venerdì 19 maggio 2023 12:21

Algoritmo Least-Recently-Used (LRU)

Consideriamo un'altra variante di algoritmo di sostituzione della pagine all'interno della RAM, che è l'algoritmo LRU.

In questo algoritmo la vittima è la pagina alla quale NON ci si riferisce da più tempo.

Attenzione: stiamo parlando di situazioni che sono accadute nel passato, non di situazioni che accadranno nel futuro. L'idea di sostituire come vittima una pagina che non utilizziamo da più lungo tempo significa dire che, le informazioni associate a quella pagina logica oramai non fanno più parte (o almeno in questo momento non fanno parte) della località dell'esecuzione delle mie applicazioni.

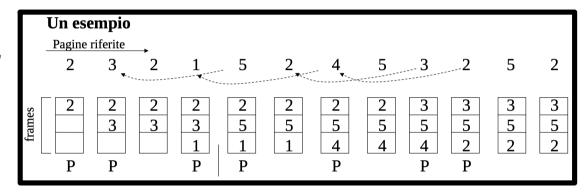
Se qualcosa non è utilizzato da più lungo tempo, probabilmente non è più nella località delle informazioni di interesse per l'applicazione.

- seleziona per la sostituzione la pagina alla quale <u>non ci si rifesce da più tempo</u> (predizione della località in base al comportamento passato)
- difficile da implementare (necessità di marcare le pagine col tempo di riferimento o di mantenere uno stack di riferimenti alle pagine)

ESEMPIO CON LA STESSA SEQUENZA DI ACCESSI DELL'ALGORITMO OTTIMO

L'insieme delle pagine in RAM per 3 frame è sempre un sottoinsieme dell'insieme di pagine in memoria per 4 frame. Per ogni r sequenza di accessi alle pagine logiche.

Algoritmo a stack.



Abbiamo alcuni page fault sistemici iniziali per portare ad avere la RAM i suoi frame occupati dalle pagine 2,3 ed 1, poi supponiamo che ad un certo punto dobbiamo accedere alla pagina 5 e quindi abbiamo necessità di portarla in RAM, in questo caso dobbiamo andare a verificare tra la 1, la 2 o la 3, quale pagina è stata utilizzata più tempo fa (NEL PASSATO), quindi la pagina a cui non ci si riferisce da più tempo: questa è la pagina 3 e per tale motivo verrà selezionata come vittima per far spazio alla pagina entrante 5.

Poi quando ci serve di portare in memoria la 4 e abbiamo un altro page fault, dobbiamo andare a vedere usando lo stesso criterio se nel passato c'era una pagina a cui non ci siamo più riferiti da più tempo, e la scelta ricade tra la 2, la 5 e la 1: bene la 1 è una vittima e fa spazio alla pagina entrante 4. E così via.

Se noi consideriamo questa stessa sequenza di accessi che avevamo nello schema precedente dell'algoritmo ottimo, nell'LRU abbiamo un page Fault in più, ma LRU è un algoritmo utilizzabile in un contesto reale, non è ideale come l'ottimo. Ma è comunque difficile da implementare perché se noi dobbiamo essere in grado di riconoscere tutti gli accessi, quindi magari riconoscere l'accesso alla pagina 2 quando abbiamo avuto in precedenza già accesso alla pagina 2 e avendo che essa era stata portata in RAM e continuiamo ad accedervi in RAM richiamandola sempre, l'identificazione di questo accesso ulteriore dovremmo utilizzarlo portando delle informazioni ad essere RESETTATE e poi ad essere SETTATE nuovamente ad 1 per indicare che questa pagina è stata acceduta.

Queste sono informazioni di controllo che devo gestire con una frequenza molto molto elevata.

Supponiamo che io accedo alla pagina 2->3->2, per riuscire a capire che il secondo accesso alla pagina 2 è avvenuto dovrei resettare delle informazioni di controllo che vengono ad essere settate nuovamente ad 1 dopo due istruzioni macchina. All'interno di una page table abbiamo dei bit che ci dicono se quella specifica entry è stata utilizzata per scoprire la posizione fisica di una pagina logica, potrei utilizzare quelle informazioni per determinare gli accessi, ma sarebbe comunque difficile da implementare perché dovrei avere un reset di quelle informazioni frequentissimo. Quindi difficilmente gestibile perché il reset viene eseguito dal software del S.O.

Dovremmo mantenere uno stack di riferimenti alle pagine difficilmente implementabile, ma ciò non toglie che questo è un algoritmo che si può comunque implementare nella pratica per creare questo stack di riferimenti con lo scopo di sapere quel è la pagina toccata da più tempo rispetto ad altre.