Il Sistema Operativo

Informatica B AA 17/18

Giacomo Boracchi

13 Dicembre 2017

giacomo.boracchi@polimi.it

REMINDER

CHI NON L'AVESSE ANCORA FATTO E' PREGATO DI RISPONDERE AL QUESTIONARIO ONLINE

http://home.dei.polimi.it/boracchi/teaching/InfoB/survey.htm

Introduzione al Sistema Operativo

Il Sistema Operativo

- Il Sistema Operativo (SO) è uno strato **software** che **nasconde** agli utenti i **dettagli dell'architettura hardware** del calcolatore
- Fornisce diverse **funzionalità** ad **alto livello** che facilitano **l'accesso alle risorse** del calcolatore
- Supporta **l'esecuzione dei programmi** applicativi **definendo una macchina virtuale**, cioè un modello ideale del calcolatore, sollevando il software applicativo dal compito di gestire i limiti delle risorse disponibili

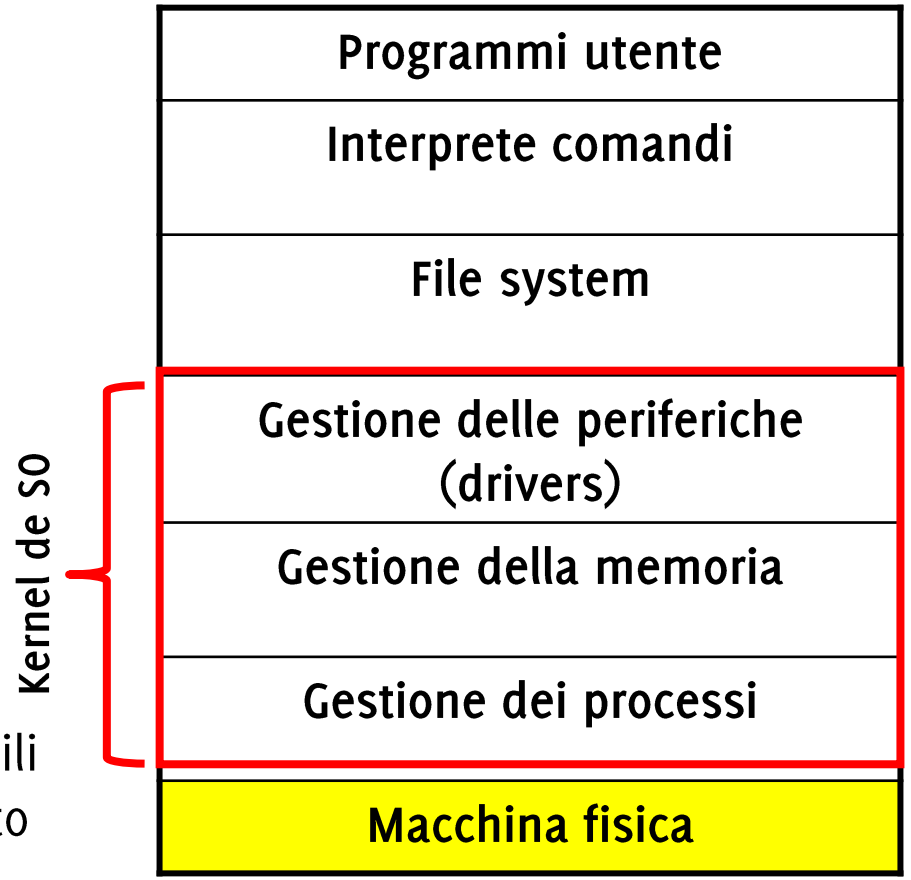
Tipi di Sistema Operativo

Esistono diversi tipi di sistema operativo, ma in generale si possono dividere in:

- **Monoutente** e **monoprogrammato**
  - Esecuzione un solo programma applicativo alla volta
  - Viene utilizzato da un solo utente per volta
  - Esempio: DOS
- **Monoutente** e **multiprogrammato** (**multitasking**)
  - Consente di eseguire contemporaneamente più programmi applicativi
  - Esempio: Windows 95
- **Multiutente**
  - Consente l'utilizzo contemporaneo da parte di più utenti
  - E' inerentemente multiprogrammato
  - Esempio: Linux e i recenti Versioni Windows

Architettura del Sistema Operativo

- Il SO è tipicamente organizzato a **strati**
- Ciascun **strato gestisce una risorsa** del calcolatore
- Le principali funzionalità offerte sono:
  - La gestione dei processi
  - La gestione della memoria
  - La gestione delle periferiche
  - La gestione del file system
  - La gestione della rete
  - La gestione dell'interfaccia utente
- Le **prime tre** funzionalità sono indispensabili per il funzionamento del sistema e pertanto **costituiscono il nucleo del SO (Kernel)**

