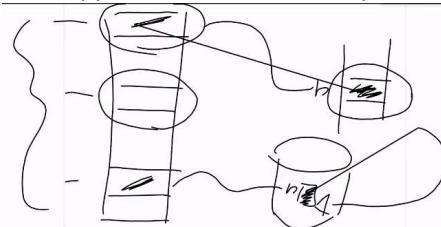


Memoria 4 Memoria virtuale + Algoritmi

mercoledì 19 novembre 2025 16:58

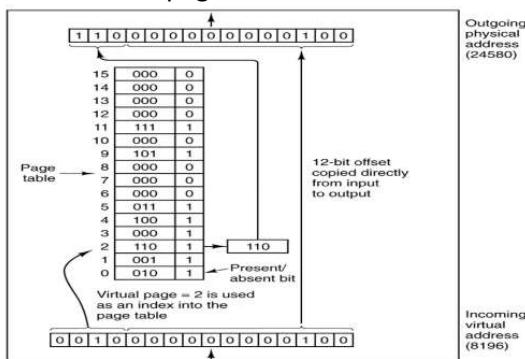
Facciamo un recap : la memoria virtuale è schema di gestione della memoria (address space) , la quale permette di avere in sottoinsieme delle pagine mappate presente in RAM . Il sottoinsieme è dovuto alla non materializzazione (anonime e non accedute e non caricate in RAM) oppure per swap-out (pagine portare fuori RAM) . Quindi con la memoria virtuale si ha che i processi possono essere parzialmente swappati fuori dalla RAM e parzialmente dentro la RAM.



Dove la prima pagina sia in address space che in RAM, la seconda solo presente in address space (non materializzata) , mentre la terza swappata fuori su hard disk. Con questa tecnica si ha che :

1. Si possono eseguire processi il cui spazio di indirizzamento eccede la dimensione della memoria di lavoro
2. Aumenta il numero di processi che si mantengono contemporaneamente in memoria
3. Riduzione del tempo per le operazioni di swapping

Quindi questa tecnica è efficace in quanto la locazione spazio-temporale aumenta le prestazioni ; inoltre all'interno dell'address space vi sono blocchi di codici usate raramente : gestione errori (per esempio zona testo ci sono delle istruzioni per gestire eventi particolari) ; inoltre si ha anche un sovradimensionamento delle strutture dati (array all'interno della sezione dati molto grande e se ne usa una porzione). Andiamo ora in dettaglio a vedere come e dove sono collocate le informazioni riguardanti il sottoinsieme delle pagine materializzate :



Dove l'indirizzo logico è composto da indice + offset , il quale viene usato per calcolare l'indirizzo in RAM (indirizzo fisico). Inoltre vi è presente un altro bit (0/1) che rappresenta se la pagina è presente o assente (non riesco ad effettuare traduzione logico-fisico), portando così al **page fault**. Ce se sono di due tipi : **major se si ha swapped out della pagina e minor fault pagina mai stata materializzata** .

Vediamo in dettaglio le prestazioni :

ma = tempo di accesso alla memoria

pft = tempo medio di caricamento della pagina da hard-drive
(tempo di page fault)

f = frequenza di page fault

Tempo di accesso effettivo: $ma + pft \times f$

Supponendo:

$ma = 5 \mu\text{sec}$ \Rightarrow Tempo di accesso effettivo: $5 + f \times 10000 \mu\text{sec}$

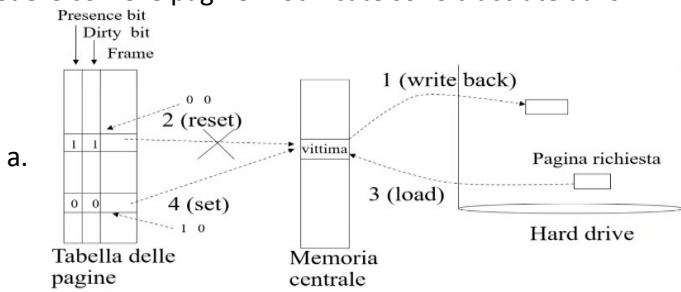
Per un rallentamento inferiore al 10%, f deve essere dell'ordine di 10^{-5}

Criticità delle prestazioni in caso di frequenza di major-fault
non minimale

Ma come possiamo mantenere la frequenza di major fault bassa? Devo stabilire la pagina da sostituire : ovvero gestione del resident set (insieme pagine per applicazione) e gestione della **vittima**; ma attenzione alle modifiche su pagina aggiornata nel frame da sovrascrivere (consistenza dei dati e delle modifiche). Ma attenzioni alle restrizioni : possibile frame bloccati (pagina può essere

eliminata / swap-out), quindi i/o asincrono ne risente. Quindi in generale per un qualunque frame esiste un bit particolare **bit lock** il quale non permette al frame di essere scelte come vittime: vale sia per frame USER e KERNEL. Andiamo ora a vedere in dettaglio:

