

## Multiprogrammazione

La multiprogrammazione nasce come un metodo per massimizzare l'uso della CPU piuttosto che come uno strumento per fornire un miglior servizio agli utenti. Solo in un secondo momento, infatti, la tecnica è stata impiegata per realizzare sistemi conversazionali. A questo proposito vale forse la pena ricordare che:

- obiettivo della *multiprogrammazione* è la massimizzazione dell'uso delle risorse di calcolo, attraverso l'esecuzione contemporanea/concorrente di più processi. In ogni caso, istante per istante, il numero di processi effettivamente in esecuzione non può eccedere il numero di processori disponibili;
- obiettivo del *time-sharing*, invece, è quello di consentire agli utenti di interagire con il proprio processo come se questo fosse l'unico presente sul sistema; questa illusione viene ottenuta eseguendo frequentemente la commutazione da un processo ad un altro.

### **Un modello per la multiprogrammazione**

Vediamo la ragione per la quale, attraverso il ricorso alla multiprogrammazione, si incrementa l'efficienza nell'uso della CPU.

Supponiamo che un processo spenda una frazione  $p$  del suo tempo di esecuzione in attesa del completamento di operazioni di I/O; quindi, il tempo in cui la CPU è effettivamente impegnata dal processo è  $1-p$ . Se  $p$  è la frazione di tempo per un determinato intervallo di tempo di riferimento  $T$  in cui il processo è bloccato per I/O, allora  $p$  è anche la probabilità che, istante per istante, il processo sia bloccato per I/O; infatti, supponendo una distribuzione uniforme, si ha

$$P\{\text{processo bloccato per I/O}\} = \frac{pT}{T} = p$$

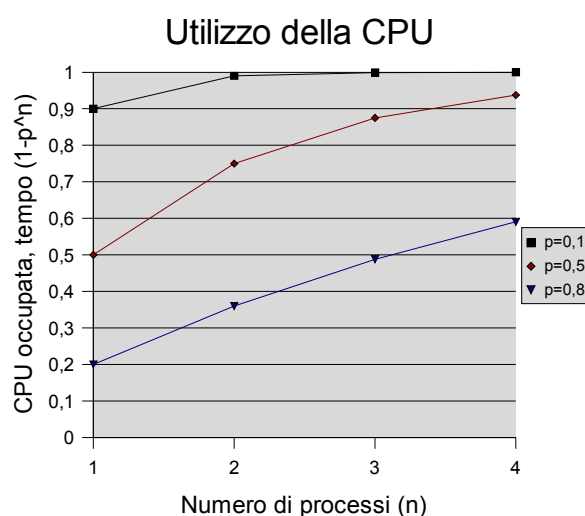
Se si hanno  $n$  processi indipendenti, la probabilità che la CPU risulti inutilizzata è esattamente quella per cui tutti i processi sono impegnati in operazioni di I/O, e cioè  $p^n$ .<sup>1</sup> Quindi, la probabilità che la CPU sia occupata è pari a  $1-p^n$ ; il parametro  $n$

<sup>1</sup> Il passaggio dal dominio del tempo a quello delle probabilità si rende necessario per consentire l'estensione al caso

prende il nome di **grado di multiprogrammazione**. Tornando nel dominio del tempo, per un periodo di riferimento T il tempo di utilizzo effettivo della CPU è pari a  $(1-p^n)T$ .

Il seguente diagramma mostra il grado di utilizzo del processore per diversi valori di n e p. In particolare, esso mette in luce il diverso grado di utilizzo della CPU per due diverse categorie di processi:

- *I/O-bound*. Processi che fanno un uso intensivo dei dispositivi periferici, e la cui esecuzione è caratterizzata da molti brevi periodi (*bursts*) di utilizzo della CPU;
- *CPU-bound*. Sono processi che spendono più tempo in elaborazioni che non nello svolgimento di operazioni di I/O; la loro esecuzione è caratterizzata da pochi e prolungati periodi di utilizzo della CPU.



CPU time ( $1-p^n$ )	0,9	0,990	0,999	1,000	CPU bound
	0,5	0,750	0,875	0,938	
	0,2	0,360	0,488	0,590	I/O bound
# processi	1	2	3	4	

**Figura 1:** andamento dell'utilizzo della CPU al variare del grado di multiprogrammazione, per diversi valori di  $1-p$ .

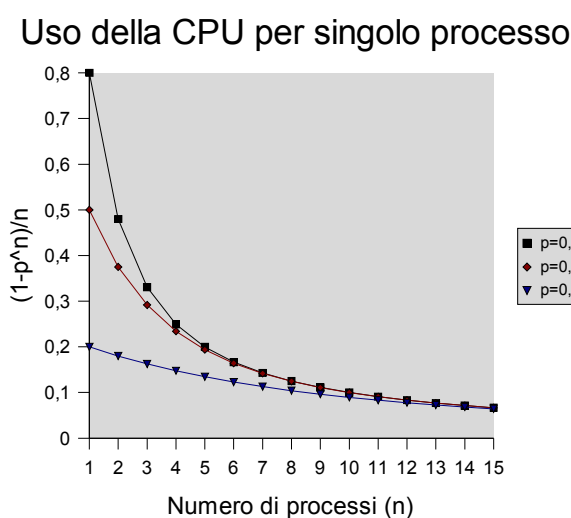
In realtà quello presentato è un modello ideale, dal quale la realtà si discosta perché i processi non sono tra loro indipendenti; infatti concorrono all'uso di risorse