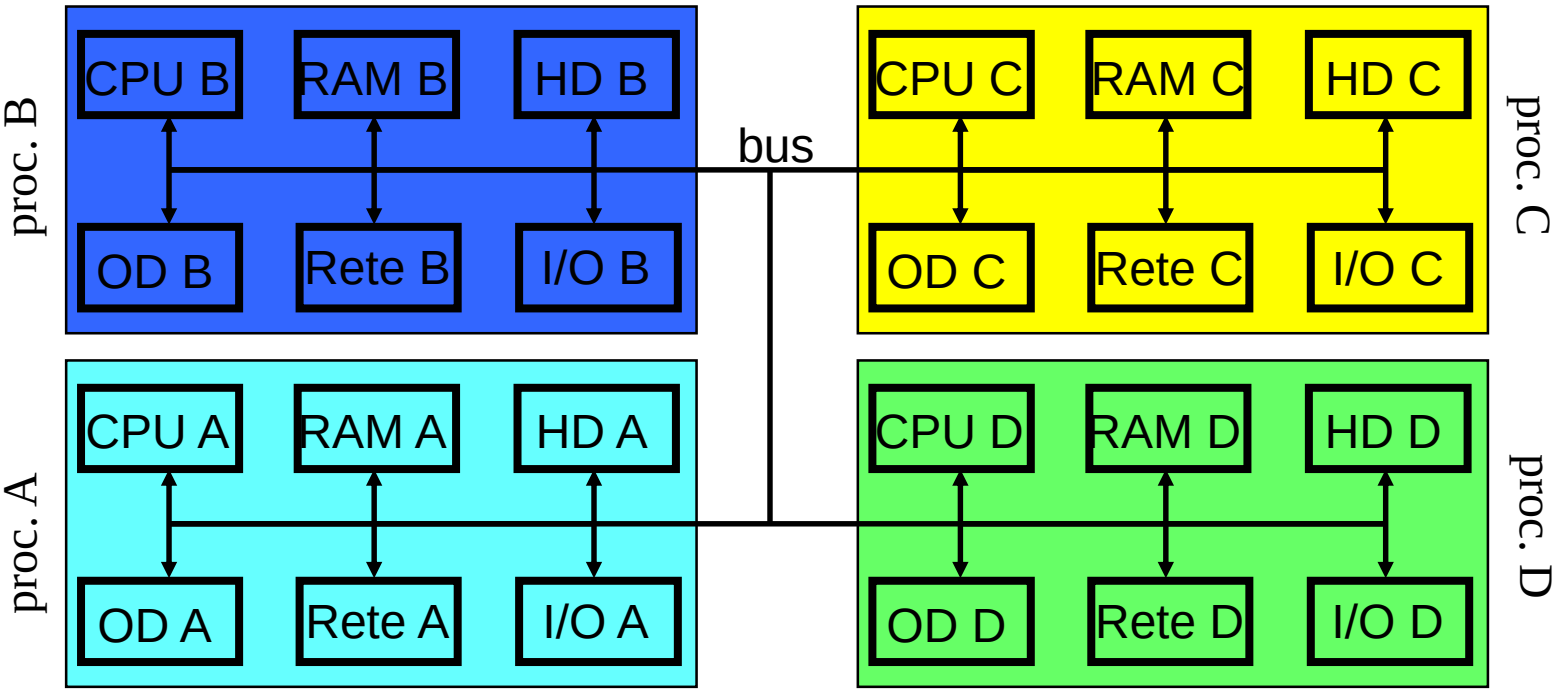


Kernel del SO: Gestione dei Processi

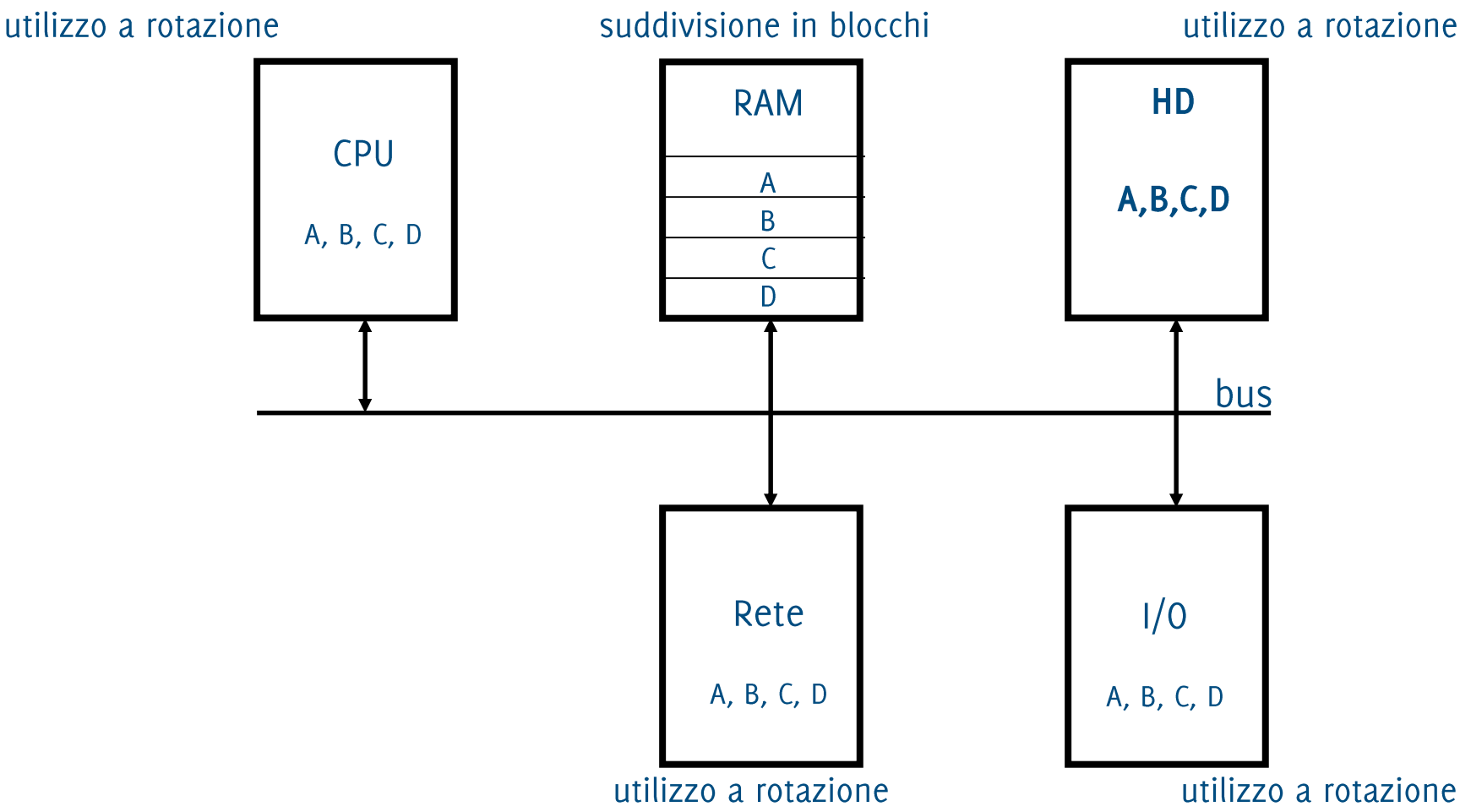
- Il SO si occupa di **gestire l'esecuzione** concorrente di più **programmi** (quantomeno nei sistemi multitasking).
- I programmi durante la loro esecuzione danno luogo a uno o più **processi**
- Il SO offre ad **ogni processo** una **macchina virtuale** interamente **dedicata** a esso.
  - Ogni processo opera come se fosse l'unico in esecuzione sulla macchina, avendo quindi disponibilità esclusiva delle risorse
- La **CPU** del calcolatore (o le CPUs nei sistemi multiprocessore) **viene suddivisa** in maniera opportuna fra i programmi da eseguire.

Il Sistema Operativo e le Macchine Virtuali

- Il SO permette di gestire **più processi simultaneamente**
- Rende quindi visibile ad ogni processo **una macchina virtuale** ad esso interamente dedicata e quindi con risorse proprie



Il Sistema Operativo e la Macchina Reale



Kernel del SO: Gestione della Memoria

- La gestione concorrente di molti programmi applicativi comporta la **presenza di molti programmi in memoria centrale** (la MM contiene i programmi in esecuzione ed i dati ad essi relativi)
- Il **Gestore della memoria del SO** offre a ogni programma in esecuzione la visione di una **memoria virtuale**, che può avere dimensioni maggiori di quella fisica.
- Per gestire la memoria virtuale e suddividerla tra i vari processi, il SO dispone di diversi meccanismi:
  - Rilocazione
  - Paginazione
  - Segmentazione

