

COPIATURA SU SCRITTURA ➤ 10.3 (copy on write)

Viene usata nei contesti di fork di un processo:

• Fork Trad:

Tutte le pagine vengono copiate dal processo padre al figlio.

• Copy-on-write:

Alla fork(), ogni pagina è shared.

Le pagine modificabili rimangono shared

Fino a quando 1 dei processi non va a scriverci.

Nella scrittura si crea una copia locale

• Vfork ➔ execve perché non servono pages.

Variante del copy-on-write in cui non viene fatta la copia alla scrittura. Il padre si sospende mentre il figlio usa le shared pages.

PAGE REPLACEMENT ➤ 10.4

Siccome spesso si allocano più pagine di quante ne servono, il sistema tende ad allocare nuovi processi sfruttando le pagine non usate degli altri processi (over-allocation).

PROBLEMA DI OVER ALLOCATION

L'over allocation genera problemi quando:

- Un processo richiede tutte le sue pagine
- L'area assegnata all'I/O cresce di fatto

comportando che la free frame list si svuoti.

All'arrivo di nuovi processi, l'S.O. può:

✗ terminare il processo: male

✗ swap out di proc. meno usati: Comporta costi alti

✓ swap + page replacement.

SWAP + PAGE REPLACEMENT

Arrivati alla situazione di processo che richiede mem e Ø frame:

1. seleziono victim frame: Uscito meno di recente.

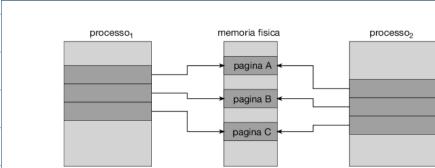


Figura 10.7 Prima della modifica alla pagina C da parte del processo1.

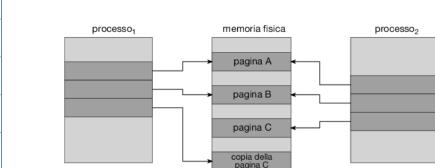


Figura 10.8 Dopo la modifica alla pagina C da parte del processo1.

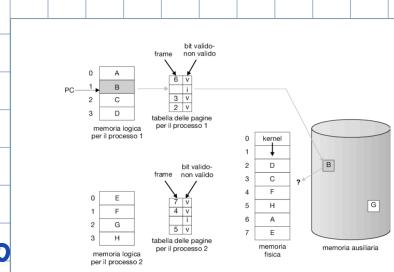


Figura 10.9 Necesidad de sustitución de páginas.

2. swap out ↗

3. aggiorno la symbol table:

Del proprietario del frame, invalid

4. swap in del req. page nel frame libero

S. Aggiorno tutte le altre Page Table.

Questa operazione ha 1 problema: vengono fatti 2 swap che radoppiano il tempo di Page fault.

OTTIMIZZAZIONE

Per risolvere il problema, si associa un dirty Bit / modify Bit ad ogni Page / Frame.

- 0 : La pagina combacia con quanto salvato in mem. secondaria
 - 1 : La pagina non combacia con mem. secondaria

Così facendo, si dimezza il tempo di replacement se uguali.

ALGO. DI PAGE REPLACEMENT > P.T. 1

La mem. virtuale necessita algoritmi di

- Frame Allocation: # di frame da assegnare al processo
 - Page Replacement: Selezione dei frame da sostituire

La metrica dell'algoritmo di Page Replacement è di minimizzare i Page Fault.

REL. FRAME / PAGE FAULT

Gli algoritmi si esaminano guardando una

sequenza di accessi, identificata per num. di pagina.

Si ha bisogno anche dei #frame disponibili. Più sono e meno forge Fault ci saranno.

ESEMPIO Dati gli ind₇, con 100 byte per pagina:

0100, 0432, 0101, 0612, 0102, 0103, 0104, 0101, 0611, 0102, 0103, 0104, 0101, 0610, 0102, 0103, 0104, 0101, 0609, 0102, 0105

(1) Transform. Seq. Page:

1, 4, 1, 6, 1, 1, 1, 1, 1, 6, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1

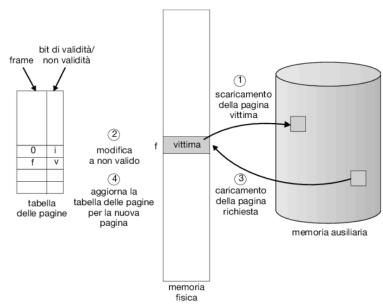


Figura 10.10 Sostituzione di una pagina

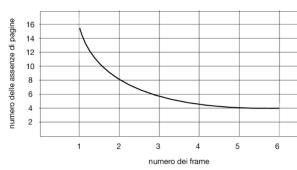


Figura 10.11 Grafico che illustra il numero di page fault rispetto al numero dei frame.

