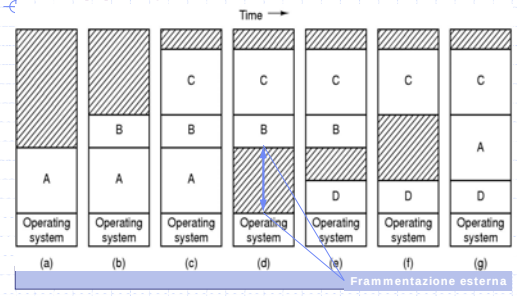
Gestione della memoria

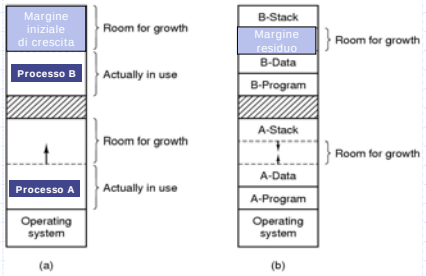
Ricapitolazione e discussione in aula:

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

Swapping – 1



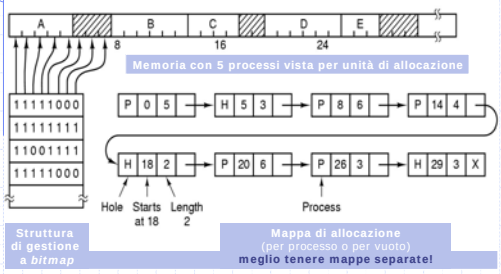
Swapping – 2



Strutture di gestione – 1

- Quando la memoria principale viene allocata dinamicamente è essenziale tenere traccia del suo stato d'uso
- Due strategie principali
  - (A) **Mappe di *bit***
  - (B) **Liste collegate**

Strutture di gestione – 2



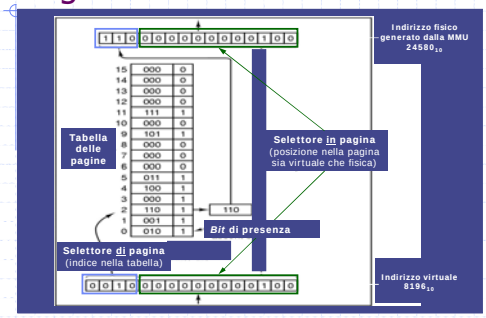
Strutture di gestione – 3

- Due strategie principali
  - (A) **Mappe di *bit***
    - Memoria vista come insieme di **unità di allocazione** (1 *bit* per unità)
      - Unità piccole → struttura di gestione grande
        - Esempio: Unità da 32 *bit* e RAM ampia 512 MB → struttura ampia 128 M *bit* = 16 MB → 3.1 % (= 1/32)
  - (B) **Liste collegate**
    - Nella sua versione più semplice la memoria è vista a segmenti
      - Segmento = processo oppure spazio libero tra processi
      - Ogni elemento di lista rappresenta un segmento
        - Ne specifica punto di inizio, ampiezza e successore
        - Liste ordinate per indirizzo di base

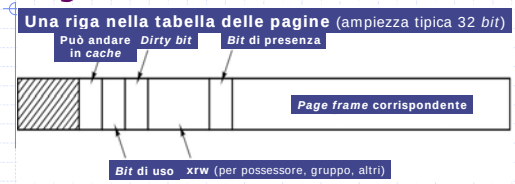
Strutture di gestione – 4

- Varie strategie di allocazione
  - ***First fit*** : il primo segmento libero ampio abbastanza
  - ***Next fit*** : come *First fit* ma cercando sempre avanti
  - ***Best fit*** : il segmento libero più adatto
  - ***Worst fit*** : sempre il segmento libero più ampio
  - ***Quick fit*** : liste diverse di ricerca per ampiezze "tipiche"

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

Paginazione: rimpiazzo – 1

- **NRU** (*not recently used*)
  - Per ogni *page frame* vengono aggiornati
    - *Bit* M (*modified*), inizializzato a 0 dal S/O
    - *Bit* R (*referenced*), posto a 0 **periodicamente** dal S/O per stimare la frequenza d'uso
  - Le pagine nei *page frame* sono classificate in
    - **Classe 0**: non riferita, non modificata
    - **Classe 1**: non riferita, modificata
    - **Classe 2**: riferita, non modificata
    - **Classe 3**: riferita, modificata
  - NRU sceglie una pagina **a caso** nella classe non vuota a indice più basso

