

CAPITOLO A6 - pag 95

GESTIONE DEI PROCESSI

MULTI-PROCESSING



INTRO

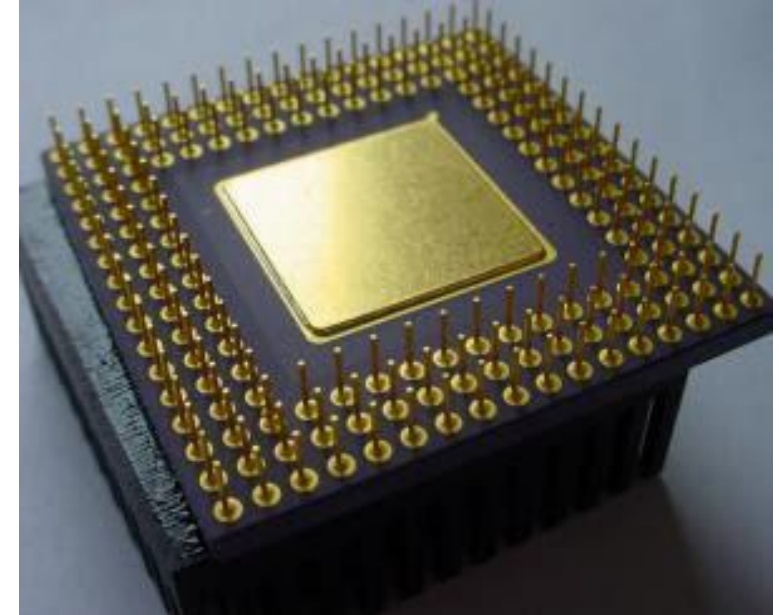
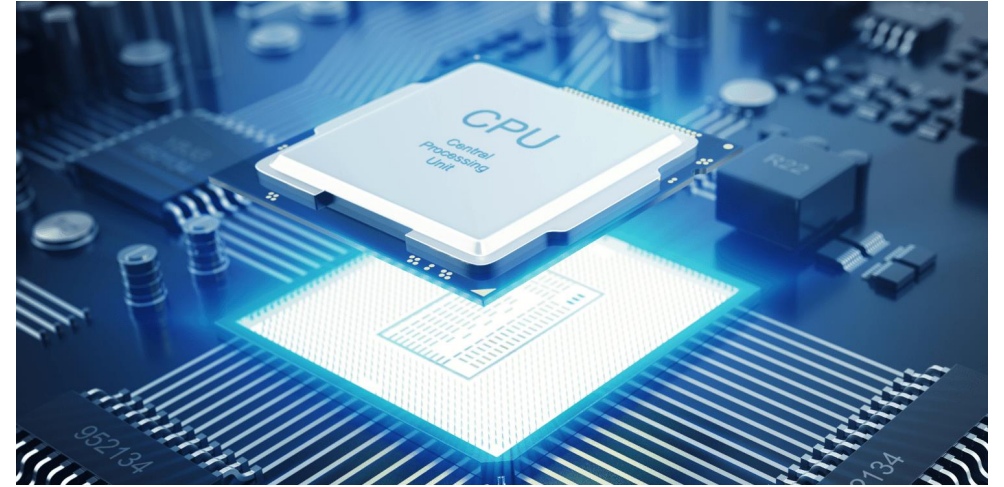
- Tutti i sistemi operativi moderni sono in grado di eseguire «contemporaneamente» un elevato numero di programmi
- la maggior parte dei computer ha un singolo processore multi-core, cioè un singolo processore fisico suddiviso al proprio interno in 2 o 4 unità di elaborazione autonome

HAI MAI SENTITO LA PAROLA CORE???

CORE

Core è un termine utilizzato in informatica per indicare il "nucleo elaborativo" di un microprocessore.

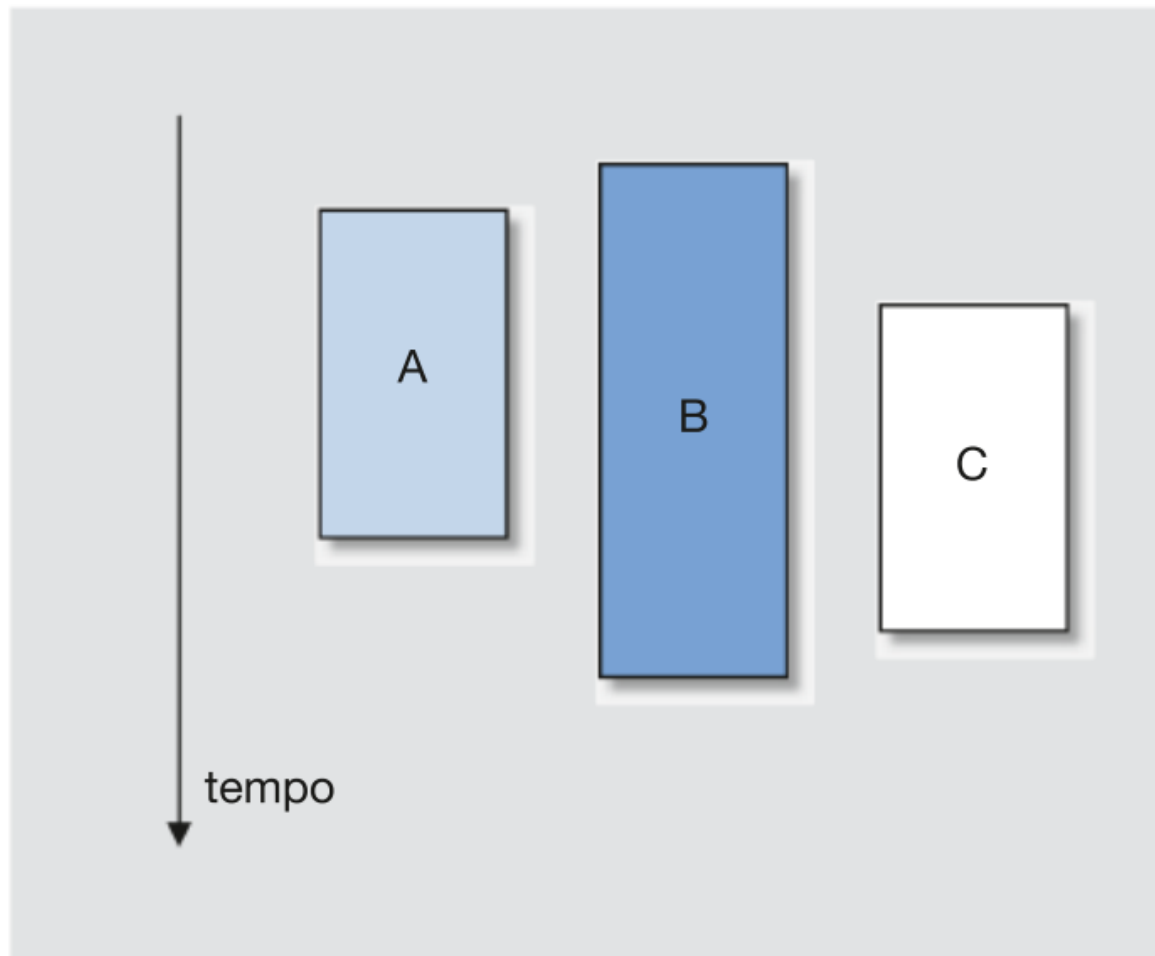
Questo infatti è costituito in realtà da 2 componenti principali: il core appunto, e il package che lo contiene.



