Lezione 9

LA MEMORIA VIRTUALE

- Background
- Paginazione su richiesta
- Sostituzione delle pagine
- Algoritmi di sostituzione delle pagine
- Allocazione dei frame
- Thrashing

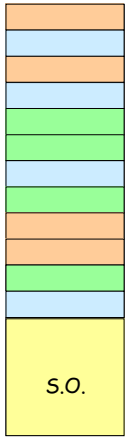
Background

- La memoria e' di solito gestita mediante tecniche di allocazione non contigua:
  - Paginazione
  - Segmentazione
  - Tecniche ibride
- Motivazione principale : Riduzione della frammentazione

Ci sono altri vantaggi ad usare
tecniche di allocazione non contigua?

Esempio:

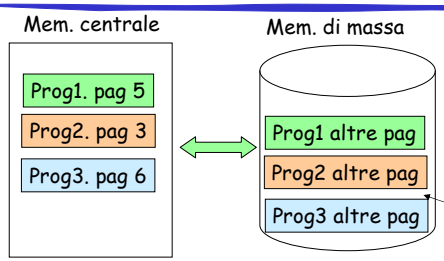
- Memoria 32 MB
- S.O. 8 MB
- 3 processi di 8 MB
- Pagine di 2 MB



Altri processi non possono utilizzare la memoria

Soluzione:
Tenere in memoria centrale solo la pagine correntemente in esecuzione di ogni processo, mentre le altre possono risiedere in memoria di massa

La memoria virtuale



La memoria virtuale e' lo spazio di indirizzamento effettivamente utilizzabile dall'utente

1. Lo spazio logico degli indirizzi e' più grande dello spazio fisico
2. Un maggior numero di processi puo' concorrere all'uso della CPU

Esempi:

Intel Pentium :

pagine da 2^{12} B = 4KB
indirizzamento a 32 bit
} 2^{20} pagine

memoria totale = 2^{32} B ~ 4GB

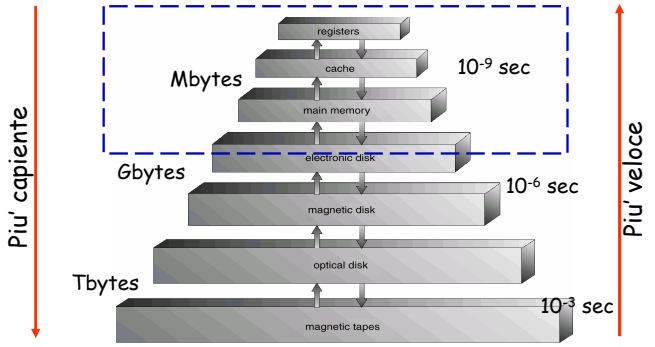
Spazio effettivamente indirizzabile (la memoria centrale puo' essere molto piu' piccola !!)

IBM PowerPC 970:

pagine da 2^{24} B = 16MB
indirizzamento a 64 bit
} 2^{40} pagine

memoria totale = 2^{64} B ~ 16×10^{18}B

Memoria gerarchica

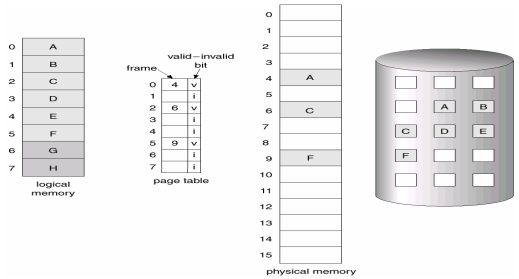


Direttamente accessibile dalla CPU

Se un riferimento alla memoria virtuale e' ad una pagina presente sul disco, la pagina deve essere portata in memoria centrale