Paginazione: rimpiazzo – 2

- **FIFO**
  - Rimuove la pagina di ingresso più antico in RAM
    - Basta una lista ordinata di *page frame*
      - Ogni inserimento viene marcato in coda e la rimozione avviene dalla testa
- ***Second chance***
  - Corregge FIFO rimpiazzando solo le pagine con *bit* R = 0
    - Altrimenti il *page frame* viene considerato come appena caricato, posto in fondo alla coda e R viene posto a 0
    - Degenera in FIFO quando tutti i *page frame* siano stati recentemente riferiti

Paginazione: rimpiazzo – 3

- **Orologio**
  - Come SC ma i *page frame* sono mantenuti in una lista **circolare**
    - L'indice di ricerca si muove come una lancetta
- **LRU** (*least recently used*)
  - Necessita di *hardware* dedicato
- **Aging** (*not frequently used* modificato)
  - Realizzabile a *software*
  - Per ogni *page frame* aggiorna periodicamente un "contatore" C che cresce di più se R = 1
    - Non incrementa C con R ma gli inserisce R a sinistra
  - Approssima LRU con differenze importanti
    - Valuta solo periodicamente (a grana grossa)
    - Usando N *bit* per C perde memoria dopo N aggiornamenti

Paginazione: rimpiazzo - 4

- **WS approssimato**
  - Simile all'*Aging*
    - Ogni *page frame* in RAM ha un attributo temporale che indica se a un dato istante appare come riferita (R = 1)
      - Tale attributo prende il valore **t** del **tempo virtuale corrente** all'arrivo di un *page fault*
      - R e M sono posti a 1 dall'*hardware*
      - R è posto a 0 (se non in uso) da un controllo periodico e al *page fault*
    - Al *page fault* sono rimpiazzabili le pagine con R = 0 e valore di attributo **antecedente** all'intervallo (**t** − Δt, **t**)
      - Se all'istante **t** tutti i *page frame* avessero R = 1 verrebbe rimpiazzata una pagina scelta a caso, con M = 0
    - Nel caso peggiore bisogna scandire l'**intera** RAM!

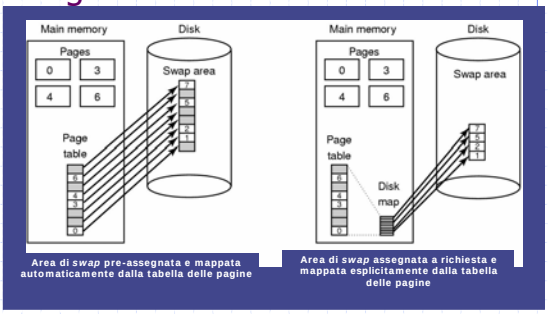
Paginazione: rimpiazzo - 5

- **WS approssimato con orologio**
  - *Page frame* organizzati in lista circolare
    - Come per l'orologio semplice
    - Ma con le informazioni del WS approssimato
  - Una "lancetta" indica il *page frame* corrente
    - Al *page fault* se R = 1 la lancetta avanza e R = 0
    - Se R = 0 si valuta l'attributo temporale
      - Se fuori da **w**(**k**,**t**) e con M = 0 allora rimpiazzo
      - Altrimenti il *page frame* va in una coda di trasferimento su disco e la lancetta avanza
        - Alla ricerca di un *page frame* rimpiazzabile direttamente
        - Quando **N** pagine in coda si trasferisce su disco
    - Se nessun *page frame* è rimpiazzabile allora si sceglie una pagina con M = 0 altrimenti quella cui punta la lancetta

Paginazione: rimpiazzo - 6

Algorithm	Comment
Optimal	Not implementable, but useful as a benchmark
NRU (Not Recently Used)	Very crude
FIFO (First-In, First-Out)	Might throw out important pages
Second chance	Big improvement over FIFO
Clock	Realistic
LRU (Least Recently Used)	Excellent, but difficult to implement exactly
NFU (Not Frequently Used)	Fairly crude approximation to LRU
Aging	Efficient algorithm that approximates LRU well
Working set	Somewhat expensive to implement
WSClock	Good efficient algorithm

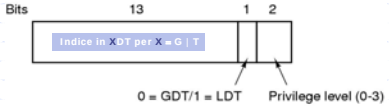
Paginazione: realizzazione – 1



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 1

Dato un sistema di *swapping* e una memoria con zone disponibili di ampiezza: 8, 4, 14, 18, 17, 9, 12, 15 KB, in questo ordine, indicare quale area venga prescelta dalla politica ***Next Fit*** a fronte della richiesta di caricamento di un segmento di ampiezza 3 KB dopo aver caricato un segmento ampio 12 KB:
A: 4 KB
B: 18 KB
C: 14 KB
D: 9 KB.