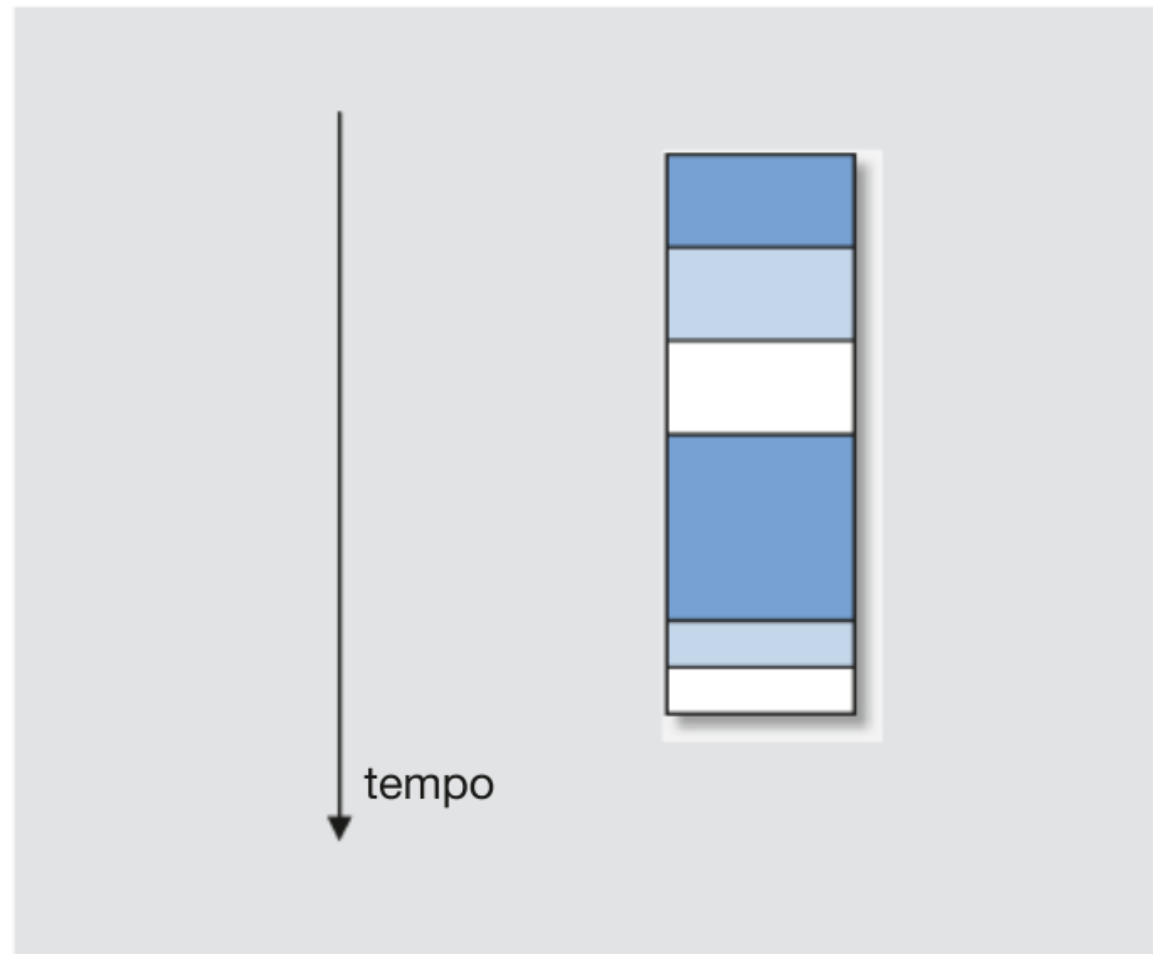
MULTI-PROCESSING

- L'utente ha la sensazione che le proprie applicazioni siano eseguite contemporaneamente dal computer: può, per esempio, ascoltare un brano musicale mentre scrive un documento
- In realtà l'esperienza di contemporaneità è data dall'elevata velocità di esecuzione dei programmi da parte del computer che esegue le applicazioni singolarmente passando rapidamente il controllo dall'una all'altra e, di conseguenza, interrompendone continuamente l'esecuzione

In realtà...



percepiamo questo



ma accade questo

PROGRAMMI E PROCESSI

Programmi e processi - PROCESSO

Un **processo** è definito come un programma in esecuzione.