Page fault

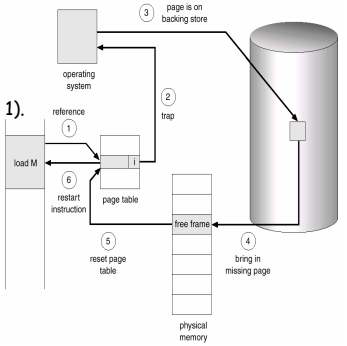
- Un bit di validità viene associato a ciascun elemento della tabella delle pagine (1 ⇒ in memoria, 0 ⇒ non in memoria).
- Inizialmente il bit di validità viene posto a 0 per tutte le pagine.
- Esempio



- In fase di traduzione degli indirizzi, se il bit di validità vale 0 ⇒ si ha un **page fault**.

Gestione del Page fault

1. Viene effettuato un riferimento ad una pagina,
2. Se la pagina non e' in memoria viene mandata una interruzione al SO che consulta una tabella per decidere se si tratta di…
   1. riferimento non valido ⇒ abort;
   2. pagina non in memoria.
3. Si accede alla pagina sul disco.
4. Si sposta la pagina in un frame libero
5. Si aggiorna la tabella (bit di validità = 1).
6. Viene riavviata l'istruzione che era stata interrotta.



Avvicendamento delle pagine

Non sempre e' presente in memoria un frame libero

3. Individuazione della pagina richiesta su disco.
4. Si sposta la pagina in un frame libero
   A. Se esiste un frame libero, viene utilizzato.
   B. Altrimenti viene utilizzato un algoritmo di sostituzione per selezionare un frame vittima.
   C. La pagina vittima viene scritta sul disco e la pagina richiesta viene letta nel frame appena liberato;
5. modifica delle tabelle delle pagine e dei frame.
6. Riavvio del processo utente.

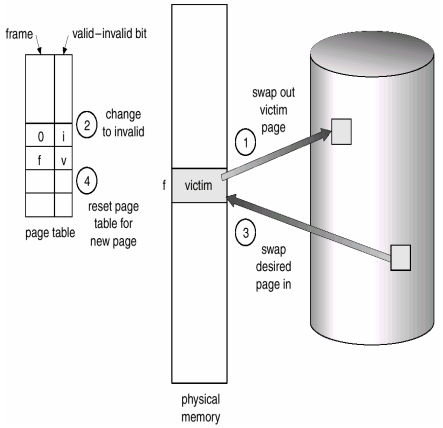
Avvicendamento delle pagine

C.1 La pagina vittima viene scritta sul disco

C.2 il bit di validita' della pagina vittima viene posto = 0

C.3 la pagina richiesta viene letta nel frame appena liberato

C.4 il bit di validita' della pagina richiesta viene posto = 1



Paginazione su richiesta

- porta una pagina in memoria solo quando è richiesta:
  - Vantaggi:
    - in memoria ci sono tutte e sole le pagine necessarie
    - Maggior grado di multiprogrammazione
  - Svantaggi
    - possibilita' di notevole overhead

Prestazioni della paginazione su richiesta

- *Page Fault Rate:* $0 \le p \le 1.0$ e' il rapporto tra riferimenti in memoria e page fault
  - se $p = 0$ non si hanno page fault;
  - se $p = 1$, ciascun riferimento è un fault.
- Tempo medio di accesso (EAT):

$$EAT = (1 - p) \times t[\text{accesso alla memoria}] + p \times t[\text{page fault}]$$

I/O su disco: t[swap out di pagina] + t[swap in di pagina] + overhead di restart

è richiesto un metodo che produca il minimo page fault rate.

Esempio di paginazione su richiesta

- Tempo di accesso alla memoria = 1 μsec
- Tempo di page fault = 10 $msec$ = 10000 μsec
- p=0.5 (il 50% dei riferimenti produce un page fault)

EAT = 0.5 x 1 + 0.5 x 10000 ~ 5000 (in μsec)

- Se un accesso su 1000 causa un page fault (p=0.001),
  EAT = 0.999 x 1 + 0.001 x 10000 ~ 11 μsec $\Rightarrow$
  l'accesso in memoria viene rallentato di un fattore 11.
- Se si desidera un rallentamento inferiore al 10%:
  EAT = (1-p) x 1 + p x 10000 < 1.1 μsec $\Rightarrow$
  $9999p < 0.1 \Rightarrow p < 0.00001\ldots$
  può essere permesso al piu'
  un page fault ogni 100000 accessi in memoria.

