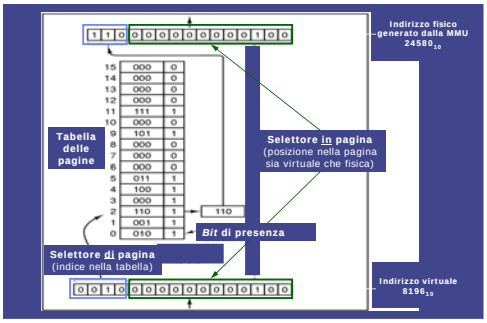
“L’angolo della posta”
(discussione su vostri quesiti)

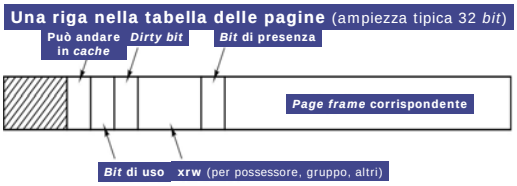
Discussione in aula:

Claudio Palazzi – *cpalazzi@math.unipd.it*

Paginazione: strutture – 1



Paginazione: strutture – 2

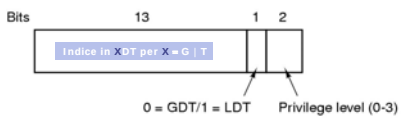


- L’indirizzo di disco ove la pagina si trova quando non è in RAM **non** è nella tabella!
  - La tabella delle pagine serve alla MMU (*hardware*)
  - Il caricamento della pagina da disco viene effettuato dal S/O (*software*)
  - L’informazione dell’uno **non serve** all’altro

Segmentazione: realizzazione – 1

- Vista la grande ampiezza potenziale i segmenti sono spesso **paginati**
- Nel caso del Pentium di Intel
  - Fino a 16 K segmenti indipendenti
    - Di ampiezza massima 4 GB (32 *bit*)
  - Una LDT per processo
    - *Local Descriptor Table*
      - Descrive i segmenti del processo
  - Una singola GDT per l’intero sistema
    - *Global Descriptor Table*
      - Descrive i segmenti del S/O

Segmentazione: realizzazione – 2



- 6 registri di segmento
  - Di cui 1 denota il segmento corrente
- LDT e GDT contengono $2^{13} = 8$ K descrittori di segmento
  - I descrittori di segmento sono espressi su 8 B
    - La **base** del segmento in RAM è espressa su 32 *bit*
    - Il **limite** su 20 *bit* per verificare la legalità dell’*offset* fornito dal processo
      - Consente ampiezza massima a 1 MB (per **granularità** a B)
      - Oppure 1 M pagine da 4 KB ovvero 4 GB (per granularità a pagine)

Segmentazione: realizzazione – 3

- L’indirizzo **lineare** ottenuto da (base di segmento + *offset*) può essere interpretato come
  - Indirizzo **fisico** se il segmento considerato non è paginato
  - Indirizzo **logico** altrimenti
    - Nel qual caso il segmento viene visto come una memoria virtuale paginata e l’indirizzo come virtuale in essa
      - 10 *bit* : indice in catalogo di tabelle delle pagine
        - 2^{10} righe da 32 *bit* ciascuna (base di tabella denotata)
      - 10 *bit* : indice in tabella delle pagine selezionata
        - 2^{10} righe da 32 bit ciascuna (base di *page frame*)
      - 12 *bit* : posizione nella pagina selezionata
        - *Offset* in pagina da 4 KB

“Esercizio 2”

Sia data memoria dotata di 4 *page frame*, inizialmente libere, e 8 pagine di memoria virtuale. Utilizzando la politica FIFO per il rimpiazzo delle pagine, indicare quanti *page fault* si verifichino a fronte della stringa di riferimenti: 0 1 7 2 3 2 7 1 0 3:

A: 4
B: 3
C: 6
D: 2

“Esercizio 3”

Si consideri la seguente serie di riferimenti a pagine di memoria:
1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

Si considerino le seguenti politiche di rimpiazzo:
- LRU
- FIFO
- Optimal

Quanti *page fault* avvengono considerando un numero di *page frame* della RAM di 1, 2, 3, 4, 5, 6, 7 ?