- Il codice del programma contenuto in un file memorizzato su una unità di memoria persistente; al momento dell'attivazione il codice viene caricato dal sistema operativo nella memoria del computer, da dove può essere effettivamente eseguito: da questo momento il programma diviene un processo

Programmi e processi - PROCESSO

Durante la sua esecuzione un processo impiega, oltre al processore, altre risorse hardware o software rese disponibili dal sistema operativo: memoria allocata, file aperti, connessioni di rete attivate, dispositivi di input e di output utilizzati, ecc.

Queste risorse descrivono lo STATO di un processo.

STATO DEL PROCESSO

I descrittori di tutte le risorse usate da un processo costituiscono lo stato del processo a un certo istante della propria esecuzione.

Inoltre un processo usa e/o produce dati (per esempio un'applicazione multimediale legge il contenuto di un file che contiene un filmato e produce i dati necessari alla scheda video e alla scheda audio per la sua riproduzione)

I **dati** elaborati da un processo sono registrati nelle aree di memoria riservate dal sistema operativo al processo stesso

In sintesi...

un processo è individuato dal programma in esecuzione più l'ambiente di esecuzione costituito dallo stato e dai dati.

OSSERVAZIONE

Due istanze contemporaneamente attive di una specifica applicazione (per esempio di un programma per la redazione di documenti) hanno in comune lo stesso programma, ma hanno un distinto ambiente in quanto i dati (per esempio il testo visualizzato) e le risorse (per esempio il file aperto) sono diverse: sono quindi due processi distinti.

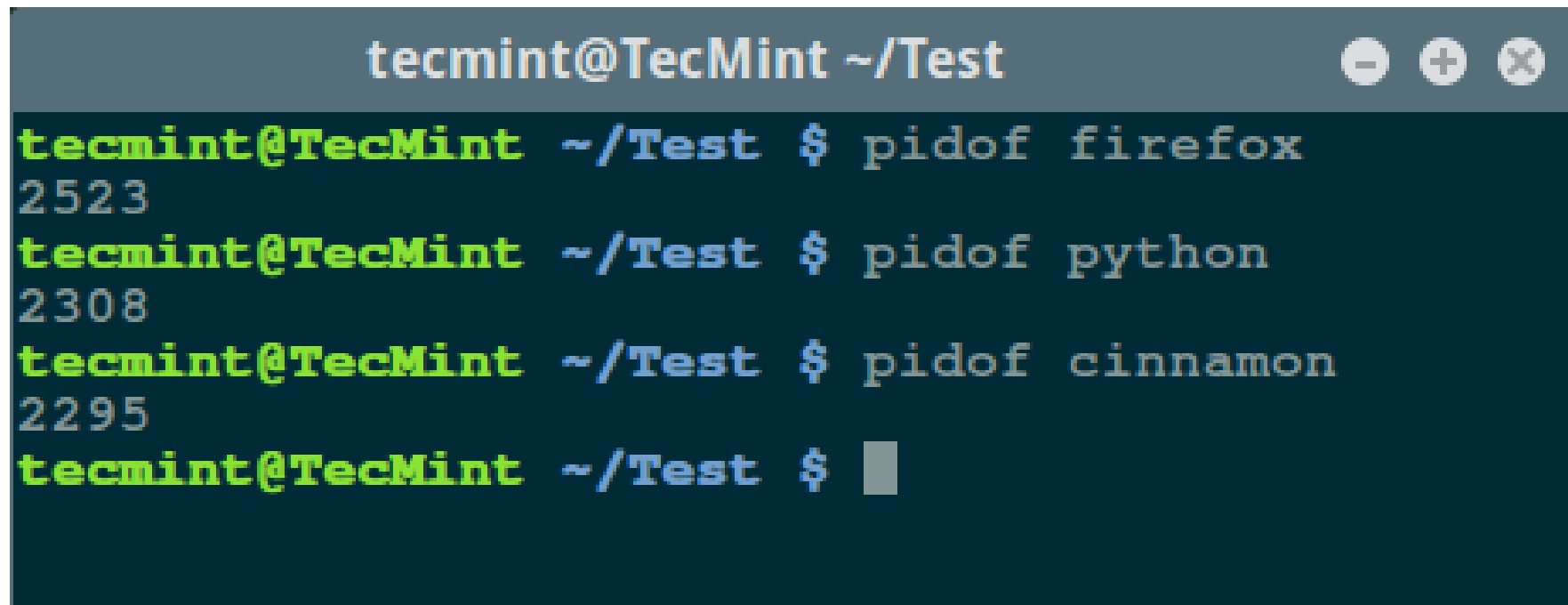
ESEMPIO??????

PID e PCB dei processi

Il sistema operativo associa a ogni processo fin dal momento della sua creazione un identificativo numerico univoco denominato PID (Process ID) e un insieme di informazioni definito Process Control Block (PCB).

PID dei processi

Il sistema operativo associa a ogni processo fin dal momento della sua creazione un identificativo numerico univoco denominato PID (Process ID).

A terminal window with a dark blue background and light blue text. The title bar at the top reads 'tecmint@TecMint ~/Test' and has three window control buttons (minimize, maximize, close) on the right. The terminal shows three commands being executed: 'pidof firefox' returns '2523', 'pidof python' returns '2308', and 'pidof cinnamon' returns '2295'. The prompt is 'tecmint@TecMint ~/Test \$' and a cursor is visible after the last command.

```
tecmint@TecMint ~/Test $ pidof firefox
2523
tecmint@TecMint ~/Test $ pidof python
2308
tecmint@TecMint ~/Test $ pidof cinnamon
2295
tecmint@TecMint ~/Test $
```

Process Control Block (PCB) dei processi

Il PCB di un processo consente al sistema operativo di interromperlo in qualsiasi momento per riprenderne successivamente l'esecuzione e comprende informazioni relative a:

- il PID e l'identificativo dell'utente (può trattarsi del sistema operativo stesso) che ha eseguito il processo
- le risorse hardware e software attualmente utilizzate dal processo
- lo stato del processore al momento della sua ultima interruzione, in particolare l'indirizzo dell'ultima istruzione di codice eseguita (cioè il contenuto del registro Program Counter della CPU)
- i dati relativi al processo e necessari al sistema operativo stesso che vedremo nel dettaglio in seguito

