esercizi
memoria virtuale

tabella delle pagine (8.1)

- data la tabella delle pagine (pagina di 1024 bytes) di un processo tradurre i seguenti riferimenti in indirzzi fisici: 1052, 5499, 2221

Numero della pagina virtuale	Valid bit	Reference bit	Modify bit	Numero di frame
0	1	1	0	4
1	1	1	1	7
2	0	0	0	-
3	1	0	0	2
4	0	0	0	-
5	1	0	1	0

1052, frame 4, offset 1052-1024

5499, frame 0, offset ...

2221, page fault

dove mettiamo le page tables?

- su molti sistemi le user page tables di ciascun processo sono parte dell'immagine del processo
- quali sono i vantaggi?
- quali sono gli svantaggi?
  - ciascun riferimento a memoria quanti page fault può generare?

vantaggi: si può trattare la tabella delle pagine alla stregua delle altre pagine del processo.

svantaggi: ciascun riferimento a memoria può generare un numero di page fault pari ai livelli della gerarchia delle tabelle

Attenzione la directory deve essere sempre residente!

quanti page faults?

- considera la seguente istruzione assembly
  mov32 0x00800FFE → 0x00900000
- tale istruzione sposta il contenuto della parola di 4 byte all'indirizzo 0x00800FFE nella parola all'indirizzo 0x00900000
- l'architettura prevede pagine di 4KB e page table a un livello, PTE di 4 byte
  - assumi che le pagine con la tabella delle pagine sia sempre residente
  - indirizzo virtuale: 20 bit di pagenumber, 12 bit di offset
- l'istruzione è all'indirizzo 0x00400FFB ed è codificata con 9 bytes
- supponi che il PC punti a 0x00400FFB e si deve ancora eseguire il fetch
- quanti page fault possono al più aver luogo per l'esecuzione di tale istruzione?

4

Fetch: accede a 2 pagine (0x00400-0x00401), read: accede a 2 pagine (0x00800-0x00801), write: accede a 1 pagina (0x00900)

gli indirizzi sono in pagine diverse

totale max 5 page faults

quanti page faults?

- stesse ipotesi di prima
- considera la seguente istruzione assembly
  mov32 0x00800FFE → 0x00801100
- quanti page faults? cosa è cambiato?

considera ora

mov32 0x00800FFE → 0x00801100

l'indirizzo destinazione è nella seconda pagina acceduta da “read”

4 page faults

quanti page faults?

- considera la seguente istruzione assembly
  mov32 0x00800FFE → 0x00801FFE
- tale istruzione sposta il contenuto della parola di 4 byte all'indirizzo 0x00800FFE nella parola all'indirizzo 0x00801FFE
- l'architettura prevede pagine di 4KB e page table a due livelli, PTE di 4 byte, ciascuna pagina contiene 2^{10} PTE
  - assumi che la pagina che contiene la directory sia sempre residente
  - indirizzo virtuale: 10 bit x liv. 1, 10 bit x liv. 2, 12 bit di offset
- l'istruzione è all'indirizzo 0x003FFFFE ed è codificata con 9 bytes
- supponi che il PC punti a 0x003FFFFE e si deve ancora eseguire il fetch
- quanti page fault possono al più aver luogo per l'esecuzione di tale istruzione?

6

l'istruzione e i due operandi sono a cavallo della separazione tra due pagine.

Fetch: accede a due pagine, read: accede a due pagine, write: accede a due pagine

ciascun accesso a pagina può dare luogo a 2 page fault potenziali, max 12 page fault totali

tuttavia notiamo che questa quantità di page fault è una sovra stima:

•Nota: indirizzo di 32 bit in esadecimale 0x 11½ (directory) ½11 (user page table) 111 (offset)

•le pagine in cui è contenuta l'istruzione hanno PTE in due distinte user pages e quindi possono entrambe generare 2 page fault

Fetch: accede a due pagine 0x003ff 000 e 0x00400 000→ 2 page fault + 2 page fault

•il primo operando è alle pagine 0x00800 000 e 0x00801 000

•entrambe queste pagine hanno la PTE all'interno della stessa user page table. L'accesso alla prima rende la user page table disponibile per la seconda.

read: accede a due pagine con PTE nella stessa user page table → 2 page fault + 1 page fault (user page table è già dentro)

•il secondo operando è alle pagine 0x00801 000 e 0x00802 000

•entrambe queste pagine hanno la PTE all'interno della stessa user page table della read.