“Esercizio 3” – soluzione FIFO

Consideriamo **FIFO** con 3 *page frame*
serie di riferimenti a pagine di memoria:

	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
NEW	1	2	3	4	4	1	5	6	2	1	1	3	7	6	6	2	1	1	3	6
		1	2	3	3	4	1	5	6	2	2	1	3	7	7	6	2	2	1	3
OLD			1	2	2	3	4	1	5	6	6	2	1	3	3	7	6	6	2	1
	P	P	P	P		P	P	P	P	P		P	P	P		P	P		P	P

16

“Esercizio 3” – soluzione LRU

Consideriamo **LRU** con 3 *page frame*
serie di riferimenti a pagine di memoria:

	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
MRU	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
		1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3
LRU			1	2	3	4	2	1	5	6	6	1	2	3	7	6	3	3	1	2
	P	P	P	P		P	P	P	P	P		P	P	P		P	P			P

15

“Esercizio 3” – soluzione OPT

Consideriamo **optimal** con 3 *page frame*
serie di riferimenti a pagine di memoria:

	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
	1	1	1	1	1	1	1	1	1	1	1	3	3	3	3	3	3	3	3	3
		2	2	2	2	2	2	2	2	2	2	7	7	7	2	2	2	2	2	2
			3	4	4	4	5	6	6	6	6	6	6	6	6	1	1	1	6	
	P	P	P	P			P	P				P	P			P	P			P

11

“Esercizio 5” (modificato dopo lezione)

Si consideri la matrice (o *array* bidimensionale):

```
int A[][] = new int[100][100];
```

Si assuma che la posizione **`A[0][0]`** di tale matrice sia posta alla locazione **200** di una memoria paginata con **3** pagine di dimensione **200** B.
Si assuma che un valore **`int`** occupi **1** B.
Si assuma che le pagine siano inizialmente tutte vuote e che il processo (il cui codice occupa esattamente **200** B) abbia la seguente esecuzione:

```
for (int i = 0; i < 100; i++){
   for (int j = 0; j < 100; j++){
      A[i][j] = 0;}}
```

Quanti *page fault* saranno generati usando LRU?

“Esercizio 5” – soluzione (1/2)

La matrice `A[i][j]` è scritta linearmente in memoria:

`A[0][0],A[0][1],A[0][2],…,A[0][99],A[1][0], A[1][1],…,A[99][99]`

Il processo azzera le celle della matrice proprio nel loro ordine di memorizzazione.

Quindi inizialmente verranno caricati nelle pagine
Pagina 0: Il programma (che occupa esattamente 200 B)
Pagina 1: Celle da A[0][0] a A[1][99] incluse
Pagina 2: Celle da A[2][0] a A[3][99] incluse

“Esercizio 5” – soluzione (2/2)

A ogni iterazione del programma la memoria viene acceduta per leggere l'istruzione successiva
→ la pagina relativa al processo verrà continuamente acceduta

I primi 200 azzeramenti faranno accesso a Pagina 1, i successivi 200 a Pagina 2; l'ulteriore azzeramento (cella `A[4][0]`) causerà un *page fault* ; la pagina usata meno recentemente è Pagina 1 che verrà sostituita con le celle da `A[4][0]` a `A[5][99]` incluse. Arrivati all'azzeramento di `A[6][0]` sarà Pagina 2 ad essere stata usata meno di recente
E così via, causando in tutto 1 *page fault* iniziale per caricare il processo in Pagina 0 e 50 *page fault* di Pagina 1 e 2 (25 ciascuno) per caricare le porzioni di matrice.

TOTALE = 51 *page fault*