Vedremo nel dettaglio la Rilocazione e la Paginazione

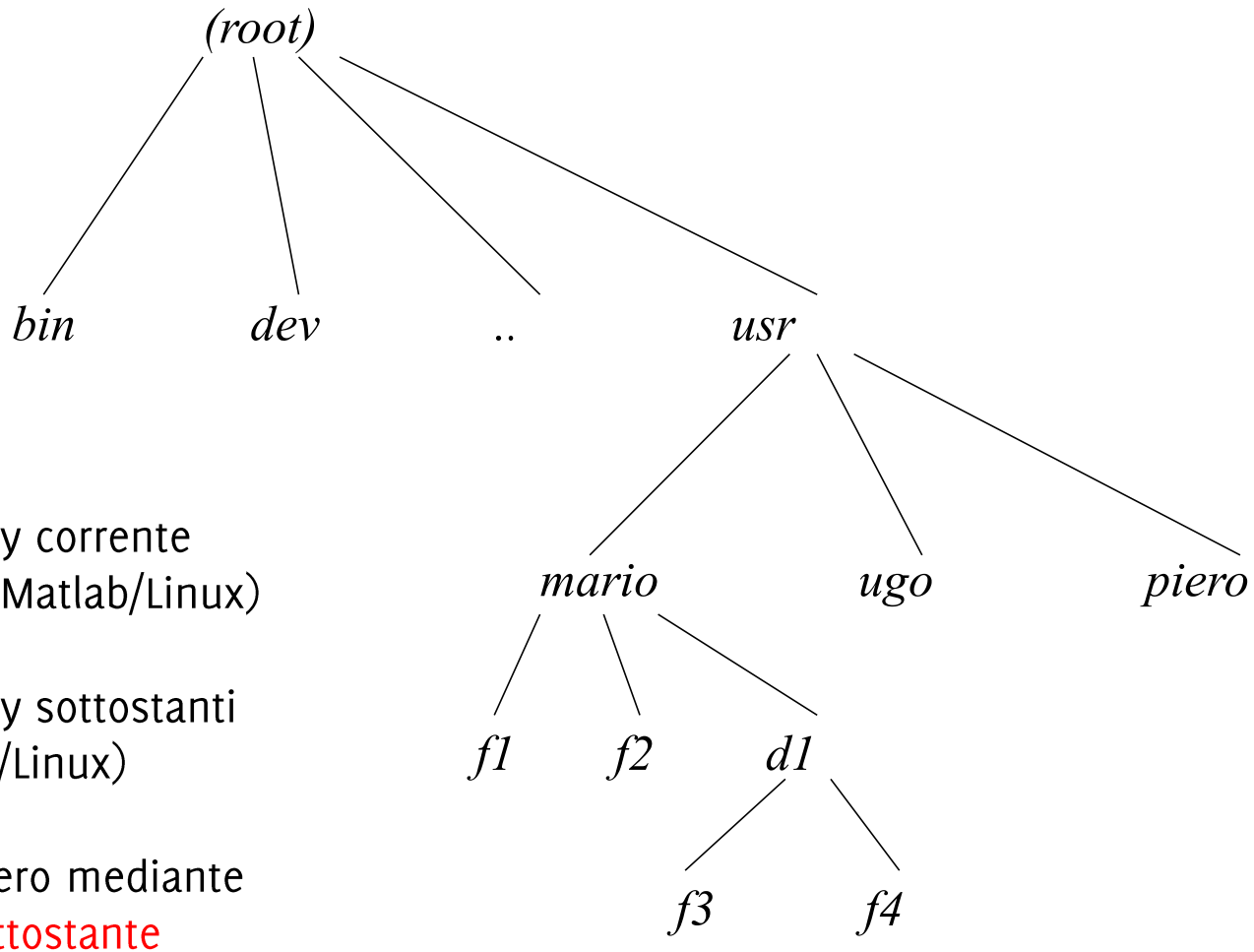
Kernel del SO: Gestione delle Periferiche, i DRIVER

- I **driver** sono meccanismi software a cui è affidato il compito **di trasferire dati da e verso le periferiche**
- Consentono ai programmi applicativi di leggere o scrivere i dati mediante **istruzioni di alto livello** che **nascondono la struttura fisica delle periferiche** (e.g., nel sistema Unix le periferiche sono viste come file speciali)
- Danno all'utente l'impressione che la periferica sia interamente dedicata all'utente.

Gestione del File System

- Il SO si occupa di gestire i **file** sulla **memoria di massa**:
  - Creare / Eliminare un file
  - Dare / Modificare il nome
  - Collocarlo in una posizione nella memoria di massa
  - Accedervi in lettura e scrittura
- Il tutto mediante istruzioni di alto livello che permettono di ignorare i dettagli di come il file viene scritto
- Gestione dei file **indipendente dalle caratteristiche fisiche** della **memoria di massa** (HD, SD CARD, CD…)
- I file vengono inclusi all'interno di *directory* (o *cartelle*):
  - Hanno una tipica organizzazione ad albero
  - Alcuni SO hanno una struttura a grafo

La Struttura ad Albero



Per visualizzare la directory corrente digitare solo cd o pwd (in Matlab/Linux)

Per visualizzare le directory sottostanti digitare dir o ls (in Matlab/Linux)

Ci si può spostare nell'albero mediante il comando cd DirectorySottostante oppure cd .. per salire di un livello nell'albero

Organizzazione dei File

- A ciascun utente è normalmente associata una directory specifica, detta ***home directory***
- Il livello di ***protezione*** di un file indica quali operazioni possono essere eseguite da ciascun utente
- Ciascun file ha un ***pathname*** (o nome completo) che include l'intero cammino dalla radice dell'albero
- Il ***contesto* di un utente** all'interno del file system è la directory in cui correntemente si trova

Esempi di pathname

C:\Windows
C:\Users\Giacomo Boracchi\Desktop
C:\Users\Giacomo Boracchi\Documents
C:\Users\Giacomo Boracchi\Documents\MATLAB

Gestione dell'Interfaccia Utente

- Il SO fornisce un interprete dei comandi inseriti dall'utente attraverso la tastiera o il mouse
- L'interfaccia utente può essere
  - Testuale (esempio: DOS)
  - Grafica (esempio: Windows)
- Consente l'inserimento di diversi comandi:
  - Esecuzione di programmi applicativi
  - Operazioni sulle periferiche
  - Configurazione dei servizi del SO
  - Operazioni sul file system (creazione, rimozione, copia, ricerca, ecc.)

Il Sistema Operativo ed i Processi

Che Cosa è un Processo?

- Processo ≠ programma !
- Un processo viene generato durante l'**esecuzione di un programma**.
- Un processo è composto da:
  - **codice eseguibile** (il programma stesso)
  - **dati dell'esecuzione** del programma
  - informazioni relative allo **stato** del processo
- Un processo è quindi un'entità **dinamica**, mentre il programma è un'entità **statica.**
- I processi vengono eseguiti dal **processore**
  - uno alla volta nelle architetture a processore singolo
  - vedremo il Round-robin come politica di scheduling

Programmi e Processi

Lo stesso **programma** può avere **più processi associati**:

- Un **programma** può essere **scomposto** in varie parti e ognuna di esse può essere associata a un diverso processo.
- Lo stesso **programma** può essere associato a diversi processi quando esso viene **eseguito più volte**, anche simultaneamente

I Processi e il Sistema Operativo

- Anche il **SO** è **implementato** tramite **processi**;
- Il **SO** è **garante** che i **conflitti** tra i **processi** siano controllati e **gestiti** correttamente;
- I processi possono essere eseguiti in due modalità:
  - **Kernel** (o **Supervisor**) **mode**: la CPU può eseguire qualsiasi istruzione, iniziare qualsiasi operazione I/O, accedere a qualsiasi area di memoria, etc.
  - **User mode**: certe istruzioni, che alterano lo stato globale della macchina, non sono permesse, e.g., operazioni I/O, accesso a certe aree di memoria, etc.
- Il SO viene **eseguito in modalità privilegiata** (kernel mode o supervisor), così da avere processi in grado di controllare altri processi eseguiti in modalità user.

Chiamate al Supervisor

I processi utente per eseguire operazioni privilegiate, es.

- accesso a file,
- accesso a periferiche,
- operazioni di I/O,
- accesso ad altre risorse

invocano il supervisor tramite chiamate di sistema (System Call)

Chiamate al Supervisor (cnt)

Perché usare la **modalità supervisor** (i.e, privilegiata)?