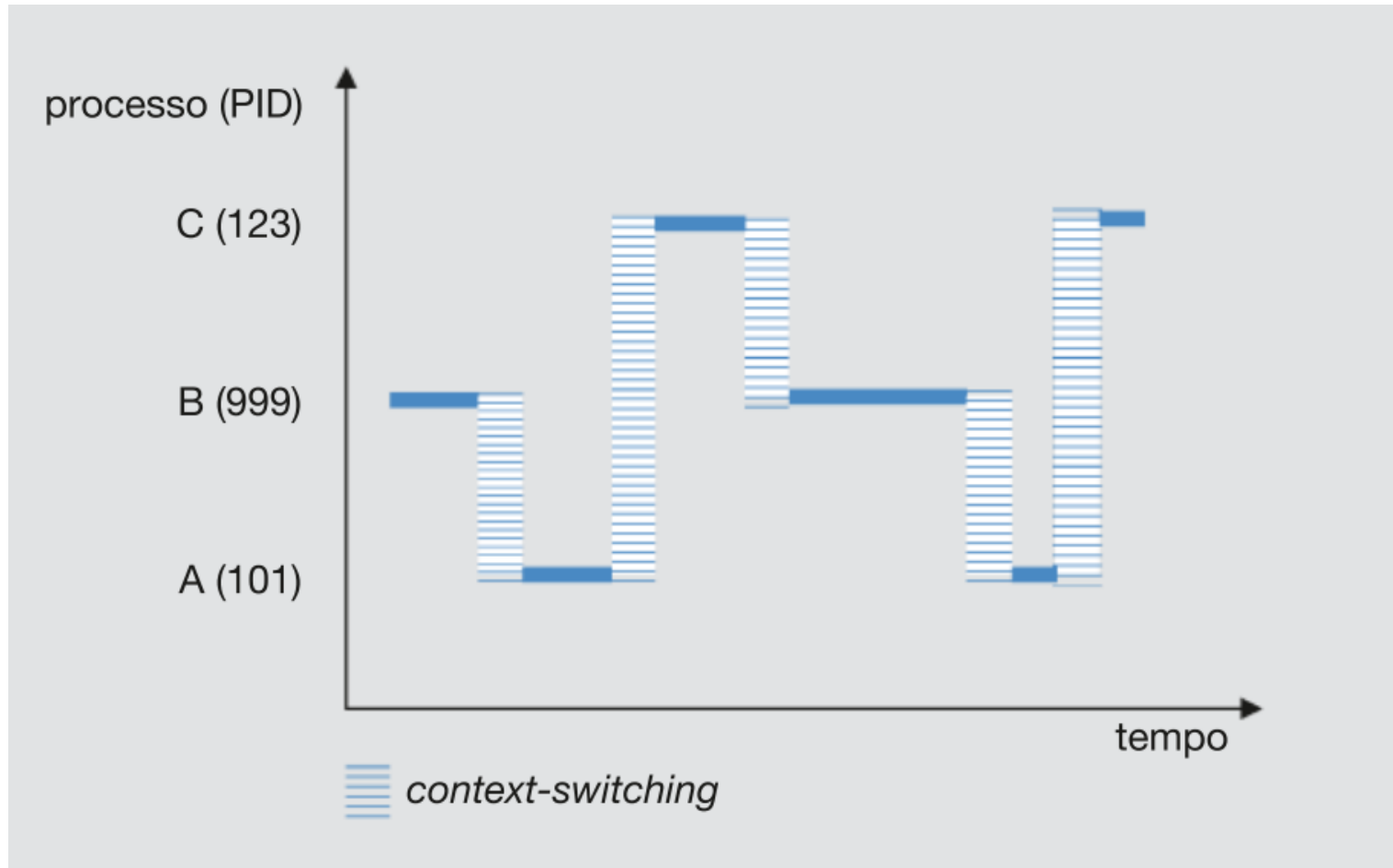
CONTEXT - SWITCHING

Le informazioni contenute nel PCB permettono al sistema operativo di passare la risorsa processore da un processo a un altro, interrompendo momentaneamente l'esecuzione del primo per riprendere temporaneamente l'esecuzione del secondo.

Questo passaggio prende il nome di **context-switching** e prevede il salvataggio del PCB del processo interrotto per ripristinare le informazioni contenute nel PCB del processo da eseguire.

CONTEXT - SWITCHING

Il context-switching è un'attività del sistema operativo, in particolare del gestore dei processi, che impiega la risorsa processore per essere svolta



Osservazione

nella fase di context-switching il gestore dei processi non si limita quindi a effettuare il passaggio della risorsa processore da un processo all'altro, ma eventualmente invoca anche altre e diverse funzioni del kernel del sistema operativo.

Un processo si può interrompere per vari motivi:

