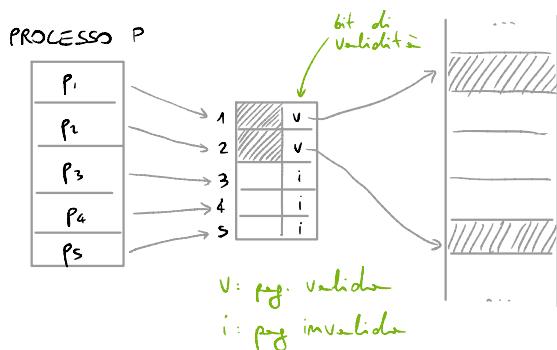


Memoria Virtuale

MEMORIA RAM:

- Può essere allocata ai processi con un **caricamento totale**
- VINCOLO**
- sullo spazio occupabile dai processi
 - sul programmer
 - la memoria necessaria potrebbe essere inferiore a quella caricata (es. librerie non completamente utilizzate).
- Il grado di multiprogrammazione cresce

Per ovviare a questi vincoli si può far uso di **memoria virtuale** con contesto di **paginazione**

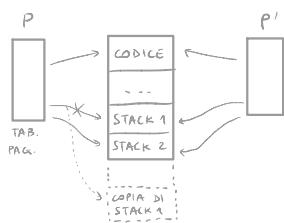


Abbiamo pagine caricate in RAM (v) e pagine presenti solo sul file system o sull'area di swap (i)

PAGINA INVALIDA:

- Pagina non caricata in RAM
- Pagina al quale si effettua l'accesso in memoria illegale (ad esempio quando l'OS assegna ad un processo spazio in più per frammentazione e si effettua la lettura su queste pagine in eccedenza)

COPIATURA SU SCRITTURA:



Aumenta l'efficienza di gestione della RAM.
In caso di fork, il processo P' CLONA la tabella delle pagine, P e P' condivideranno lo stack fino a quando uno dei due non lo modifica. Si copia quindi lo stack modificato in un frame libero.

POOL OF FREE FRAMES

In caso di PAGE FAULT, la pagina da caricare viene scritta in un frame libero mentre la pagina vittima diventa un frame libero

- In questo modo se il processo ha bisogno della pagina vittima questa è già presente in RAM e può essere linkata al processo (evitiamo così la copiatura da disco)

MEMORIA SECONDARIA - area di SWAP

- Vista come estensione della RAM ma la CPU può accedervi solo in caso di page fault

Contiene dati (stack), non codice (file system)

REALIZZAZIONE DELLA MEMORIA VIRTUALE

- ① Algoritmo di sostituzione pagine
- ② Algoritmo di allocazione di frame

Algoritmi di Sostituzione delle Pagine

La richiesta porta a PAGE FAULT:

- ① Verifica la validità
- ② Identifica un frame libero
- ③ Se lo trova:
 - Avvia caricamento pagina (processo in waiting)
 - Aggiorna la tabella delle pagine del processo
 - Processo in ready

durante la fase di fetch o execute
operazione costosa

Gestione del page fault: (tempo di scanso della RAM $\approx 10^{-3}$ secondi)

- ① Gestione dell'interruzione $\approx 10^{-6}$ secondi
- ② Lettura della pagina mancante $\approx 10^{-3}$ secondi
- ③ Riconv. del processo $\approx 10^{-6}$ secondi

Caso page fault ma nessun frame libero:

- ① Trova una pagina vittima
 - ② Effettua la sostituzione:
 - Salvo la pagina vittima su disco
 - Copia la pagina che ha causato page fault nello stesso frame
- se ha associato un dirty bit con valore 1
che sta ad indicare che la copia in RAM è
diversa da quella su disco

MECCANISMI PER SCEGLIERE LA PAGINA VITTIMA:

- FIFO - First In First Out
- Algoritmo Ottimale
- Least Recently Used - LRU
- Algoritmo di Seconda Chance
- Algoritmo con bit Supplementare di Riferimento] approssimazioni di LRU
- Least Frequently Used] - algoritmi basati su conteggio
- Most Frequently Used]

FIFO - associa ad ogni pagina un timestamp

esempio

p → $\begin{array}{ccccccccc} t_F & t_F & t_F & t_F & t_F \\ 7 & 0 & 1 & 2 & 0 & 3 & 0 & 4 & 2 \end{array}$... # pagine caricate con paginazione su richiesta

