

Algoritmi di Sostituzione

- Se non ci sono pagine libere il page fault richiede una scelta su quale pagina deve essere rimossa
 - Libera memoria per la pagina da caricare
- Pagine modificate devono essere salvate
 - Quelle non modificate vengono semplicemente sovrascritte
- Deve evitare di selezionare una pagina riferita spesso
 - Potrebbe essere necessario ricaricarla in breve tempo
- Deve minimizzare il numero di page fault

↳ questo è buono se calcola da
% dei page fault

1

Algoritmo di Sostituzione Ottimo

- Sostituisce la pagina che sarà riferita nell'istante più lontano nel tempo
 - Ottimo ma non realizzabile
 - Si potrebbe stimare l'ordine di caricamento delle pagine in esecuzioni precedenti del processo (neanche questa soluzione è applicabile in pratica)
 - L'algoritmo può essere utilizzato per valutare le prestazioni di altri algoritmi di sostituzione

2

Algoritmo di Sostituzione Ottimo: esempio

- Esempio: 4 frame disponibili (nessuna pagina precaricata)
pagine richieste: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1	4
2	
3	
4	5

6 page fault

Algoritmo appena dell'ott. Lo posso usare perché conosco sequenza di richieste da pagine logiche che vedo + ocfwnt.
Ho 4 frame

- Algoritmo FIFO: si sostituisce la pagina più vecchia

1	1	5	4
2	2	1	5
3	3	2	
4	4	3	

10 page fault

3

* Inizialmente nessuna pagina era in memoria quindi 6 page fault.

ALGORITMO DI SOSTITUZIONE

Not Recently Used

Si utilizzano due bit per ogni frame:

R = bit di riferimento (diventa 1 quando la pagina viene letta/scritta)

M = bit di modifica (diventa 1 quando la pagina viene scritta)

R	M	
0	0	Classe 0
0	1	Classe 1
1	0	Classe 2
1	1	Classe 3

Mpfa per alg. di sostituzione vengono salvate nella tabella delle pagine.
Bit di modifica e bit di riferimento

Soltanto solo le pagine presenti in memoria

R e M vengono impostati dall'hardware e azzerati dal sistema op.

R viene periodicamente azzerato dal sistema operativo (per es. ogni k timer interrupt).

Quando si verifica page fault, se c'è bisogno di una sostituzione, l'algoritmo sceglie in ordine crescente di classe (prima i frame in classe 0, poi in classe 1, ecc.).

4

bit si diventa 1 quando p. logica viene riferita. Una volta messo a 1, viene azzerato completamente
↳ Se è a 1 significa che pagina è stata riferita molto recentemente
Posso classificare pagine in 4 classi diverse.
NOTA: Critico è molto grossolano, perché le pg. di classe 0 possono avere tempi di futura uso molto diversi

SO dovranno avere un modo hardware riservato per registrare il tempo di accesso a pagina. Non posso permettermi di fare operazioni di scrittura a meno che non ha hardware dedicato, altrimenti è troppo costoso.

↑
pagina usata molto di recente

Least Recently Used (LRU)

- Assume che le pagine usate di recente siano riferite di nuovo in breve tempo (principio di località temporale)
 - Scarica le pagine inutilizzate da più tempo
- Implementazione diretta: mantiene una lista di pagine
 - Le pagine usate più di recente in cima
 - Aggiorna la lista ad ogni riferimento della memoria (troppo costoso)!!
- Impl. hardware: mantiene un contatore per ogni record della tabella delle pagine
 - L'hw incrementa il contatore centrale C ad ogni istruzione
 - Se accedo alla pagina p , C viene copiato nel record corrispondente
 - Scarica la pagina fisica con il più piccolo valore nel campo contatore

5

Mettendo da 0 a 1 che un HW che fa? Immagina di avere 4 pagine in memoria. Se accedo a pagina molto tutti 1 nella riga della pagina riferita e tutti 0 sulle

Colonna delle pagine

1. Scelgo valore più basso sulla riga, perché ho tutti 0 se questa è stata usata poco. Mettore tutti 1 = dare il valore massimo. Mettere tutti 0 su colonna + significato togliere 8 a tutti.