•la prima parte è nella stessa pagina della seconda metà della read.

write : accede a due pagine con PTE nella stessa user page table già residente, la prima pagine è già residente → 0 page fault (la pagina è già dentro vedi read) + 1 page fault (user page table è già dentro)

TOTALE: 8

page replacement (8.4)

Numero della pagina virtuale	Page Frame	Tempo di caricamento	Tempo di riferimento	R bit	M bit
2	0	60	161	0	1
1	1	130	160	0	0
0	2	26	162	1	0
3	3	20	163	1	1

- page fault alla pagina 4 e tempo 164
- quale pagina viene sostituita?
  - rispondi usando FIFO, LRU, Clock (con uso del bit di modifica M), Optimal
  - page reference string: 4 0 0 0 2 4 2 1 0 3 2

7

fifo: 3

LRU: 1

corretto: Clock: l'ultima pagina inserita è stata la pagina 1 nel frame 1 quindi clock parte dal frame 2 che però ha R bit pari a 1. Clock semplice metterebbe R bit pari a zero per frames 2 e 3 e seglierebbe il frame 0. Il clock modificato non mette R bit pari a zero per frame 2 e 3, trova il frame 0 che e' stato modificato e quindi sceglie il frame 1 che non e' stato ne modificato ne referenziato.

M=1, il frame 2 ha R=M=0 e viene sostituito.

Optimal: 3

LRU (8.4)

Numero della pagina virtuale	Page Frame	Tempo di caricamento	Tempo di riferimento	R bit	M bit
2	0	60	161	0	1
1	1	130	160	0	0
0	2	26	162	1	0
3	3	20	163	1	1

- page reference string: 4 0 0 0 2 4 2 1 0 3 2
- 4 frames, page replacement LRU
- quanti page fault avvengono? quando?

8

	coda per LRU
3	3 0 2 1
4f	4 3 0 2
0	0 4 3 2
0	0 4 3 2
0	0 4 3 2
2	2 0 4 3
4	4 2 0 3
2	2 4 0 3
1f	1 2 4 0
0	0 1 2 4
3f	3 0 1 2
2	2 3 0 1

working set (8.4)

Numero della pagina virtuale	Page Frame	Tempo di caricamento	Tempo di riferimento	R bit	M bit
2	0	60	161	0	1
1	1	130	160	0	0
0	2	26	162	1	0
3	3	20	163	1	1

- page reference string: 4 0 0 0 2 4 2 1 0 3 2
- Δ=4, variable allocation, RS=WS
  - cioè, numero di frame |W| e replacement LRU
- quanti page fault avvengono? quando?
- come varia |W|?

9

	ultimi Δ accessi	|W|
3	3 0 2 1	4
4f	4 3 0 2	4
0	0 4 3 0	3
0	0 0 4 3	3
0	0 0 0 4	2
2f	2 0 0 0	2
4f	4 2 0 0	3
2	2 4 2 0	3
1f	1 2 4 2	3
0f	0 1 2 4	4
3f	3 0 1 2	4
2	2 3 0 1	4

upper/lower bound (8.12)

- stringa di riferimenti a pagine di lunghezza P
- le pagine distinte nella stringa sono N≤P
- i frames allocati al processo sono M
  - tutti inizialmente vuoti
- algoritmo di replacement non specificato!
- lower bound sul numero di page faults?
- upper bound sul numero di page faults?