(2) Prendo solo i "cambiamenti"

1 2 7 5 6 8 9 10
1 4 1 6 2 6 1 6 1

(3) Calcolo # di Page fault

$\rightarrow 1 \text{ frame} = 1 \text{ seq} = 11$

$\rightarrow 3 \text{ frames} = 3 \text{ (1 per ogni "1° accesso")}$

FIFO ALG 10.4.2

Prevede di usare una coda in cui le Page Repl. avverranno in Testa (Head) che diventerà la Coda (Tail) della FIFO

ESEMPIO: (Quella presente da più tempo)

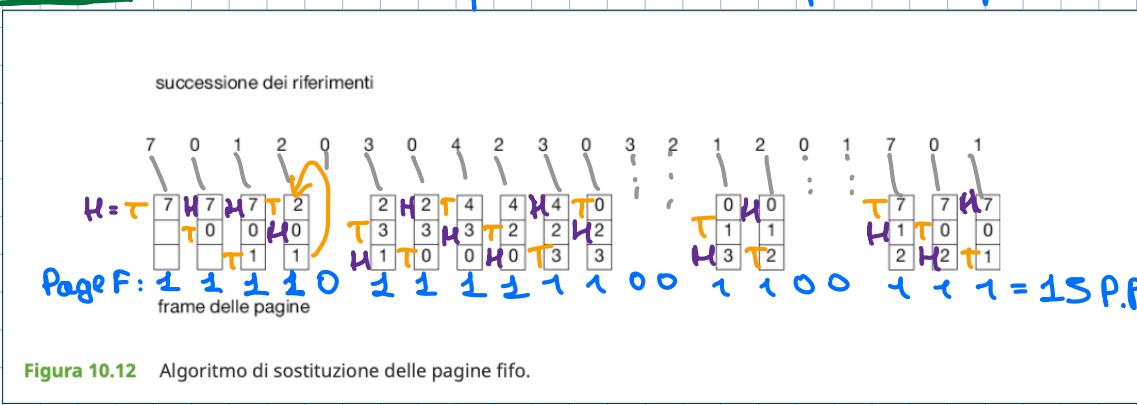


Figura 10.12 Algoritmo di sostituzione delle pagine fifo.

Lo svantaggio principale delle FIFO è che non è ottimo perché selezionare la pagina rimasta "più tempo" non equivale a selezionare quella meno usata (cattiva scelta $\rightarrow +\text{tempo}$)

ANOMALIA DI BELADY

Si verifica su alcuni algo. di Page Rep.

L'effetto è che aumentando i frames,

aumentano i Page fault su seq. particolari

ALG. OTTIMALE 10.4.3

L'algo. min / opt sostituisce la pagina

che non verrà usata per il tempo più lungo

ESEMPIO

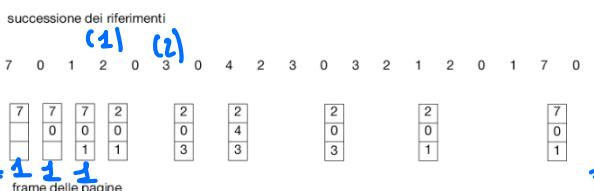


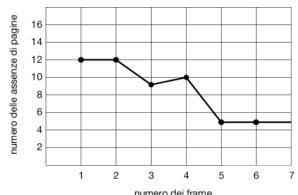
Figura 10.14 Algoritmo ottimale di sostituzione delle pagine.

(1) I tempi futuri sono

$\sqrt{T}: 18^\circ, 0: 5^\circ, 1: 12^\circ$

(2) T fut:

$0: 1, 2 = 3, 1 = 8 \checkmark$



1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
1 1 1 1 1 1 0 0 1 1 0
4 2 3 4 1 3 4 1 2
5 1 2 5 3 2 5 3 4

Lo svantaggio è che bisogna conoscere le esecuzioni future

ALGO. LAST RECENTLY USED (Opt. inverso opt(s^n) = LRU)

È un'approssimazione di opt. Sostituisce la pagina meno usata del passato:

ESEMPIO



$$(1) \check{z} = 3 \text{ fa}, 0 = 2 \text{ fa} \quad \check{v}$$

$$1 = 1 \text{ fa}$$

$$(2) \check{z} = 2 \text{ fa}, 0 = 1 \text{ fa}$$

$$\underline{1 = 3 \text{ fa}} \quad \check{v}$$

+ Vantaggi: - Non soffre di A. Belkamp

- Pochi Page Fault

- Svantaggi: - Necessita HW per def. ordine frame su uso

IMPLEMENTAZIONE

• Contatori: Sost. min. mom. utilizzo

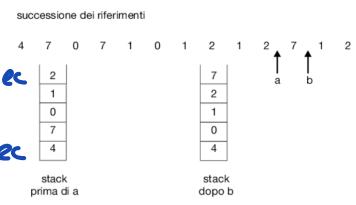
Si inserisce campo mom. utilizzo per ogni elem. della Page Table. La CPU avrà un contatore che aumenta ad ogni accesso in mem.

