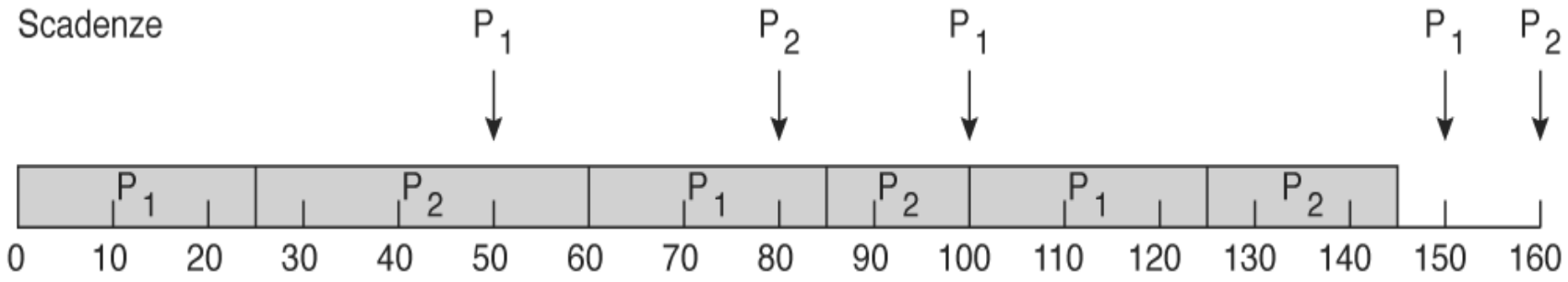
Scheduling EDF (Earliest Deadline First)

- Lo scheduling EDF attribuisce le priorità (dinamica) sulla base delle scadenze:
  - **Più vicina è la scadenza, maggiore sarà la priorità**;
  - Più lontana è la scadenza, minore sarà la priorità;
- Esempio:
  - P_1 = 50 and t_1 = 25
  - P_2 = 80 and t_2 = 35
  - Scadenze = periodi



Scheduling EDF (Earliest Deadline First)

- Lo scheduling EDF non è basato sulla periodicità dei processi (richiesta della CPU a intervalli periodici).
- Lo scheduling EDF richiede che i processi informino lo scheduler della loro prossima scadenza.
- **E' uno scheduling ottimale**: tutti i processi entro le loro scadenze (se possibile). Nessun altro algoritmo con priorità statiche può fare meglio, ma EDF ha costi di esecuzione non trascurabili.



Scheduling EDF (Earliest Deadline First)

- Esempio:

Processo	Tempo di esecuzione	Periodo
P1	1	8
P2	2	5
P3	3	10

- Utilizzazione:

$$U = (1/8+2/5+ 3/10) = 82,5\%$$

- Poichè l'utilizzazione è pari all'82,5%, quindi minore del 100%, il sistema è scedulabile con EDF.

Scheduling a quote proporzionali

- Un ***insieme di Q*** quote sono assegnate/distribuite tra tutte le applicazioni (processi del sistema).

- Una applicazione riceve ***N*** quote, con ***N < Q***

- Questa strategia assicura che ogni applicazione riceverà

  N / Q

  del tempo totale della CPU.

- Lo scheduler accetta richieste di esecuzione (con le relative quote) a condizione che il numero di quote richieste siano disponibili (non si usa più del 100% delle quote!).

