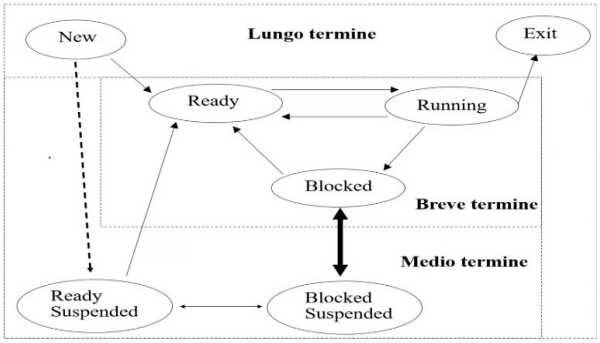
Scheduling 1 Algoritmi

martedì 21 ottobre 2025 12:10

Andiamo ora a concentrarci sullo **scheduling** (pianificazione delle risorse sulle cpu). Ce ne sono vari tipi :

A lungo termine	Decisioni sull'aggiunta di un nuovo processo all'insieme dei processi attivi
A medio termine	Decisioni sull'inserimento, totale o parziale di un processo attivo in memoria di lavoro
A breve termine (CPU-scheduling o dispatching)	Decisioni su quale processo debba impegnare la CPU
I/O scheduling	Decisioni sulla sequenzializzazione di richieste da servire sui dispositivi (logici e fisici)

Nota : la prima nei sistemi operativi attuali non viene più usata ; la seconda invece è attualmente utilizzata nei sistemi attuali, decidendo come usare la memoria (swap in /swap out) ; nel terzo tipo si utilizzano i thread ; l'ultimo invece si focalizza sull'ordine delle richieste da eseguire . quindi ora torniamo a vedere che relazione c'è tra lo scheduling e lo stato di un processo :



In dettaglio : quello a lungo termine si basano sull'attivazione di un processo : di solito con questo tipo di scheduling si arrivava alla multiprogrammazione, puntando ad una mistura di processi **CPU BOUND e I/O BOUND** dove il primo si basa su utilizzo cpu per fare il lavoro , mentre il secondo si basa su utilizzo di altre risorse. Quindi l'attivazione di questo scheduler veniva fatta o alla terminazione di un processo oppure su richiesta (invocazione system call), oppure quando l'utilizzo della cpu scende sotto certi valori (**mistura scorretta**) -> molti processi sono blocked. Come già detto questo scheduler non viene usato nei sistemi moderni in quanto non consente di controllare applicazioni interattive in quanto i processi venivano attivati basandosi sulle condizioni di carico del sistema ; questo tipo di scheduling quindi si aveva nei sistemi batch multi programmati. Quindi in generale il dispatching si basa su :

Orientamento all'utente
- decisioni di dispatching funzione di come gli utenti percepiscono il comportamento del sistema (es. tempo di risposta)

Orientamento al sistema
- decisioni di dispatching tese a ottimizzare il comportamento del sistema nella sua globalita' (es. utilizzazione di risorse)

Orientamento a metriche prestazionali
- approccio quantitativo
- parametri facilmente misurabili (monitorabili), analizzabili

Orientamento a metriche non prestazionali
- parametri tipicamente qualitativi o non facilmente misurabili

Andiamo ora a vederli in dettagli queste metriche :

Prestazionali

• **tempo di risposta**	ovvero il tempo necessario affinche' un processo inizi a produrre l'output
• **tempo di turnaround**	ovvero del tempo totale intercorrente tra l'istante di creazione e l'istante di completamento di un processo
• **scadenze**	ovvero una deadline di completamento

Altri

• **prevedibilita'**	possibilita' di supportare esecuzioni conformi a determinati parametri indipendentemente dal livello di carico del sistema

Per quanto riguarda invece quelli orientati al sistema:

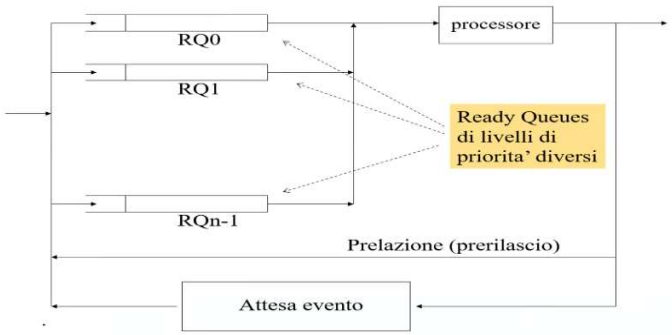
Prestazionali

• **throughput**	processi completati per unita' di tempo
• **utilizzazione del processore**	percentuale del tempo in cui la CPU risulta impegnata