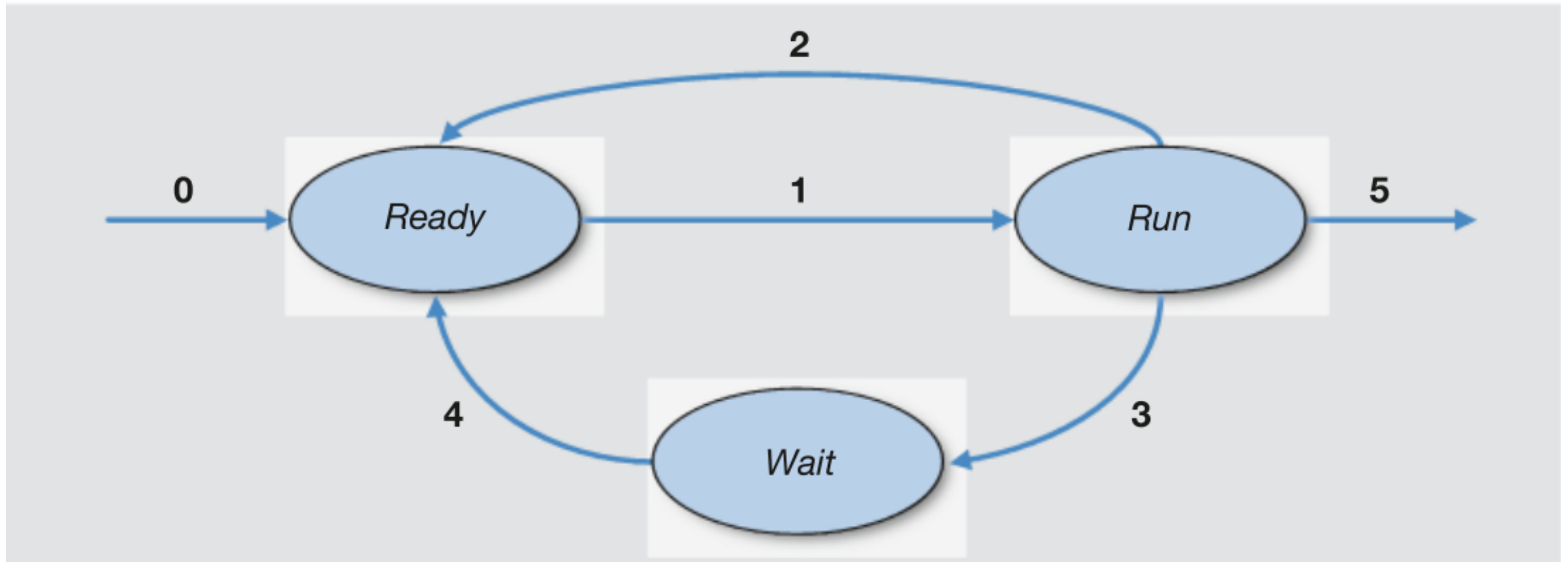
- Risorse che occupano per troppo tempo il processore
- Il processo chiede l'intervento del SO

STATI DI UN PROCESSO

Stati di un processo

- **READY:** Il processo non è attualmente in esecuzione, ma è pronto per essere eseguito in quanto l'unica risorsa mancante è la CPU
- **WAIT:** Il processo ha invocato un servizio del sistema operativo (per esempio ha iniziato un'operazione di lettura/scrittura da/su un file o è in attesa di una segnalazione da parte di un altro processo, cioè ha richiesto una risorsa non ancora disponibile: non può essere eseguito fino al termine della richiesta)
- **RUN:** Il processo è attualmente in esecuzione, cioè dispone di tutte le risorse necessarie al suo avanzamento

Stati di un processo



Stati di un processo

1. Il processo viene creato nello stato Ready
2. Il sistema operativo seleziona il processo per l'esecuzione: passa nello stato Run
3. Il sistema operativo interrompe l'esecuzione del processo che ritorna nello stato Ready
4. Nel corso dell'esecuzione il processo richiede un servizio del sistema operativo: viene posto nello stato Wait
5. Il sistema operativo completa la richiesta di un processo che può così riprendere l'esecuzione: dallo stato Wait passa nuovamente allo stato Ready
6. Il processo termina l'esecuzione e il sistema operativo lo distrugge

Osservazione

- Nello stato di Run vi possono essere al massimo tanti processi quanti sono i processori
- negli stati di Wait e di Ready vi possono essere numerosi processi in attesa di essere sbloccati o selezionati dal sistema operativo
- I PID dei processi sono mantenuti in code che possono o meno essere gestite con una politica First-In First-Out (FIFO) ma ne vedremo altri

Per il sistema operativo non tutti i processi sono «uguali», in particolare rispetto alla politica di selezione per l'esecuzione

COMPITI

- Fai una ricerca di materiali aggiuntivi(video, pdf, animazioni, meme) sui termini visti in questa lezione, sul libro sono evidenziati in blu, e posta quello che trovi nello stream del canale

POLITICHE DI SCHEDULING

pag 100 -103

Obiettivi

- ridurre il tempo complessivo di esecuzione di un processo;
- minimizzare il tempo di attesa da parte dell'utente;
- massimizzare il carico di lavoro dei processori del sistema;
- rispettare le priorità di esecuzione dei processi, impedendo che un processo con bassa priorità non venga mai eseguito.
- massimizzare il risparmio energetico da parte del processore;
- minimizzare il riscaldamento del processore.

Processi I/O bound

Prevedono una continua interazione con l'utente—passano molto tempo nello stato di Wait ad attendere come input un'azione (per esempio l'attivazione del mouse in una specifica posizione, oppure la digitazione di un testo)

CPU bound

Devono svolgere un'intensa attività di elaborazione di dati presenti in memoria – non rilasciano quasi mai il processore passando nello stato di Wait e sono dipendenti principalmente dalla disponibilità del processore

Ottimizzazione - TIME SHARING

- a ogni processo viene assegnato un «quanto» di tempo massimo dopo il quale deve obbligatoriamente rilasciare il processore passando dallo stato di Run a quello di Ready
- Ovviamente, deve ovviamente essere superiore al tempo medio di context-switching

Un esempio: round - robin

- Algoritmo di schedulazione che abbina la tecnica del time-sharing con la politica FIFO di selezione dei processi in coda nello stato di Ready e consente all'utente del computer di avere la sensazione di esecuzione contemporanea di più programmi interattivi
- tutti i processi – compresi i processi di servizio del sistema operativo – sono selezionati per l'esecuzione in modo rigorosamente sequenziale; infatti non è possibile fare in modo che un processo assuma un'importanza maggiore rispetto a un altro.

Priorità

- Molti sistemi operativi associano a ogni processo un valore di priorità in base al quale il gestore dei processi seleziona dalla coda dei processi in stato di Ready il primo processo con priorità più alta anziché semplicemente il primo in ordine di attesa.
- Altri esempi a pag 102 - 103

Multi-threading in Windows e Linux

- Nei moderni sistemi operativi ogni singolo processo può essere composto da più elementi separati e attivi contemporaneamente – denominati thread e usualmente identificati da un valore numerico univoco – che sono singolarmente schedulati per l'esecuzione da parte del gestore dei processi: a questo scopo a ogni thread è associato un TCB (Thread Control Block).