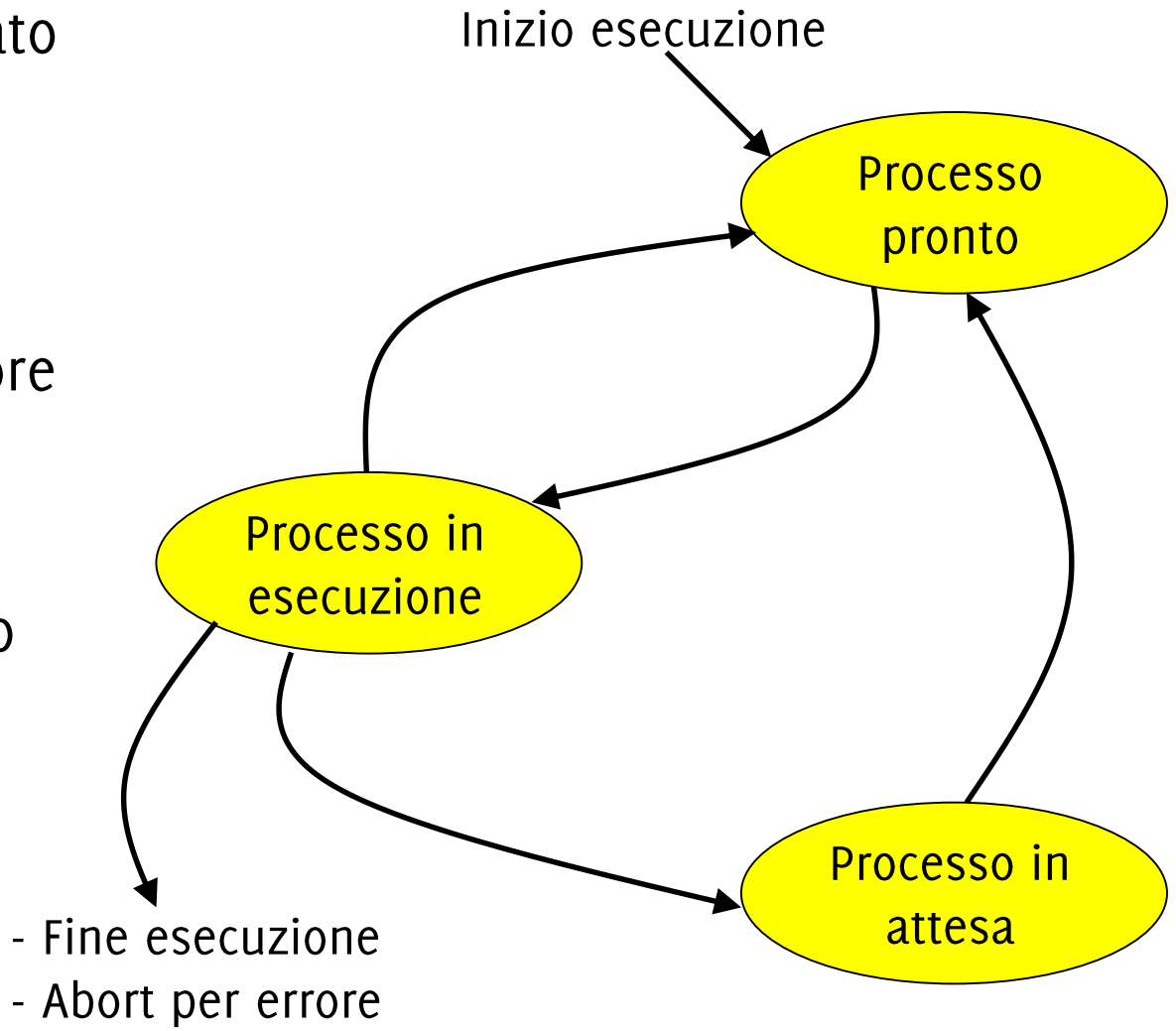
- un processo A non deve poter scrivere messaggi su un terminale non associato allo stesso processo A;
- un processo A non deve poter leggere caratteri immessi da un terminale non associato allo stesso processo A
- Un processo non deve poter sconfinare al di fuori del proprio spazio di memoria:
  - per non accedere allo spazio di memoria associato a un altro processo, modificando codice e dati di quest'ultimo;
  - per non occupare tutta la memoria disponibile nel sistema, bloccandolo e rendendolo così inutilizzabile da altri processi.
- La condivisione di risorse (dischi, CPU, ecc.) deve essere tale da cautelare i dati di ogni utente;

Lo Stato di un Processo

- Lo stato del processo può essere distinto fra stato ***interno*** e stato ***esterno***.
- Lo **stato interno** indica:
  - la **prossima istruzione del programma** che deve essere eseguita;
  - i **valori delle variabili e dei registri** utilizzati dal processo.
- Lo **stato esterno** indica se il processo è:
  - in **esecuzione** (il processore è assegnato al processo);
  - **pronto** per l'esecuzione, e quindi in attesa di accedere alla CPU, quando il SO lo deciderà.
  - in **attesa** di un evento, ad es. la lettura da disco o l'inserimento di dati da tastiera;

Stato di un Processo (1)

- ***In esecuzione***: assegnato al processore ed eseguito da esso
- ***Pronto***: può andare in esecuzione, se il gestore dei processi lo decide
- ***In attesa***: attende il verificarsi di un evento esterno per andare in stato di *pronto*



Algoritmi di Scheduling: Approccio FIFO

- La schedulazione avviene in modo semplice
  - Il primo processo viene mandato in esecuzione
  - Quando termina è il turno del secondo processo
- Semplice da realizzare,
- Inefficiente e inutilizzabile su un sistema interattivo (viene usato nelle workstation per il calcolo scientifico)

Algoritmi di Scheduling: Approccio Ideale

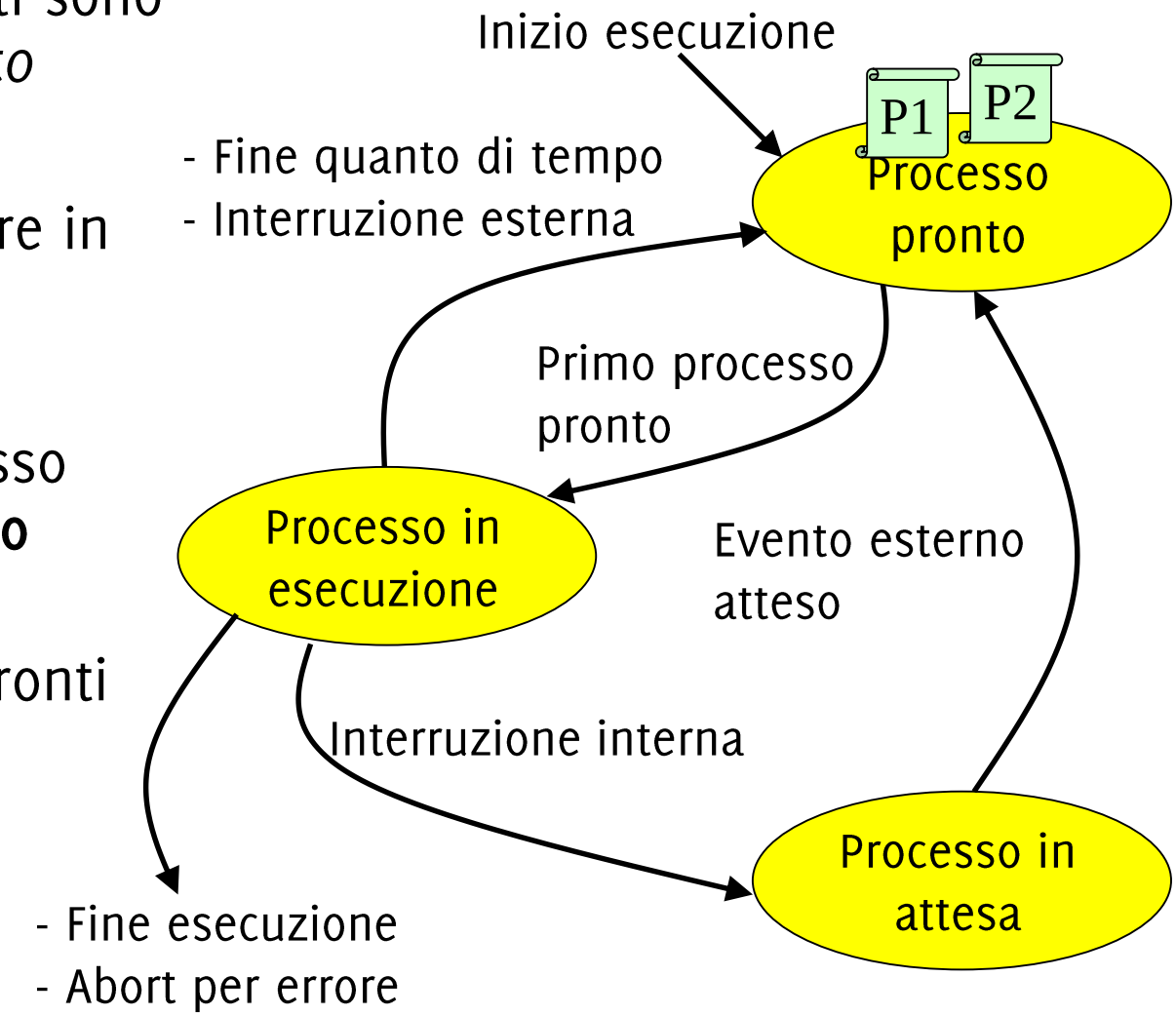
- Il minimo tempo medio di attesa si ottiene eseguendo prima i processi la cui esecuzione è più rapida
- Il problema è che non si conosce a priori quanto un programma rimarrà in esecuzione
- Si corre il rischio di non vedere eseguiti i processi troppo lunghi