Esercizio 1 - soluzione

Dato un sistema di *swapping* e una memoria con zone disponibili di ampiezza: 8, 4, 14, 18, 17, 9, 12, 15 KB, in questo ordine, indicare quale area venga prescelta dalla politica ***Next Fit*** a fronte della richiesta di caricamento di un segmento di ampiezza 3 KB dopo aver caricato un segmento ampio 12 KB:
A: 4 KB
B: 18 KB
C: 14 KB
D: 9 KB

E con ***First fit? Best fit? Worst fit?***

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 2 - soluzione

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 4 7 3 7 4 1 0 3:
A: 4
B: 3
C: 6
D: 2

Posso decidere anche per altre politiche di rimpiazzo (NRU, *Second Chance*, LRU, ecc.)? Oppure ho bisogno di altre informazioni (in caso affermativo, quali?) ?

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

# page frame	FIFO	LRU	*Optimal*
1	20	20	20
2	18	18	15
3	16	15	11
4	14	10	8
5	10	8	7
6	10	7	7
7	7	7	7

Esercizio 4.1

Sia dato un sistema di gestione della memoria principale basato su segmentazione con indirizzi logici e fisici espressi su 16 *bit*. Si consideri la tabella dei segmenti riportata di seguito, ove il prefisso 0x denota l'uso di notazione Esadecimale:

Segmento	Base	Limite
$0x0$	$0x0219$	$0x600$
$0x1$	$0x2300$	$0x014$
$0x2$	$0x0090$	$0x100$
$0x3$	$0x1327$	$0x580$
$0x4$	$0x1952$	$0x096$
...	...	...
$0xF$	...	...

Si mostri graficamente la mappa di memoria corrispondente, indicando anche il suo grado percentuale di occupazione.

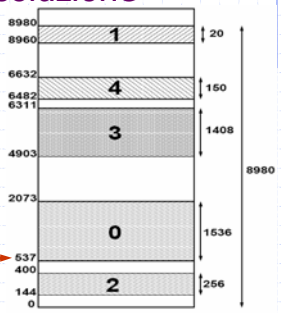
Esercizio 4.1 - soluzione

Con un po' di calcoli:

0x219 =
2 × 256 + 1 × 16+9 × 1 =
537

Eccetera ...

L'area di memoria occupata va dall'indirizzo 0 all'indirizzo 8980. L'ampiezza complessiva dei segmenti in tale zona misura 3370 B con grado di occupazione: **37, 53%**.



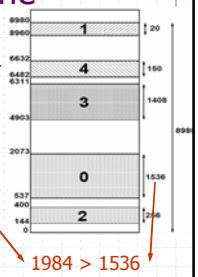
Esercizio 4.2

Si fornisca l'indirizzo fisico corrispondente ai seguenti indirizzi virtuali espressi in notazione decimale:

a. $< 0, 1984 >$
b. $< 1, 18 >$
c. $< 2, 250 >$
d. $< 3, 1400 >$
e. $< 4, 112 >$

Esercizio 4.2 - soluzione

Ricordando che ...



Allora:

indirizzo virtuale	indirizzo fisico
$< 0, 1984 >$	Errore: indirizzo illegale
$< 1, 18 >$	$8960 + 0018 = 8978$
$< 2, 250 >$	$0144 + 0250 = 0394$
$< 3, 1400 >$	$4903 + 1400 = 6303$
$< 4, 112 >$	$6482 + 0112 = 6594$

Esercizio 4.3

Indicare il modo nel quale i segmenti di indice 0–4 possano essere mappati su disco, assumendo blocchi di ampiezza 1 KB, calcolando anche la quantità percentuale di memoria *sprecata* di conseguenza.

Esercizio 4.3 - soluzione

Blocchi di ampiezza 1 KB = 1024 B, quindi:

Segmento	Ampiezza	# Blocchi occupati
0	1536 B	2
1	20 B	1
2	256 B	1
3	1408 B	2
4	150 B	1
Totale	3370 B	7

Totale B × Blocchi = 1024 B × 7 = 7168 B
Totale spreco = (7168 – 3370) B = 3798 B
Spreco di memoria: 3798 B / 7168 B = **53%**

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*