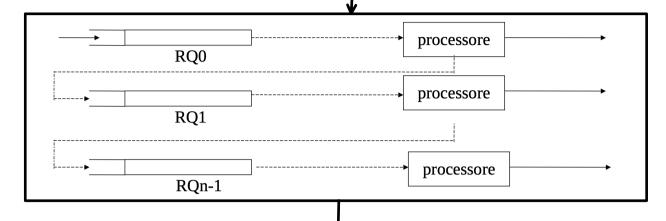
martedì 11 aprile 2023 17:41

## Multi-level feedback-queue scheduling

differentemente da SPN, SRTN e HRRN non necessita di informazioni (predette o monitorate)

## Abbiamo N code di priorità diverse.



Quando un processo nasce e diventa "Ready" a lui viene assegnata una priorità zero e viene inserito all'interno della prima coda.

Quando si libera la CPU io vado a pescare il primo elemento da quella coda e lo assegno al processore.

Che fine fa questo processo quando poi viene assegnato al processore?

In realtà gli viene dato un quanto di tempo. Supponiamo che questo quanto di tempo lo usiamo TUTTO, quindi arriva un <u>interrupt</u> da timer che indica al sistema che non c'è più tempo per lui e che bisogna riassegnare la CPU.

Che fine fa questo processo che avevamo precedentemente selezionato dal livello 0 e poi passato in CPU? NON LO METTIAMO al livello 0, perché lui ha utilizzato tutto il quanto e ipotizziamo che questo sia un processo CPU BOUND ossia voglia utilizzare realmente la CPU in maniera massiva, quindi gli do una priorità più bassa, e lo inserisco all'interno di una coda di livello inferiore che è quella associata all'indice 1.

Poi nell'istante dopo il software del sistema va sempre nella coda di livello zero, prende il processo successivo e lo assegna alla CPU.

Nella coda di livello 0 che è la coda più importante noi manteniamo cose che primariamente vengono mandate in CPU rispetto a cose che manteniamo nella coda di livello 1.

Soltanto quando la coda di livello zero sarà vuota prenderemo un elemento dalla coda di livello uno e lo manderemo in CPU.

Supponiamo di pescare un oggetto dalla coda con indice 1, e lo mandiamo in CPU, questo oggetto può continuare ad utilizzare tutto il quanto di tempo che gli assegno? Chiaramente SI, e così posso decifrare che questo processo sta impegnando in maniera importante la CPU, ancora una volta. E quindi quando gli tolgo la CPU perché il quanto finisce e dovrò eventualmente riassegnare la CPU lo inserisco ancora più giù nei livelli di priorità, sino ad arrivare al livello N-1.

Ma se un processo va in esercizio sulla CPU e non ha usato tutto il quanto di tempo che gli era stato <u>assegnato, perchè</u> magari ha chiamato il kernel del S.O per andare in stato di attesa, cosa succede?

Quando ritorno <u>ready</u>, vengo riportato esattamente nell'ultima coda in cui ero stato. Quindi se io rilascio la CPU prima dello scadere del quanto, quando poi verrò riportato "<u>ready</u>" verrò riportato allo stesso livello, ma MESSO IN CODA.

Se io invece avevo preso originariamente la CPU e usato tutto quanto il "quanto" è arrivato <u>l'interrupt</u> da timer, il kernel ha preso il controllo ha assegnato eventualmente la CPU a qualcun altro, in questo caso ero stato portato dalla coda di alta priorità 0 alla coda 1.

Una volta declassato alo livello 1, quando rientro ready, rientrerò al livello 1.

Così via.......

Più volte mostro di utilizzare tutto il quanto e più volte perdo nei livelli di priorità.

Più volte mostro di utilizzare tutto il quanto e più volte perdo nei livelli di priorità.

Ci troveremo nella coda di livello 0, <u>sperabilmente</u>, tutti i processi che sono I/O BOUND. Per cui quando vanno in CPU la usano pochissimo, la rilasciano e vanno in blocco, e quando vengono sbloccati li rimetto nella stessa coda, in coda.

Verranno declassati i processi che hanno manifestato di usare la CPU in maniera più intensa.

PER STABILIRE LA PRIORITÀ DI UN THREAD IN QUESTO ALGORITMO, IO DEVO VEDERE SOLO SE IL PROCESSORE HA UTILIZZATO TUTTO IL QUANTO CHE GLI HO DATO, OPPURE NO.

Se ha utilizzato tutto il quanto la sua priorità viene declassata, e lo segno ad un livello di priorità più bassa.

Quando un'applicazione scende nei livelli di priorità in realtà questa discesa può essere dovuta al fatto che questa applicazione potrebbe avere un comportamento di questo tipo: inizialmente vorrebbe usare la CPU per tanto tempo, e lo fa, ma ad un certo punto magari dopo tanti intervalli d'esecuzione potrebbe chiamare il kernel per eseguire dei servizi, quindi abbiamo una fase iniziale che è CPU BOUND e una fase da un punto in poi che è I/O BOUND. Nel CPU BOUND, LA CAPIAMO, E DECLASSIAMO L'APPLICAZIONE. Ma l'altra fase non la capiamo e quindi non riportiamo l'applicazione su nei livelli di priorità.

Non abbiamo un feedback migliorativo delle priorità quando un'applicazione cambia comportamento. Perché i processi possono cambiare il loro comportamento.

Quanto di tempo <u>fisso per tutte le priorità:</u> starvation sui processi molto lunghi soluzione parziale: quanto di tempo pari a  $2^i$  dove i è la priorità

Poi magari sono ad un livello di priorità scarsa e mi viene dato lo stesso quanto di tempo di un'applicazione con priorità alta, chiaramente vengo sparito ulteriormente.

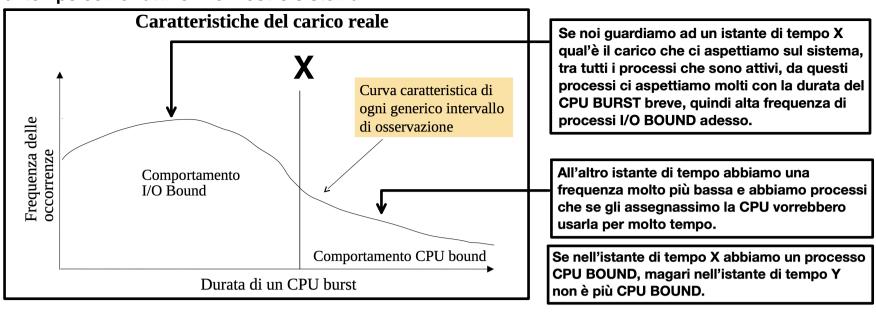
Magari i quanti di tempo da assegnare ai processi che vengono pescati in code diverse, sono diversi. Più scendiamo a pescare un processo all'interno di questo schema di priorità, assegnamo un quanto di tempo sempre un po' più lungo.

soluzione parziale: quanto di tempo pari a  $2^i$  dove i è la priorità

Il problema di questo scheduler è che noi possiamo essere DECLASSATI ma non c'è un modo per risalire all'interno di questo schema di priorità. Perché in fin dei conti un processo attivo P in una fase può essere CPU BOUND e in un'altra fase I/O Bound. Quando P comincia a comportarsi come I/O BOUND non abbiamo la possibilità di tirarlo su.

Per far fronte a questo problema, nel mondo UNIX è stata proposta una variazione di questo algoritmo basata sulle caratteristiche di carico reale che noi possiamo osservare ad un certo istante

di tempo come "attivo" nel nostro sistema.



Per risolvere questo problema abbiamo lo scheduling UNIX tradizionale.