Pagina riferita per ultima va in cima alla classifica e poi tolgo lo stesso valore e tutti quanti. Fisso valore max e setto a 0. Pagina in fondo non è mai uscita

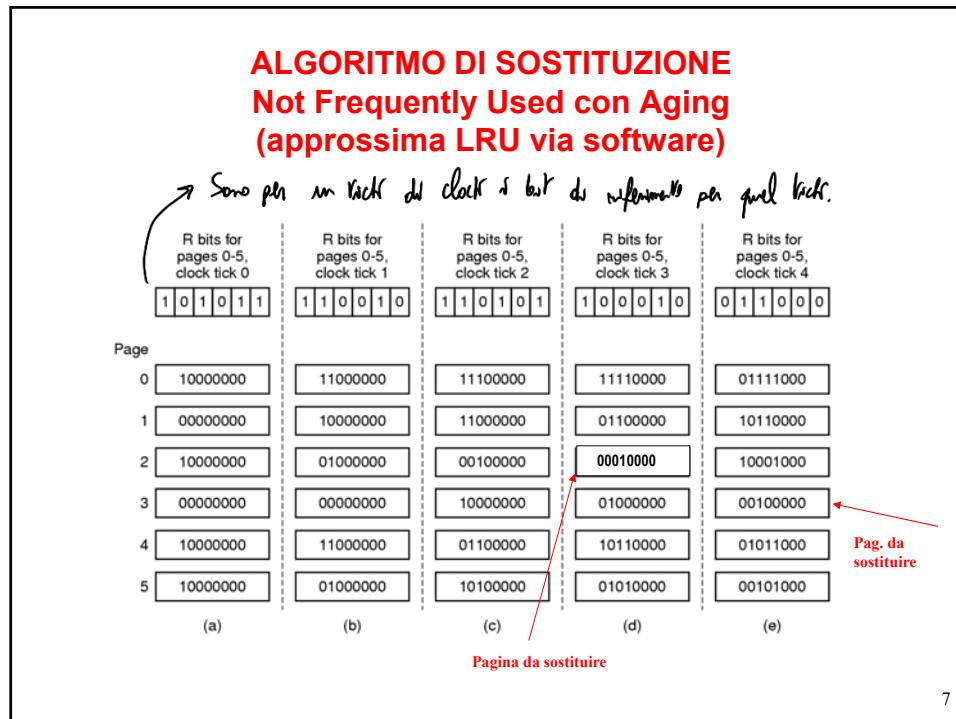
LRU in hardware

	Page 0 1 2 3				
0	0 1 1 1	0 0 1 1	0 0 0 1	0 0 0 0	0 0 0 0
1	0 0 0 0	1 0 1 1	1 0 0 1	1 0 0 0	1 0 0 0
2	0 0 0 0	0 0 0 0	1 1 0 1	1 1 0 0	1 1 0 1
3	0 0 0 0	0 0 0 0	0 0 0 0	1 1 1 0	1 1 0 0
(a)					
	Page 0 1 2 3				
0	0 0 0 0	0 1 1 1	0 1 1 0	0 1 0 0	0 1 0 0
1	1 0 1 1	0 0 1 1	0 0 1 0	0 0 0 0	0 0 0 0
2	1 0 0 1	0 0 0 1	0 0 0 0	1 1 0 1	1 1 0 0
3	1 0 0 0	0 0 0 0	1 1 1 0	1 1 0 0	1 1 1 0
(b)					
	Page 0 1 2 3				
0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
1	1 0 0 0	0 0 0 0	0 0 0 0	1 1 0 0	1 1 0 0
2	1 0 0 0	0 0 0 0	1 1 0 0	1 1 0 0	1 1 1 0
3	1 0 0 0	0 0 0 0	1 1 1 0	1 1 0 0	1 1 1 0
(c)					
	Page 0 1 2 3				
0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
1	1 0 0 0	0 0 0 0	0 0 0 0	1 1 0 0	1 1 0 0
2	1 0 0 0	0 0 0 0	1 1 0 0	1 1 0 0	1 1 1 0
3	1 0 0 0	0 0 0 0	1 1 1 0	1 1 0 0	1 1 1 0
(d)					
	Page 0 1 2 3				
0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
1	1 0 0 0	0 0 0 0	0 0 0 0	1 1 0 0	1 1 0 0
2	1 0 0 0	0 0 0 0	1 1 0 0	1 1 0 0	1 1 1 0
3	1 0 0 0	0 0 0 0	1 1 1 0	1 1 0 0	1 1 1 0
(e)					
(f)					
	Page 0 1 2 3				
0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
1	1 0 0 0	0 0 0 0	0 0 0 0	1 1 0 0	1 1 0 0
2	1 0 0 0	0 0 0 0	1 1 0 0	1 1 0 0	1 1 1 0
3	1 0 0 0	0 0 0 0	1 1 1 0	1 1 0 0	1 1 1 0
(g)					
	Page 0 1 2 3				
0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
1	1 0 0 0	0 0 0 0	0 0 0 0	1 1 0 0	1 1 0 0
2	1 0 0 0	0 0 0 0	1 1 0 0	1 1 0 0	1 1 1 0
3	1 0 0 0	0 0 0 0	1 1 1 0	1 1 0 0	1 1 1 0
(h)					
	Page 0 1 2 3				
0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
1	1 0 0 0	0 0 0 0	0 0 0 0	1 1 0 0	1 1 0 0
2	1 0 0 0	0 0 0 0	1 1 0 0	1 1 0 0	1 1 1 0
3	1 0 0 0	0 0 0 0	1 1 1 0	1 1 0 0	1 1 1 0
(i)					
	Page 0 1 2 3				
0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
1	1 0 0 0	0 0 0 0	0 0 0 0	1 1 0 0	1 1 0 0
2	1 0 0 0	0 0 0 0	1 1 0 0	1 1 0 0	1 1 1 0
3	1 0 0 0	0 0 0 0	1 1 1 0	1 1 0 0	1 1 1 0
(j)					

