SISTEMI OPERATIVI (parte prima - gestione dei processi)

Tra i compiti di un sistema operativo sicuramente troviamo i seguenti:

- Gestione dei processi
- Gestione della memoria
- Gestione del file-system

Ci occuperemo adesso della gestione dei processi.

Definizione di processo.
Un processo è un programma (o una parte di una programma) in corso di esecuzione

E' doveroso fare una distinzione precisa fra *programma* e *processo*.

Mentre il *programma* è una entità statica composta da una serie di istruzioni memorizzata su memoria di massa, il *processo* è una entità dinamica, cioè è una parte di un programma che viene caricata in RAM ed eseguita dalla CPU e che durante la usa esecuzione può assumere vari stati di evoluzione (stati di un processo)
L'esecuzione di un processo è costituita da una successione di fasi di elaborazione sulla CPU e fasi di attesa per l'esecuzione di altre operazioni su altre risorse che di fatto lasciano inattiva la CPU

Es. l'istruzione cout del C++ genera un processo che richiede alla CPU di processare l'istruzione stessa (CPU attiva) ma anche la richiesta di utilizzo della periferica di output (CPU inattiva).

Tutti i moderni sistemi operativi cercano di sfruttare al massimo le potenzialità dell' hardware (soprattutto della CPU) presente in un computer allo scopo di:

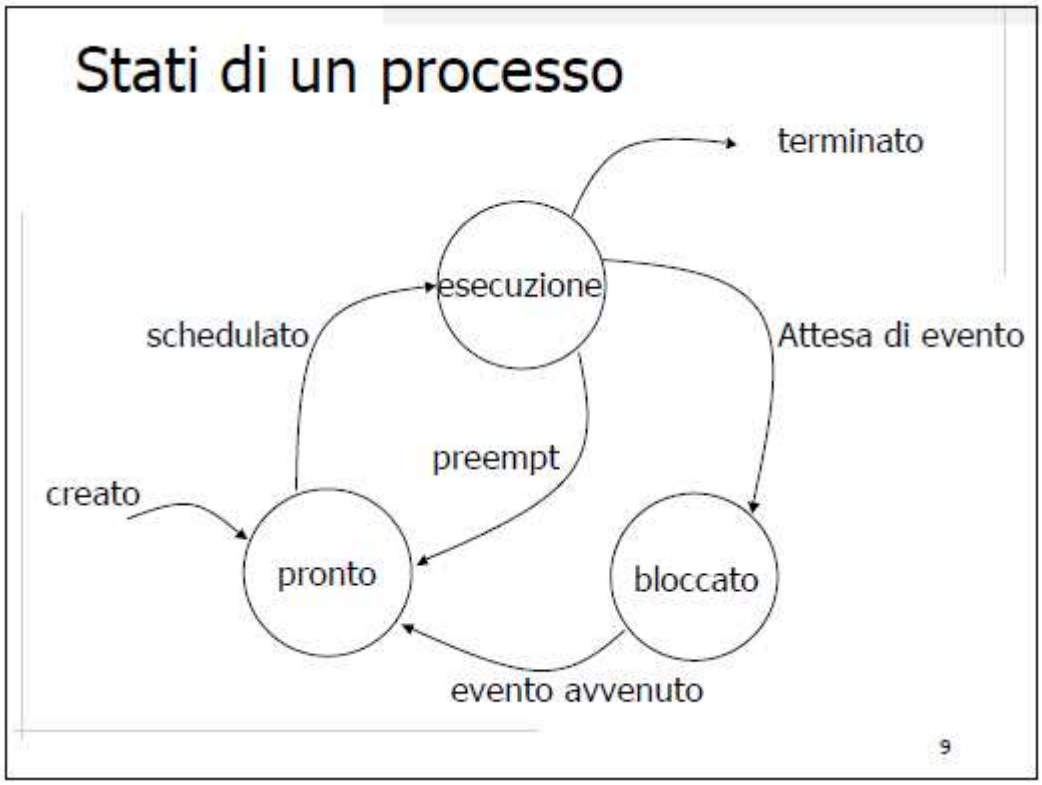
- massimizzare il numero dei processi (detti anche task, dall'inglese "compito") eseguiti nell'unità di tempo (throughtput);
- minimizzare i tempi di risposta di esecuzione dei vari processi

Sono stati quindi creati sistemi operativi *multitasking* che sono in grado di caricare in RAM e di "portare avanti in parallelo" più processi contemporaneamente.
In realtà se si dispone di una sola CPU, il parallelismo è solo virtuale (e non fisico, come nelle complesse architetture multiprocessore), in quanto un solo processo alla volta potrà trovarsi nello stato di esecuzione. Ma allora come avviene il multitasking?