Algoritmi di Scheduling: Round Robin

- Molto efficiente, suddivide la CPU tra i vari processi in base al quanto di tempo cadenziato dal clock di sistema.

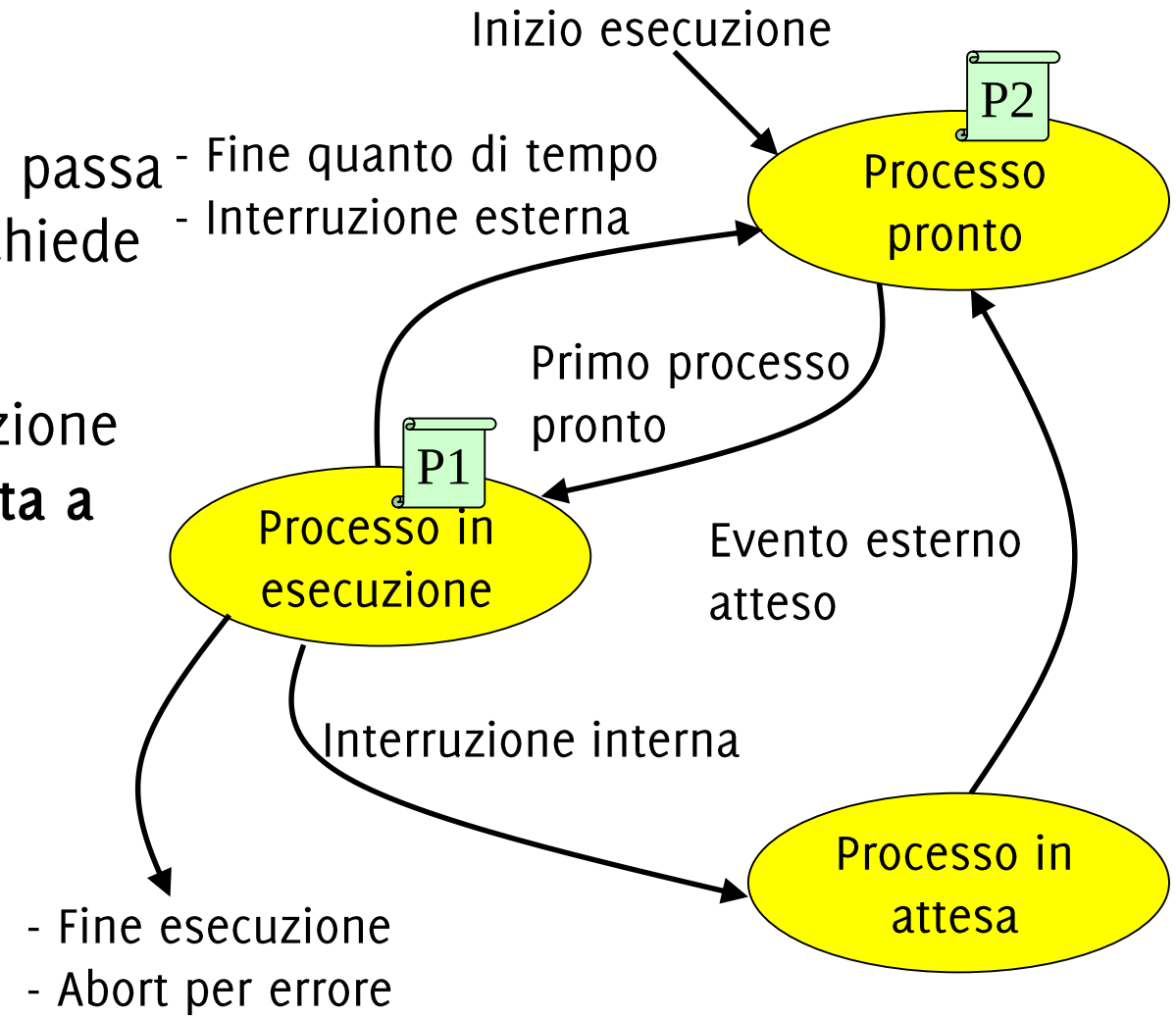
Round Robin – Stato di un Processo (2)

- I processi appena creati sono messi in stato di *pronto*
- Il **kernel** decide quale processo **pronto** mettere in stato **di esecuzione**
- Il **kernel** assegna il processore a un processo per un **quanto di tempo**
  - Round-robin
  - Coda dei processi pronti (FIFO)
  - Priorità dei processi



Round Robin - Stato di un Processo (3)