La quantità di frame che può utilizzare il process è limitata, es. MAX 3 frame



⊕ Facile da implementare

± Le prestazioni possono essere molto scarse in certi casi

⊖ I timestamp crescenti possono andare in overflow

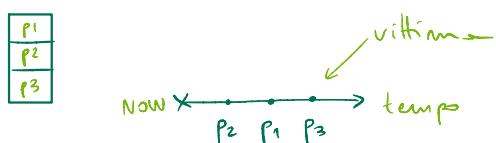
⊖ Statisticamente, soffre di ANOMALIA DI BELADY: Il numero dei page fault non decresce sempre all'aumentare dei frame



ALGORITMO OTTIMALE - OPT/MIN

Si basa sul comportamento futuro dei processi

esempio



⊕ Minimizza il numero di PAGE FAULT

- Non è implementabile

LEAST RECENTLY USED

Si parte dall'assunzione che le pagine utilizzate di recente saranno utilizzate anche nell'immediato futuro e sceglie come vittime quelle usate meno recentemente

esempio

p → $\begin{array}{ccccccccc} t_F & t_F & t_F & t_F & t_F \\ 7 & 0 & 1 & 2 & 0 & 3 & 0 & 4 & 2 \end{array}$... # pagine caricate con paginazione su richiesta



⊕ Non soffre dell'anomalia di Belady

⊖ Possibilità di overflow

⊖ Necessaria ricerca sequenziale di pagine valide (possibile stack di pagine)

ALGORITMO CON BIT SUPPLEMENTARE DI RIFERIMENTO

Soluzione che richiede un'implementazione HW

Utilizzo di un bit impostato a 1 quando viene effettuato il riferimento a una pagina e 0 quando non viene effettuato nessun riferimento per un certo LASSO DI TEMPO

esempio



Inizio con un registro di tutti zeri

000 000 00
000 000 00
000 000 00

Salvo A nella prima cella, imposta il bit ad 1, avvia un timer. Dopo un tempo t shifts il primo registro

A	1	<u>1</u> 00 000 00
	0	<u>0</u> 00 000 00
	0	<u>0</u> 00 000 00

Passa il lasso di tempo determinato dal timer e scrivo in p₂ e p₃, shifts i registri

A	0	<u>0</u> 10 000 00
B	0	<u>1</u> 00 000 00
C	0	<u>1</u> 00 000 00

Se avviene page fault vero → sostituisco la pagina alla quale corrisponde il registro col numero minore

ALGORITMO DI SECONDA CHANCE

Bit di riferimento + FIFO

Le pagine in RAM sono organizzate con code FIFO circolare.

Non definisce nessun meccanismo temporale.

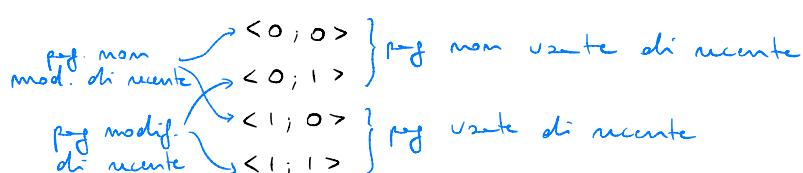
Associa ad ogni pagina solamente il bit di riferimento insieme un registro

Quando devo cercare una vittima:

(segno l'ordine FIFO)

- ① Se il bit vale 1 lo impasto a 0
- ② Se trovo un bit 0 scelgo quella pagina come vittima

IMPROVEMENT → 2bit (RIFERIMENTO; MODIFICA)



00 < 01 < 10 < 11

LEAST FREQUENTLY USED

Assegna a ciascuna pagina un contatore che conta le volte in cui una pagina è utilizzata

- ⊕ Riduce il rischio di overflow poiché il contatore cresce più lentamente
 - La pagina vittima è scelta tra le pagine con contatore più basso
 - Due page fault consecutivi possono portare all'immediata eliminazione della prima pagina caricata in quanto possiede un contatore basso, appena inizializzato
- ⊖ Sfavoreisce le pagine caricate recentemente in RAM

MOST FREQUENTLY USED

Stessa implementazione di "Least Frequently Used" ma sceglie come vittima una delle pagine più utilizzate

- > Si suppone che una pagina con contatore molto alto è in RAM da tanto tempo e a breve non gli servirà più
- ⊕ Si comporta meglio rispetto a Least Frequently Used

Algoritmi di Allocazione dei Frame

PAGINAZIONE SU RICHIESTA:
(Lazy swapping, paginazione on demand)

Assegna 0 pagine alla creazione di un processo. Quando la CPU fa riferimento a una pagina invalida, questa viene caricata.