Tecnica del time-sharing (condivisione di tempo)
Il principio di funzionamento di questa tecnica, introdotta al MIT nel 1963, è sostanzialmente il seguente:
Ad ogni processo (detto anche task o job) viene assegnato un intervallo di tempo uguale per tutti (**time-slice**), generalmente pochi decine di millisecondi,
Se un processo non si conclude entro il tempo a lui assegnato dalla CPU, viene sospeso (mediante interrupt)e messo in una lista di attesa, dopodichè si passa ad eseguire il processo successivo che necessita della CPU per un altro time-slice e cosi via per tutti gli altri.
Quindi, con il time sharing, un task può terminare la sua elaborazione per tre motivi:

- ha terminato la sua elaborazione entro il time-slice a lui assegnato,
- non terminato la sua elaborazione entro il time-slice a lui assegnato e viene messo in attesa
- ha bisogno di una periferica di I/O , viene messo in attesa e lì rimane finchè la risorsa non si rende disponibile.

Stato dei processi
Durante il ciclo di vita di un processo è possibile individuare un insieme di cinque situazioni in cui il processo si può trovare rispetto alla CPU e che definiremo come
Diagramma degli stati di un processo



- **creato** (new): è lo stato in cui si trova un processo appena creato dall'utente quando esegue un programma che risiede su disco fisso.
- **esecuzione** (running): la CPU sta eseguendo le sue istruzioni. Nei sistemi a monoprocessore soltanto un processo può essere in questo stato.
- **bloccato** (wait): un processo è in stato di attesa quando gli manca una risorsa per poter evolvere quindi sta aspettando che si verifichi un evento che gli liberi la risorsa
- **pronto** (ready): un processo è nello stato di pronto se ha tutte le risorse necessarie per evolvere tranne la CPU e sta solo aspettando che gli venga concesso il suo time-slice
- **terminato** (terminated): tutto il codice del processo è stato eseguito e quindi ha terminato l'esecuzione, la CPU provvederà a liberare le risorse che il processo utilizzava.

Osservazioni
Un processo può assumere una volta sola gli stati **creato** e **terminato** e molte volte gli altri stati.
Il sistema operativo gestisce due liste: una dei processi pronti ad essere eseguiti RL (ready list) ed una dei processi in attesa WL (waiting list)
Dallo stato di **bloccato** non si può tornare nello stato di **esecuzione** se non passando prima attraverso lo stato di **pronto**.

La modalità di esecuzione di processo può essere:
- USER-MODE quando il processo è creato dall'utente,
- KERNEL-MODE quando è il processo è creato dal sistema operativo

Un processo in esecuzione in modalità USER-MODE può essere interrotto prima che abbia terminato il suo time-slice da un processo supervisore di tipo KERNEL-MODE. In questo caso, il primo processo viene messo subito nello stato di pronto per essere immediatamente ripreso (operazione di preemptive , pre-rilascio)
Un processo di tipo KERNEL-MODE non può essere interrotto e deve portare a termine la sua esecuzione (non preemptive process).

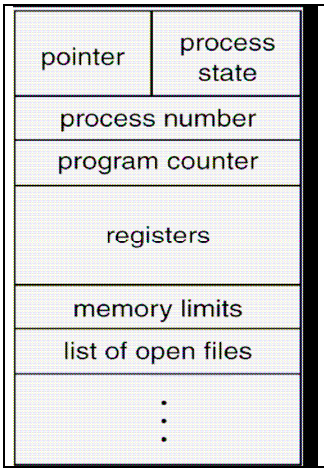
Descrittore di processo

Il sistema operativo deve mantenere per ogni processo alcune informazioni che costituiscono la cosiddetta fotografia "istante per istante di un processo", l'insieme di queste informazioni è detto descrittore di processo (PD) o Process Control Block (PCB).