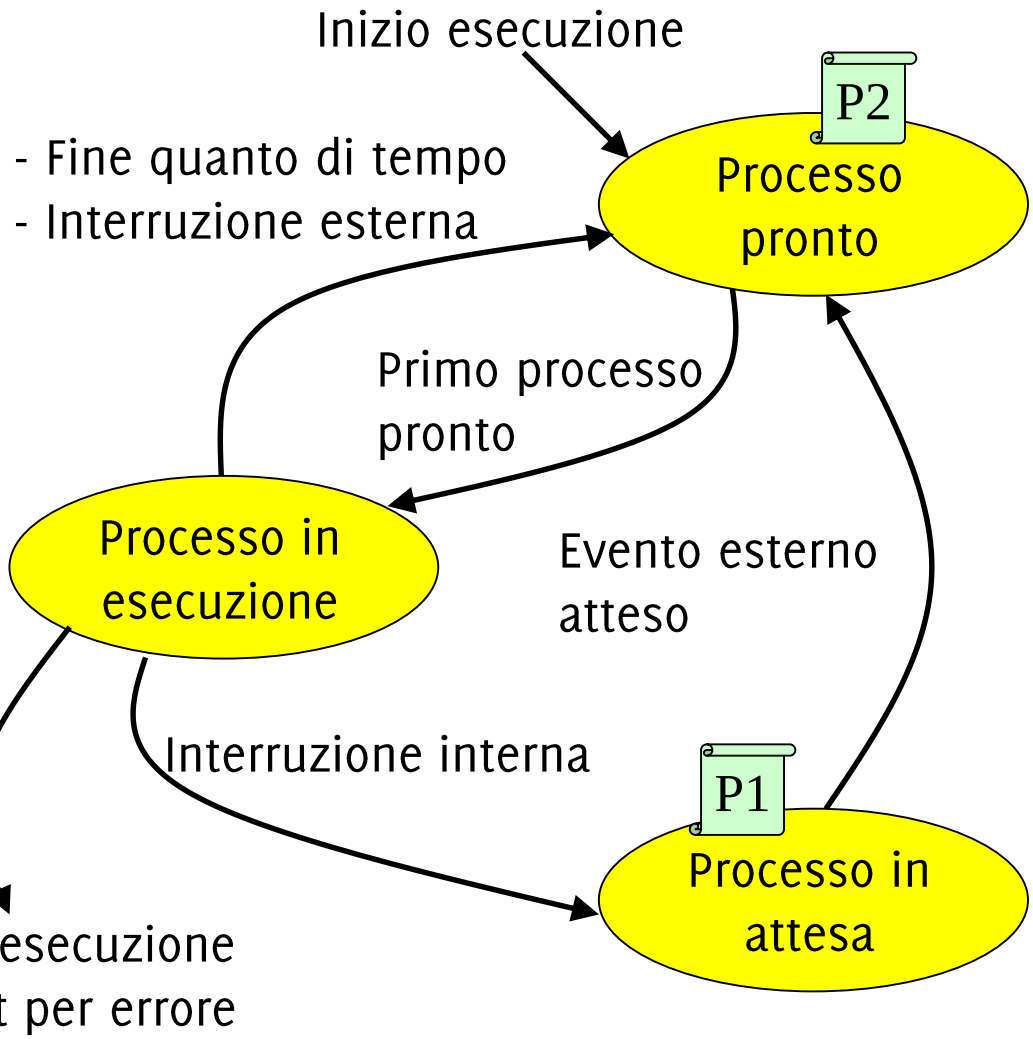
- ***interruzione interna***: Il processo in esecuzione passa in stato di *attesa* se richiede operazioni di I/O
- Corrisponde alla esecuzione dell'istruzione "**chiamata a supervisore**" (*SuperVisor Call*, SVC)



Round Robin - Stato di un Processo (4)

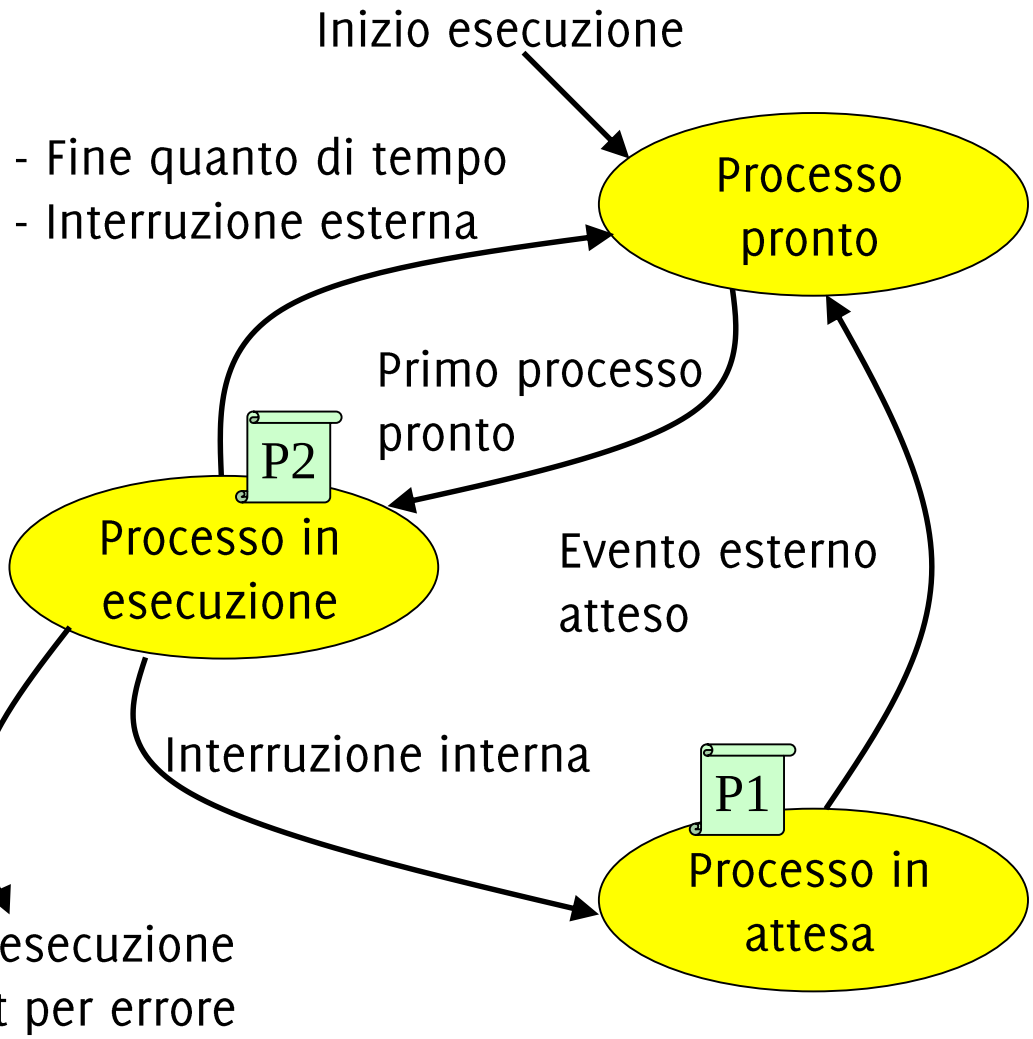
- ***Cambiamento di contesto***:
  Salvare il *contesto* di P1, copia il contenuto dei registri del processore, il codice in esecuzione, etc…, in una zona di memoria, il *descrittore di processo*

