10 - Memoria Virtuale

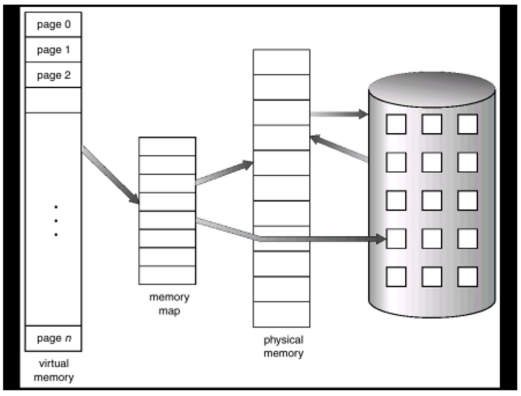
giovedì 5 maggio 2022 13:07

•Fino ad ora abbiamo detto che l'intero programma deve essere caricato in memoria per essere eseguito, ma **questo non è sempre fattibile e/o necessario**
-Molti programmi moderni pesano molti più GB di quanto la RAM sia fisicamente in grado di contenere
-Un programma pesa di più dentro la RAM perché diventando un processo verranno aggiunte cose come la stack

•Questo problema è stato risolto tramite l'introduzione della memoria virtuale
•Memoria Virtuale: RAM + backing store (disco)
•Può essere implementata tramite:
-**Paginazione su domanda** (demand paging)
-**Segmentazione su domanda** (demand segmentation)

PAGINAZIONE SU DOMANDA

•Sostanzialmente mi permette di eseguire programmi più grandi della RAM **caricando dentro quest'ultima SOLO una parte delle pagine**
-le pagine restanti vengono caricate su disco (backing store) e prese tramite swapping



PAGINE DENTRO LA RAM

•Per capire se una pagina è dentro la ram o su disco utilizzo i bit di validità
-**1:** pagina in memoria
-**0:** pagina NON in memoria
•Inizialmente sono tutti 0
•Se cercando una pagina trovo un bit a 0, ho un page fault

Frame #	valid bit
	1
	1
	1
	1
	0
...	
	0
	0

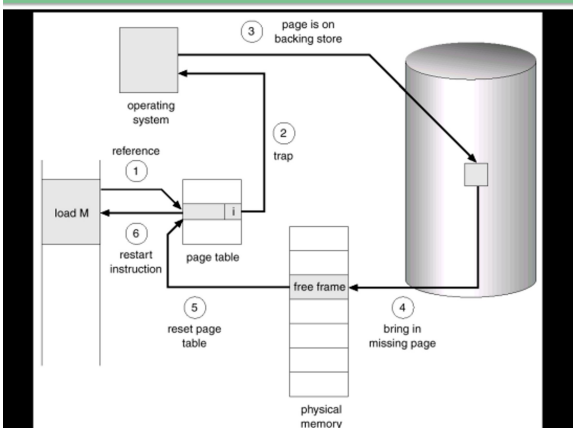
page table

PAGE FAULT

•Un page fault causa un interrupt al SO che avviene in queste 5 fasi:
1)SO verifica una tabella: se trova il bit a 1 carica la pagina, altrimenti passa allo step 2
2)Cerca un frame vuoto nella RAM
3)Swap della pagina da disco al frame
4)Modifica il bit di validità a 1 perché ora la pagina è in memoria
5)Ripristina l'istruzione che ha causato il page fault

NOTA: il primo accesso in memoria di un programma risulta sempre in page fault (perché la tabella è ancora vuota)

Gestione dei page fault

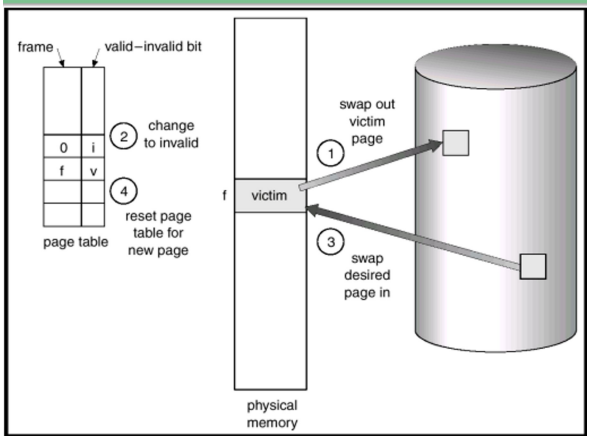


- La paginazione su domanda influenza il tempo di accesso effettivo alla memoria (Effective Access Time, **EAT**)
  - la sua velocità dipende da quanti page fault avvengono
- Il tempo di page fault è dato da:
  - **servizio dell'interrupt**
  - **swap in** (può essere rallentato se la ram è piena e bisogna fare swap out per liberare un frame)
  - **costo di riavvio del processo**