10

lower bound

bene che vada ciascuna delle N distinte pagine va portata in memoria prima di essere utilizzata, quindi un lower bound è N

upper bound

supponiamo che M>=N

in questo caso il numero di page faults è proprio N in qualsiasi caso.

supponiamo che M<N (il caso significativo)

in questo caso un avversario che conosce la nostra strategia può costruire una stringa di riferimenti che provochi un fault per ciascun riferimento, infatti c'è sempre almeno una pagina che non è residente, ad esempio quella eliminata dal riferimento precedente, su cui si deve avere necessariamente un page fault. Per cui l'upper bound è P

page buffering

- considera la seguente stringa di riferimenti
- 1 2 3 4 1 2 5 2 1 3 2 3
- considera i tre casi
  - FIFO con 4 frame
  - FIFO con 2 frame e PAGE BUFFERING fifo con 2 frame (il buffering è anch'esso gestito con una coda FIFO)
  - LRU con 4 frame
- conta i page fault e gli accessi a disco nei tre casi

11

Seq	FIFO4	FIFO2+ PB2	LRU4
1	1 FD	1 FD	1 FD
2	2 1 FD	2 1 FD	2 1 FD
3	3 2 1 FD	3 2 * 1 FD	3 2 1 FD
4	4 3 2 1 FD	4 3 * 2 1 FD	4 3 2 1 FD
1	4 3 2 1	1 4 * 3 2 F	1 4 3 2
2	4 3 2 1	2 1 * 4 3 F	2 1 4 3
5	5 4 3 2 FD	5 2 * 1 4 FD	5 2 1 4 FD
2	5 4 3 2	5 2 * 1 4	2 5 1 4
1	1 5 4 3 FD	1 5 * 2 4 F	1 2 5 4
3	1 5 4 3	3 1 * 5 2 FD	3 1 2 5FD
2	2 1 5 4 FD	2 3 * 1 5 F	2 3 1 5
3	3 2 1 5 FD	2 3 * 1 5	3 2 1 5

FIFO: ogni fault un accesso a disco #F=#D=8

LRU: ogni fault un accesso a disco #F=#D=6

FIFO+PB: #F=10, #D=6

anomalia di belady

- considera la seguente stringa di riferimenti
  - 0 1 2 3 0 1 4 0 1 2 3 4
- usa FIFO
- considera 3 frame allocati al processo
  - conta i page fault
- considera 4 frame allocati al processo
  - conta i page fault
- noti qualcosa di strano?

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

FIFO worst case

- considera l'algoritmo FIFO con 3 frame allocati al processo
- descrivi una famiglia di stringhe di riferimenti (di lunghezza infinita) che generi un fault ad ogni riferimento
- qual'è il numero minimo di pagine che deve contenere la stringa?

1 2 3 4 1 2 3 4...

da 4 in poi ciascun riferimento genera un page fault che sostituisce un'altra pagina. Viene sostutuita sempre la prossima pagina che verrà referenziata.

Almeno devo avere 4 pagine (con 3 frame) altrimenti avrei solo i primi 3 fault.

Optimal: fault ad ogni accesso?

- considera l'algoritmo Optimal con 3 frame allocati al processo
- esiste una stringa di riferimenti infinita che genera un page fault ad ogni accesso? fornisci un esempio o una dimostrazione di non esistenza.

14

Una tale stringa non esiste.

Considera una stringa di riferimenti che inizia con tutte pagine diverse fino alla richiesta i-1. La richiesta i-sima è la prima richiesta che viene fatta ad una pagina già richiesta in precedenza. Chiamiamo p tale pagina. La richiesta precedente a p è stata fatta all'istante $j<i$.

Vogliamo dimostrare che all'istante i-esimo non c'è mai page fault, cioé che all'istante i-esimo p è già in memoria, cioè che dall' istante j+1 in poi p rimane in memoria, cioè che optimal sceglie sempre una pagina diversa da p negli istanti j+1...i-1.