Idle thread - Windows

- Windows a ogni singolo thread associa una priorità (un valore numerico compreso tra 0 e 31): le priorità da 1 a 15 sono adattate automaticamente dal gestore dei processi, mentre le priorità da 16 a 31 (denominate real-time priority) sono costanti
- il thread da eseguire viene selezionato tra quelli in stato di Ready aventi la priorità più alta; se non ci sono thread nello stato di Ready viene eseguito un thread speciale denominato idle thread

PARTE INTEGRATIVA

Processi Pre-emptive e Non pre-emptive

Non tutti i processi possono essere sospesi dal SO in ogni istante della loro esecuzione: per loro natura alcuni processi devono terminare la loro esecuzione

Questi processi vengono chiamati **non pre-emptive**, a differenza di quelli che possono essere interrotti che sono i processi **pre-emptive**.

I sistemi a divisione di tempo (time sharing) hanno uno scheduling pre-emptive


Algoritmo di scheduling FCFS

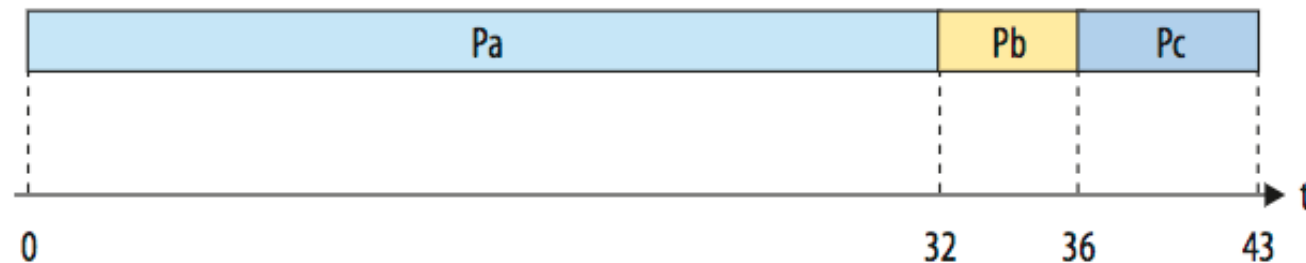
- First-Come-First-Served, cioè “il primo arrivato è il primo a essere servito”.
- i processi in coda secondo l'ordine d'arrivo, FIFO (First In First Out), e indipendentemente dal tipo e dalla durata prevista per la loro esecuzione; inoltre questo algoritmo è di tipo **non pre-emptive**
- DIFETTO: tutti gli altri devono aspettare la naturale terminazione dei processi che li precedono

Algoritmo di scheduling FCFS – calcolo del tempo di attesa medio – esempio 1

Supponiamo di avere tre processi che rispettivamente richiedono:

- $P_a = 32$ unità di tempo/CPU
- $P_b = 4$ unità di tempo/CPU
- $P_c = 7$ unità di tempo/CPU

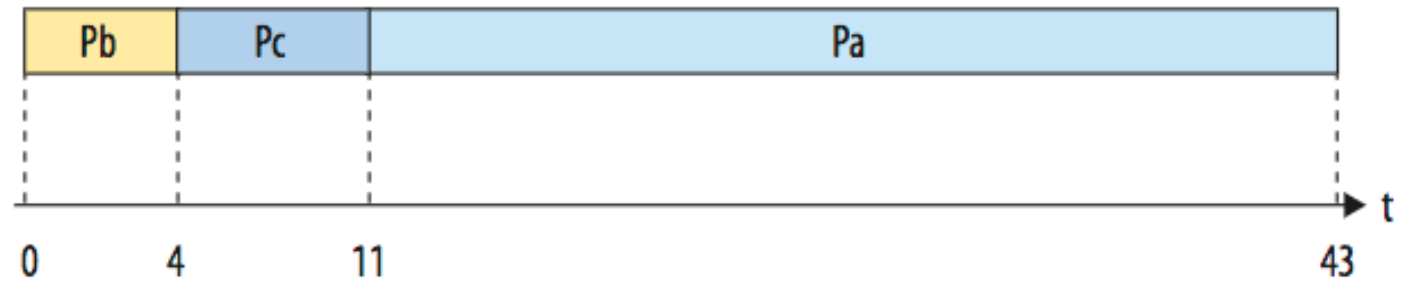
Nel primo caso i tre processi arrivano nell'ordine P_a , P_b e P_c . Calcoliamo il tempo medio di attesa in questa situazione rappresentandolo su di un diagramma di **Gantt** :



$$ta_{m1} = \frac{ta_a + ta_b + ta_c}{3} = \frac{0 + 32 + 36}{3} = 22.7$$

Algoritmo di scheduling FCFS – calcolo del tempo di attesa medio – esempio 2

Nel secondo caso l'ordine è diverso: Pb, Pc e Pa. Calcoliamo ora il nuovo tempo medio di attesa:



$$ta_{m2} = \frac{ta_a + ta_b + ta_c}{3} = \frac{0 + 4 + 11}{3} = 5$$

Dopo questo esempio notiamo che il tempo medio di attesa non è un parametro significativo perché risulta essere totalmente casuale

Algoritmo di scheduling SJF

- scegliere tra la lista dei processi pronti quello che occuperà per meno tempo la cpu(Shortest Job First)
- Inoltre potrebbe verificarsi la situazione in cui mentre un processo è in esecuzione se ne aggiunge uno in coda con un tempo stimato minore; in questo caso possiamo avere due possibilità:
 - nel caso di situazione non pre-emptive non si fa nulla;
 - nel caso di situazione pre-emptive è necessario stimare per quanto tempo ancora il processo deve rimanere in esecuzione e confrontarlo con il tempo previsto per il nuovo processo: se questo è minore, si deve effettuare la sospensione e il cambio del contesto assegnando la CPU al nuovo processo

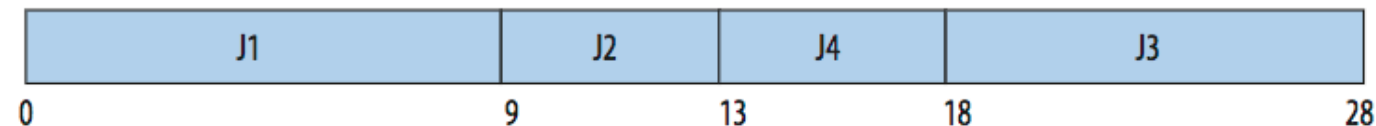
Algoritmo di scheduling SJF – calcolo del tempo di attesa medio – esempio 1

CALCOLO DEL TEMPO DI ATTESA NEL CASO SJF

Calcoliamo il tempo medio per la situazione presentata di seguito, dapprima utilizzando l'algoritmo **SJF** e quindi migliorandolo con il **SRTF**.