- Il processore è ora libero, un altro processo passerà in *esecuzione*



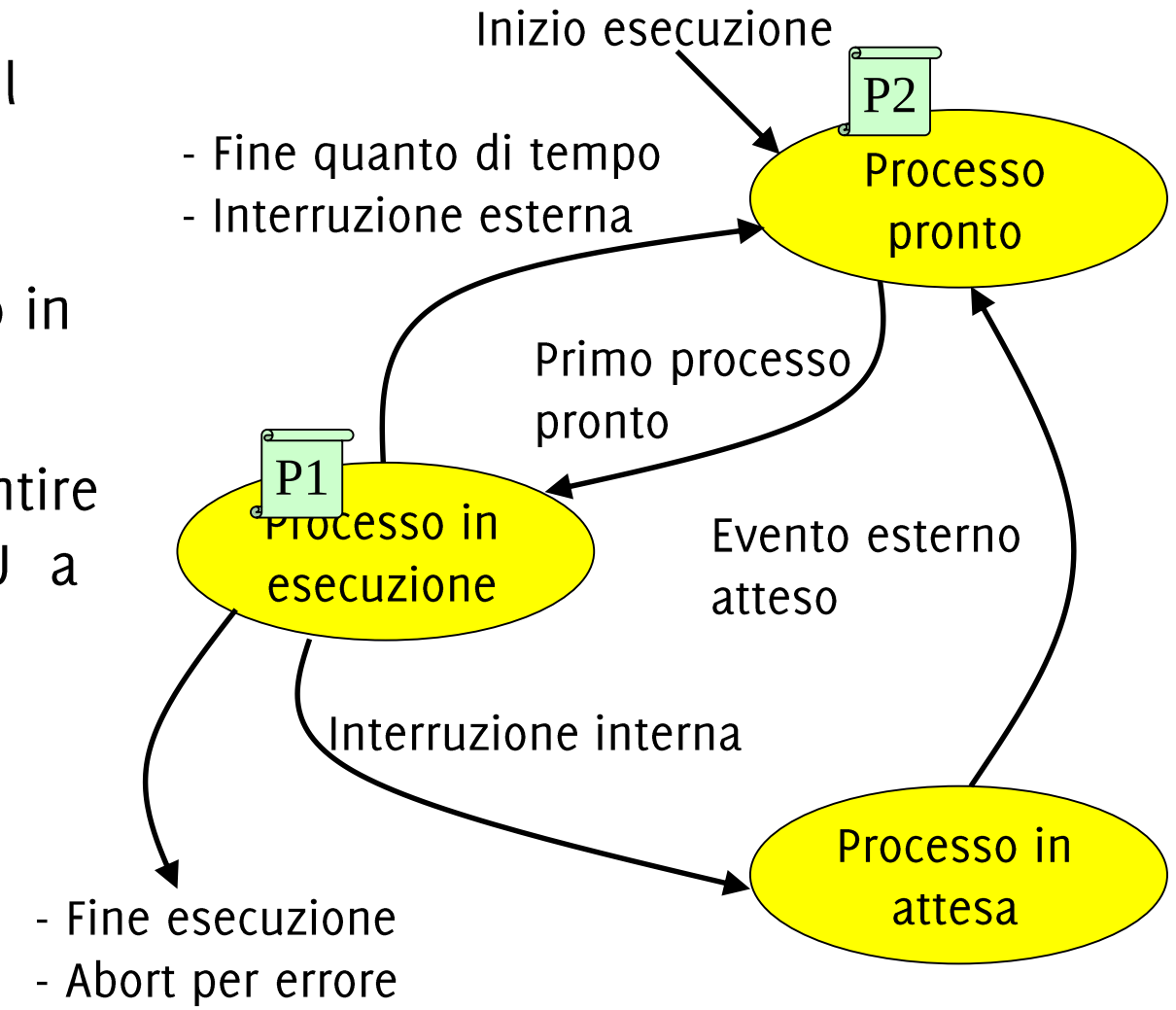
Round Robin - Stato di un Processo (5)

- Quando l'operazione di I/O è finita viene generata un'***interruzione esterna***
- Il processo in esecuzione viene **interrotto**
- Il kernel esegue il ***gestore delle interruzioni*** che esegue le azioni opportune
  - load dati da periferiche
  - **Riporta P1 *pronto***
- Il kernel sceglie quale processo mandare in esecuzione



Round Robin - Stato di un Processo (6)

- ***Pre-emption***: quando il quanto di tempo è scaduto, il nucleo interrompe il processo in esecuzione
- Il kernel cerca di garantire un uso equo della CPU a tutti i processi



La gestione del quanto di tempo

Il quanto di tempo è gestito da una particolare interruzione, generata dall'orologio di sistema:

- **a una frequenza definita**, il dispositivo che realizza **l'orologio** di sistema genera **un'interruzione**.
- Se **il quanto di tempo non è scaduto quando il processo in esecuzione termina,** il processore prosegue nell'esecuzione. I quanti vengono rischedulati

La gestione del quanto di tempo (cnt)

Se invece il **quanto di tempo è scaduto** viene invocata una particolare funzione del kernel (**preemption**) che

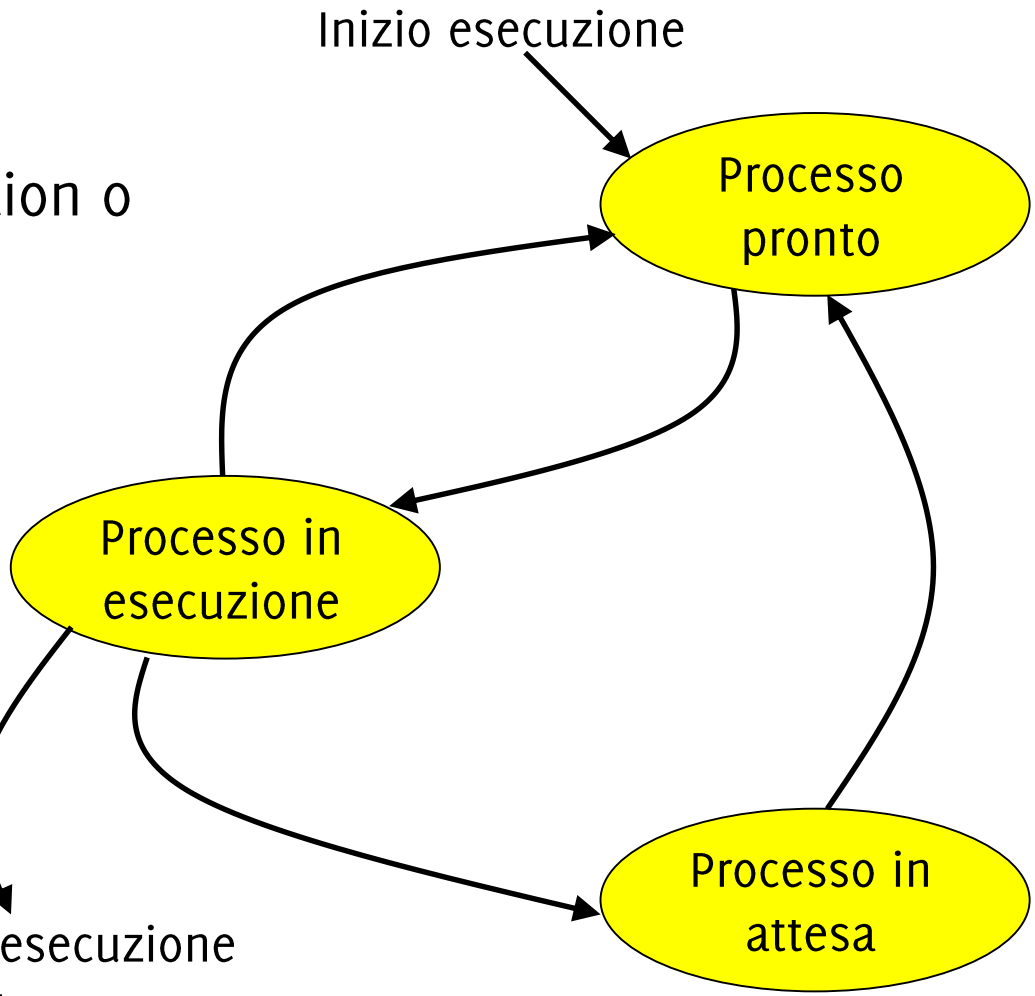
1. cambia lo stato del processo da esecuzione a pronto,
2. **salva il contesto** del processo
3. attiva una particolare funzione del kernel (**change**) che esegue una commutazione di contesto e manda in esecuzione uno dei processi in stato di pronto.

Riepilogando...