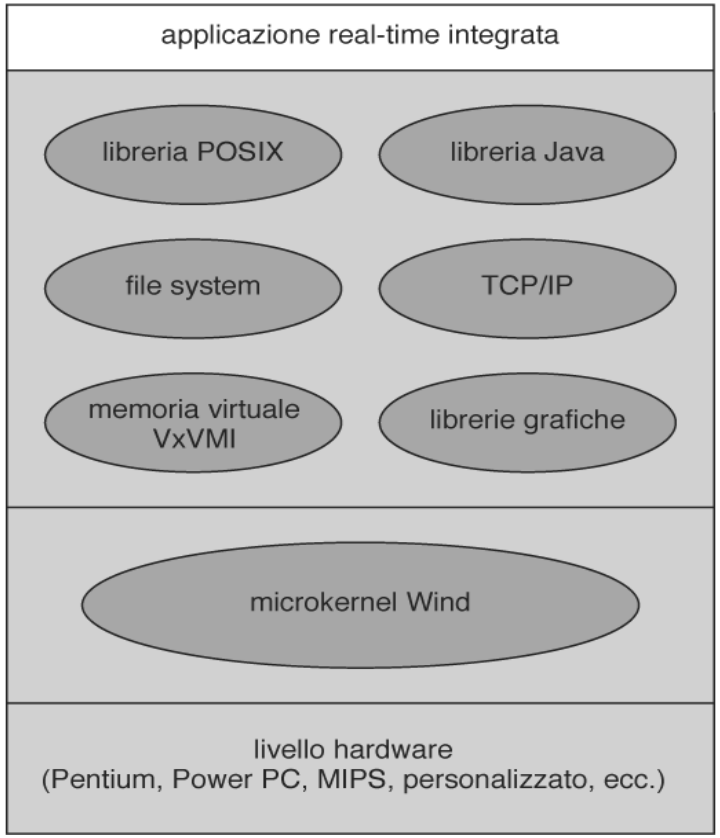
- Esempio: Se *Q = 250* e *N = 20* ogni processo potrà ricevere *20/250 = 8%* del tempo.

Scheduling a quote proporzionali

- Esempio: *Q* = 100 quote a disposizione da ripartire fra i processi *A*, *B* e *C*.
- Se *A* dispone di 50 quote, *B* di 15 e *C* di 20, la CPU è occupata all'85%.
- Lo scheduler opera in sinergia con un meccanismo di controllo dell'ammissione, per garantire che ogni processo possa effettivamente ricevere le quote di tempo Q che gli sono state destinate (altrimenti non viene momentaneamente ammesso nel sistema).

Esempio di S.O. hard-real time: VxWorks

- **VxWorks** è un S.O. molto usato per applicazioni real-time in ambito controllo industriale, automotive, sonde spaziali (NASA).
- Ha una struttura basata su un microkernel dedicato alle funzioni principali e alla gestione dei processi.
- Disponibile su molte architetture hardware (Intel, POWER, Arm).

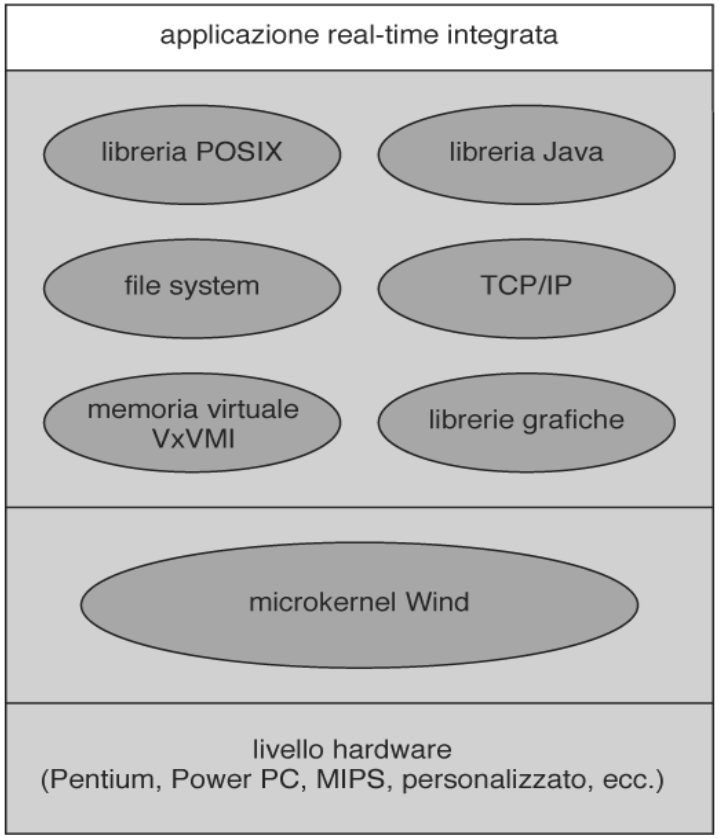


Esempio di S.O. hard-real time: VxWorks

VxWorks è stato progettato nel 1987 per l'utilizzo in sistemi embedded con vincoli di tempo reale.