All'accesso alla Page, si imposta il val. del contatore

• Stack

Si usa uno stack in cui l'elem in cima è quello più recente.

Richiede agg. dello stack a ogni nodo.



ALGO. PAGE APPROX LRU ➔ 10.4.5

Siccome l'algo LRU necessita HW per agg. counter/stack, non tutti i sistemi ce li hanno. Senza HW rallenterebbero troppo

IDEA DI BASE

Si vuole approssimare l'algoritmo LRU. Per fare ciò si usano i reference bit, presenti in ogni riga E Page Table.

All'inizio, l'S.O. azzera i RB e man mano che il processo li

usa, li imposta a 1.

Così facendo si sa quali pagine sono state usate (senza ord.).

APPROX. BIT SUPPL. 10.4.5.1

Si definisce una tabella con n bit (dim = PageTable).

Ogni ∞ viene mandato un interrupt in cui si censiscono le Pages usate:

(1) Si inserisce "1" nel bit più sign. (∞)

(2) shift a dx (scarto meno sign.) e "0" sul nuovo asr

Così facendo si può prendere il min unsigned int.

ESEMPIO

00000000 → Non usato in 8 tempi

10010010 più recente di 01111111

APPROX. SECONDA CHANCE o "Clock" 10.4.5.2

È un FIFO con HW per bit di riferimento.

Creare una coda circolare in cui il

Puntatore è la vittima.

- se è 0: la vittima è scelta per PageR.

- se è 1: Si imposta 0 e si va avanti.

Nel caso peggiore, si ripercorre tutta la coda fino a se.

APPROX. SECONDA CHANCE MIGLIORE 10.4.5.3

Variante della 2° chance che sfrutta il modify / dirty bit implementato per dimezzare lo swap:

Rif bit Dirty Bit

0

0

Non usato e già in mem. Ottimo

0

1

Non usato ma now in mem → I/O

1

0

Usato di recente ma in mem

1

1

Usato di recente e now in mem **Male**

Controllare anche il dirty bit consente di ridurre lo swap.

PAGE REPL. SU CONTEGGIO > 10.4.6

Ci sono algoritmi alternativi che si basano su un campo

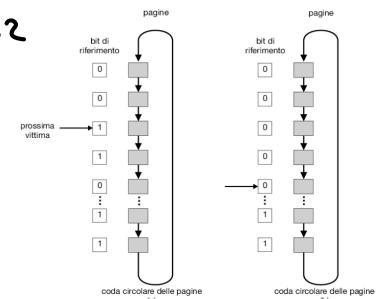


Figura 10.17 Algoritmo di sostituzione delle pagine con seconda chance (orologio).

contatore di utilizzo & riga & Page Table:

- LEAST FREQUENTLY USED (LFU)

La vittima è la page col contatore min

? vantaggi: Dovrebbe preferire le page meno usate

- svantaggi: Se una page viene usata tanto all'inizio, e poi niente → Non verrà tolta per un bel po'

- MOST FREQ. USED (MFU)

La vittima è la page col contatore max.

Basata su presupposto che le nuove pages hanno count = 0 e quindi non sono state ancora usate

Entrambe non si avvicinano a opt.

ALGO. REPL. SU BUFFERING > 10.4.7

Si tiene un pool di free frames. Ad ogni Page Fault si andrà a salvare il contenuto della page nel frame pool.

In secondo piano avverrà lo swap in della Page.

Da cui si possono creare 2 variazioni

- SENZA NOTE

Terminato lo swap in, si rimette il frame nei frame liberi **senza** segnare a chi appartenevano i frame.

- CON NOTE

Terminato lo swap in, si rimette il frame nei frame liberi **segnando** la page che conteneva.

Nelle richieste successive, si guarda il frame pool che fa da "mem. associativa" per evitare lo swap.

Usando questo principio si può tenere una lista di Page modificate e fare swap in quando la mem **non** è in uso.

<10.4.8> Ignored: Aspetto lezione