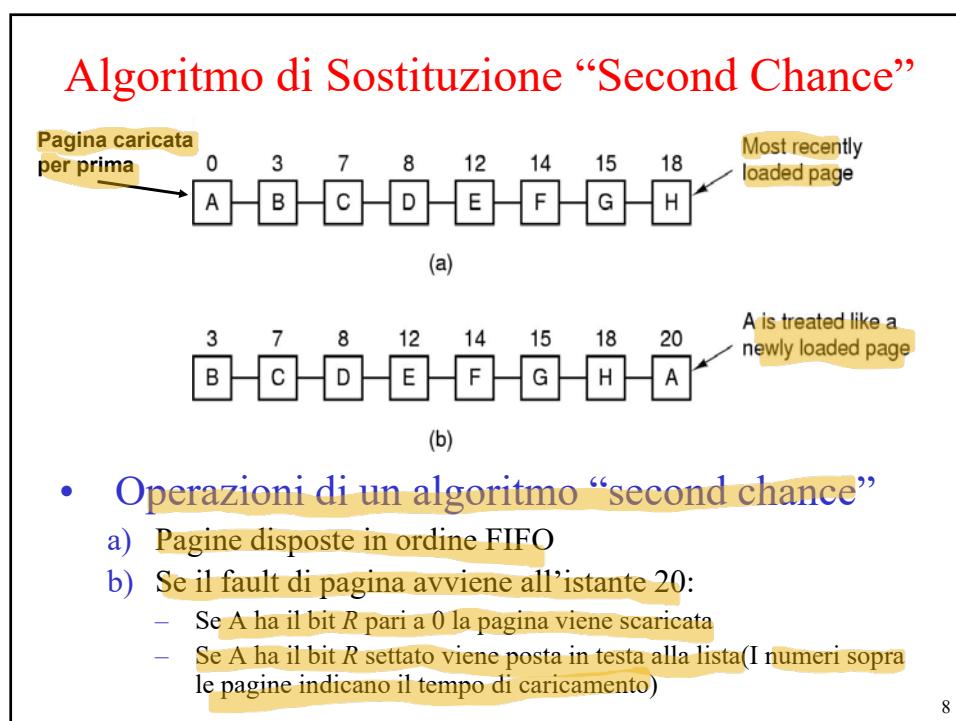
LRU using a matrix – pages referenced in order
0,1,2,3,2,1,0,3,2,3

6

Qui bit 0 vengono memorizzati
in registo a scorrimento. Ogni
pagina ha registrato il suo bit.
Ogni volta che ho