Job	Durata	Arrivo
J1	9	0
J2	4	1
J3	10	2
J4	5	3

La sequenza di attivazione con l'algoritmo **SJF** è la seguente:



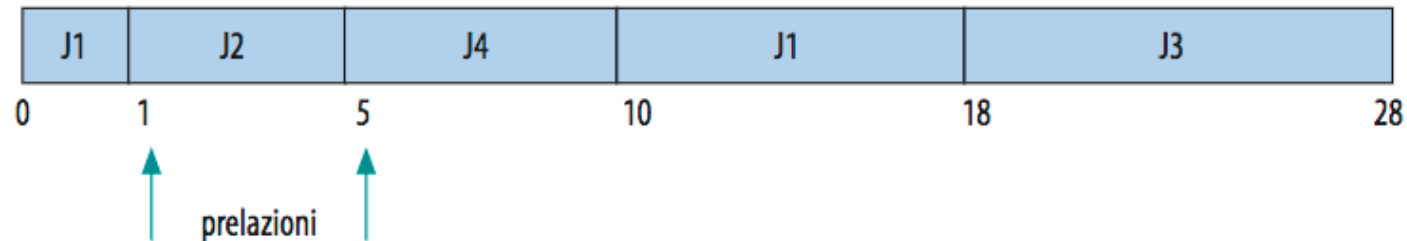
Il tempo di attesa è così calcolato, sottraendo a ogni job il "tempo di arrivo in coda":

$$Tw = (J1, J2, J4, J3) = [0 + (9 - 1) + (13 - 3) + (18 - 2)] / 4 = 8.5$$

Algoritmo di scheduling Shortest Remaining Time First, calcolo del tempo di attesa medio, esempio

Nel caso precedente si è ipotizzata una situazione di **SJF non pre-emptive**: vediamo ora come si comporterebbe un **SFRT pre-emptive**: se arriva un nuovo processo con **burst** di esecuzione più corto questo effettua una "prelazione" rispetto al processo in esecuzione.

Il job **J1** va in esecuzione ma all'istante di tempo 1 giunge il processo **J2** che ha un **burst** minore del **burst** rimanente di **J1** (4 contro 9-1): **J1** in esecuzione viene sospeso e viene mandato in esecuzione **J2**; analoga situazione avviene all'arrivo del job **J4** che ha un **burst** di durata = 5, inferiore sia a quello di **J3** che del residuo di **J1**. Il diagramma di **Gantt** completo è il seguente:



Il tempo di attesa medio è:

$$T_w = (J1, J2, J4, J1, J3) = [0 + (1 - 1) + (5 - 3) + 10 + (18 - 2)] / 4 = 7$$

Job	Durata	Arrivo
J1	9	0
J2	4	1
J3	10	2
J4	5	3

Scheduling con priorità(prelazione)

Nella procedura di scheduling con priorità a ogni processo viene associato un numero intero che corrisponde a un livello di priorità con il quale deve essere poi mandato in esecuzione: lo scheduler seleziona tra tutti i processi in coda quello a priorità più alta che avrà la precedenza di esecuzione su tutti.

Alla sua terminazione verrà scelto il processo che ha la massima priorità tra quelli rimasti e via di seguito; nel caso di due processi con medesima priorità si serve per primo il primo arrivato in coda, come nella FCFS. La priorità viene assegnata ai processi dallo stesso sistema operativo, in base a criteri legati al tipo di processo e all'utente che lo ha mandato in esecuzione.

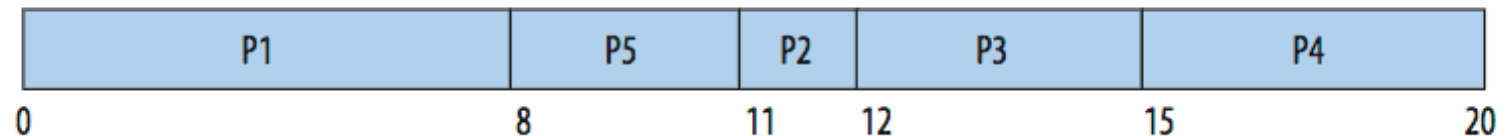
Algoritmo di scheduling con priorità – calcolo del tempo di attesa medio – esempio 1

SEQUENZA DI ATTIVAZIONE

In questo esempio visualizziamo la sequenza dei processi in base alla seguente tabella di priorità.

Processo	Durata	Priorità
P1	8	1
P2	1	3
P3	3	4
P4	5	5
P5	3	2

La sequenza di attivazione è la seguente:



Il tempo di attesa medio è:

$$T_w = (P1, P5, P2, P3, P4) = (0 + 8 + 11 + 12 + 15) / 5 = 9.2$$

Algoritmo di scheduling con priorità

Gli algoritmi con priorità possono essere di tipo sia non pre-emptive sia pre-emptive: in questo caso, se si sta servendo un processo con priorità più bassa di uno nuovo appena giunto in coda, si cede la CPU a quello con priorità maggiore sospendendo il processo in esecuzione in quel momento.

