SERT - 08/10/2020 - Algoritmi priority-driven - R04

4.1

Benvenuti, siamo alla quarta lezione dedicata ai sistemi RT. Oggi parliamo di algoritmi priority driven.

4.2

In realtà ci porteremo appresso questi algoritmi per tutto il resto del corso. Questi sono gli algoritmi più diffusi quando si tratta di schedulare i task nei sistemi RT. Inizieremo a parlare di algoritmi round-robin giusto per dare un contesto al nostro discorso. Dopo di che affronteremo un discorso più proprio degli algoritmi priority driven utilizzati nei sistemi RT.

4.3

Che cosa è un algoritmo **round-robin**? E' gestito tramite una coda di job, di processi, *FIFO*(First in first out). Il primo job che arriva in coda è anche il primo ad essere servito. In realtà, quindi, abbiamo che quando un job arriva è messo in coda di esecuzione e, se devo scegliere fra tanti job, seleziono quello che è in cima alla coda, cioè quello inserito da più tempo. Lo rimuovo dalla coda e lo eseguo. Tipicamente non lo eseguo fino a che termina, ma lo eseguo per un certo tempo predefinito, chiamato *time slice* o *quanto di tempo*. Nel momento in cui il quanto di tempo scade ed il job non ha finito, viene interrotto ed inserito nuovamente in coda. In realtà passa indietro a tutti gli altri job nel sistema.

4.4

Quindi se nella coda ci sono n job attivi nel sistema, un gruppo di questi n time slice è chiamato un **round**, intervallo di tempo pari ad n volte il quanto di tempo dedicato ad ogni processo attivo. Ciascun job ottiene un time slice determinato. Questo è lo schema di base. In realtà in generale si adotta uno schema leggermente più complicato, che è il **round-robin pesato**. E' possibile fare in modo che i job differenti abbiano in qualche modo una percentuale di allocazione del processore diversa. Quindi posso assegnare un peso. Un job più pesante ottiene più processore di un job meno pesante. Come possiamo definire formalmente questo tipo di algoritmo? Un **round** lo definisco come un numero di time slice pari alla somma di tutti i pesi dei job nella coda. Un job con peso w, in ogni round ottiene w time slices. Quindi è chiaro che un job più pesante, in ogni round ottiene più processore di un job più leggero.

Quali sono i vantaggi degli scheduler round-robin? Innanzitutto sono abbastanza equi. Cioè fondamentalmente job dello stesso peso ottengono la stessa quantità di risorse, di processore. Questa è una proprietà, il **time sharing** cioè la suddivisione del tempo del processore in modo equo tra i processi del sistema, una caratteristica importante dei sistemi operativi general purpose. Un'altra caratteristica importante di questi algoritmi è la loro semplicità di implementazione,

in quanto si usano delle code FIFO molto semplici. Tipicamente se mi limito a questo algoritmo di base, è estremamente efficiente e veloce da implementare. Gli algoritmi round-robin sono algoritmi utilizzati sopratutto nei sistemi operativi general purpose, già visti in una forma o nell'altra nei corsi di sistemi operativi di base ed avanzati. In realtà, se ne usano delle versioni molto più sofisticate perché in qualche modo voglio cercare di discriminare job di tipologie differenti, o posso cercare di fare assegnare all'utente stesso delle priorità, dei pesi, su questi job. Oppure posso cercare di contemplare il caso in cui i job effettuano operazioni che sono particolarmente onerose rispetto ad altri che non lo sono. Gli algoritmi che usano nei sistemi operativi reali sono molto sofisticati, ma alla base di tutto c'è questo schema round-robin in cui i processi si alternano sul processore.

4.5

A questo punto quindi possiamo domandarci: ma nell'ambito dei sistemi RT, cioè dei sistemi in cui la schedulazione deve occuparsi del fatto che tutti i job devono rispettare le scadenze, quali sono gli svantaggi di questi scheduler round-robin? Ovviamente lo svantaggio principale è che questi algoritmi non consentono di valutare le scadenze del job. Se io inserisco un job in coda, quando arriva il suo turno esegue, in realtà non sto considerando quanto è vicina o lontana la scadenza di quel job. E quindi poiché non considera la scadenza dei job, fa un pessimo lavoro nel cercare di far rispettare le scadenze dei job. C'è un altro problema. Un problema abbastanza serio. Questi algoritmi non gestiscono bene i job che hanno dei vincoli di precedenza. I vincoli di precedenza è una situazione per la quale un certo job non può cominciare se un job precedente non ha terminato. Sappiamo che chiamo questa relazione d'ordine *precedenza immediata* con il simbolo sulle slide. Supponiamo di avere $J_{1,1}$ e $J_{1,2}$ con vincolo di precedenza, dunque il secondo non può iniziare se il primo non è finito. Poi abbiamo anche $J_{2,1}$ e $J_{2,2}$. Supponiamo di avere due processori. Questa semplice situazione, quando questi algoritmi sono schedulati sul processore con round robin, fondamentalmente le cose vanno come in slide. Siccome tutti i job che devono eseguire sul processore due non possono partire fin quando i job del processore 1 non sono terminati, nella prima parte per due unità di tempo si alternano sul processore 1 i job $J_{1,1}$ e $J_{2,1}$. Fondalmente possiamo assumere che il time slice sia così piccolo che questi fanno un'alternanza molto rapida sul processore. E' come "se andassero avanti in parallelo". Dopo di che però il loro tempo di esecuzione totale è più o meno la somma dei singoli job. Quindi all'istante 2, $J_{1,1}$ e $J_{2,1}$ terminano più o meno. Quindi all'istante 2 possono iniziare gli altri job che finiscono all'istante 4. Con il round robin, l'ultimo job completa all'istante 4. Il tempo di completamento dell'ultimo job è quattro.

Ma questo algoritmo è particolarmente cattivo se lo confrontiamo con un algoritmo che non usa il round-robin. Se io non lo uso, io posso dire che sul processore 1 eseguo tutto prima tutto il job $J_{1,1}$, ed a questo punto all'istante 1 posso eseguire il job $J_{2,1}$ e $J_{1,2}$. All'istante 2 terminate $J_{2,1}$ e può essere eseguito $J_{2,2}$. L'istante di terminazione dell'ultimo job è l'istante 3. Dove è il guadagno del tempo? Sta nel fatto che tra 1 e 2, ci sono stati due job in esecuzione

contemporaneamente. Cioè ho sfruttato veramente il sistema multiprocessore. Mentre con l'algoritmo round-robin, questo sistema multiprocessore non l'ho potuto utilizzare a causa dei vincoli di precedenza.

10.16