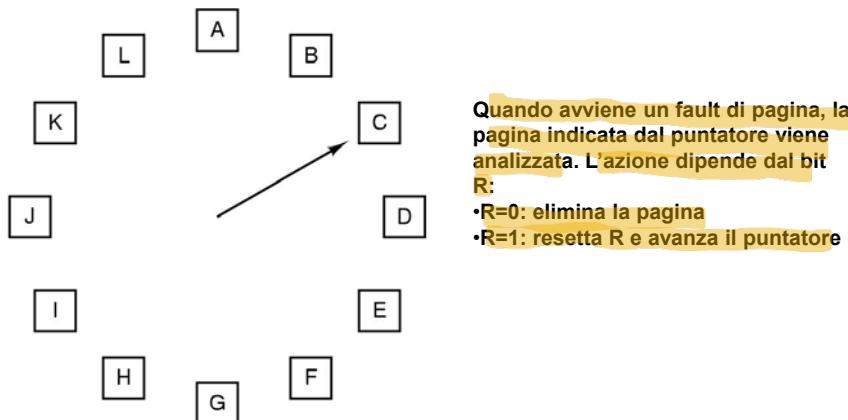
Memorizzo tutti i bit precedenti in un registro a scorrimento. Bit meno significativo (sono più significativi) quindi vengono presi.



Criterio di vecchiaia potrebbe andare bene con accorgimento su: immagina che so segno l’tempo di caricamento. Ho lista concatenata di pagine ordinata per tempo di caricamento. Criterio: se pagina in coda alla lista ha bit R pari a 1 viene rimossa dalla coda. È come se rimanessero l’tempo di accesso.

Potrei avere più pagine messe in coda alla lista per ogni page fault.

Algoritmo di Sostituzione "Clock"



9

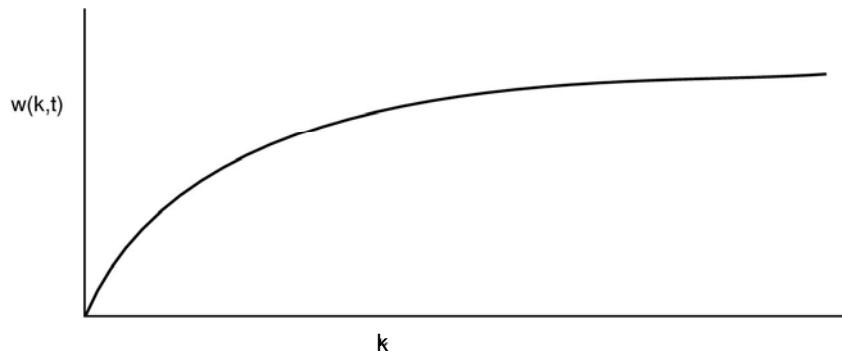
FORMALMENTE: k =istante di tempo (fisico distinto e processore, tempo attribuito a guardare gli ultimi k accessi fatti da un processo). Al crescere di K si è temporisticamente. $w(k,t)$ =numero di pag. diverse che processo ha riferito negli ultimi k accessi.

$w(1,t)=1$, $w(2,t)=1$ oppure 2. Grapho dice che saltano da mille quando tornano all'inizio, ma per grapho si appella se.

Un numero di pagine usato è pari ad assalto sortiti dalla curva.

Suppose che ogni processo userà 10 pagine, si può fare dimostrato che processo affitta sempre il 10 pag. del working set, che possono cambiare, ma i numeri sono sempre le stesse. Punto dunque è capire quale pagina servirà più al processo e anche capire il numero di pagine per working set.

Working Set



- Il "working set" è l'insieme di pagine riferite negli ultimi k accessi in memoria
- $w(k,t)$ è la dimensione del "working set" al tempo t
- Idealmente bisognerebbe che in ogni momento in memoria vi fosse l'intero working set (WS) di ogni processo

10

1. Princípio da località: i processi vendono a rispettare questo principio: se hanno fatto un accesso a una zona di memoria è probabile che rimangano intorno a quella zona durante l'esecuzione (vale sia per codice che per dati).

WORKING SET DI UN PROCESSO: insieme delle pagine che processo sta usando in quel momento.

Devo fare in modo che ogni processo abbia un numero di pagine corrispondenti al suo working set.

Il working set può cambiare, così come il numero di pagine. Algoritmo dovrebbe togliere la pagina che non fa/farà uso di una pagina del working set.