Come migliorare le prestazioni ?

- IDEA: Se una pagina selezionata come vittima non e' stata modificata, si puo' semplicemente riscriverci sopra, evitando lo swap-out e "dimezzando" l'I/O
- L'informazione sull'eventuale modifica di una pagina e' conservata in un bit associato alla pagina (dirty bit)
  - dirty bit = 0 pagina non modificata
  - dirty bit = 1 pagina modificata
- Al momento dell'avvicendamento della pagina, il S.O. esamina il dirty bit e decide se salvare la pagina vittima sul disco
- Per aumentare la probabilita' che il dirty bit = 0 al momento della sostituzione, il S.O. periodicamente
  - Copia su disco le pagine con dirty bit=1
  - Pone il d.b. =0

Un'alternativa: paginazione anticipata

- Il S.O. cerca di prevedere quali pagine saranno richieste dal processo in esecuzione e le carica prima di quando vengono effettivamente richieste
  - Vantaggi
    - Se il sistema e' capace di effettuare correttamente la previsione si evita l'overhead della paginazione su richiesta
  - Svantaggi
    - Difficile prevedere quali pagine serviranno al processo
    - Deve essere progettato con attenzione per evitare di portare in memoria pagine inutili al posto di quelle che effettivamente serviranno

Una possibile soluzione: approccio combinato

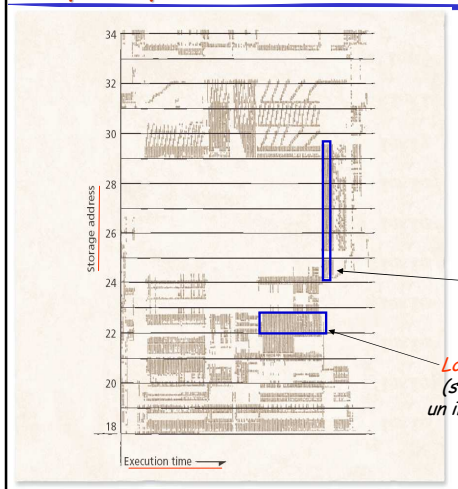
- Spesso si combina la paginazione anticipata con la paginazione su richiesta:
- ESEMPIO:
  - In caso di page fault, il s.o. Linux carica in memoria non solo la pagina mancante ma anche un piccolo gruppo di pagine adiacenti dello stesso processo
    - 4 pagine (16 KB) se la memoria < 16 MB
    - 8 pagine (32 KB) altrimenti
- Idea alla base di tale scelta: se viene referenziata una pagina, probabilmente in un prossimo futuro serviranno anche quelle adiacenti (principio della localita' spaziale)

Paginatore

- Il modulo del S.O. Incaricato dell'avvicendamento delle pagine e' chiamato PAGINATORE
- Esistono molti algoritmi alla base dei paginatori con l'obiettivo comune di minimizzare la frequenza di page fault.

- Obiettivo: Favorire i processi che esibiscono il principio di localita'
  - Localita' spaziale : processi che in ogni istante usano pagine vicine
  - Localita' temporale: processi che per un intervallo di tempo usano le stesse pagine

Il principio di localita'



Esempio di riferimenti in memoria di un programma in esecuzione

Localita' spaziale (pagine vicine allo stesso istante)

Localita' temporale (stesse pagine per un intervallo di tempo)

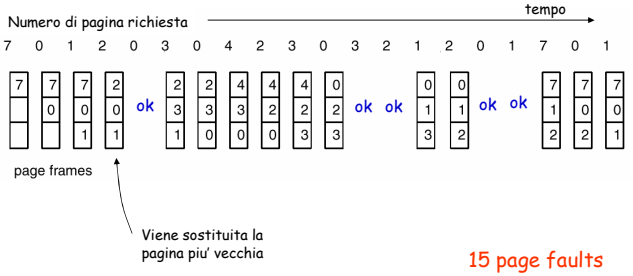
problema

Con che criterio avvicendare le pagine in memoria?