Applicazioni:

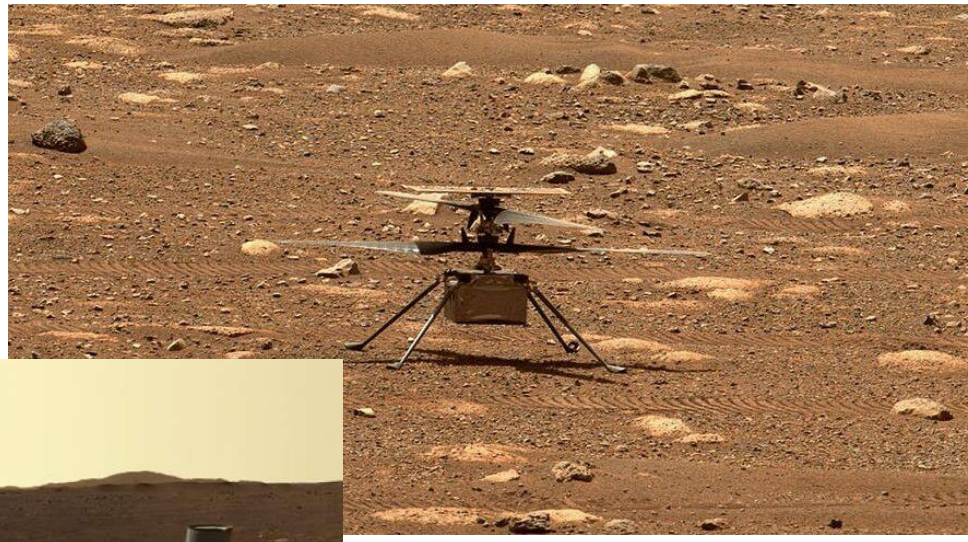
- Sicurezza, aerospazio, difesa,
- dispositivi medici, apparecchiature industriali,
- robotica, energia, trasporto, infrastrutture di rete, automotive, e elettronica di consumo, telescopi, robotica, ... rover NASA Mars.



Esempio di S.O. hard-real time: VxWorks

VxWorks è stato su Marte 4 volte.

Questa volta insieme a Linux (che gestisce l'elicottero).



Microkernel Wind

- Il microkernel Wind supporta:
  - Processi e threads (chiamati task).
  - Lo scheduling round-robin non-preemptive e **preemptive** (con 256 livelli di priorità).
  - La gestione degli interrupts (con latenza di interrupt e dispatch limitati) per rispettare vincoli real-time.
  - Inter-process communication con shared memory e semafori, con scambio messaggi e segnali.

VxWorks: Dimensioni e Caratteristiche

- Nella versione minima VxWorks occupa uno spazio di circa 36 KB incluso il Board Support Package;
- Un'installazione che include il supporto per applicazioni di alto livello, rete e file system può raggiungere i 600 KB.
- Dalla versione 6.0, VxWorks ha introdotto, oltre alla modalità di esecuzione dei task 'Kernel Mode', anche la modalità 'User Mode' indicata con il termine RTP (Real-Time Processes).
- Questa aumenta i tempi di latenza e peggiora le funzionalità real-time, ma consente la virtualizzazione dei processi con la possibilità di una maggiore protezione del sistema operativo e una maggiore facilità di utilizzo grazie alla gestione automatica e protetta delle risorse.


The End