IL PROBLEMA DEL THRASHING

L'overhead (in termini di tempo) introdotto dalla paginazione può diventare enorme in determinate situazioni: questo fenomeno è conosciuto con il nome di "thrashing".

La causa del thrashing può essere la presenza contemporanea in memoria di n processi tali che la somma delle dimensioni dei loro working set eccede la dimensione totale della RAM disponibile.

I "sintomi" di questo fenomeno sono un altissimo page fault rate, bassa utilizzazione della CPU, alta utilizzazione del disco.

Gli "effetti" sono di rallentare l'esecuzione di tutti i processi, dato che il sistema spende gran parte del tempo a gestire i page fault.

11

Il modello "Working Set"

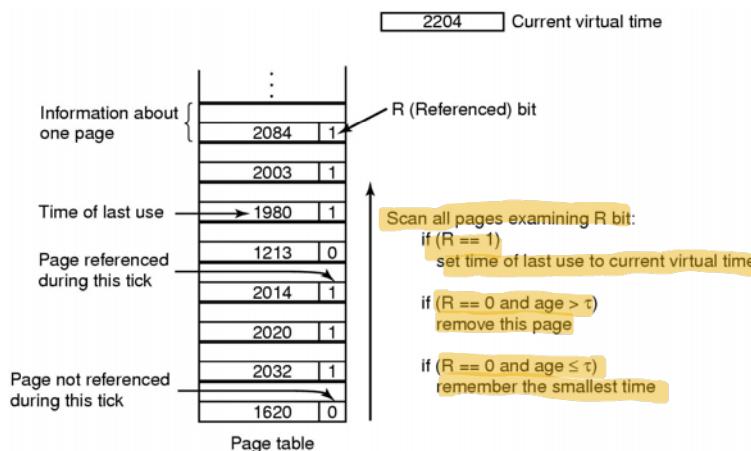
- Paginazione su domanda
 - un processo passa in esecuzione senza alcuna pagina in memoria
 - le pagine vengono caricate quando avviene un page fault
 - lento finché non è stato caricato il working set
- Pre-paginazione (prepaging) \Rightarrow *Propriamente appello che le carico, quindi 10 page fault.*
 - il sistema cerca di tenere traccia del working set in cache all'page
 - l'ultimo working set viene caricato in memoria prima di riavviare il processo
 - Là forse tendo a scommettere su quale pagina mi servono.*

12

Dove capire ninfo su ultimo utilizzo. Quanto tempo è passato. Dovendo fare delle assunzioni:
fissa costante τ che rappresenta soglia oltre quale page non fa più parte del working set.
Scelta di τ è delicata assai molto. Causale è mettere pagina che ha età maggiore di τ .
Con algoritmo di orologio modificante, aggiornando tempo utilizzo vivere un'alternativa.

28/04/2017

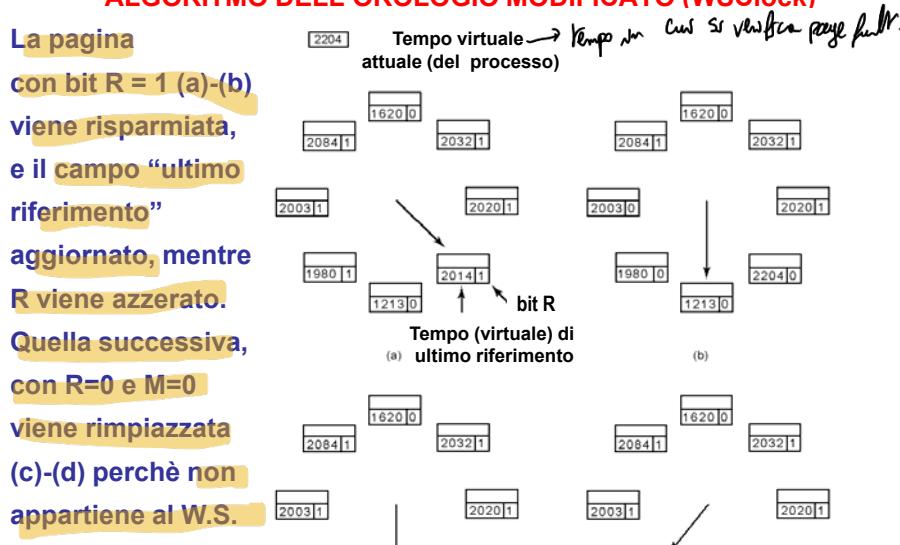
Algoritmo di Sostituzione basato sul “Working Set”



L'algoritmo cerca di eliminare pagine che hanno alta probabilità di non appartenere al WS

13

ALGORITMO DELL'OROLOGIO MODIFICATO (WSclock)



14

Succo lista concorrente partendo dalla lista delle disk. Obiettivo trovare $\text{eta} > \tau$. Progr. pagina ha bit R, M e tempo di ultimo utilizzo.

