02-05-2023

Obiettivi degli algoritmi di scheduling

Per progettare un algoritmo di schedulig è cruciale **cosa** deve fare il SO e **quali sono i suoi obiettivi** e questi ultimi dipendono dall'ambiente (*macchina applicativa*). Si distinguono vari ***ambienti di lavoro***:

1. **BATCH**: sono sistemi in cui la velocità della risposta è **meno importante**. Risulta fondamentale **l'esecuzione LUNGA del processo**. Di conseguenza sono **più adatti gli algoritmi non preemptive** e quindi vengono applicati maggiormente negli ambienti *INDUSTRIALI*. (*potrebbe esserci anche un algoritmo preemptive MA con una **LUNGA ESECUZIONE***)
   - In questo caso si **riducono** il numero di *cambi di contesto*
2. **INTERATTIVI**: sono macchine che interagiscono con l'utente. Di conseguenza **la velocità della risposta** alle richieste ricevute ha un'**ALTA PRIORITA'**. Classici esempi sono l'apertura di un sito web, di una cartella o simili. In questo caso risultano più adatti gli **algoritmi preemptive**.
3. **REAL-TIME**: E' determinante l'esecuzione di un processo in un **preciso istante di tempo** (classico esempio è la *catena di montaggio*) e **NON è sempre utile** usare metodi **preemptive**. In questo caso non si usa un algoritmo di scheduling come gli altri perchè ogni singolo processo sa di per sé che verrà eseguito per periodi non troppo lunghi

A prescindere dall'ambiente di lavoro usato, gli algoritmi di scheduling hanno degli obiettivi comuni:

1. **equità nell'assegnazione della CPU**: processi comparabili, cioè *appartenenti alla stessa categoria* e quindi che hanno una **stessa priorità**, devono avere un uguale trattamento. Di conseguenza, per sempio, in un *sistema interattivo*, tutti i processi che *interagiscono con l'utente*, devono essere comunque eseguiti **prima** di qualsiasi altro processo
2. **bilanciamento** nell'uso delle risorse hardware: **tutte le parti** del sistema devono essere **impegnate** e bisogna **evitare** che la CPU venga occupata da *processi inattivi*.

> Domanda ->*Come si fa a valutare se si è sviluppato un algoritmo di scheduling o uno scheduler buono o non buono?*

Metriche delle prestazioni

Se l'algoritmo è applicato in un sistema **BATCH**, si devono verificare tutte le seguenti metriche allo stesso tempo:

- **MASSIMIZZARE** il numero di processi che vengono **eseguiti e completati** in **un'unità di tempo** (***THROUGHPUT*** O *produttività*)
- **MINIMIZZARE il tempo di *turnaround***, cioè il **tempo medio** di completamento ed è il tempo fra **l'istante di arrivo della richiesta** del processo (e quindi il suo *accodamento*) e l'**istante del suo completamento**
- **MINIMIZZARE** il **tempo di attesa** ciopè il **tempo medio** che il processo mediamente trascorre fra i processi **READY**

> *Fra le 3 metriche si ha un **peso maggiore** sul **tempo di attesa** e su questa metrica vi è una **maggiore influenza** da parte dell'algoritmo di scheduling*

> Inoltre, *massimizzare il throughput* **non vuol dire** necessariamente *minimizzare il tempo di turnaround*

Se l'algoritmo è applicato in un sistema **INTERATTIVO**, si devono verificare tutte le seguenti metriche allo stesso tempo:

- E' fondamentale rispondere nel minor tempo possibile alle richieste dell'utente
- Bisogna quindi **MINIMIZZARE** il **tempo di risposta**, cioè il tempo che passa fra *richiesta ricevuta* e l'*esecuzione effettiva della richiesta*

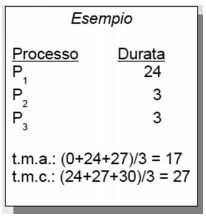
Se l'algoritmo è applicato in un sistema **REAL-TIME**, si devono verificare tutte le seguenti metriche allo stesso tempo:

- Si devono **RISPETTARE LE SCADENZE** dell'avvio o della terminazione di un processo in modo da permettere la **regolarità di esecuzione**
- **PREVEDIBILITA'** dei tempi di esecuzione dei processi

Algoritmi di Scheduling in sistemi BATCH

First-Come First-Served (*FCFS*)

- Si prende una coda (coda dei processi *READY*) e si mandano in esecuzione i processi **IN BASE ALL'ORDINE DI ARRIVO**. Questo sistema si chiama **First-Come First-Served** (*FCFS*). Questo tipo di algoritmo risulta penalizzante per i processi I/O bounded.
- E' un algoritmo **NON-PREEMPTIVE** e quindi i processi rilasciano autonomamente la CPU
- Se, durante l'esecuzione, arrivano altre richieste, queste ultime verranno aggiunte in coda e poi eseguite fino allo svuotamento della coda



`t.m.c.` = tempo di completamento medio = 27

- I processi che arrivano in coda si aspettano a vicenda -> Per completare P3 si deve aspettare P1 e P2.
  `t.m.a.` = tempo medio di attesa = 17
- Il primo processo attende 0 secondi per essere eseguito -> P2 aspetta il tempo di completamento di P1 (24).