Altri

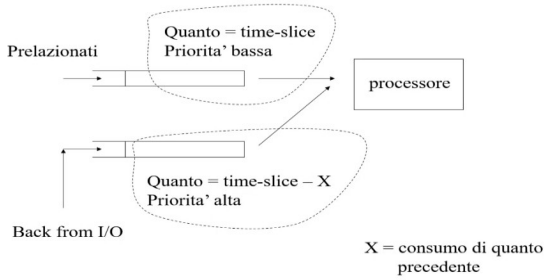
• **fairness**	capacita' di evitare **starvation** di processi attivi
• **priorita'**	capacita' di distinguere tra livelli di priorita' multipli per i processi attivi
• **bilanciameno delle risorse**	capacita' di equilibrare l'utilizzo delle risorse al fine di aumentarne lo sfruttamento

Dove per **starvation** si intende che il sistema non sta gestendo in modo giusto le applicazioni (faccio girare sempre le stesse) . Quindi riassumendo : per schedulare un processo bisogna tenere conto di **priorità** (livello con il quale il processo / thread viene schedulato) che parte dal livello 0 fino a n-1, ma attenzione alla starvation :



Quindi in base alla priorità si arriva ad un algoritmo molto importante (**ROUND ROBIN-> carosello**). Andiamo ora a vedere i vari tipi di scheduler (processi su unico processore) :

1. **First come first served** (FCFS)
    a. Il primo che arriva nella coda ready viene servito (unica ready queue).
    b. Quindi quel singolo processo impegnava la cpu (finchè non temrina) : non prelazione
    c. Questo scheduler non minimizza il tempo di attesa
    d. Inadeguato per gestione processi interattivi
    e. Causa sottoutilizzo dei dispositivi di i/o in quanto i processi interattivi non vengono favoriti
2. **Round robin** (time sliced): carosello
    a. Quindi la CPU viene usata a turno : quanto di tempo (al termine viene tornato il controllo al so)
    b. Vi è prelazione
    c. Anche questo algoritmo non è fair in quanto sfavorisce i processi i/o bound rispetto ai cpu bound
    d. Non gestisce in modo opportuno i processi interattivi
    e. Sottoutilizzo del processore
    f. Criticità : scelta del time slice
3. **Round robin virtuale**
    a.



    b. Entrambe le code sono di processi ready
    c. Nota : la priorità migliori ce l'hanno i processi nella coda back from i/o
    d. Quindi il quanto di tempo cambia , permettendo il recupero del residuo del quanto assegnato
    e. Anche questo algoritmo non basta : come faccio a capire la priorità tra i processi i/o bound? E come faccio a capire quale tipo di i/o ha provocato questo swap (quindi arriva interrupt)

f. Inoltre ci deve essere un meccanismo che permette di vare un timer programmabile

4. **Shortest process next (SPN)**
   a. Si basa su **CPU BURST** : quantità di tempo di cpu che un'applicazione in esecuzione utilizza fino a che non rilascia le risorse e/o arriva interrupt
   b. Quindi tutti i processi ready vengono ordinati secondo il CPU BURST.
   c. Vi è una variante che la prelazione : **STRN->shortest remaining time next**
   d. In caso di non preemption , il processo rimane attivo fino al suo completamento
   e. Quindi con questo algoritmo si minimizza il tempo di attesa , quindi adeguato per processi interattivi , quindi non si ha sottoutilizzo del processore in caso di prelazione dei processi
   f. Attenzione però : non si sa predizione del cpu burst , arrivando così alla starvation (occhio alla priorità)
   g. Vediamo ora come stimare la durata del cpu burst :

Media aritmetica $$S_{n+1} = \frac{1}{n}\sum_{i=1}^{n} S_i$$

---

i. **Media esponenziale** $$S_{n+1} = \alpha T_n + (1-\alpha) S_{n-1}$$

α vicino all'unita' determina maggior peso per osservazioni recenti

↳ Impatto sulla stabilita' in presenza di alta varianza

   ii. Ma attenzione alla varianza α

5. **Highest response ratio next (HRRN)**
   a. Questo algoritmo permette di bypassare la starvation

Processi selezionati in base al Rapporto di Risposta $$RR = \frac{w+s}{s}$$

Dove : w = tempo di attesa (permanenza nello stato ready)
s = tempo di servizio (di esecuzione)

   b. ↳
   - favorisce gli I/O bound (caratterizzati da piccoli valori di s)
   - affronta il problema della starvation dovuto alle priorita'

   c. Attenzione al tempo di attesa , cambiando così la priorità
   d. Dobbiamo "moderare" il tempo di ricalcolo di questo rapporto