Università di Udine — Facoltà di Scienze MM.FF.NN.

Laurea in Informatica — A.A. 2003/04

Trasparenze del Corso di *Sistemi Operativi*

Marino Miculan

Università di Udine

Memoria Virtuale

Memoria virtuale: separazione della memoria logica vista dall'utente/programmatore dalla memoria fisica

Solo *parte* del programma e dei dati devono stare in memoria affinché il processo possa essere eseguito (*resident set*)

Memoria Virtuale: perché

Molti vantaggi sia per gli utenti che per il sistema

- Lo spazio logico può essere molto più grande di quello fisico.
- Meno consumo di memoria ⇒ più processi in esecuzione ⇒ maggiore multiprogrammazione
- Meno I/O per caricare i programmi

Porta alla necessità di caricare e salvare parti di memoria dei processi da/per il disco al runtime.

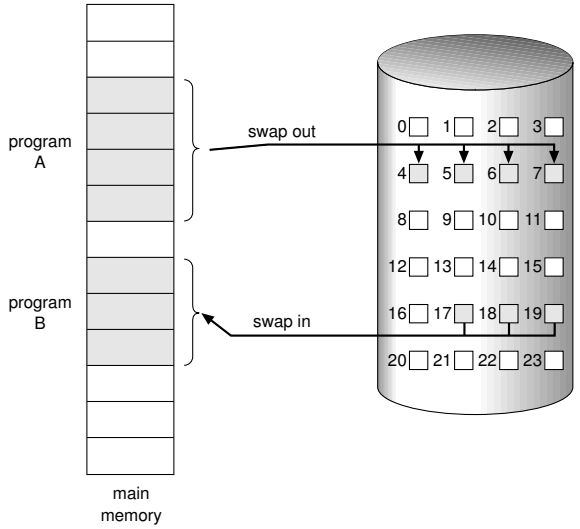
La memoria virtuale può essere implementata come
paginazione su richiesta oppure
segmentazione su richiesta

Paginazione su richiesta

Schema a paginazione, ma in cui si carica una pagina in memoria solo quando è necessario

- Meno I/O
- Meno memoria occupata
- Maggiore velocità
- Più utenti/processi

Una pagina è richiesta quando vi si fa riferimento
- viene segnalato dalla MMU
- se l'accesso non è valido ⇒ abortisci il processo
- se la pagina non è in memoria ⇒ caricala dal disco



Swapping vs. Paging

Spesso si confonde *swapping* con *paging*

- Swapping: scambio di interi processi da/per il backing store

  Swapper: processo che implementa una politica di swapping (scheduling di medio termine)

- Paging: scambio di gruppi di pagine (sottoinsiemi di processi) da/per il backing store

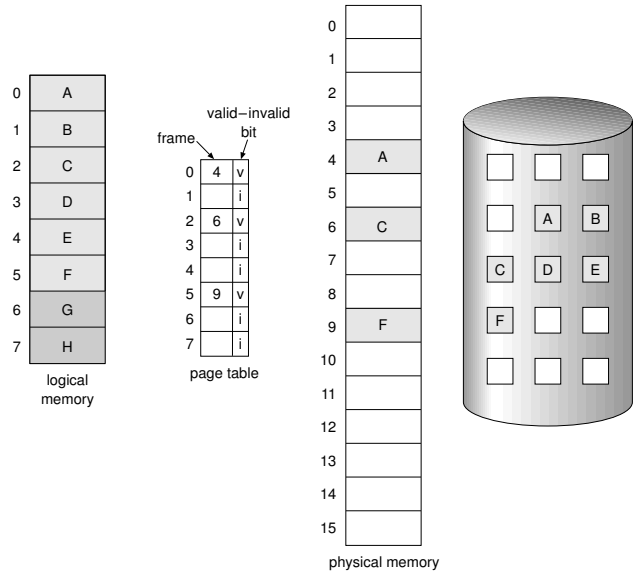
  Pager: processo che implementa una politica di gestione delle pagine dei processi (caricamento/scaricamento).

Sono concetti molto diversi, e non esclusivi!

Purtroppo, in alcuni S.O. il pager viene chiamato "swapper" (es.: Linux: `kswapd`)

Valid–Invalid Bit

- Ad ogni entry nella page table, si associa un bit di validità. (1 ⇒ in-memory, 0 ⇒ not-in-memory)
- Inizialmente, il bit di validità è settato a 0 per tutte le pagine.
- La prima volta che si fa riferimento ad una pagina (non presente in memoria), la MMU invia un interrupt alla CPU: *page fault.*



Routine di gestione del Page Fault

- il S.O. controlla guarda in un'altra tabella se è stata un accesso non valido (fuori dallo spazio indirizzi virtuali assegnati al processo) ⇒ abort del processo ("segmentation fault")

- Se l'accesso è valido, ma la pagina non è in memoria:

  – trovare qualche pagina in memoria, ma in realtà non usata, e scaricarla su disco (*swap out*)

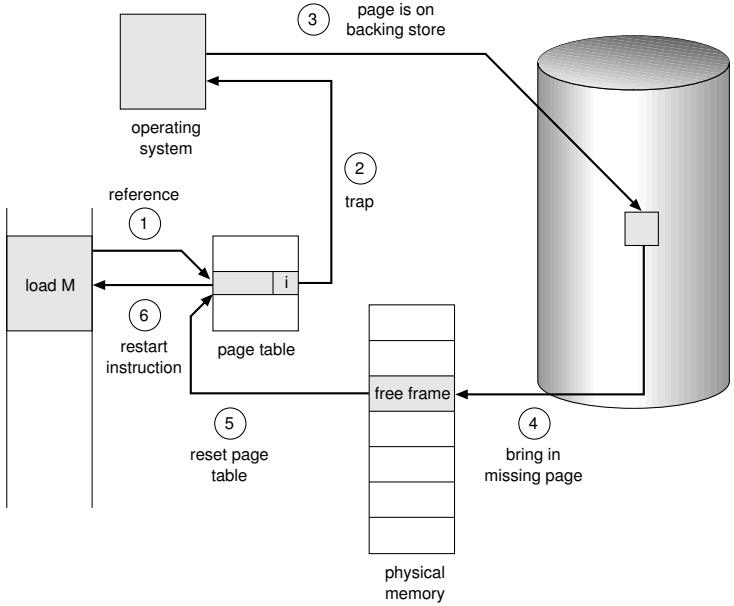
  – Caricare la pagina richiesta nel frame così liberato (*swap in*)

  – Aggiornare le tabelle delle pagine

- L'istruzione che ha causato il page fault deve essere rieseguita in modo consistente

  ⇒ vincoli sull'architettura della macchina. Es: la MVC dell'IBM 360.

Page Fault: gestione



Performance del paging on-demand

- p = Page fault rate; $0 \leq p \leq 1$
  - $p = 0 \Rightarrow$ nessun page fault
  - $p = 1 \Rightarrow$ ogni riferimento in memoria porta ad un page fault
- Tempo effettivo di accesso (EAT)

$$\begin{aligned} EAT \;=\; & (1-p) \times \text{accesso alla memoria} \\ & +p(\text{overhead di page fault} \\ & \quad [+\text{swap page out}] \\ & \quad +\text{swap page in} \\ & \quad +\text{overhead di restart}) \end{aligned}$$

Esempio di Demand Paging

- Tempo di accesso alla memoria (comprensivo del tempo di traduzione): 60 nsec
- Assumiamo che 50% delle volte che una pagina deve essere rimpiazzata, è stata modificata e quindi deve essere scaricata su disco.
- Swap Page Time = 5 msec = 5e6 nsec (disco molto veloce!)
- $EAT = 60(1-p) + 5e6 * 1.5 * p = 60 + (7.5e6 - 60)p$ in nsec
- Si ha un degrado del 10% quando $p = 6/(7.5e6 - 60) = 1/1250000$

Considerazioni sul Demand Paging

- problema di performance: si vuole un algoritmo di rimpiazzamento che porti al minor numero di page fault possibile
- l'area di swap deve essere il più veloce possibile $\Rightarrow$ meglio tenerla separata dal file system (possibilmente anche su un device dedicato) ed accedervi direttamente (senza passare per il file system). Blocchi fisici = frame in memoria.
- La memoria virtuale con demand paging ha benefici anche alla creazione dei processi

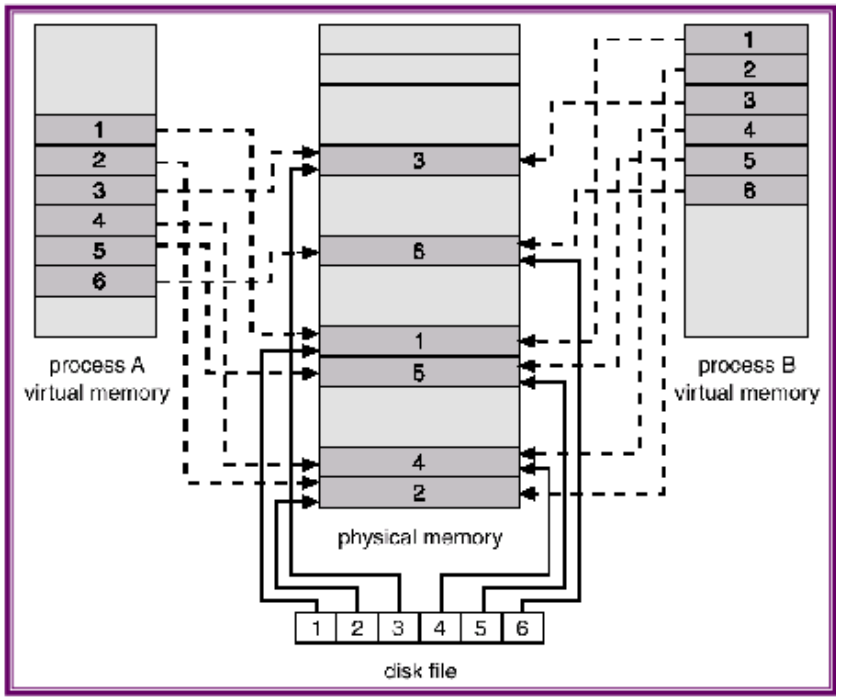
Creazione dei processi: Copy on Write

- Il Copy-on-Write permette al padre e al figlio di condividere inizialmente le stesse pagine in memoria.

  Una pagina viene copiata *se* e *quando* viene acceduta in scrittura.
- COW permette una creazione più veloce dei processi
- Le pagine libere devono essere allocate da un set di pagine azzerate

Creazione dei processi: Memory-Mapped I/O

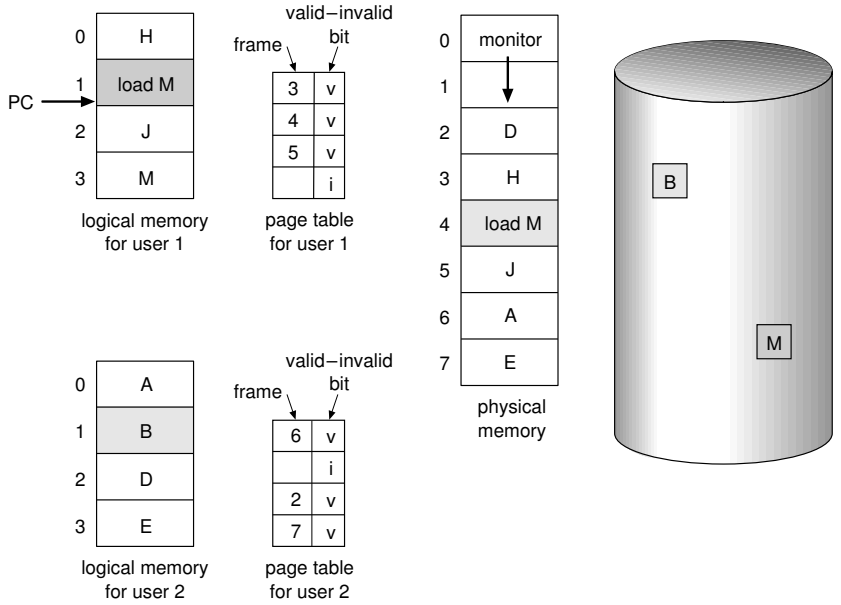
- Memory-mapped file I/O permette di gestire l'I/O di file come accessi in memoria: ogni blocco di un file viene *mappato* su una pagina di memoria virtuale
- Un file (es. DLL, .so) può essere così letto come se fosse in memoria, con demand paging. Dopo che un blocco è stato letto una volta, rimane caricato in memoria senza doverlo rileggere.
- La gestione dell'I/O è molto semplificata
- Più processi possono condividere lo stesso file, condividendo gli stessi frame in cui viene caricato.

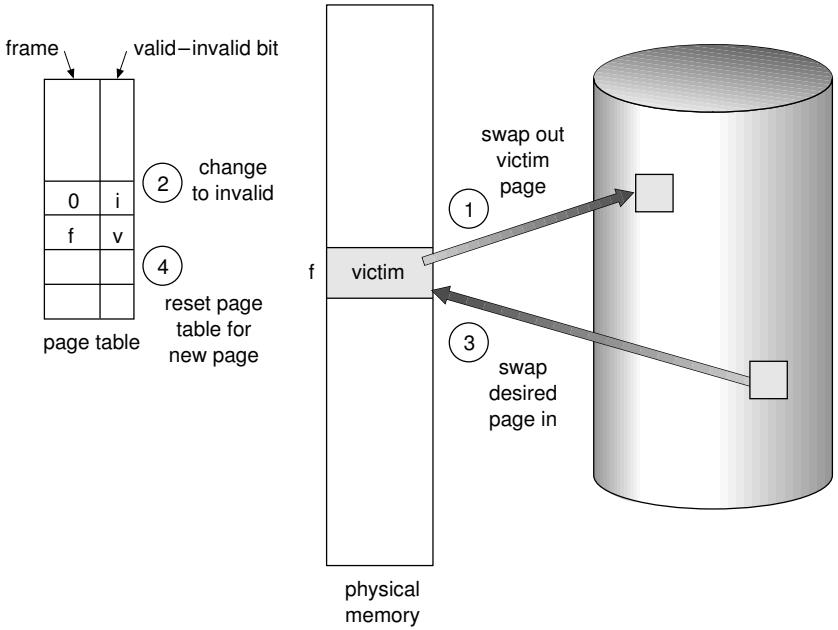


Sostituzione delle pagine

- Aumentando il grado di multiprogrammazione, la memoria viene *sovrallocata:* la somma degli spazi logici dei processi in esecuzione è superiore alla dimensione della memoria fisica
- Ad un page fault, può succedere che non esistono frame liberi
- Si modifica la routine di gestione del page fault aggiungendo la *sostituzione delle pagine* che libera un frame occupato (*vittima*)
- Bit di modifica (*dirty bit*): segnala quali pagine sono state modificate, e quindi devono essere salvate su disco. Riduce l'overhead.
- Il rimpiazzamento di pagina completa la separazione tra memoria logica e memoria fisica: una memoria logica di grandi dimensioni può essere implementata con una piccola memoria fisica.

Sostituzione delle pagine (cont.)





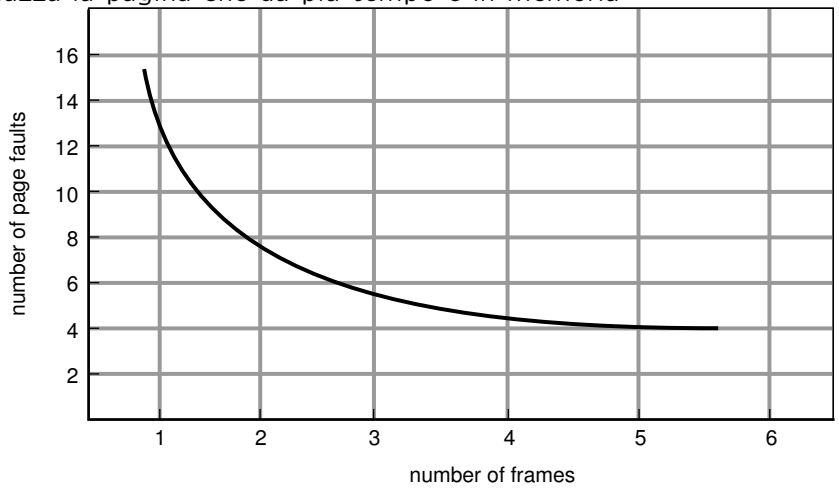
Algoritmi di rimpiazzamento delle pagine

- È un problema molto comune, non solo nella gestione della memoria (es: cache di CPU, di disco, di web server. . .)
- Si mira a minimizzare il page-fault rate.
- Un modo per valutare questi algoritmi: provarli su una sequenza prefissata di accessi alla memoria, e contare il numero di page fault.
- In tutti i nostri esempi, la sequenza sarà di 5 pagine in questo ordine

  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

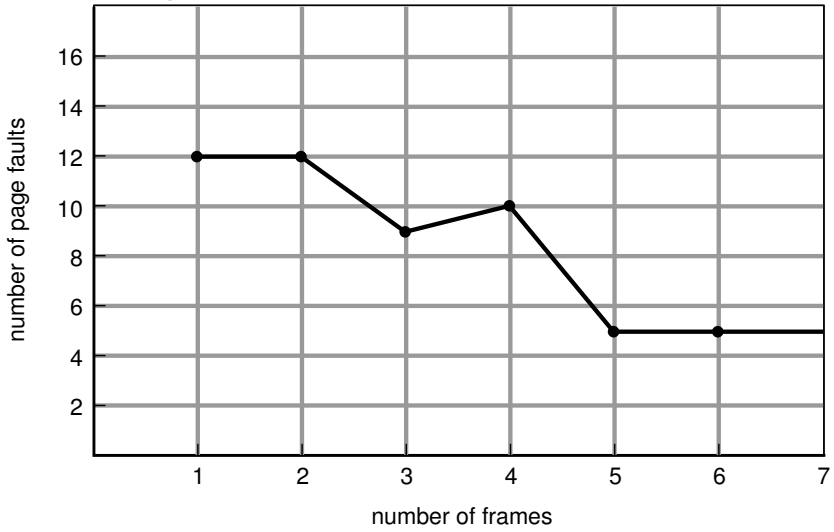
Algoritmo First-In-First-Out (FIFO)

- Si rimpiazza la pagina che da più tempo è in memoria



- Con 3 frame (3 pagine per volta possono essere in memoria): 9 page fault

- Con 4 frame: 10 page fault



- Il rimpiazzamento FIFO soffre dell'*anomalia di Belady:* + memoria fisica $\not\Rightarrow$ − page fault!

Algoritmo ottimale (OPT o MIN)

- Si rimpiazza la pagina che non verrà riusata per il periodo più lungo
- Con 4 frame: 6 page fault
- Tra tutti gli algoritmi, è quello che porta al minore numero di page fault e non soffre dell'anomalia di Belady
- Ma come si può prevedere quando verrà riusata una pagina?
- Algoritmo usato in confronti con altri algoritmi

Algoritmo Least Recently Used (LRU)

- Approssimazione di OPT: studiare il passato per prevedere il futuro
- Si rimpiazza la pagina che da più tempo non viene usata
- Con 4 frame: 8 page fault
- È la soluzione ottima con ricerca *all'indietro* nel tempo: LRU su una stringa di riferimenti r è OPT sulla stringa $reverse(r)$
- Quindi la frequenza di page fault per la LRU è la stessa di OPT su stringhe invertite.
- Non soffre dell'anomalia di Belady (è un *algoritmo di stack*)
- Generalmente è una buona soluzione
- Problema: LRU necessita di notevole assistenza hardware

Matrice di memoria

Dato un algoritmo di rimpiazzamento, e una reference string, si definisce la **matrice di memoria**: $M(m, r)$ è l'insieme delle pagine caricate all'istante r avendo m frames a disposizione

Esempio di matrice di memoria per LRU:

Reference string	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	1	3	4	1
	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	1	3	4	1
		0	2	1	3	5	4	6	3	7	4	7	7	3	3	5	3	3	3	1	7	1	3	4
			0	2	1	3	5	4	6	3	3	4	4	7	7	7	5	5	5	3	3	7	1	3
				0	2	1	3	5	4	6	6	6	6	4	4	4	7	7	7	5	5	5	7	7
					0	2	1	1	5	5	5	5	5	6	6	6	4	4	4	4	4	4	5	5
						0	2	2	1	1	1	1	1	1	1	1	6	6	6	6	6	6	6	6
							0	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
									0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Page faults	P	P	P	P	P	P	P		P					P			P						P	
Distance string	∞	∞	∞	∞	∞	∞	∞	4	∞	4	2	3	1	5	1	2	6	1	1	4	2	3	5	3

Algoritmi di Stack

Un algoritmo di rimpiazzamento si dice *di stack* se per ogni reference string r, per ogni memoria m:

$$M(m, r) \subseteq M(m + 1, r)$$

Ad esempio, OPT e LRU sono algoritmi di stack. FIFO non è di stack

Fatto: Gli algoritmi di stack non soffrono dell'anomalia di Belady.

Implementazioni di LRU

Implementazione a contatori

- La MMU ha un contatore (32-64 bit) che viene automaticamente incrementato dopo ogni accesso in memoria.
- Ogni entry nella page table ha un registro (*reference time*)
- ogni volta che si riferisce ad una pagina, si copia il contatore nel registro della entry corrispondente
- Quando si deve liberare un frame, si cerca la pagina con il registro più basso

Molto dispendioso, se la ricerca viene parallelizzata in hardware.

Implementazioni di LRU (Cont.)

Implementazione a stack

- si tiene uno stack di numeri di pagina in un lista double-linked
- Quando si riferisce ad una pagina, la si sposta sul top dello stack (Richiede la modifica di 6 puntatori).
- Quando si deve liberare un frame, la pagina da swappare è quella in fondo allo stack: non serve fare una ricerca

Implementabile in software (microcodice). Costoso in termini di tempo.

Approssimazioni di LRU: reference bit e NFU

Bit di riferimento (*reference bit*)

- Associare ad ogni pagina un bit R, inizialmente =0
- Quando si riferisce alla pagina, R viene settato a 1
- Si rimpiazza la pagina che ha $R = 0$ (se esiste).
- Non si può conoscere l'ordine: impreciso.

Variante: **Not Frequently Used** (NFU)

- Ad ogni pagina si associa un contatore
- Ad intervalli regolari (*tick*, tip. 10-20ms), per ogni entry si somma il reference bit al contatore.
- Problema: pagine usate molto tempo fa contano come quelle recenti

Approssimazioni di LRU: aging

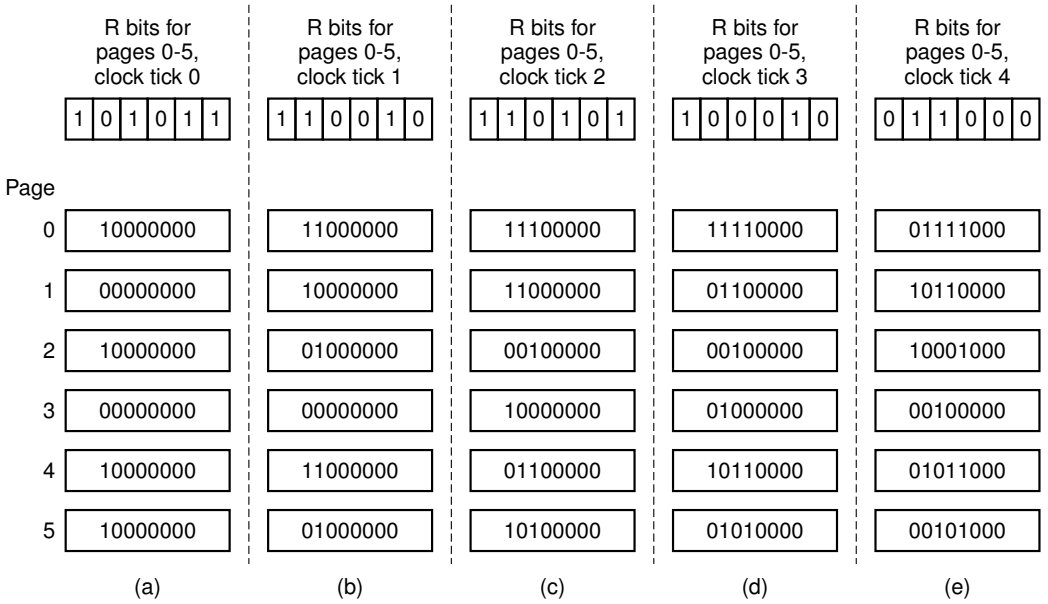
Aggiungere bit supplementari di riferimento, con peso diverso.

- Ad ogni pagina si associa un array di bit, inizialmente =0
- Ad intervalli regolari, un interrupt del timer fa partire una routine che shifta gli array di tutte le pagine immettendovi i bit di riferimento, che vengono settato a 0
- Si rimpiazza la pagina che ha il numero più basso nell'array

Differenze con LRU:

- Non può distingure tra pagine accedute nello stesso tick.
- Il numero di bit è finito $\Rightarrow$ la memoria è limitata

In genere comunque è una buona approssimazione.



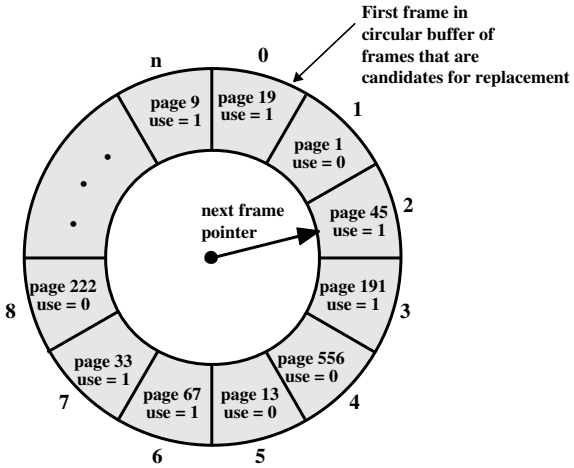
Approssimazioni di LRU: CLOCK (o "Second chance")

Idea di base: se una pagina è stata usata pesantemente di recente, allora probabilmente verrà usata pesantemente anche prossimamente.