Questo algoritmo è semplice da implementare ma **penalizza fortemente i processi I/O**. In questo caso il **disco risulta essere inattivo** perchè **non viene usato**

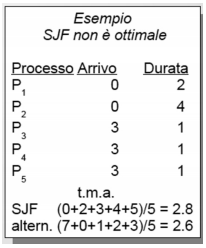
Shortest Job First (*SJF*)

Un altro algoritmo è **Shortest Job First** (*SJF*):

- Si eseguono prima i processi che richiedono **MENO DI ESECUZIONE** e in questo modo si hanno **tma più brevi**
- Bisogna conoscere, all'inizio, i tempi di esecuzione dei processi

- E' un algoritmo **NON-PREEMPTIVE** ed è ottimale **ESCLUSIVAMENTE** se i **lavori** sono tutti **subito disponibili** ma comunque resta facile da implementare

> Se si riprende il primo esempio si ha un $tmc = 13$ e un $tma = 3$



Esempio
SJF non è ottimale

Processo	Arrivo	Durata
P_1	0	2
P_2	0	4
P_3	3	1
P_4	3	1
P_5	3	1

	t.m.a.
SJF	(0+2+3+4+5)/5 = 2.8
altern.	(7+0+1+2+3)/5 = 2.6

In questo caso:

- tempo di completamento t.c = :
  - p1 = 2
  - p2 = 6
  - p3 = 6+1 **-3** = 4 (**perchè p3 non è arrivato al tempo 0**)
  - p4 = 7+1 **-3** = 5
  - p5 = 8+1 **-3** = 6
- tempo di attesa t.a = 2.8:
  - p1 = 0
  - p2 = 2
  - p3 = 2+4 **-3** = 3
  - p4 = 6+1 **-3** = 4
  - p5 = 7+1 **-3** = 5

Dalla versione alternata, cioè partendo da p2 e poi p1,p3,p4,p5 allora si ha:

- il tempo di attesa $t.a$ = **2.6**:
  - p2 = 0 (*dura 4 e nel frattempo arrivano p3,p4,p5*)
  - p3= 1
  - p4=2
  - p5=3
  - p1=7

Quindi, quando *i processi arrivano **durante l'esecuzione***, allora questo algoritmo **NON** è più ottimale perchè, invertendo il due processi, allora si è ottenuto un risultato migliore

Esempio: Dati 4 processi dove ognuno impiega tempo a, b, c, d, il tmc di $P1 = a$, $P2 = a + b$, $P3 = a + b + c$, $P4 = a + b + c + d$. In generale, con ordine `P1 P2 P3 P4` viene $t.m.c = \frac{4a+3b+2c+d}{4}$

Shortest Remaining Time Next (*SRTN*)

Un altro algoritmo è **Shortest Remaining Time Next** (*SRTN*)

- si basa sulla politica SJF (dare priorità ai processi che durano di meno)
- è un algoritmo **PREEMPTIVE**
- Inizialmente, quando si hanno tutti i processi a tempo 0, si **applica la politica SJF**
- Se nel frattempo arrivano altri processi, il SO guarda il tempo dei nuovi processi e, se il tempo è più basso, allora si esegue quello che ha tempo più corto. Il tempo tolto va riconfrontato in futuro

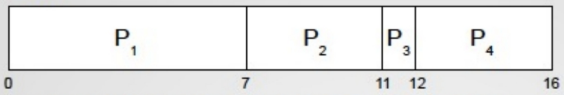
> *Nell'esempio di prima si esegue P1 e poi P2. Quando parte P2, dipo 1 ms arrivano P3 P4 e P5. P2 ha ancora 3 ms di lavoro ma i processi nuovi ne hanno 1 ciascuno. P2 viene stoppato e vengono eseguiti i nuovi processi. P2 verrà poi confrontato con il tempo rimanente (3 ms)*

Confronto fra i 3 algoritmi con l'uso del Diagramma Gantt (*ESAME*)

Si considerano 4 processi che arrivano in tempi diversi:

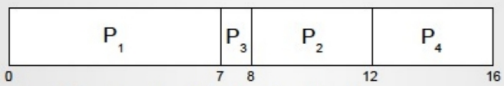
Processo	Arrivo	Durata
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

FCFS:



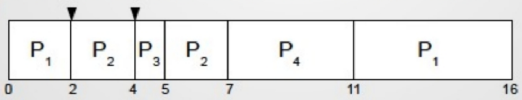
- tempi di attesa: P_1=0; P_2=5; P_3=7; P_4=7 (media 4.75);
- tempi di completamento: P_1=7; P_2=9; P_3=8; P_4=11 (media 8.75);

SJF:



- tempi di attesa: P_1=0; P_2=6; P_3=3; P_4=7 (media 4);
- tempi di completamento: P_1=7; P_2=10; P_3=4; P_4=11 (media 8);

SRTN:



- tempi di attesa: P_1=9; P_2=1; P_3=0; P_4=2 (media 3);
- tempi di completamento: P_1=16; P_2=5; P_3=1; P_4=6 (media 7).

In particolare, in *SRTN*, quando si calcolano i **tempi di attesa**, si devono sottrarre anche i tempi che i processi hanno **consumato**. Invece, nel calcolo del tempo di completamento, si devono **sottrarre i millisecondi in cui il processo in questione non era presente.**

L'algoritmo **SRTN risulta essere il più efficiente** rispetto agli altri 2

ESERCIZIO: