04-05-2023

Scheduling nei sistemi interattivi

Round-Robin (*RR*)

- E' un algoritmo di scheduling **PREEMPTIVE** *FCFS* (*First Come First Served*)
- Assegna un quanto di tempo a un processo.
  - Appena scade il tempo assegnato, il processo **preleva** tale processo e lo rimuove.
- Se **il processo si blocca** prima (*I/O*) verrà anche rimosso
- Si basa su una **coda di processi** e il primo che si trova in testa è il successivo da eseguire (*concetto di* `Queue`)
  - Se il processo **B termina il suo quanto di tempo**, viene prelevato e rimesso in fondo alla coda

> *Quanto tempo si deve assegnare (**timeslice**) a ogni processo?*
> *Se il timeslice è troppo breve, allora, allora aumento il numero di cambi di processi e di contesto (switch) e si spreca tempo in cambi*
> *Se il timeslice è troppo lungo, allora riduco il numero di cambi di processi (switch) e di contesto. In questo caso incremento il tempo di attesa fra processi.*

In media uno switch dura 1ms mentre il timeslice è 20-50ms

Bisogna valutare 2 concetti che influiranno:

- `context switch` = scambio di modalità fra utente e kernel
- `process switch` = scambio fra processi

> *Se i switch sono molti, si ha un grande spreco di tempo perchè, chiaramente, ogni switch richiede del tempo.*

- Più si **aumenta il timeslice** e più **riduco gli switch fra processi** e quindi un processo resta **in attesa per molto più tempo**.
- Quindi risulta deterministico **ricavare il giusto timeslice** per aumentare le prestazioni dell'algoritmo

> *Quando il timeslice finisce, allora interviene un **INTERRUPT DI CLOCK*** sul processo.

Se il timeslice è "corretto" allora ci sono dei vantaggi e svantaggi:

- +E' **semplice da implementare** (*basta usare una **coda di processi***) e ad ogni processo si assegna il *timeslice*
- -**Tutti i processi sono uguali** fra loro a **livello di importanza** e questo non è possibile in caso di sistema interattivo
  - *Un processo per email non è uguale a un processo che richiede la digitazione di testo su schermo*

Algoritmo di scheduling con priorità

Bisogna eseguire uno **scheduling a priorità** (i processi demoni o simili (*per esempio*) devono avere priorità minore rispetto a una richiesta di I/O). Quindi serve **differenziare i processi**:

- I processi con **alta priorità** devono avere una **maggior possibilità di essere scelti** se si trovano nella coda dei processi `READY`
    - Con questo sistema, però, si ha una **difficoltà** nella gestione dei processi con **priorità bassa**. Se ci sono sempre processi con priorità alta, **quelli con priorità bassa** rimangono **in attesa all'infinito**

L'idea è: quando scade il timeslice, si va a scegliere il prossimo processo con priorità alta. **Ad ogni clock** si deve **RIDURRE LA PRIORITA'** del processo in esecuzione e in questo modo si garantirà che anche i processi di bassa priorità verranno eseguiti.
![[]]

Priorità statiche e dinamiche

Le priorità possono essere **STATICHE** (`nice`) o **DINAMICHE** ($\frac{1}{f}$) e per assegnarle/modificarle si usa `nice` (Unix) su un processo e si usa solo da `root` perchè è un comando delicato.

> *Per vedere i dettagli del comando si usa* `man nice`

Con l'uso della *priorità dinamica* si usa dare **priorità maggiore ai processi I/O bounded**.

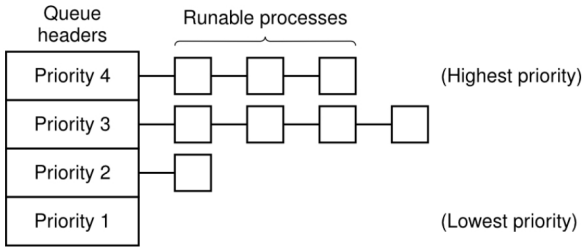
Si usa anche assegnare una priorità in base al timeslice consumato f in particolare $\frac{1}{f}$. Si può fare anche $\frac{timeslice\_complessivo}{timeslice\_consumato}$.

> *Generalmente meno timeslice se ne consuma, più alta e la priorità e viceversa. **In questo modo** si assegnano priorità **DINAMICHE***

SJF si può rivedere con un algoritmo di priorità dove **più lunga è l'esecuzione, più bassa è la sua priorità** dove, in generale, si considera **l'inverso della durata del processo**. In questo senso SJF può essere visto come un **ALGORITMO DI PRIORITA'.**

Classi di priorità

Se i processi hanno la **stessa priorità**, allora c'è un sistema che suddivide i processi in **CLASSI DI PRIORITA'**: in una singola classe ci sono i processi in una coda (*FIFO*) che hanno la stessa priorità.



Se si esegue un processo di priorità 3 e ne arriva uno di priorità 4 allora si deve prelevare (*PREEMPTIVE*) ed eseguire il nuovo processo

Problema: i processi di classi inferiori rischiano di non essere mai eseguiti (***STARVATION***) e una possibile soluzione è detta ***AGING*** che consiste nell'***aumentare gradualmente la priorità dei processi che attendono*** a lungo in coda nella stessa classe.

I processi possono avere **task differenti** (*processo interattivo vs processo background*) e quindi *si distingue* ulteriormente *per gruppi* dove vengono individuate le categorie di ogni processo in questo modo **si distingue l'attività specifica del processo**

All'interno delle singole classi (quindi ***a partità di priorità***), si ha una selezione dei processi che devono essere eseguiti utilizzando lo scheduling **ROUND-ROBIN** (*eseguo per prima i processi che si trovano in testa alla coda*).

> *Si può usare un sistema di assegnazione del timeslice differente per ogni categoria di processo*