- 1^a pagina ha bit $R=1$ rispetto a ultimo aggiornamento. Lasciala passare avanti, ma valuta prima posizionato al tempo di ultimo utilizzo a tempo corrente. Bit $R=1 \Rightarrow$ puoi avanti. Se $r=0$, allora guarda alla pagina: (t_{age} - t_{ultimo}) e vedere se è suff. grande. Se $M=0$ e età è alta, mi fermo. Vai alla solle da sostituire. τ

7

* R=1 vede orario. R=0, M=0, età > P va bene, R=0, M=1, età > P vede avanti perché non regola perdo tempo.
 Però intanto programma operazione di scrittura su disco. Se fatto giro completo, riprende il giro, visto 28/04/2017
 che ho schedulato scrittura ho paura con M=0 ora e la scelgo. Se non trova età > P, toglie pugna più vecchia.
 Se R=0, ma età < P, continua a scrivere.
 Se R=1, ma età > P aggiunge tempo all'ultimo accesso.

ALGORITMO DELL'OROLOGIO MODIFICATO (WSclock)

- Se la pagina individuata come non compresa nel WS è stata modificata ($M=1$), non può essere rilasciata ma va prima copiata sul disco. Per evitare un cambio di processo, in questo caso viene schedulata la scrittura e la lancetta avanza
 - Viene cercata una pagina fuori dal WS e non modificata
 - Se la lancetta torna al punto di partenza, vi sono due casi:
 - È stata schedulata almeno una scrittura: la lancetta continua a girare fino a quando la prima scrittura viene completata (trovo una pagina con $M=0$)
 - Non è stata schedulata nessuna scrittura: tutte le pagine fanno parte del WS. Viene scelta una pagina non modificata o se sono tutte con $M=1$, quella corrente.

15

Politiche di Allocazione Locali e Globali

	Age	Political locale
A0	10	AC
A1	7	A1
A2	5	A2
A3	4	A3
A4	6	A4
A5	3	A5
B0	9	BC
B1	4	B1
B2	6	B2
B3	2	B3
B4	5	B4
B5	6	B5
B6	12	B6
C1	3	C1
C2	5	C2
C3	6	C3

1

In un sistema multiprogrammato si possono usare due approcci al sostituzione:

- **sostituzione locale**: ad ogni processo viene assegnato un numero di frame definito, quando P1 causa un page fault la vittima per il sostituzione può essere scelta solo tra le pagine di P1;

- sostituzione globale: la vittima per la sostituzione può essere scelta fra tutte le pagine in memoria.

7) offinalezzazione che non prende in considerazione tutte le possibilità.

10

Riassunto degli Algoritmi di Sostituzione

Algoritmo	Commento
Ottimo	Non implementabile, si usa come benchmark
NRU (Not Recently Used)	Molto rozzo
LRU (Least Recently Used)	Eccellente ma va implementato in hardware
Aging	Approssima bene in software LRU
Second Chance	Costoso da realizzare
Clock	Meno costoso del precedente
Working Set	Costoso da implementare
WSClock	Buono ed efficiente

17

Poco effettuare misurazioni prendendo process con comportamento diverso e li testa in base all'algoritmo ottimo. NOTA: dato un determinato benchmark posso avere comportamenti diversi degli algoritmi. Dopo trovare compromesso tra costo, uso hardware ecc. e efficienza.

NOTA: LIMITI PAGINAZIONE: pagine possono essere protette solo singolarmente (pero' suddivisione logica fulla non funziona: area resto, area dati ecc... Però suddivisione logica).

Dividiamo sp. indirizzabile in segmenti di lunghezza diversa, suddiviso logicamente spazio di indirizzamento. Vogliamo però ottenere pezzi in segmenti di dimensioni diverse se devo fare allocazioni consecutive.