Un processo in esecuzione può
- terminare
- tornare in pronto (preemption o interruzione esterna)
- andare in attesa (SVC)

Un processo pronto può
- andare in esecuzione (per andare in attesa deve fare una SVC, quindi essere eseguito)

Un processo in attesa può
- andare in pronto (no esecuzione subito)



Esempio di esecuzione processi in Round Robin

- I seguenti processi in coda con relativa durata in millisecondi, il quanto di tempo dura di 20 ms:

  p1 30 → p2 15 → p3 60 → p4 45

  Assumendo che non nessuno di questi richieda una system call e che i processi in stato di pronto vengano gestiti con una FIFO (first in first out). Visualizzare la sequenza di esecuzione secondo uno schema di Round Robin

Esempio di esecuzione processi in Round Robin

- I seguenti processi in coda con relativa durata in millisecondi, il quanto di tempo dura di 20 ms:

  p1 30 → p2 15 → p3 60 → p4 45

  Verranno eseguiti nel seguente ordine:

  1. p1 (interrotto dopo 20 ms, ne rimangono altri 10 ms)
  2. p2 (termina dopo 15 perché dura meno di 20 ms)
  3. p3 (interrotto dopo 20 ms, ne rimangono altri 40 ms)
  4. p4 (interrotto dopo 20 ms, ne rimangono altri 25 ms)
  5. p1 (termina dopo 10,necessitava di meno di 20 ms)
  6. p3 (interrotto dopo 20 ms, ne rimangono altri 20 ms)
  7. p4 (interrotto dopo 20 ms, ne rimangono altri 5 ms)
  8. p3 (termina dopo 20 ms, necessitava di 20 ms)
  9. p4 (termina dopo 5 ms, necessitava meno di 20 ms)

Esempio di esecuzione processi in Round Robin

- I seguenti processi in coda con relativa durata in millisecondi, il quanto di tempo dura di 20 ms:

  p1 30 → p2 15 → p3 60 → p4 45

  Verranno eseguiti nel seguente ordine:

		Tempo dall'inizio
		0
1.	p1 (interrotto dopo 20 ms, ne rimangono altri 10 ms)	20
2.	p2 (termina dopo 15 perché dura meno di 20 ms)	35
3.	p3 (interrotto dopo 20 ms, ne rimangono altri 40 ms)	55
4.	p4 (interrotto dopo 20 ms, ne rimangono altri 25 ms)	75
5.	p1 (termina dopo 10,necessitava di meno di 20 ms)	85
6.	p3 (interrotto dopo 20 ms, ne rimangono altri 20 ms)	105
7.	p4 (interrotto dopo 20 ms, ne rimangono altri 5 ms)	125
8.	p3 (termina dopo 20 ms, necessitava di 20 ms)	145
9.	p4 (termina dopo 5 ms, necessitava meno di 20 ms)	150

Esempio di esecuzione processi in Round Robin

- I seguenti processi in coda con relativa durata in millisecondi, il quanto di tempo dura di 20 ms:

  p1 30 → p2 15 → p3 60 → p4 45

  Verranno eseguiti nel seguente ordine:

		Tempo dall'inizio
1.	p1 (interrotto dopo 20 ms, ne rimangono altri 10 ms)	0
2.	p2 (termina dopo 15 perché dura meno di 20 ms)	20
3.	p3 (interrotto dopo 20 ms, ne rimangono altri 40 ms)	35
4.	p4 (interrotto dopo 20 ms, ne rimangono altri 25 ms)	55
5.	p1 (termina dopo 10,necessitava di meno di 20 ms)	75
6.	p3 (interrotto dopo 20 ms, ne rimangono altri 20 ms)	85
7.	p4 (interrotto dopo 20 ms, ne rimangono altri 5 ms)	105
8.	p3 (termina dopo 20 ms, necessitava di 20 ms)	125
9.	p4 (termina dopo 5 ms, necessitava meno di 20 ms)	145
		150

5 Contex switches

- Il tempo medio «perso» è la media dei tempi in cui ciascun processo finisce e la sua durata effettiva

[(85-30)+(35-15)+(145-60)+(150-45)]/4 è 66,25 ms.

Esercizio

Siano P e Q due processi lanciati su un sistema monoprocessore. P contiene una scanf, mentre Q non comporta alcuna chiamata al supervisor. Dire se ciascuna delle seguenti affermazioni é vera o falsa. Giustificare le risposte.

« Il processo P potrebbe terminare senza mai essere mai essere nello stato “in attesa” »

Esercizio

Siano P e Q due processi lanciati su un sistema monoprocessore. P contiene una scanf, mentre Q non comporta alcuna chiamata al supervisor. Dire se ciascuna delle seguenti affermazioni é vera o falsa. Giustificare le risposte.

« Il processo P potrebbe terminare senza mai essere mai essere nello stato “in attesa” »

Falso.

Dal momento che contiene una scanf dovrà necessariamente effettuata una supervisor call e il suo stato diverrà “in attesa”

Esercizio

Siano P e Q due processi lanciati su un sistema monoprocessore. P contiene una scanf, mentre Q non comporta alcuna chiamata al supervisor. Dire se ciascuna delle seguenti affermazioni é vera o falsa. Giustificare le risposte.

«Se il processo Q viene lanciato prima di P allora Q termina sicuramente prima di P»

Esercizio

Siano P e Q due processi lanciati su un sistema monoprocessore. P contiene una scanf, mentre Q non comporta alcuna chiamata al supervisor. Dire se ciascuna delle seguenti affermazioni é vera o falsa. Giustificare le risposte.
«Se il processo Q viene lanciato prima di P allora Q termina sicuramente prima di P»

Falso.
Non è possibile sapere quale processo terminerà prima a priori dal momento che ad ogni processo è garantito un solo quanto di tempo alla volta.

Esercizio

Siano P e Q due processi lanciati su un sistema monoprocessore. P contiene una scanf, mentre Q non comporta alcuna chiamata al supervisor. Dire se ciascuna delle seguenti affermazioni é vera o falsa. Giustificare le risposte.

«Una volta lanciato Q rimarrà sempre nello stato “in esecuzione»

Esercizio

Siano P e Q due processi lanciati su un sistema monoprocessore. P contiene una scanf, mentre Q non comporta alcuna chiamata al supervisor. Dire se ciascuna delle seguenti affermazioni é vera o falsa. Giustificare le risposte.

«Una volta lanciato Q rimarrà sempre nello stato “in esecuzione»

Falso:

Non è possibile affermarlo con certezza: se Q dovesse terminare prima dello scadere del quanto di tempo allora rimarrà sempre nello stato “in esecuzione”, viceversa sarà posto nello stato di “pronto”. Potrebbe inoltre subentrare l’interruzione esterna

Si consideri un sistema monoprocessore con i 4 processi in stato di pronto aventi la seguente durata:
P1 75ns
P2 40ns
P3 20ns
P4 40ns
La CPU adotta una politica di tipo Round Robin con quanto di tempo 30ns, nessun processo esegue chiamate al supervisor.
Si assuma che i processi siano in ordinati in una FIFO da P1 a P4, tuttavia P3 e P4 hanno priorità alta, mentre P1 e P2 hanno priorità normale. A parità di priorità viene mandato in esecuzione il processo avanti nella FIFO.

Si descriva la sequenza di esecuzione fino a quando tutti i processi terminano, riportando:

- dopo quanti ns termina ogni processo a partire dall'inizio
- quanti context switch richiede
- il tempo di attesa medio per ogni processo

Processo in Esecuzione	Durata Es.	Tempo trasc.	Termina/CS
P3	20	20	Termina
P4	30	50	CS
P4	10	60	Termina
P1	30	90	CS
P2	30	120	CS
P1	30	150	CS
P2	10	160	T
P1	15	175	T

Tempo medio attesa:
$\frac{1}{n}\sum_{i}^{n}(T_i - D_i)$ dove D_i è la durata prevista del processo P_i, T_i è l'istante in cui termina P_i

((175 - 75) + (160 -40) + (60 -40) + (20 - 20)) / 4 = 80ns