ALGORITMI DI AVVICENDAMENTO

Algoritmo First-In-First-Out (FIFO)

- Viene sostituita la pagina presente in memoria da piu' tempo
- Esempio: processo di 8 pagine (0,..,7)
- 3 frame (3 pagine per ciascun processo possono trovarsi in memoria contemporaneamente).



15 page faults

Algoritmo FIFO

- Idea di base: una pagina presente in memoria da molto tempo non verra' piu' refernziata
- Vantaggi:
  - Facile da implementare (coda FIFO)
  - Basso overhead
- Svantaggi
  - Rischio di sostituire pagine molto utilizzate e per questo presenti in memoria da molto tempo
  - Anomalia di Belady

Anomalia di Belady

- Si potrebbe pensare che con l'algoritmo FIFO aumentando il numero di frame si ha un minor numero di page faults
- Esempio: sequenza: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frame

1	1	1	4	4	4	5			5	5		
	2	2	2	1	1	1	ok	ok	3	3	ok	9 page fault
		3	3	3	2	2			2	4		

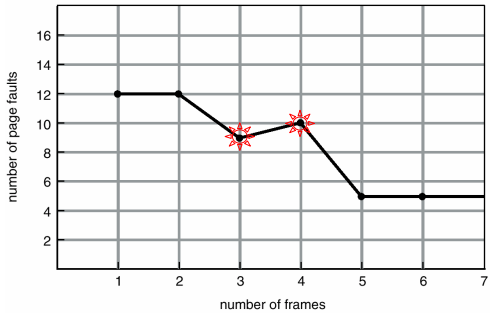
- 4 frame

1	1	1	1			5	5	5	5	4	4	
	2	2	2	ok	ok	2	1	1	1	1	5	10 page fault
		3	3			3	3	2	2	2	2	
			4			4	4	4	3	3	3	

più frame ⇒ più page fault

Anomalia di Belady

Richiesta alle pagine 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
al variare del numero di frame disponibili



Algoritmo ottimo (OPT)

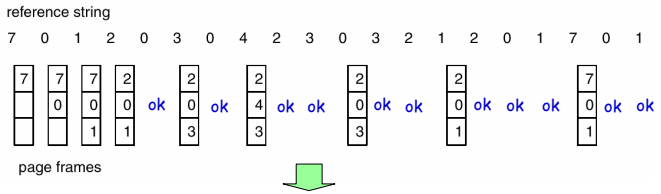
- Viene sostituita la pagina che non verrà usata per il periodo di tempo più lungo.
- Esempio: 4 frame, con stringa dei riferimenti
  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1	1	1	1			1			4		
	2	2	2	ok	ok	2	ok	ok ok	2	ok	6 page fault
		3	3			3			3		
			4			5			5		

- Si puo' dimostrare che minimizza il numero di page faults
- Problema: Come si può conoscere l'identità della pagina?
- Di solo interesse teorico: viene impiegato per misurare le prestazioni (comparative) degli algoritmi con valenza pratica
- Esempio: FIFO/OPT = 10/6 = 1.67 ➔ FIFO 67% piu' lento di OPT

Algoritmo ottimo

ESEMPIO



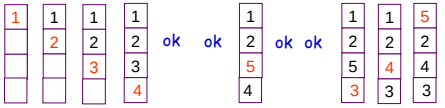
9 page faults

Con la stessa sequenza l'algoritmo FIFO produce 15 page faults

FIFO/OPT = 15/9 = 1.66 ➔ FIFO 66% piu' lento di OPT

Algoritmo Least-Recently-Used (LRU)

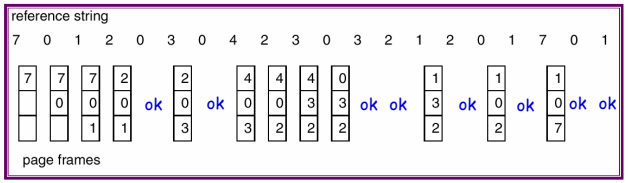
- Viene sostituita la pagina che non e' referenziata da piu' tempo
- Esempio: 4 frame con stringa di riferimenti
  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



8 page fault

- LRU/OPT = 8/6 = 1.33 ➔ LRU 33% piu' lento di OPT
- migliori prestazioni rispetto all'algoritmo FIFO ma con un maggiore overhead
- Favorisce i processi che esibiscono il principio di localita' temporale

ESEMPIO algoritmo LRU



12fault

LRU/OPT = 12/9 = 1.33 = LRU 33% piu' lento di OPT

Implementazione dell'algoritmo LRU

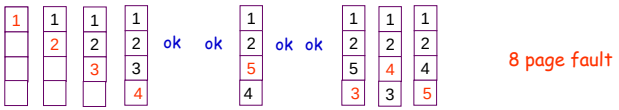
- Implementazione con registro
  - Ciascuna pagina ha un registro; ogni volta che si fa riferimento alla pagina si copia l'orologio nel registro.
  - Quando si deve rimuovere una pagina, si analizzano i registri per scegliere quale pagina cambiare.
  - Overhead per la ricerca della pagina "piu' vecchia" non refernziata
- Implementazione con lista :
  - Le pagine in memoria sono organizzate secondo una lista; ogni volta che si fa riferimento alla pagina, la si sposta in testa alla coda
  - Quando si deve rimuovere una pagina, si elimina la pagina in coda alla lista
  - Overhead per la gestione della lista.

Algoritmo LRU

- Idea di base: una pagina presente in memoria che non e' referenziata da molto tempo non verra' piu' refernziata
- Vantaggi:
  - Piu' efficiente dell'algoritmo FIFO
- Svantaggi
  - Difficile da implementare
  - Alto overhead

Algoritmo Least-Frequently-Used (LFU)

- Viene sostituita la pagina meno frequentemente utilizzata
- Esempio: 4 frame con stringa di riferimenti
  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



- LFU/OPT = 8/6 = 1.33 ➔ LFU 33% piu' lento di OPT
- migliori prestazioni rispetto all'algoritmo FIFO ma con un maggiore overhead

Algoritmo LFU

- Idea di base: una pagina presente in memoria che non e' molto referenziata sara' poco referenziata anche in futuro
- Vantaggi:
  - Piu' efficiente dell'algoritmo FIFO
- Svantaggi
  - Rischio di sostituire pagine da poco presenti in memoria e per questo poco referenziate rispetto ad altre (NO localita' temporale)
  - Alto overhead

Algoritmo Not recently used (NRU)

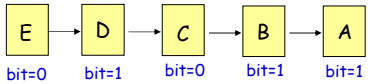
- Viene sostituita la pagina meno usata recentemente
- E' considerato una approssimazione dell'algoritmo LRU
- Divide le pagine in 3 classi
  a) Non referenziate e non modificate
  b) Referenziate e non modificate
  c) Referenziate e modificate
- viene usato l'algoritmo FIFO per ogni classe
- Si parte dalla classe a). Se e' vuota si esamina la classe b), se anch'essa e' vuota si esamina la c).
- Implementazione mediante due bit hw (o sw) di riferimento e di modifica

1a variante dell'algoritmo FIFO: seconda chance

Il principale difetto dell'algoritmo FIFO e' che sostituisce pagine presenti in memoria da molto tempo
e che potrebbero essere molto utilizzate