Dati contenuti in un PCB

- identificatore unico del processo (process ID o PID)
- stato corrente (pronto, bloccato,...)
- program counter (PC = registro della CPU che contiene l'indirizzo di memoria RAM della successiva istruzione da eseguire)
- registri della CPU (IR istruction register, SP stack pointer, ecc..)
- priorità (legata alla politica della schedulazione dei processi)
- puntatori alla memoria per i dati e le variabili (stack, area globale)
- puntatori alle risorse allocate dinamicamente dal programma (heap)
- puntatori per l'accounting e per lo stato dell'I/O (lista dei file e delle periferiche associati al processo)

PCB



Process management	Memory management	File management
Registers	Pointer to text segment	Root directory
Program counter	Pointer to data segment	Working directory
Program status word	Pointer to stack segment	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

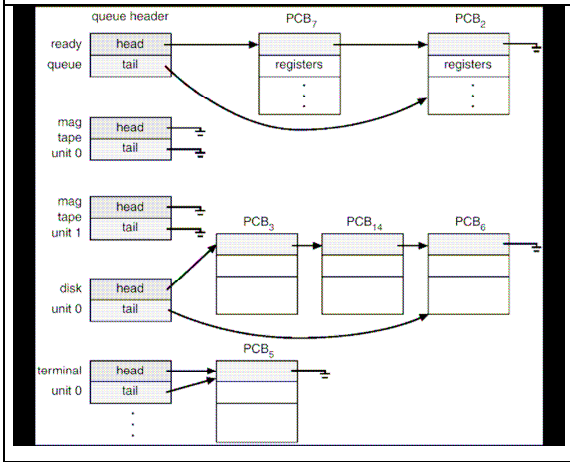
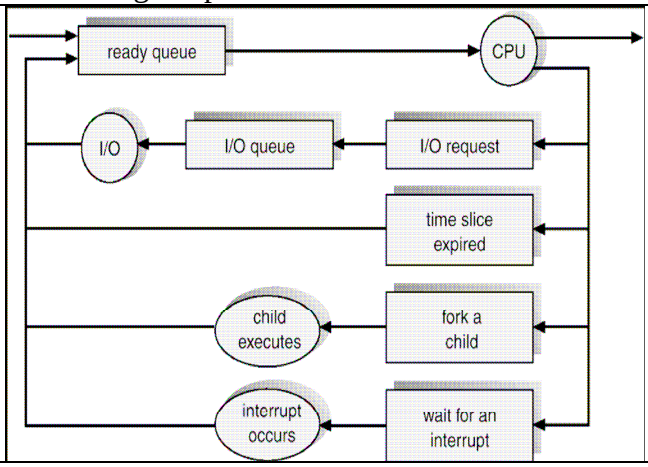
Scheduling e scheduler

I meccanismi con i quali i processi vengono scelti prendono il nome di politiche di gestione e di scheduzione (*Scheduling*)

Lo *scheduler* è un software che si occupa di scegliere quale processo mandare in esecuzione.

Tutti i processi del sistema sono posizionati in code (lista FIFO), delle quali ricordiamo almeno

- la **coda dei processi pronti** (ready queue list, RL), cioè dei processi in memoria pronti per essere eseguiti
- la **coda di attesa di un evento** (wait queue list, WL), dove vengono inseriti i processi in attesa di una risorsa

Code del sistema	Scheduling dei processi
	

Context-process e context-switching
Il c*ontext-process* **(***contesto di un processo)* è l'insieme dei valori contenuti nel suo PCB in un dato istante.
Quando l'esecuzione di un processo P1 viene sospesa per eseguire un altro processo P2, avviene una fase delicata, della durata di pochi millisecondi, che si chiama *context-switching (cambio di contesto)*, nella quale il sistema operativo "fa una fotografia" del processo P1, salvando i dati nel suo PCB e passa ad esaminare i dati del PCB del processo P2. La parte di S.O. che realizza il cambio di contesto si chiama *dispatcher*.