RIMPIAZZARE PAGINE

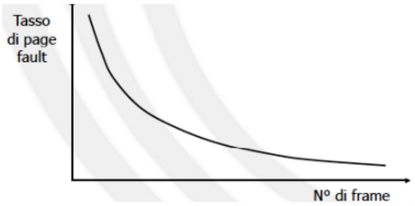
- Nel caso la memoria sia piena (non ci sono pagine libere), è necessario fare uno **swap out** su disco delle pagine
  - le pagine non vengono mai cancellate ma sovrascritte
- In assenza di frame liberi, sono necessari **2 accessi alla memoria**:
  - uno per lo swap out della "vittima"
  - uno per lo swap in del frame da caricare
- Questo raddoppia il tempo di page fault
- Per ottimizzare le prestazioni si può usare un bit di modifica (dirty bit)
  - **1:** pagina modificata
  - **2:** pagina NON modificata (non va spostata su disco)

Rimpiazzamento delle pagine



ALGORITMI DI RIMPIAZZAMENTO DELLE PAGINE

- Questi algoritmi hanno il compito di minimizzare il numero di page fault



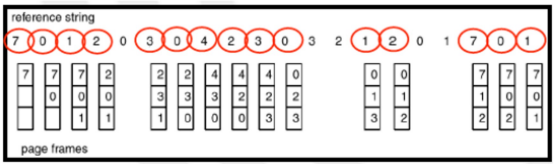
NOTA: All'aumentare dei frame ci sono meno page fault

ALGORITMO FIFO (FIRST-IN-FIRST-OUT)

- La prima pagina introdotta è anche la prima ad essere tolta in caso di swap out
- Problemi:
  - non viene valutata l'importanza delle pagine che vengono rimosse
  - soffre dell'anomalia di Belady: aumentano i frame possono esserci più page fault

Algoritmo FIFO - esempio

- Consideriamo una memoria con 3 frame
- Usando FIFO si hanno 15 page fault



Anomalia di Belady - esempio

- Reference string

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

- Con 3 frame 9 page fault

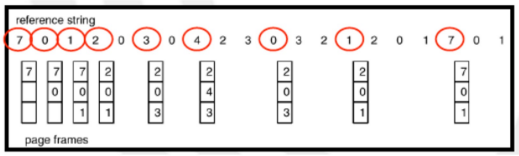
1	1	1	4	4	4	5	5	5	5	5	5
	2	2	2	1	1	1	1	1	3	3	3
		3	3	3	2	2	2	2	2	4	4

ALGORITMO IDEALE

- Rimpiazza le pagine che non saranno usate per il periodo di tempo più lungo
- Questo algoritmo non esiste
  - non può essere implementato perché non sappiamo quali pagine non verranno usate per più temp

Algoritmo ideale - esempio

- Con l'algoritmo ideale abbiamo 9 page fault



ALGORITMO LRU (LEAST RECENTLY USED)

- È **l'algoritmo migliore** che si possa implementare
- Viene rimpiazzata la pagina che non viene usata da più tempo
- Problemi:
  - risulta difficile da implementare
  - può richiedere hardware apposito

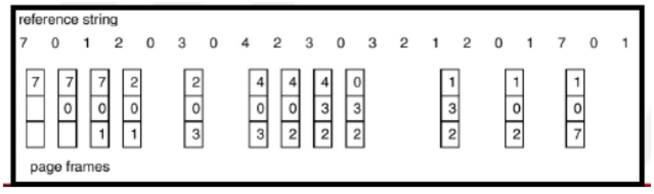
Algoritmo LRU – (Least Recently Used)

- Approssimazione dell'algoritmo ottimo
  - Usare il passato recente come previsione del futuro
  - Si rimpiazza la pagina che non viene usata da più tempo (*)
- Esempio: 4 frame → 8 page fault

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

Algoritmo LRU - esempio

- Con l'algoritmo LRU abbiamo 12 page fault
  - Migliore del FIFO
  - Peggiore dell'ideale

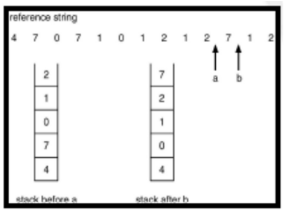


LRU - IMPLEMENTAZIONE TRAMITE CONTATORE

- Ogni pagina ha un contatore che indica il tempo dell'ultimo accesso (è un singolo numero)
- La pagina con il tempo di accesso più vecchio viene sostituita

LRU - IMPLEMENTAZIONE TRAMITE STACK

- Le pagine sono inserite in uno stack
- Quando una pagina viene referenziata, questa viene messa in cima
  - le pagine referenziate meno volte rimangono in fondo allo stack



Modifica stack o copia del tempo di sistema richiede supporto HW!