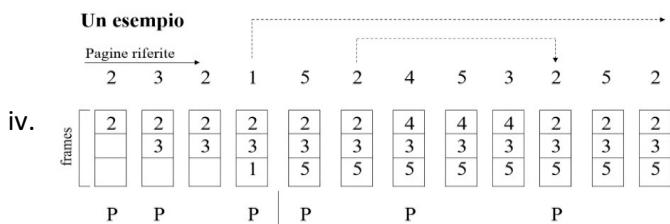
1. Vedere come le pagine modificate sono tracciate dal sw



- b. Si usa ulteriore bit oltre al presence bit il **dirty bit** (originariamente zero ma posto ad 1 se vi è qualunque scrittura); se questo bit è uno la pagina copiata su hard disk
2. Vediamo ora gli algoritmi per la scelta della vittima basati su metrica di valutazione (frequenza di page fault -> major fault), valutando tracce reali di esecuzione o scenari simulati, focalizzandoci sull'aspettativa degli algoritmi di scelta della vittima. Vediamoli

a. Algoritmo ottimo

- i. Algoritmo ideale per scegliere vittima all'interno di un sistema virtuale basato su paginazione
- ii. La scelta della vittima è basata sul fatto che mi riferirò a quella pagina dopo il più lungo tempo possibile
- iii. **Impossibile!!!** in quanto non ho conoscenza di eventi futuri

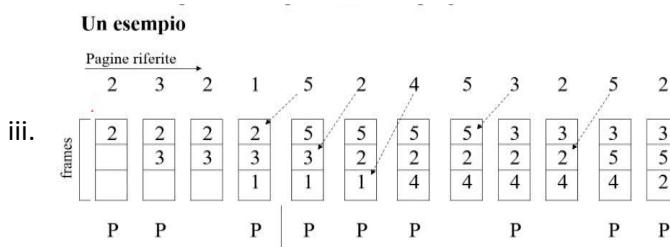


b. Lru (least recently used)

- i. La vittima è la pagina alla quale non ci si riferisce da più tempo (non è nella località dell'applicazione)
 - ii.
- | | |
|---|--|
| Un esempio
Pagine riferite →
2 3 2 1 5 2 4 5 3 2 5 2 | |
|---|--|
- iii. Si ha un page fault in più rispetto a quello ottimo; difficile da implementare
- iv. Non soffre dell'anomalia di Belady **essendo algoritmo a stack**

c. FIFO (first in first out)

- i. Sostituisce la pagina presente da più lungo tempo
- ii. Basta una lista dei frame basato su ordine caricamento/materializzazione



d. Anomalia di Belady

- i. Si ha quando aumentando i frames aumenta il numero di page fault dovuti all'algoritmo di sostituzione (per esempio FIFO)

0	1	2	3	0	1	4	0	1	2	3	4
0	1	2	3	0	1	4	4	4	2	3	3
0	1	2	3	0	1	1	1	1	4	2	2
0	1	2	3	0	0	0	0	0	1	4	4

9 Page Faults

ii.

0	1	2	3	0	1	4	0	1	2	3	4
0	1	2	3	3	3	4	0	1	2	3	4
0	1	2	2	2	3	4	0	1	2	3	
0	1	1	1	2	3	4	0	1	2		
0	0	0	0	1	2	3	4	0	1		

10 Page Faults

e. Algoritmi a stack

i. Non soffrono dell'anomalia di Belady

per qualsiasi sequenza di riferimenti l'insieme delle pagine in memoria per n frames è sempre un sottoinsieme dell'insieme di pagine in memoria per $n+1$ frames

ii.

$$\forall \text{ sequenza } r : M(n+1, r) \supseteq M(n, r)$$

f. Algoritmo dell'orologio (not recently used)

i. Approssimazione di LRU

ii. La vittima è una pagina non utilizzata di recente

iii. Per ogni frame vi è associato un reference bit (frame utilizzato o no), rappresentato all'interno delle page table, il quale viene impostato ad 1 se c'è traduzione da indirizzo logico a fisico (caricamento nel tlb).

iv. Periodicamente il so scorre (una porzione) tutti i reference bit e li resetta (li setta a 0)

v. Per sostituire la pagina si sceglie uno impostato a 0

vi. Durante la ricerca vengono resettati tutti i reference bit

vii. Approssima algoritmo a stack

g. Algoritmo dell'orologio con bit addizionali

i. Variante che considera sia il reference bit che il dirty

ii. Si preferisce fare la ricerca dove il dirty bit è 0

iii. Quattro possibilità (rb, db) = (0,0), (0,1), (1,0), (1,1)

h. Nota: è possibile sempre identificare la vittima ? Diciamo di sì se esecuzione atomica, no se si hanno macchine single core o time-sharing. In dettaglio :