Algoritmi di scheduling
Tutti i criteri di scheduling si pongono gli stessi obiettivi:

- massimizzare la percentuale di utilizzo della CPU
- massimizzare il troughtput del sistema

Algoritmo FCFS (First-Come-First-Served)
I processi pronti vengono messi in coda secondo il loro ordine di arrivo (FIFO) e schedulati uno dopo l'altro indipendentemente dal tipo e dalla durata prevista per la loro esecuzione.
Vantaggi
L'algoritmo è di tipo non-preemptive, i processi non possono essere interrotti
Svantaggi
Basso uso della CPU (CPU burst), se viene eseguito per primo un processo lungo, gli altri, anche se più brevi, devono attendere

Es. Calcolo tempo medio di attesa tre processi P1=32, P2=4 , P3 = 7 unità/CPU

processo	p1	p2	p3	fine
tempo	0	32	36	43

ta = (0 + 32 + 36) / 3 = 22,7

Algoritmo SJF (Shortest Job First)
Per migliorare l'algoritmo FCFS basta scegliere tra i processi pronti quello che occuperà per meno tempo la CPU e che quindi verrà mandato per primo in esecuzione.

Potrebbe però verificarsi la situazione in cui mentre un processo è in esecuzione se ne aggiungo uno in coda con un tempo stimato minore, in questo caso possiamo avere due possibilità:

nel caso di situazione non-preemptive non si fa nulla,
nel caso di situazione preemptive 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 ed il cambio del contesto assegnando la CPU al nuovo processo. Questo algoritmo è detto **SRFT (Shortest Remaining Time First)**
Es.

Processo	Durata (tempi/CPU)	Arrivo (tempi/CPU)
P1	9	0
P2	4	1
P3	10	2
P4	5	3

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

P1	P2	P4	P3	fine
0	9	13	18	28

il tempo di attesa è così calcolato:
ta = (P1,P2,P4,P3) = ((0 + (9-1) + (13-3) + (18-2)) / 4 = 8,5

Con l'algoritmo SRTF, il processo P1 viene sospeso perchè arriva P2 con un tempo stimato più basso di 9-1 = 8 unità/CPU. Pertanto la sequenza è la seguente:

P1	P2	P4	P1	P3	fine
0	1	5	10	18	28

con

ta = (P1,P2,P4,P1,P3) = ((0 + (1-1) + (5-3) + 10 + (18-2)) / 4 = 7

Scheduling con priorità

Nella procedura di scheduling con priorità ad 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 con priorità più alta che avrà la precedenza di esecuzione su tutti. Nel caso di due processi con la stessa priorità si serve quello arrivato per primo come nella FCFS. Gli algoritmi con priorità possono essere sia non-preempite sia preemptive: in questo caso se si sta servendo un processo con priorità più bassa di uno nuovo appena entrato in cosa, si cede la CPU a quello con priorità maggiore, sospendendo il processo precedente.

In questi tipi di scheduling si possono verificare fenomeni di **starvation** (fame o morte per inedia), nei quali processi a bassa priorità restano in attesa per tempi indefiniti.

Es

Processo	Durata (tempi/CPU)	Priorità
P1	8	1
P2	1	3
P3	5	4
P4	4	5
P5	3	2

la sequenza di attivazione è:

P1	P5	P2	P3	P4	fine
0	8	11	12	17	21

ta = (P1, P5, P2, P3, P4) = (0+8+11+12+17) / 5 = 9,6

Algoritmo di scheduling Round Robin (RR)

Implementa la tecnica del time-sharing. Tutti i processi vengono inseriti in una coda circolare ed eseguiti per un quanto di tempo uguale per tutti (time-slice) indipendentemente dalla loro priorità.

RR = FCFS con preemptive periodica ad ogni scadenza del quanto di tempo.

Es. quanto di tempo = 4

Processo	Durata (tempi/CPU)
P1	20
P2	3
P3	7

P1	P2	P3	P1	P3	P1	P1	P1	fine
0	4	7	11	15	18	22	26	30

ta = (P1, P2, P3) = (0 + 4 + 7) / 3 = 3,6

Occorre però prestare attenzione al dimensionamento del time-slice perchè:

a) se troppo piccolo, ho tempi di risposta brevi ma un elevato numero di context-switching che crea uno spreco di tempo e risorse (overhead)

b) se troppo grande, algoritmo RR ha tempi di risposta elevati e degenera in FCFS

CONCLUSIONE

La soluzione ideale non esiste: i moderni SO combinano tra loro gli algoritmi qui presentati in modo da ottenere soluzioni mediamente buone per tutte le situazioni.

Scheduling Windows XP

- CPU Scheduling preemptive
- 32 livelli di priorità di Round-Robin
- Due classi di priorità: real-time (priorità 16-32), variable (priorità 1-15), la priorità si abbassa se si esaurisce il quanto di tempo, in seguito ad uno sblocco la priorità viene alzata di molto