LRU - IMPLEMENTAZIONE TRAMITE BIT DI REFERENCE

- Ogni pagina ha un bit di reference che indica se la pagina è stata referenziata o no
  - **bit a 0:** non referenziata
  - **bit a 1:** referenziata
- Si possono implementare più bit di reference per ogni pagina
- I bit possono essere aggiornati periodicamente (es. ogni 100ms)

LRU - APPROSSIMAZIONI

- **Algoritmo LFU (Least Frequently Used):** rimpiazza la pagina che è stata utilizzata meno
- **Algoritmo MFU (Most Frequently Used):** rimpiazza la pagina che è stata utilizzata di più

ALLOCAZIONE DEI FRAME

- Per rendere efficiente la memoria virtuale, è necessario decidere **quanti frame allocare per ogni processo**
  - ogni processo ha bisogno di un numero minimo di pagine per essere eseguito
- In caso di page fault, i frame vittime possono essere scelte:
  - **localmente:** vengono rimpiazzati SOLO frame del processo che ha causato il page fault
  - **globalmente:** vengono rimpiazzati frame di un qualunque processo
- I frame possono essere allocati in 2 modi:
  - **allocazione fissa:** il processo ha sempre lo stesso numero di frame
  - **allocazione variabile:** il processo ha un numero di frame che può variare durante l'esecuzione

ALLOCAZIONE FISSA

- Dati M frame ed N processi, **vengono allocati ad ogni processo M/N frame**
- Semplice da gestire
- Problema: poco flessibile

ALLOCAZIONE VARIABILE

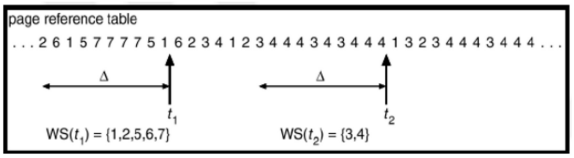
- Il numero di frame che alloco per un processo può **variare costantemente** in base alle esigenze
- Può essere implementata in 2 modi:
  - working set
  - page fault frequency

WORKING SET

- Il working set si basa sulla località: pagine referenziate in un determinato tempo
  - non possiamo mai saperla, ma possiamo avvicinarci attraverso delle approssimazioni calcolati con i riferimenti passati
- In base al working set varia il numero di frame allocati

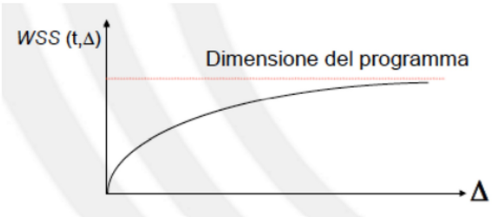
Modello del working set

- frame sufficiente a mantenere in memoria il suo working set WS_i (t,**Δ**)
  - Numero di pagine referenziate nell'intervallo di tempo [t-**Δ**, t] più recente (finestra del working set)
    - se **Δ** piccolo = poco significativo
    - se **Δ** troppo grande = può coprire varie località
    - **Δ** = ∞ ⇒ tutto il programma



Modello del working set

- WSS_i (t,**Δ**) = dimensione di WS_i (t,**Δ**) in funzione del tempo

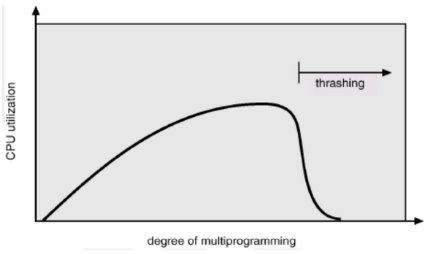


- Per misurare il working set vengono utilizzati **timer e bit di reference**
  - un timer interrompe periodicamente la CPU
  - ad ogni interruzione vengono scandite delle pagine (se non hanno il bit a 1 queste vengono scartate)
- L'utilizzo del working set può portare al thrashing

THRASHING

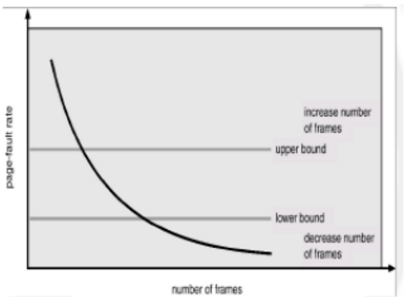
- Thrashing: fenomeno dovuto a un processo che non ha abbastanza frame e genera più page fault
- Porta ad un declino costante dell'utilizzo della CPU
- I nuovi processi "rubano" frame ai vecchi processi **aumentando il numero di page fault**

Diagramma del trashing



PAGE FAULT FREQUENCY

- Si basa sul presupposto che deve evitare il thrashing
- Viene stabilito un **tasso di page fault** "accettabile" che non deve essere superato
  - se un processo supera questo tasso riceve più frame
  - se un processo ha un tasso troppo basso, gli vengono tolti frame



NOTE FINALI - FRAME LOCKING

- Alcuni frame non devono essere MAI rimpiazzati
  - frame corrispondenti a pagine del kernel
  - frame corrispondenti a pagine usate per trasferire dati da/verso I/O