ALTRI MECCANISMI DI PAGINAZIONE:

- L'OS assegna almeno il numero libero di frame calcolato sull'architettura per eseguire una singola istruzione

ALLOCAZIONE RAM

PROCESSI UTENTE
PROCESSI KERNEL

→ necessitano di conservare tabelle molto grandi

ALLOCAZIONE UNIFORME

Ogni processo riceve la stessa quantità di frame

ALLOCAZIONE PROPORZIONALE

Calcolo $U = UM_1 + UM_2 + \dots + UM_i + \dots + UM_n$

→ virtual memory richiede che ogni processo, sommati, per eseguire

Calcolo $m_i = \frac{UM_i}{U} M$ per calcolare il numero di frame da assegnare a ciascun processo

Se nasce un nuovo processo si riesegue l'assegnazione per redistribuire i frame (possibilmente non utilizzati) ← allocazione dinamica

ALLOCAZIONE DEI FRAME - tipologie



> LOCALE

Normalmente le richieste di una pagina da un processo con tutti i frame occupati porta a page fault

> GLOBALE

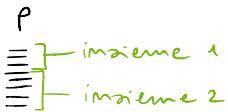
Invece di effettuare page fault, un processo a priorità più alta di un altro può "rubare" da questi un frame libero

- Rischio di Threshing:
 - Processi a priorità bassa con troppi pochi frame per lavorare senza continui page fault - CPU inutilizzata perché si aspetta continuamente la lettura dal disco
- "CPU poco utilizzata" può portare al caricamento in RAM di ulteriori processi, nonostante quelli già caricati siano già a conto di memoria. Stallo

MODELLO DEL WORKING SET

Usa il concetto di pagine attive, voglio evitare il threshing

- I processi sono insiemi di pagine diversi all'interno della propria esecuzione



- Il processo esegue senza problemi se gli assegno almeno il numero di pagine attive per ciascun insieme (WORKING SET)
ricalcolato a intervalli regolari
- Devo memorizzare le pagine alle quali il processo ha effettuato l'accesso nelle ultime 8 richieste
 $\approx 10^4$
- L'OS deve "indovinare" il momento in cui il processo cambia working set



- Se la PAGE FAULT FREQUENCY cresce si aumenta il numero di frame assegnati dall'OS

$IWSpi$: cardinalità del WF di un processo

$\sum_i IWSpi$: numero complessivo di frame necessari per tutti i processi
 serve a decidere se è necessario swappare alcuni processi

PREPAGINAZIONE - swapping

- Quando un processo viene "swapped out" dalla memoria RAM, memorizza il Working set del processo in memoria secondaria.
 Il processo viene "swapped in" quando le pagine del working set si liberano

MAPPATURA VIRTUALE

- mappatura di file
- mappatura di device

- Il processo P effettua open/read(file)

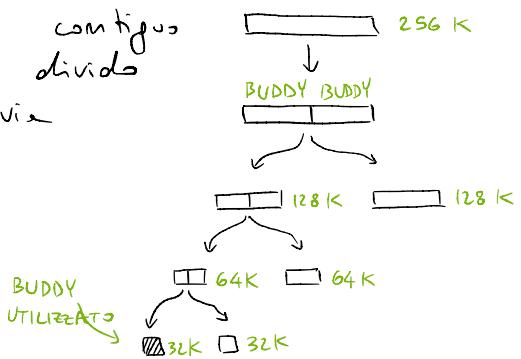


- Mella RAM c'è una porzione riservata ai device

ALLOCAZIONE RAM PER PROCESSI KERNEL

SISTEMA BUDDY (gemelli)

- inizia con un solo segmento contiguo
- se necessita di ulteriori segmenti divide
- metà il segmento e così via



- Se un buddy e' il suo gemello sono libri li ricompongo

L'efficienza di questo sistema non è ottima, es. un segmento da 65K necessita di un buddy da 128K nell'es. di prima

CACHE DI SLAB (virtuale, non fisica)

 Inizializza un array uniforme, inizialmente con tutti i segmenti delle slab con descrittori liberi
← la contiguità riduce la frammentazione interna

- Se la SLAB si riempie ne crea un'altra
- L'insieme di slab che contengono elementi dello stesso tipo prende il nome di cache