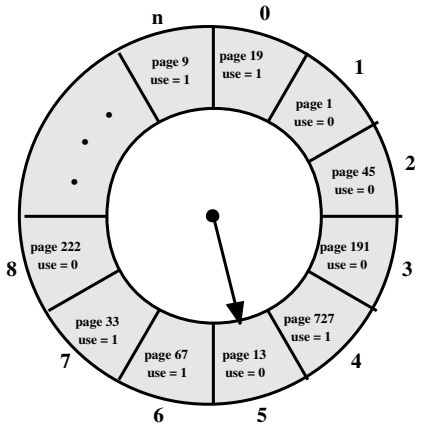
- Utilizza il reference bit.
- Si segue un ordine "ad orologio"
- Se la pagina candidato ha il reference bit = 0, rimpiazzala
- se ha il bit = 1, allora
  - imposta il reference bit 0.
  - lascia la pagina in memoria
  - passa alla prossima pagina, seguendo le stesse regole

Nota: se tutti i bit=1, degenera in un FIFO

Buona approssimazione di LRU; usato (con varianti) in molti sistemi



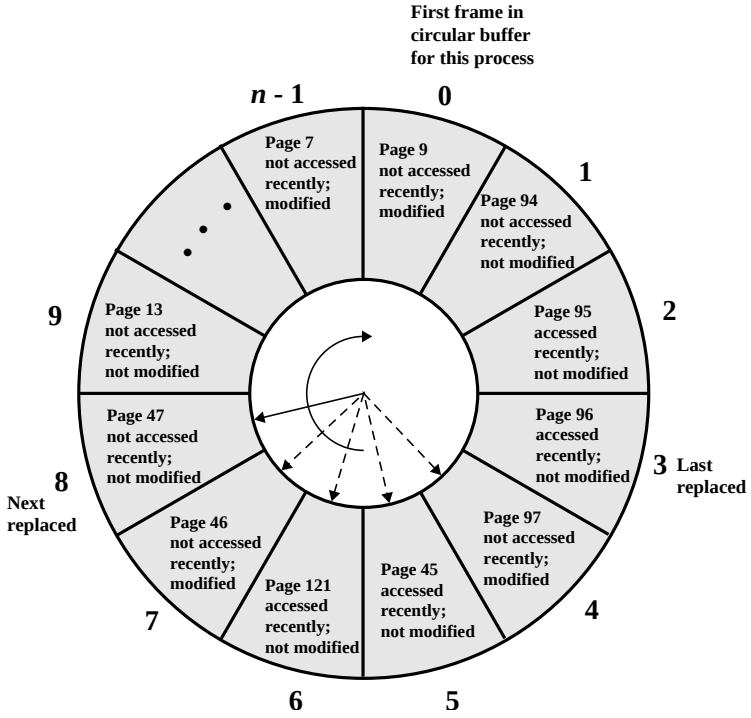
(a) State of buffer just prior to a page replacement



(b) State of buffer just after the next page replacement

Approssimazioni di LRU: CLOCK migliorato

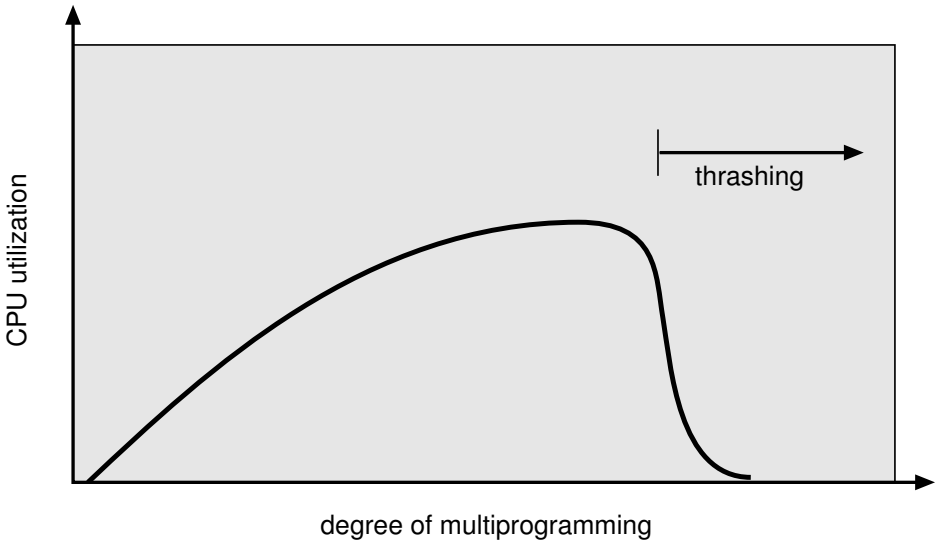
- Usare due bit per pagina: il reference (r) e il dirty (d) bit
  - non usata recentemente, non modificata ($r = 0, d = 0$): buona
  - non usata recentemente, ma modificata ($r = 0, d = 1$): meno buona
  - usata recentemente, non modificata ($r = 1, d = 0$): probabilmente verrà riusata
  - usata recentemente e modificata ($r = 1, d = 1$): molto usata
- si scandisce la coda dei frame più volte
  1. cerca una pagina con (0,0) senza modificare i bit; fine se trovata
  2. cerca una pagina con (0,1) azzerando i reference bit; fine se trovata
  3. vai a 1.
- Usato nel MacOS tradizionale (fino a 9.x)



Thrashing

- Se un processo non ha "abbastanza" pagine, il page-fault rate è molto alto. Questo porta a
  - basso utilizzo della CPU (i processi sono impegnati in I/O)
  - il S.O. potrebbe pensare che deve aumentare il grado di multiprogrammazione (errore!)
  - un altro processo viene caricato in memoria
- *Thrashing:* uno o più processi spendono la maggior parte del loro tempo a swappare pagine dentro e fuori
- Il thrashing di un processo avviene quando la memoria assegnatagli è inferiore a quella richiesta dalla sua località

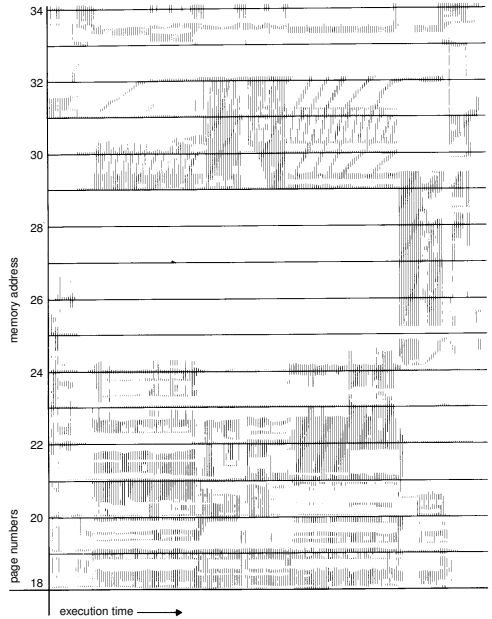
- Il thrashing del sistema avviene quando la memoria fisica è inferiore somma delle località dei processi in esecuzione. Può essere causato da un processo che si espande e in presenza di rimpiazzamento globale.



Principio di località

Ma allora, perché la paginazione funziona? Per il principio di località

- Una *località* è un insieme di pagine che vengono utilizzate attivamente assieme dal processo.
- Il processo, durante l'esecuzione, migra da una località all'altra
- Le località si possono sovrapporre



Impedire il thrashing: modello del working-set

- $\Delta \equiv$ working-set window $\equiv$ un numero fisso di riferimenti a pagine

  Esempio: le pagine a cui hanno fatto riferimento le ultime 10,000 instruzioni

- WSS_i (working set del processo P_i) = numero totale di pagine riferite nell'ultimo periodo Δ. Varia nel tempo.
  - Se Δ è troppo piccolo, il WS non copre l'intera località
  - Se Δ è troppo grande, copre più località
  - Se $\Delta = \infty \Rightarrow$ copre l'intero programma e dati
- $D = \Sigma\ WSS_i \equiv$ totale frame richiesti.
- Sia $m =$ n. di frame fisici disponibile. Se $D > m \Rightarrow$ thrashing.

Algoritmo di allocazione basato sul working set

- il sistema monitorizza il ws di ogni processo, allocandogli frame sufficienti per coprire il suo ws
- alla creazione di un nuovo processo, questo viene ammesso nella coda ready solo se ci sono frame liberi sufficienti per coprire il suo ws
- se $D > m$, allora si sospende uno dei processi per liberare la sua memoria per gli altri (diminuire il grado di multiprogrammazione — scheduling di medio termine)

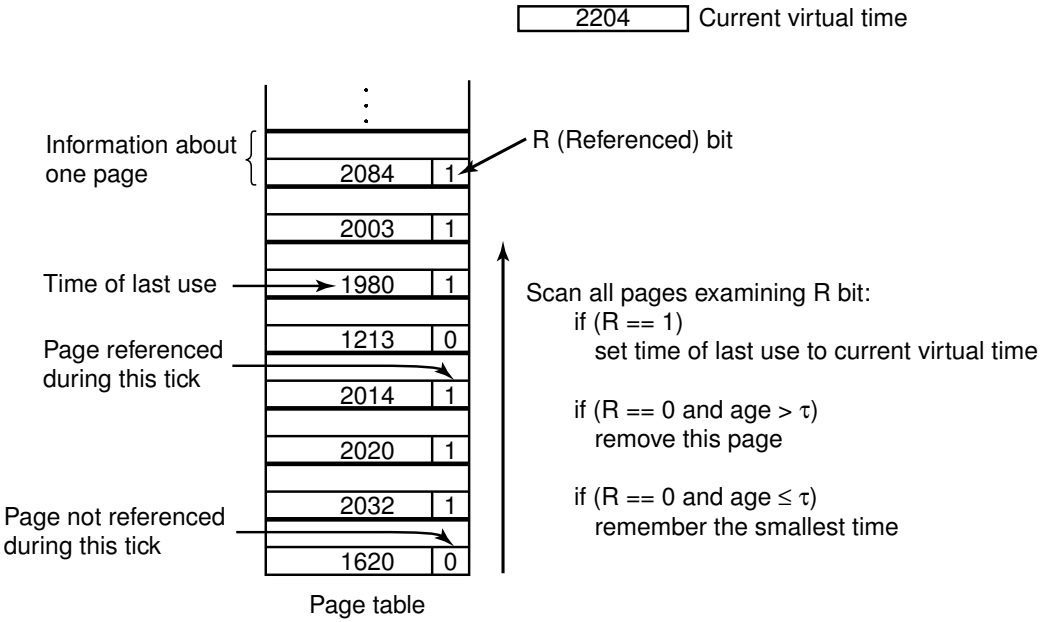
Si impedisce il thrashing, massimizzando nel contempo l'uso della CPU.

Approssimazione del working set: registri a scorrimento

- Si approssima con un timer e il bit di riferimento
- Esempio: $\Delta = 10000$
  - Si mantengono due bit per ogni pagina (oltre al reference bit)
  - il timer manda un interrupt ogni 5000 unità di tempo
  - Quando arriva l'interrupt, si shifta il reference bit di ogni pagina nei due bit in memoria, e lo si cancella
  - Quando si deve scegliere una vittima: se uno dei tre bit è a 1, allora la pagina è nel working set
- Implementazione non completamente accurata (scarto di 5000 accessi)
- Miglioramento: 10 bit e interrupt ogni 1000 unità di tempo $\Rightarrow$ più preciso ma anche più costoso da gestire

Approssimazione del working set: tempo virtuale

- Si mantiene un *tempo virtuale corrente* del processo (=n, di tick consumati da processo)
- Si eliminano pagine più vecchie di τ tick
- Ad ogni pagina, viene associato un registro contenente il tempo di ultimo riferimento
- Ad un page fault, si controlla la tabella alla ricerca di una vittima.
  - se il reference bit è a 1, si copia il TVC nel registro corrispondente, il reference viene azzerato e la pagina viene saltata
  - se il reference è a 0 e l'età $> \tau$, la pagina viene rimossa
  - se il reference è a 0 e l'età $\leq \tau$, segnati quella più vecchia (con minore tempo di ultimo riferimento). Alla peggio, questa viene cancellata.



Algoritmo di rimpiazzamento WSClock

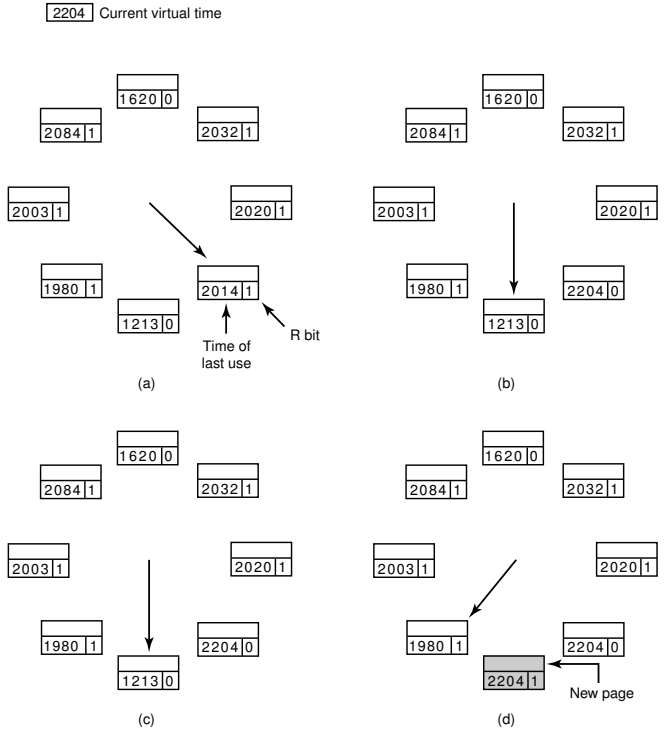
Variante del Clock che tiene conto del Working Set. Invece di contare i riferimenti, si tiene conto di una finestra temporale τ fissata (es. 100ms)

- si mantiene un contatore T del tempo di CPU impiegato da ogni processo
- le pagine sono organizzate ad orologio; inizialmente, lista vuota
- ogni entry contiene i reference e dirty bit R, M, e un registro *Time of last use*, che viene copiato dal contatore durante l'algoritmo. La differenza tra questo registro e il contatore si chiama *età* della pagina.
- ad un page fault, si guarda prima la pagina indicata dal puntatore
  - se $R = 1$, si mette $R = 0$, si copia $TLU = T$ e si passa avanti
  - se $R = 0$ e età $\leq \tau$: è nel working set: si passa avanti

  - se $R = 0$ e età $> \tau$: se $M = 0$ allora si libera la pagina, altrimenti si schedula un pageout e si passa avanti

- Cosa succede se si fa un giro completo?
  - se almeno un pageout è stato schedulato, si continua a girare (aspettando che le pagine schedulate vengano salvate)
  - altrimenti, significa che tutte le pagine sono nel working set. Soluzione semplice: si rimpiazza una quasisiasi pagina pulita.

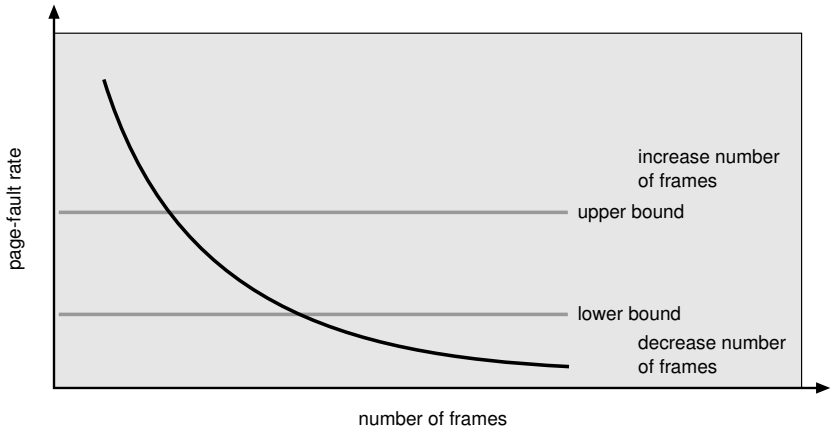
    Se non ci sono neanche pagine pulite, si rimpiazza la pagina corrente.



(a-b): cosa succede quando $R == 1$.

(c-d): cosa succede quando $R == 0$ e $T - TLU > \tau$.

Impedire il thrashing: frequenza di page-fault



Si stabilisce un page-fault rate "accettabile"

- Se quello attuale è troppo basso, il processo perde un frame
- Se quello attuale è troppo alto, il processo guadagna un frame

Nota: si controlla solo il n. di frame assegnati, non quali pagine sono caricate.

Sostituzione globale vs. locale

- *Sostituzione locale:* ogni processo può rimpiazzare solo i proprî frame.
  - Mantiene fisso il numero di frame allocati ad un processo (anche se ci sono frame liberi)
  - Il comportamento di un processo non è influenzato da quello degli altri processi
- *Sostituzione globale:* un processo sceglie un frame tra tutti i frame del sistema
  - Un processo può "rubare" un frame ad un altro
  - Sfrutta meglio la memoria fisica
  - il comportamento di un processo dipende da quello degli altri
- Dipende dall'algoritmo di rimpiazzamento scelto: se è basato su un modello di ws, si usa una sostituzione locale, altrimenti globale.

Algoritmi di allocazione dei frame

- Ogni processo necessita di un numero minimo di pagine imposto dall'architettura (Es.: su IBM 370, possono essere necessarie 6 pagine per poter eseguire l'istruzione MOV)

Diversi modi di assegnare i frame ai varî processi

- Allocazione libera: dare a qualsiasi processo quante frame desidera.

  Funziona solo se ci sono sufficienti frame liberi.

- Allocazione equa: stesso numero di frame ad ogni processo

  Porta a sprechi (non tutti i processi hanno le stesse necessità)

- Allocazione proporzionale: un numero di frame in proporzione a
  - dimensione del processo
  - sua priorità (Solitamente, ai page fault si prendono frame ai processi a priorità inferiore)

Esempio: due processi da 10 e 127 pagine, su 62 frame:

$$\frac{10}{127+10} * 62 \cong 4 \qquad \frac{127}{127+10} * 62 \cong 57$$

L'allocazione varia al variare del livello di multiprogrammazione: se arriva un terzo processo da 23 frame:

$$\frac{10}{127+10+23} * 62 \cong 3 \qquad \frac{127}{127+10+23} * 62 \cong 49 \qquad \frac{23}{127+10+23} * 62 \cong 8$$

Buffering di pagine

Aggiungere un insieme (*free list*) di frame liberi agli schemi visti

- il sistema cerca di mantenere sempre un po' di frame sulla free list
- quando si libera un frame,
  - se è stato modificato lo si salva su disco
  - si mette il suo dirty bit a 0
  - si sposta il frame sulla free list *senza cancellarne il contenuto*
- quando un processo produce un page fault
  - si vede se la pagina è per caso ancora sulla free list (*soft page fault*)
  - altrimenti, si prende dalla free list un frame, e vi si carica la pagina richiesta dal disco (*hard page fault*)

Altre considerazioni

- Prepaging: caricare in anticipo le pagine che "probabilmente" verranno usate
  - applicato al lancio dei programmi e al ripristino di processi sottoposti a swapout di medio termine
- Selezione della dimensione della pagina: solitamente imposta dall'architettura. Dimensione tipica: 4K-8K. Influenza
  - frammentazione: meglio piccola
  - dimensioni della page table: meglio grande
  - quantità di I/O: meglio piccola
  - tempo di I/O: meglio grande
  - località: meglio piccola
  - n. di page fault: meglio grande

Altre considerazioni (cont.)

- La struttura del programma può influenzare il page-fault rate
  - Array A[1024,1024] **of** integer
  - Ogni riga è memorizzata in una pagina
  - Un frame a disposizione

Programma 1	Programma 2
for $j := 1$ to 1024 **do**	**for** $i := 1$ to 1024 **do**
for $i := 1$ to 1024 **do**	**for** $j := 1$ to 1024 **do**
A[i, j] := 0;	A[i, j] := 0;
1024×1024 page faults	1024 page faults

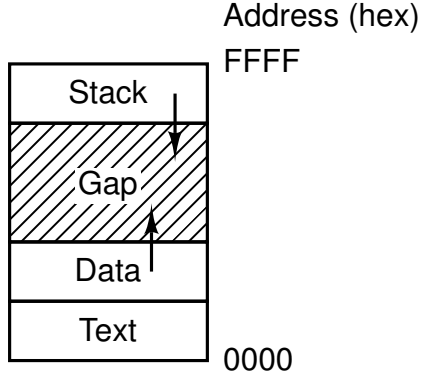
- Durante I/O, i frame contenenti i buffer non possono essere swappati
  - I/O solo in memoria di sistema $\Rightarrow$ costoso
  - Lockare in memoria i frame contenenti buffer di I/O (*I/O interlock*) $\Rightarrow$ delicato (un frame lockato potrebbe non essere più rilasciato)

Modello della memoria in Unix

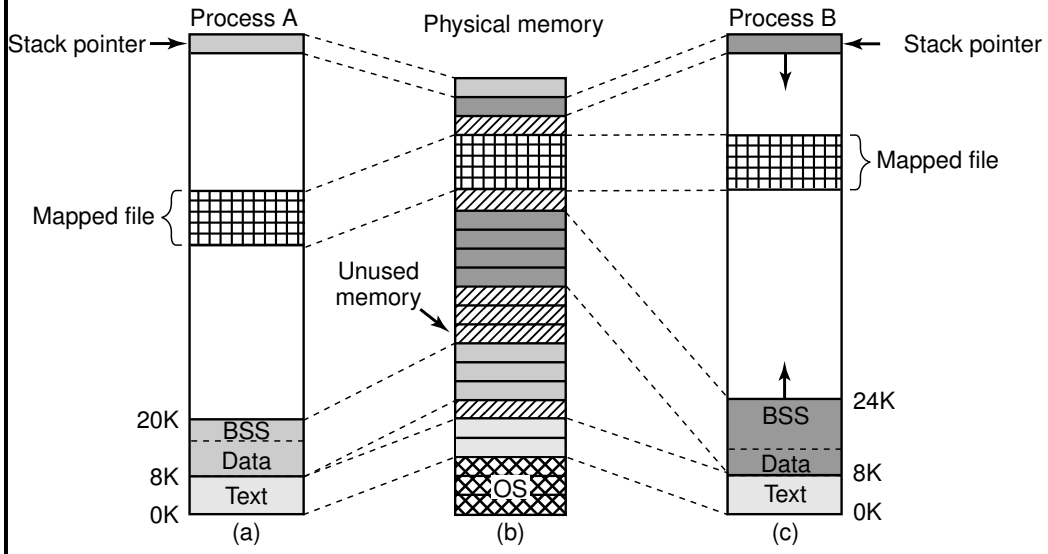
Ogni processo UNIX ha uno spazio indirizzi separato. Non vede le zone di memoria dedicati agli altri processi.

Un processo UNIX ha tre segmenti:

- Stack: Stack di attivazione delle subroutine. Cambia dinamicamente.
- Data: Contiene lo heap e i dati inizializzati al caricamento del programma. Cambia dinamicamente su richiesta esplicita del programma (es., con la **malloc**).
- Text: codice eseguibile. Non modificabile, protetto in scrittura.



Mappatura dei segmenti in memoria reale



Gestione della memoria in UNIX

- Dipende dall'hardware sottostante
  - Fino a 3BSD (1978): solo segmentazione con swapping.
  - Dal 3BSD: paginazione; da 4.2BSD (1983): paginazione on demand
  - Esistono anche UNIX senza memoria virtuale
- Viene mantenuta una free list di frame con buffering e prepaging
- Le pagine vengono allocate dalla lista libera dal kernel, su base libera, con *copy-on-write*
- La free list viene mantenuta entro un certo livello dal processo *pagedaemon*. Applica varianti del CLOCK
- Lo swapping rimane come soluzione contro il thrashing: uno *scheduler a medio termine* decide il grado di multiprogrammazione. Si attiva automaticamente, in situazioni estreme

Quando si alloca la memoria

- Ulteriore memoria può essere richiesta da un processo in corrispondenza ad uno dei seguenti eventi:
  1. Una fork, che crea un nuovo processo (allocazione di memoria per i segmenti data e stack);
  2. Una brk (e.g., in una `malloc`) che estende un segmento data;
  3. Uno stack che cresce oltre le dimensioni prefissate.
  4. Un accesso in scrittura ad una pagina condivisa tra due processi (*copy-on-write*)

  Oppure, per un processo che era swapped da troppo tempo e che deve essere caricato in memoria.

Quando si alloca la memoria (cont.)

- *minfree* = parametro fissato al boot, in proporzione alla memoria fisica. Tipicamente, 100K–5M.
- se la free list scende sotto *minfree*, il kernel si rifiuta di allocare nuove pagine di memoria.
- Le pagine vengono cercate sulla free list, prima di caricarle da disco
- Quando un processo viene lanciato, molte pagine vengono precaricate e poste sulla free list (prepaging)
- Quando un processo termina, le sue pagine vengono messe sulla lista libera
- I segmenti di codice condiviso non vengono swappati (solitamente) $\Rightarrow$ meno I/O e meno memoria usata
- Le pagine vengono lockate in memoria per I/O asincrono

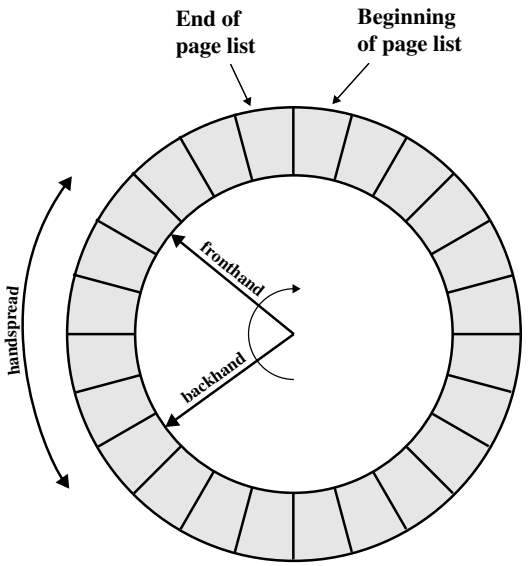
Quando si libera memoria

- La sostituzione di pagina viene implementata da un processo, il *pagedaemon*, noto anche come *pageout* (PID=2) su Solaris, *kswapd* (PID=5) su Linux, . . . Spesso è un thread di kernel
- Viene lanciato al boot e si attiva ad intervalli regolari (tip. 2–4 volte al secondo) o su richiesta del kernel
- Parametro: *lotsfree* = n. limite di frame liberi. Fissato al boot; 5–10% dei frame totali (1M–64M)
- Il pagedaemon interviene quando # frame liberi $<$ *lotsfree*
- Applica un rimpiazzamento globale
- In alcuni UNIX (specie quelli più vecchi): algoritmo dell'orologio. Inadeguato per grandi quantità di memoria: libera pagine troppo lentamente

CLOCK a due lancette

Negli UNIX moderni: algoritmo dell'orologio *a 2 lancette*.
Due puntatori scorrono la lista delle pagine allocate

- il primo azzera il reference bit
- il secondo sceglie la pagina vittima: se trova $r = 0$ (la pagina non è stata usata), il frame viene salvato e posto sulla lista libera
- si fanno avanzare i due puntatori
- si ripete finché # frame liberi $\geq$ *lotsfree*



CLOCK a due lancette (cont.)

- La distanza tra i puntatori (*handspread*) viene decisa al boot, per liberare frame abbastanza rapidamente
- La velocità di scan dipende da quanto manca a *lotsfree*: meno memoria libera c'è, più velocemente si muovono i puntatori
- Variante: "isteresi"
  - ulteriore parametro *maxfree*; *maxfree* $>$ *lotsfree*
  - quando il livello di pagine scende sotto *lotsfree*, il pagedaemon libera pagine fino a raggiungere *maxfree*

  Permette di evitare una potenziale instabilità del CLOCK a due lancette.

Swapping di processi

- Interi processi possono essere sospesi (*disattivati*) temporaneamente, per abbassare la multiprogrammazione
- Il processo *swapper* o *sched* (PID=0) decide quale processo deve essere swappato su disco
- Viene lanciato al boot; spesso è un thread di kernel
- Parametro: *desfree*, impostato al boot. 100K–10M.
  minfree $<$ *desfree* $<$ *lotsfree*
- Lo swapper si sveglia ogni 1–2 secondi, e interviene solo se
  # frame liberi $<$ *minfree* e
  # frame liberi $<$ *desfree* nella storia recente

Swapping: chi esce...

- Regole di scelta del processo vittima:
  - si cerca tra i processi in "wait", senza considerare quelli in memoria da meno di 2 secondi
  - se ce ne sono, si prende quello con Priority+Residence Time più alto.
  - Altrimenti, si cerca tra quelli in "ready", con lo stesso criterio
- Per il processo selezionato:
  - i suoi segmenti data e stack (non il text) vengono scaricati sul device di swap; i frame vengono aggiunti alla lista dei frame liberi
  - Nel PCB, viene messo lo stato "swapped" e agganciato alla lista dei processi swappati
- Si ripete fino a che sufficiente memoria viene liberata.

Swapping: ...e chi entra

Quando *swapper* si sveglia da sé:

1. cerca nella lista dei PCB dei processi swappati e ready, il processo swappato da più tempo, ma almeno 2 secondi (per evitare thrashing);
2. se lo trova, determina se c'è sufficiente memoria libera per le page tables (*easy swap*) oppure no (*hard swap*);
3. se è un hard swap, libera memoria swappando qualche altro processo;
4. carica le page table in memoria e mette il processo in "ready, in memory"
5. Solitamente, si provvede anche a prepaginare le pagine swappate

Si ripete finché non ci sono processi da caricare.

Considerazioni

Interazione tra scheduling a breve termine, a medio termine e paginazione

- minore è la priorità, maggiore è la probabilità che il processo venga swappato
- per ogni processo in esecuzione, la paginazione tende a mantenere in memoria il suo working set
- quindi, processi che non sono idle tendono a stare in memoria, mentre si tende a swappare solo processi idle da molto tempo
- nel complesso, il sistema massimizza l'utilizzo della memoria e la multiprogrammazione, limitando il thrashing e garantendo l'assenza di starvation per i processi swappati
- (comunque, i processi non dovrebbero mai essere swappati!)
- processi real-time (i.e., in classi SCHED_FIFO e SCHED_RR) non vengono mai swappati

Monitorare i processi e la memoria: il comando ps lx

```
  F   UID   PID  PPID PRI  NI    VSZ  RSS WCHAN  STAT TTY        TIME COMMAND
100   121   632   622   0   0   1696    0 wait4  SW   ?          0:00 [Default]
000   121   643   632   0   0   5564 1024 do_pol S    ?          0:01 gnome-sessi
000   121   660     1   1   0   5588  520 do_sel S    ?          0:06 gnome-smpro
000   121   664     1   3   0   4840 2208 do_sel S    ?          1:06 enlightenme
000   121   680     1   0   0   2704  316 do_pol S    ?          0:00 gnome-name-
000   121   683     1   0   0   7852 4052 do_pol S    ?          0:04 panel --sm-
000   121   685     1   0   0   3024 1464 do_sel S    ?          0:10 xscreensave
000   121   687     1   0   0   7836  852 do_pol S    ?          0:04 gmc --sm-co
000   121   689     1   0   0   3592  884 do_sel S    ?          0:01 Eterm
000   121   697   689   0   0   1748    0 wait4  SW   ttyp1      0:00 [bash]
000   121   727     1  14   5   6424 1008 do_pol SN   ?         13:23 cpumemusage
000   121   729     1   0   0   6484  556 do_pol S    ?          0:08 gnomexmms -
000   121   731     1   3   0   6436 1584 do_pol S    ?          1:07 gnomepager_
000   121   733     1   0   0   6472  804 do_pol S    ?          0:04 clockmail_a
000   121   755   697   0   0   5180 2368 do_sel S    ttyp1      0:02 pine -i
000   121  1020     1   0   0   4796 3392 do_sel S    ?          0:00 Eterm
000   121  1023  1020   0   0   1752 1048 read_c S    ttyp0      0:00 -bash
000   121  1034  1023   4   0   7420 6136 do_sel S    ttyp0      1:19 sdr
604   121  1054  1034  10  10      0    0 do_exi ZN   ttyp0      0:02 [vic <defun
000   121  1062     1   0   0   4576 3172 do_sel S    ?          0:02 Eterm
000   121  1065  1062   0   0   1776 1092 read_c S    ttyp2      0:00 -bash
```

Monitorare i processi e la memoria: il comando top

coltrane:~

File Opzioni Aiuto

```
  4:04pm  up 4 days, 20:54,  0 users,  load average: 1.05, 1.15, 1.07
62 processes: 60 sleeping, 2 running, 0 zombie, 0 stopped
CPU states: 98.5% user,  1.3% system, 94.2% nice,  0.3% idle
Mem:   63404K av,  61624K used,   1780K free,  40340K shrd,    792K buff
Swap:  64476K av,  16332K used,  48144K free                  25024K cached

  PID USER     PRI  NI  SIZE  RSS SHARE STAT  LIB %CPU %MEM   TIME COMMAND
  311 nobody    20  19   356  316   220 R N     0 94.2  0.4  6881m rc5des
  435 root       0   0 15568  14M   920 S       0  3.7 22.6  16:29 X
 7934 miculan    3   0   808  808   616 R       0  1.5  1.2   0:00 top
 6935 miculan    0   0  3208 2356  1432 S       0  0.3  3.7   0:09 kpanel
    1 root       0   0   108   68    48 S       0  0.0  0.1   0:02 init
    2 root       0   0     0    0     0 SW      0  0.0  0.0   0:00 kflushd
    3 root     -12 -12     0    0     0 SW<     0  0.0  0.0   0:00 kswapd
    4 root       0   0     0    0     0 SW      0  0.0  0.0   0:00 nfsiod
    5 root       0   0     0    0     0 SW      0  0.0  0.0   0:00 nfsiod
    6 root       0   0     0    0     0 SW      0  0.0  0.0   0:00 nfsiod
    7 root       0   0     0    0     0 SW      0  0.0  0.0   0:00 nfsiod
  424 root       0   0    56    4     4 S       0  0.0  0.0   0:00 mingetty
  289 root       0   0   104   24    24 S       0  0.0  0.0   0:00 ypbind
   53 root       0   0   108   72    52 S       0  0.0  0.1   0:00 kerneld
  218 root       0   0   220  188   144 S       0  0.0  0.2   0:00 syslogd
  227 root       0   0   324  168   128 S       0  0.0  0.2   0:00 klogd
  238 daemon     0   0   140  104    64 S       0  0.0  0.1   0:00 atd
```


Monitorare i processi e la memoria: il comando vmstat

```
miculan@ten:~> vmstat 5
 procs      memory              page            disk          faults       cpu
 r b w   swap   free   re  mf pi po fr de sr s0 s1 s2 s3   in   sy   cs us sy id
 0 0 0 102232   7264    0 137  5  0  0  0  0  8  0  0  0  442 24730  365 26 13 61
 0 0 0  99260   6164    0 468 44  0  0 1440 0 14 1  0  0  479 21848  358 30 36 34
 0 0 0  96324   3964    0 146  4  0  0 808  0  1  0  0  0  213 24221  186 38  9 53
 0 0 0  95148   2740    0 100  4  0  0 208  0  0  0  0  0  364 25165  331 30 10 60
 0 0 0  95008   2560    0  79  0  4  4  0  0  0  0  0  0  472 25218  400 28 10 62
 0 0 0  94592   2136    0  11  1  3  3  0  0  6  0  0  0  324 25114  224 24 10 66
 0 0 0  93192   2040    0  61 76 24 45  0 17  4  0  0  0  464 27347  382 38 17 45
 0 0 0  92420   1984    0  51  0 32 81  0 44  4  0  0  0  291 26620  246 35 14 52
 0 0 0  93268   2084    3  69  7 32 56  0 23  6  0  0  0  292 23713  272 24 10 66
 0 0 0  93572   2192    2  63 10 12 12  0  0  3  0  0  0  309 23867  263 25 10 65
 0 0 0  94216   2540    0   1  5  7  7  0  0  1  0  0  0  257 23890  215 23  7 69
 0 0 0  94128   2428    0  36  0  4  4  0  0  0  0  0  0  445 25718  391 30 11 59
 0 0 0  93784   2224    2  44  1 12 13  0  0 11  1  0  0  299 23573  223 21  9 69
 0 1 0  93600   2224    4  96  1 93 100 0  7  6  0  0  0  530 23410  443 23 14 63
 0 0 0  94052   2472    0   4  0  2  2  0  0  0  0  0  0  328 24105  322 26  9 65
 0 0 0  93996   2416    0   6  4  5  5  0  0  1  0  0  0  370 24347  343 23 11 66
 0 0 0  94216   2488    0   3  3  2  2  0  0  1  0  0  0  218 24154  196 21  9 70
 0 0 0  94344   2580    2   0  4 31 31  0  0  0  0  0  0  219 23826  176 21  8 72
 0 0 0  94288   2364    0   0 124 1  1  0  0  0  0  0  0  366 23770  271 20 10 70
 0 0 0  94708   2504    0  74  0  1  1  0  0  8  1  0  0  343 24442  261 24 10 66
 0 0 0  94516   2336    2  51 34 21 28  0  6  2  0  0  0  359 24339  333 22 12 66
 0 0 0  92708   5660    0  16 36 12 73  0 60  1  0  0  0  350 23529  361 24 11 65
 0 0 0  94720   6460    0  25  4  0  0  0  0  0  0  0  0  330 23820  292 35 10 55
```

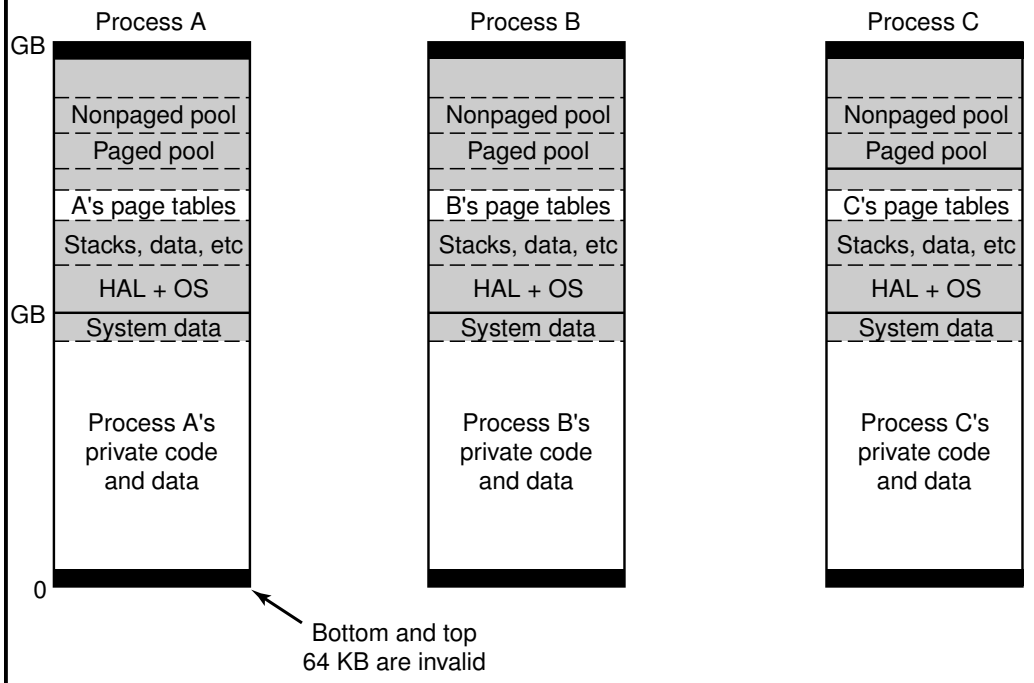

Modello della memoria in Windows 2000

Ogni task di Windows riceve uno spazio indirizzi di 4G, diviso in due parti

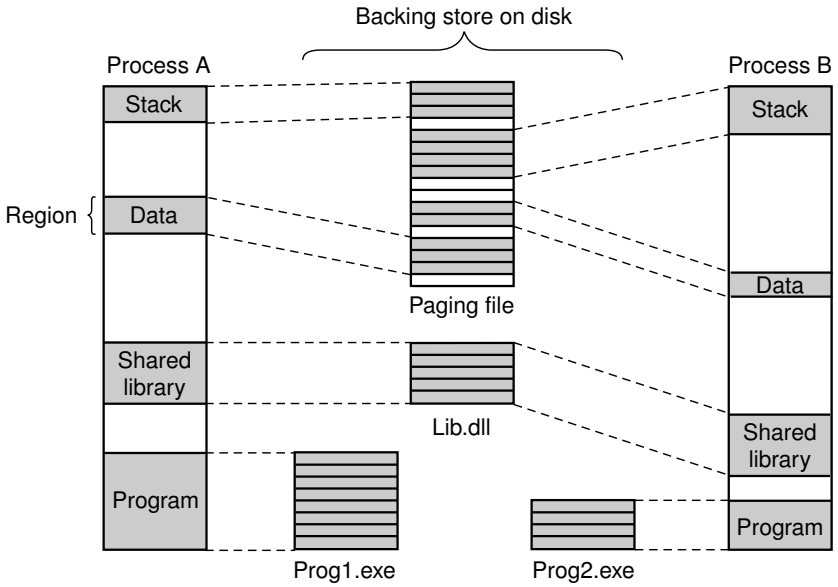
- codice e dati, nella parte bassa (< 2G) Liberamente accessibile.
- kernel e strutture di sistema (comprese le page tables) nella parte alta La maggior parte di questo spazio non è accessibile, neanche in lettura.
- i primi ed ultimi 64kb sono invalidi per individuare rapidamente errori di programmazione (puntatori a 0 e -1).

La memoria è puramente paginata (senza prepaging), con copy-on-write.

Il caricamento dei segmenti è basato fortemente sul memory mapping.



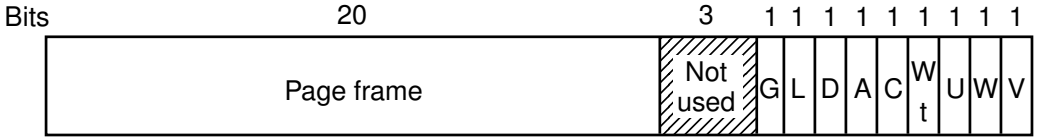
Mappatura dei segmenti in memoria reale e su file



Gestione dei page fault

Nessuna forma di prepaging: tutte le pagine vengono caricate su page fault.

Page table entry per il Pentium:



G: Page is global to all processes
L: Large (4-MB) page
D: Page is dirty
A: Page has been accessed
C: Caching enabled/disabled

Wt: Write through (no caching)
U: Page is accessible in user mode
W: Writing to the page permitted
V: Valid page table entry

Stati di una pagina

- *Available*: pagina non usata da nessun processo. Si divide in tre possibilità:
  - Free: riusabile
  - Standby: rimossa da un ws ma richiamabile (buffering)
  - Zeroed: riusabile e in più tutta azzerata
- *Reserved*: riservata da un processo ma non ancora usata. Non fa parte del ws fino a che non viene veramente usata.
- *Committed*: usata da un processo e associata ad un blocco su disco

Casi di page fault

- La pagina riferita non è committed
  ⇒ Terminazione del processo
- Violazione di protezione
  ⇒ Terminazione del processo
- Scrittura su una pagina condivisa
  ⇒ Copy-on-write su una pagina reserved
- Crescita dello stack
  ⇒ Allocazione di una pagina azzerata
- La pagina riferita è salvata ma non attualmente caricata in memoria
  ⇒ il vero page fault: pagein della pagina mancante

Algoritmo di rimpiazzamento di pagina

La paginazione è basata sul modello del Working Set

- ogni processo ha una dim. minima e massima (non sono limiti hard)
- Tutti i processi iniziano con lo stesso min e max (risp. 20-50 e 45-345, in proporzione alla RAM; modificabile dall'admin)
- Ad un page fault:
  - se $ws < max$, la pagina viene allocata ed aggiunta al working set.
  - se $ws > max$, una pagina vittima viene scelta nel working set (politica di rimpiazzamento *locale*)
- I limiti possono cambiare nel tempo: se un processo sta paginando troppo (thrashing locale), il suo max viene aumentato
- vengono mantenute sempre libere almeno 512 pagine

Le pagine allocate vengono prelevate dalla *free list*:

- ogni secondo parte un thread del kernel (*balance set manager*)
- se la free list è troppo corta, parte il *working set manager* che esamina i working sets per liberare pagine
  - prima i processi più grandi e idle da più tempo; il processo in foreground è considerato per ultimo
  - se un processo ha $ws < min$ o ha avuto molti page fault recentemente, viene saltato
  - altrimenti una o più pagine vengono rimosse
- si ripete sempre più aggressivamente finché la free list ritorna accettabile
- anche parte del kernel può essere paginata
- Eventualmente un ws può scendere sotto il min
- non esiste completo swapout di processi