IDEA:
dare una seconda opportunita' alle pagine prima di essere rimosse

Se la pagina e' utilizzata la si lascia in memoria
- Implementazione con un bit di accesso

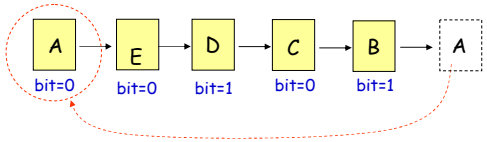


Coda FIFO

1a variante dell'algoritmo FIFO: seconda chance

Quando una pagina viene referenziata si pone bit=1

Quando le pagine raggiungono la testa della lista
- Se bit=1 si sposta la pagina in fondo alla lista e si pone bit=0
- Se bit=0 la pagina viene rimossa

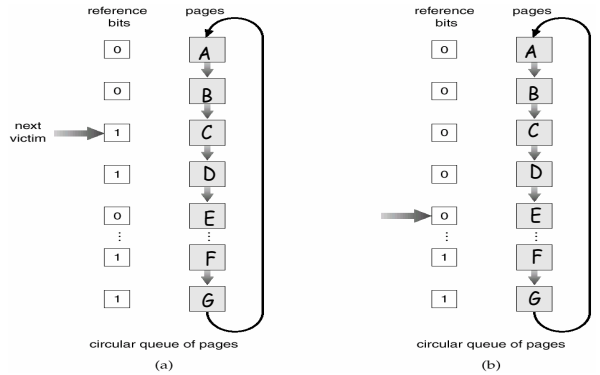


2a variante dell'algoritmo FIFO: clock algorithm

Effetto analogo all'algoritmo di seconda chance
Le pagine sono organizzate secondo una lista circolare
con un puntatore che le scorre

- Implementazione mediante un bit di accesso.
- Quando la pagina viene referenziata si pone bit=1
- Si scorrono le pagine presenti nella lista:
- Se bit=1 :
  - Si pone il bit di riferimento a 0.
  - Si lascia la pagina in memoria.
  - Si rimpiazza la pagina successiva (in ordine di clock), in base alle stesse regole.
- Se bit=0
  - Si rimpiazza la pagina

Esempio: clock algorithm



- pagina C candidata ad essere eliminata
- bit=1 ➔ C viene salvata (e si pone bit=0) e si esamina la pagina D
- bit=1 ➔ D viene salvata e si esamina la pagina E che viene eliminata

Allocazione dinamica di frame

- FIFO
- LRU / LFU
- NUR
- FIFO s.c.

Presuppongono costante il numero di frame allocati al processo (allocazione fissa)

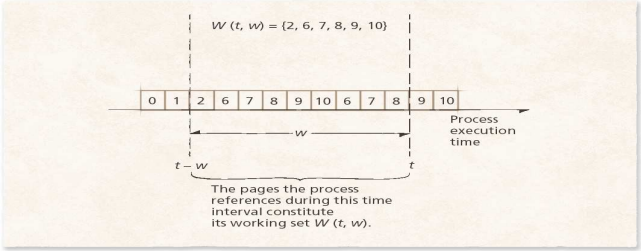
- Modello "working set"
- Avvicendamento "Page Fault Frequency (PFF)"

Numero di frame variabile
(allocazione variabile di frame)

Working set

Il modello basato su working set cerca di favorire i processi che realizzano il principio di localita' temporale

Il working set di un processo $W(t, w)$, e' l'insieme delle pagine referenziate dal processo nell'intervallo di tempo $[t - w, t]$



Dimensione del working set

- Il working set di un processo e' caratterizzato da
  - w (intervallo temporale di osservazione ➔ fissato)
  - t (istante di osservazione ➔ variabile)

insieme dinamico

page reference table

. . . 2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2 3 4 4 4 3 4 4 4 . . .