---

di N processi (per la probabilità si può dire che, nel caso di processi indipendenti, la probabilità dell'evento congiunto è il prodotto delle probabilità degli eventi semplici); a questo punto si può tornare nel dominio del tempo.

spartite, tra cui la CPU. Un'analisi più accurata richiederebbe pertanto un modello basato sulla teoria delle code. Tuttavia, anche con questo modello semplificato si riesce a cogliere il vantaggio della multiprogrammazione, e cioè che al crescere del grado di multiprogrammazione cresce l'utilizzo del processore (sempreché, come vedremo più in là, ci sia memoria sufficiente per tutti i processi!).

Da notare che al crescere del grado di multiprogrammazione  $n$  la frazione di tempo dedicata a ciascun processo diminuisce; infatti, essa è pari a  $\frac{(1-p^n)}{n}$ .



**Figura 2:** andamento della frazione di tempo dedicata a ciascun processo al variare del grado di multiprogrammazione.

**Esempio:** Sia dato un insieme di 4 processi, tutti caratterizzati da un proprio tempo di arrivo e da un tempo di CPU necessario per completarne l'esecuzione, come mostrato nella tabella riportata di seguito. Tutti e quattro i processi passano l'80% del loro tempo in attesa del completamento di operazioni di I/O ( $p=0.8$ ;  $1-p=0.2$ ).

	Tempo di arrivo	Tempo di CPU (minuti)
P1	10:00	4
P2	10:10	3
P3	10:15	2
P4	10:20	2

Se ci fosse solo il processo 1, questo richiederebbe un tempo di

esecuzione pari a  $\frac{4}{(1-0.8)}=20$  minuti,

infatti  $T_{CPU}=(1-p)T_{Tot} \Rightarrow T_{Tot}=\frac{T_{CPU}}{(1-p)}$ .

Tuttavia, poiché successivamente al suo arrivo sopraggiungono anche altri processi, le cose andranno in maniera un po' diversa.

Dalle 10:00 alle 10:10 c'è solo P1, che in quel periodo occupa la CPU per  $(1-p)*T=10*0.2=2$  minuti.

Dalle 10:10 alle 10:15 sono in esecuzione sia P1 che P2. In questo periodo, l'utilizzo della CPU è pari a  $(1-0.8^2)=0.36$  (che è un valore sicuramente superiore a 0.2, che si aveva con il solo P1 in esecuzione). In questo periodo ciascuno dei due processi sfrutta il processore per  $5*0.36/2=0.9$  minuti.

Il ragionamento si ripete finché tutti e quattro i processi non hanno completato il loro compito. La traccia di esecuzione risulta essere la seguente:

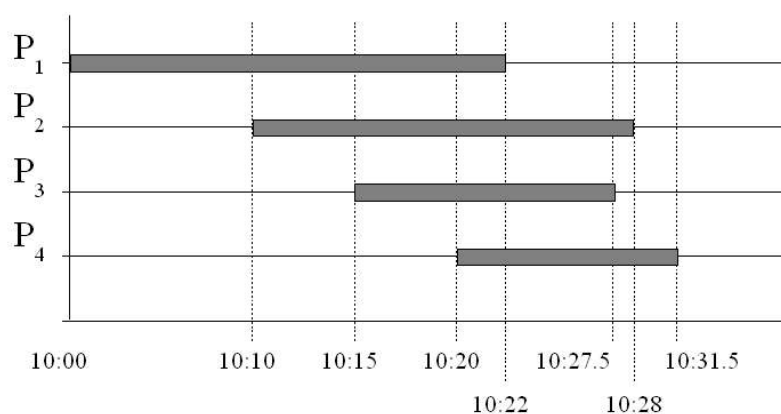


Figura 3: diagramma riportante le tracce di esecuzione dei singoli processi. Poiché la rappresentazione potrebbe ingannare, si ricorda che, disponendo di un solo processore, i processi procedono in modalità *interleaved*, e non *overlapped*.

Per completezza, nella tabella riportata di seguito sono anche indicati i tempi di CPU, e di completamento (*turn around time*) sia per il caso di monoprogrammazione (*virtual*; pari a  $T_{CPU}/(1-p)$ ), sia per il caso di multiprogrammazione visto nell'esempio (*actual*) :

	Tempo di CPU (minuti)	Virtual exec time	Actual exec time
P1	4	20	22
P2	3	15	18
P3	2	10	12,5
P4	2	10	11,5

Si osserva facilmente che il tempo di completamento per ogni singolo processo nell'esempio è superiore a quello che si avrebbe avuto in un contesto di monoprogrammazione.<sup>2</sup> Tuttavia il tempo complessivamente impiegato dai 4 processi nel caso di multiprogrammazione è inferiore alla somma dei tempi richiesti nel caso di monoprogrammazione.

---

<sup>2</sup> Questo è un effetto indesiderato nei sistemi hard real-time, in cui il tempo di completamento deve essere prevedibile. Qui, invece, dipende dal numero e dal tipo dei processi in esecuzione.