Consideriamo le scelte che optimal fa su sulle richieste j+1...i-1. Optimal sostituisce la pagina per cui il prossimo istante in cui sarà refernziata è il più lontano nel tempo (le pagine non più referenziate sono considerate come infinitamente distanti nel tempo e quindi vengono sostuite prima delle altre).

Nota che le pagine in memoria subito prima del riferimento i-esimo sono un sottoinsieme delle pagine riferite agli istanti 1...i-1. Per tali pagine, tranne per p, il successivo istante di riferimento è sempre strettamente maggiore di i poiché la pagina p è la prima ad essere referenziata per la seconda volta, quindi optimal preferisce sempre una pagina che è diversa da p e p rimarrà in memoria.

LRU=WS?

- caratterizzare la classe di stringhe di riferimenti su cui LRU con F frames da gli stessi page fault di WS con finestra Δ=F.
- mostra un esempio nella classe e uno fuori da tale classe

15

La classe è composta da

"le strighe in cui |W(t,Δ)| non varia ed è sempre pari a F al variare di t".

F=Δ=3

1 2 3 1 2 3 è nella classe

1 2 1 3 1 2 1 3 1 non è nella classe

il secondo accesso alla pagina 2 genera un fault con WS e non lo genera con LRU (sottolineati gli elementi che ricadono nella finestra temporale, di lunghezza Δ, nel momento del fault)

FIFO=LRU?

- 4 pagine e 3 frames
- dai una classe di stringhe di riferimenti che danno gli stessi page fault sia per FIFO che per LRU sostituendo le stesse pagine
  - dai una classe che genera un fault ad ogni accesso
  - prova quindi a dare una classe che NON genera un fault ad ogni accesso

16

La classe che ha page fault ad ogni accesso e sostituisce le stesse pagine in LRU e FIFO è “la classe delle stringhe che non fanno mai un accesso ad una pagina che è già residente”.

Cioè ciascuna pagina è acceduta solo quando abbiamo il page fault che la ha caricata. Infatti in questo caso LRU mantiene una coda identica a FIFO.

esempio

1 2 3 4 1 2 3 4 ...

F F F F_1 F_2 F_3 F_4 F_1 .... (F_n indica che è stata sostituita la pagina n)

che può essere espressa come (1 2 3 4)*

Per ottenere una stringa che non generi sempre fault possiamo inserire delle sottostringhe tra un fault e l'altro che non cambino l'ordine, ad esempio

1 2 3 2 3 1 2 3 1 2 3 4 1 2 3 4 ...

F F F F_1 F_2 F_3 F_4 F_1 ....

Un esempio della classe richiesta è

(1 2 3 ((1|2|3)* 2 3) 4)*

dove ((1|2|3)* 2 3) lascia invariato l'ordine all'interno della coda LRU e non genera page fault.

FIFO = CLOCK?

- 3 frame e 4 pagine
- dai una classe di stringhe di riferimenti su cui FIFO da gli stessi page fault di CLOCK
  - dai una classe che genera un fault ad ogni accesso
  - prova quindi a dare una classe che NON genera un fault ad ogni accesso

17

Una stringa con un fault ad ogni accesso sia per FIFO che per CLOCK è 1 2 3 4 1 2 3 4 ....

Per non avere un accesso senza fault bisogna far riferimento ad una pagina residente ma grazie all'uso dello "use bit" questo normalmente fa sì che CLOCK si comporti meglio di FIFO.

La classe può essere la seguente

"le stringhe in cui dopo un fault si referenziano tutte le pagine residenti "

Infatti se tutti i frame hanno use=1 CLOCK sostituisce la pagina nel frame attualmente puntato dalla lancetta e muove la lancetta al prossimo frame, cioè si comporta come FIFO.

Nella seguente espressione regolare gli accessi sottolineati sono quelli che mettono use=1.

1 2 3 (4 2 3 4 1 3 4 1 2 4 1 2 3 1 2 3)*

F F F F_1 F_2 F_3 F_4

Per tale classe di stringhe di riferimenti FIFO e CLOCK sostituiscono le stesse pagine..