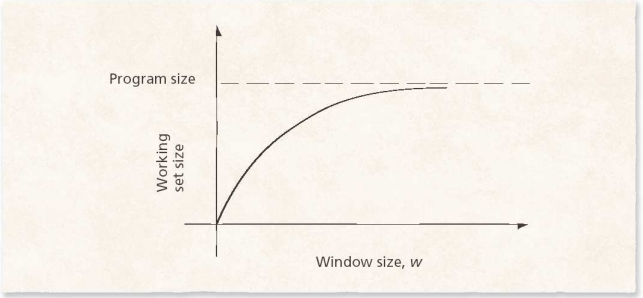
w w

t_1 t_2

$WS(t_1) = \{1,2,5,6,7\}$ $WS(t_2) = \{3,4\}$

Dimensione del working set

- La dimensione del working set cresce asinoticamente al crescere di w



Modello gestione dei w.s.

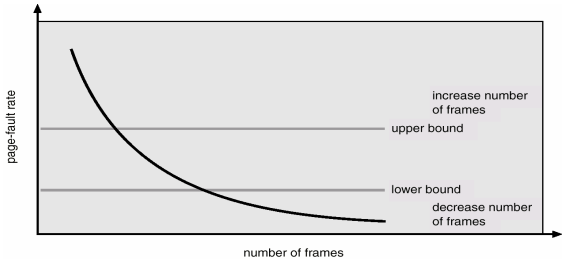
I metodi di avvicendamento basati sul modello del working set, allocano in memoria solo le pagine del processo appartenenti ai working set con w fissato

Problema: Dimensione di w

- w troppo piccolo: continuo avvicendamento di pagine utili (trashing)
- w troppo grande: rischio di tenere in memoria pagine poco utilizzate

Approccio alternativo

Basato sulla frequenza di page fault (Page Fault Frequency = PFF)



- Si stabilisce una frequenza di page fault "accettabile".
  - Se la frequenza effettiva è troppo bassa, il processo rilascia dei frame.
  - Se la frequenza è troppo elevata, il processo acquisisce nuovi frame.

w.s. vs PFF

- La strategia PFF ha alcuni vantaggi sul metodo basato su w.s. puro
  1. PFF determina le pagine da tenere in memoria solo all'occorrenza di un p.f., mentre il modello w.s. lo fa ad ogni riferimento ➔ minore overhead
  2. PFF si adatta dinamicamente all'andamento dell'utilizzo della memoria da parte del processo ➔ migliore utilizzo della memoria

Altre considerazioni

- Struttura dei programmi
  - Pagine di 4096 byte (4x1024)
  - A[ ][ ] = new int[1024][1024];
  - Ciascuna riga viene memorizzata in una pagina (1024 pagine)
  - Programma 1

```
for (j = 0; j < 1024; j++)
    for (i = 0; i < 1024; i++)
        A[i,j] = 0;
```

accesso per colonne
1024 x 1024 page fault

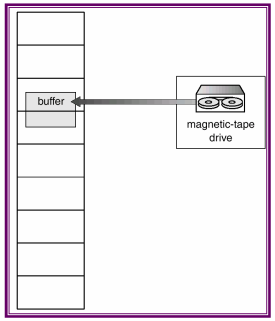
  - Programma 2

```
for (i = 0; i < 1024; i++)
    for (j = 0; j < 1024; j++)
        A[i,j] = 0;
```