Se continuano ad arrivare processi con alta priorità può avvenire il fenomeno della starvation dei processi, cioè i processi con priorità maggiore vengono sempre serviti a scapito di quelli con priorità bassa, che possono rimanere in coda anche per tempi indefiniti

Starvation

La starvation dei processi si verifica quando uno o più processi di priorità bassa rimangono nella coda dei processi pronti per un tempo indefinito in quanto sopraggiungono continuamente processi pronti di priorità più alta.

Algoritmo di scheduling Round Robin

- tutti i processi pronti vengono inseriti in una coda circolare di tipo FIFO, cioè inseriti in ordine di arrivo, tutti senza priorità, e a ogni processo viene assegnato un intervallo di tempo di esecuzione prefissato denominato quanto di tempo (o time slice) di durata che varia tra 10 e 100 millisecondi
- Se al termine di questo intervallo di tempo il processo non ha ancora terminato l'esecuzione, l'uso della CPU viene comunque affidato a un altro processo, prelevandolo sequenzialmente dalla coda e sospendendo il processo che era in esecuzione.
- Possiamo osservare che l'algoritmo RR può essere visto come un'estensione di FCFS con pre-emption periodica a ogni scadenza del quanto di tempo.
- Con questo algoritmo tutti i processi sono trattati allo stesso modo, in una sorta di "correttezza" (fairness), e possiamo essere certi che non ci sono possibilità di starvation perché tutti a turno hanno diritto a utilizzare la CPU.

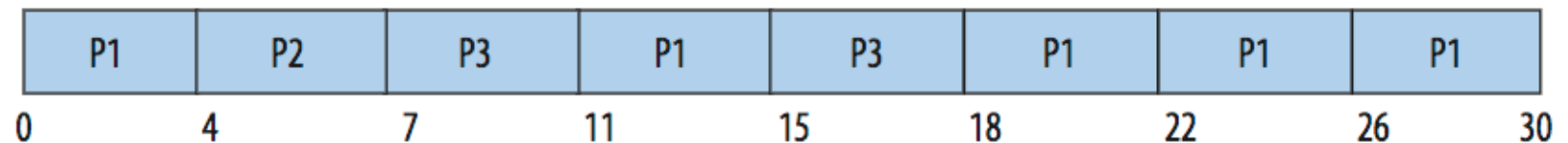
Round Robin - esempio

SEQUENZA DI ATTIVAZIONE

La sequenza di attivazione è riportata di seguito, considerando il quanto di tempo = 4:

Processo	Durata
P1	20
P2	3
P3	7

Il tempo di attesa medio è:



$$T_w = (P1, P2, P3) = 11 / 3 = 3.6$$

Real-time clock (RTC)

- dispositivo che rende possibile la sospensione di un processo ancora in esecuzione allo scadere del time slice
- un chip impiantato sulla scheda madre contenente un cristallo di quarzo che viene fatto oscillare in modo estremamente stabile con segnali elettrici: tali oscillazioni scandiscono il tempo generando periodicamente delle interruzioni da inviare al sistema operativo

TIME SLICE

- quando è piccolo abbiamo tempi di risposta ridotti ma è necessario effettuare frequentemente il cambio di contesto tra i processi, con notevole spreco di tempo e quindi di risorse (overhead)
- quando è grande i tempi di risposta possono essere elevati e l'algoritmo degenera in quello di FCFS



ADESSO IL BRAVO STUDENTE |
FARA' UNA DOMANDA....

Ma allora il migliore??????

La soluzione ottimale perciò non esiste: i moderni sistemi operativi combinano tra loro gli algoritmi qui presentati cercando in primo luogo di eliminare i problemi tipici di ogni algoritmo per ottenere una soluzione che possa essere mediamente buona per tutte le situazioni

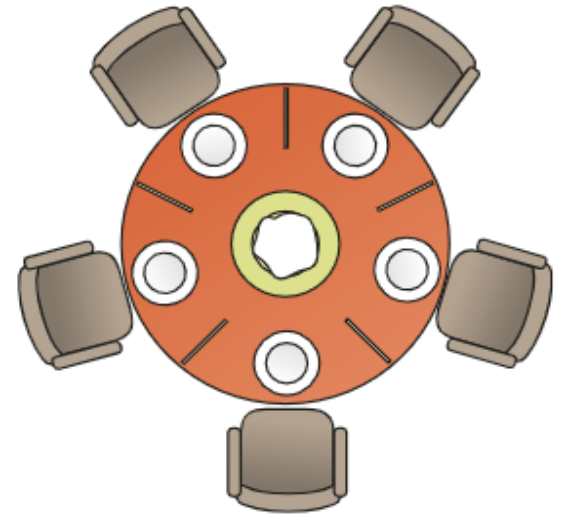
Il problema dei cinque filosofi

Cinque filosofi stanno seduti intorno a un tavolo; ciascun filosofo ha di fronte un piatto e tra ogni piatto vi è una bacchetta per il riso (cinque filosofi, cinque piatti e cinque bacchette).

I filosofi alternano il loro tempo pensando o mangiando: quando un filosofo comincia ad avere fame, per poter mangiare deve avere due bacchette per poter prendere il riso, e quindi per prima cosa cerca di prendere possesso della bacchetta di sinistra e, successivamente, se va a buon fine, di quella di destra, una alla volta in ordine arbitrario.

Inizia a mangiare quando ha a disposizione entrambe le bacchette e quando si è saziato depone le bacchette e riprende a pensare.

Supponiamo che a un certo punto tutti i filosofi si trovino nella situazione di avere una bacchetta e di attendere la seconda, che non otterranno mai in quanto tutte le cinque bacchette sono in mano ad altrettanti filosofi: il loro destino è, purtroppo, la "morte per fame"!



Nell'esempio appena descritto avviene quello che prende il nome di **dead-lock**, o abbraccio mortale: i processi (filosofi) si ostacolano a vicenda impedendo ciascuno l'avanzamento dell'altro trattenendo una risorsa (una bacchetta) e contemporaneamente danneggiando anche se stesso.



È quindi necessario che il sistema operativo gestisca anche questi casi, cioè la **sincronizzazione dei processi** tra loro indipendenti che però interagiscono in quanto utilizzano risorse in comune.