Accesso per righe
1024 page fault

Altre considerazioni

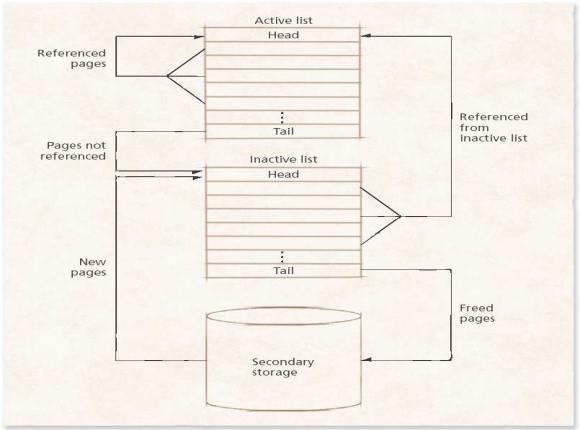
- **I/O Interlock** — Talvolta, occorre permettere ad alcune pagine di rimanere *bloccate* in memoria.
- Esempio: Le pagine utilizzate per copiare un file da un device di I/O devono essere bloccate, affinché non possano essere selezionate come vittime da un algoritmo di sostituzione.



Paginazione di Linux

- Linux usa una variante (a priorita') del clock algorithm per approsimare la strategia di avvicendamento Last Recently Used
- Il paginatore mantiene due linked list
  - Lista attiva
    - Contiene pagine molto utilizzate
    - Le pagine usate piu' recentemente sono vicine alla testa della lista attiva
  - Lista inattiva
    - Contiene pagine poco utilizzate
    - Le pagine meno usate sono vicine alla coda della lista
  - Vengono avvicendate solo le pagine nella lista inattiva

Paginazione di Linux



Paginazione di windows NT

- In Windows NT le assenze di pagina vengono gestite caricando in memoria non solo la pagina richiesta, ma più pagine ad essa adiacenti (*demand paging with clustering*).
- All'atto della creazione un processo riceve un valore di *minimo insieme di lavoro*, cioè di numero di pagine che il sistema assicura di poter caricare in memoria, fino ad un *massimo insieme di lavoro*, nel caso fossero insufficienti.
- Quando si verifica un page fault, il gestore controlla se è possibile allocare nuove pagine, o bisogna localmente rimuovere una pagina.
- Sugli 80x86 monoprocessori, per la sostituzione, si utilizza un algoritmo con seconda chance, su Alpha e multiprocessori 80x86 un algoritmo di tipo FIFO.

Paginazione di Solaris 2

- Il nucleo di Solaris2 assegna una pagina ad un thread ogni volta che si verifica un'assenza di pagina; se il numero di pagine libere scende sotto una determinata soglia (es. 1/64 della memoria fisica), inizia la paginazione.
  - Il processo paginatore **pageout**, responsabile della paginazione, implementa un algoritmo di sostituzione delle pagine ad orologio con bit di riferimento e modifica.
  - La frequenza delle scansioni è variabile e dipende dalla memoria disponibile
  - **pageout** controlla la memoria 4 volte al secondo: se la memoria disponibile scende sotto una certa soglia, aumenta la frequenza di scansione.
  - Se non si riesce a diminuire l'occupazione di memoria, il nucleo intraprende l'avvicendamento dei processi, selezionando quelli che sono rimasti inattivi per lunghi periodi.

Conclusioni

Algoritmo	commento
FIFO	Puo' avvicendare pagine importanti
OPT	Non implemnetabile
LRU	Eccellente, ma difficile da implementare e alto overhead
LFU / NRU	Approssimazioni di LRU
Seconda chanche	Buon miglioramento di FIFO
Clock	realistico
Working set	Alto overhead
PFF	Migliore del working set

Conclusioni

- E' utile poter eseguire processi il cui spazio di indirizzi logici sia maggiore dello spazio d'indirizzi fisici disponibili. La memoria virtuale è una tecnica che permette di mettere in corrispondenza uno spazio (ampio) di indirizzi logici con uno (piccolo) di indirizzi fisici, aumentando il grado di multiprogrammazione.
- La paginazione su richiesta si può usare per ridurre il numero di blocchi di memoria assegnati al processo.
- Può essere necessario sostituire pagine presenti in memoria allo scopo di liberare blocchi per nuove pagine, e per questo esistono vari algoritmi.
- Durante la scrittura dei programmi di utente e' possibile intervenire sul numero di page fault