Problemi di Progettazione per Sistemi di Paging

Politiche di allocazione globali e locali

Come dovrebbe essere allocata la memoria fra i processi eseguibili concorrenti?

> **Esempio**
>
> Se A ottiene un page fault, dobbiamo sostituire solo la pagina con l'età inferiore o tutte le pagine di A?
> Se invece considerassimo tutte le pagine anche degli altri processi ed eliminassimo la pagina di B con l'età inferiore?

Questi tipi di sostituzione sono chiamati:

- **Algoritmi di sostituzione locali delle pagine** -> Assegnano a ogni processo una frazione fissa della memoria.
- **Algoritmi di sostituzione globali delle pagine** -> Assegnano i frame in maniera dinamica.

Gli algoritmi globali funzionano solitamente meglio, specie quando la dimensione sel set di lavoro può variare durante la vita di un processo.
Negli algoritmi locali se aumenta il set di lavoro ci sta più possibilità di trashing. mentre se diminuisce ci sta uno spreco della memoria.

Algoritmi globali -> Il sistema decide continuamente quanti frame assegnare ad ogni processo.

> **Come decidiamo l'assegnazione dei frame ai processi?**
>
> Monitoriamo la dimensione del set di lavoro come indicato dai bit di aging, anche se non è detto che prevengano il trashing, perché il Working set cambia in millisecondi, mentre i bit di aging sono distribuiti su un numero di cicli di clock.

> **Altro Approccio:**
>
> Usare un algoritmo per assegnare ad ogni processo dei frame della memoria, ad ognuno è assegnato lo stesso numero di frame.

⚠ Problema: Non è equo

Non ha senso assegnare a due processi di diverse dimensioni lo stesso numero di frame, quindi assegniamo un limite minimo di pagine che possono usare.

Negli algoritmi globali si può avviare un processo con un numero di pagine assegnato proporzionale alla sua dimensione e possiamo anche aggiornarle dinamicamente.

Come gestire l'allocazione: PFF

Il **Page Fault Frequency** indica quando aumentare o diminuire l'allocazione delle pagine di un processo, non dice nulla sulla pagine da sostituire.

Per molti algoritmi di sostituzione (Anche LRU) abbiamo che il numero di page fault diminuisce all'aumentare dei frame assegnati.
Ovviamente non vogliamo avere troppi page fault, perchè rallenta tutto il sistema, ma nemmeno troppi pochi, perchè significa che il processo sta probabilemente usando troppa memoria. Nel secondo caso possono essergli tolti dal PFF.

ⓘ Quale algoritmo usare?

Alcuni algoritmi di sostituzione possono funzionare sia con politiche globali che locali (FIFO, LRU).
Per alti ha senso solo una strategia locale, per esempio il working set e WSCLOCK fanno riferimento ad un insieme di lavoro specifico di un processo, non estendibile a tutta la macchina.

Controllo del carico

Anche con il miglior algoritmo di sostituzione e con l'assegnazione ottimale dei frame possiamo imbatterci nel thrashing.

Quando avviene?

Se la combinazione dei set di lavoro di tutti i processi supera la memoria a disposizione.

Di conseguenza il **PFF** indica che alcuni processi hanno bisogno di più memoria, ma nessun processo ha bisogno di meno memoria.

Non possiamo quindi dare più memoria ad un processo senza danneggiare gli altri, la soluzione al problema sta quindi nel sbarazzarsi di alcuni processi temporaneamente.

Uccidiamo qualche processo --> Usando un processo speciale: **OOM (Out Of Memory killer).**

OOM

Si attiva se il sistema è a corto di memoria. Analizza i processi e sceglie una vittima liberandone le risorse.
Più in particolare asssegna dei voti per indicare quanto ogni processo sia "cattivo", alcuni hanno sempre dei voti bassi (root) e cerca di ridurre al minimo le vittime, pur liberando memoria a sufficienza.

Una maniera meno violenta è quella di spostare qualche processo nella memoria non volatile liberando tutte le pagine che detiene per poi spartirle ai processi in thrashing.
Se il thrasing finisce il sistema continua il suo lavoro, altrimenti deve scegliere altri processi da spostare.

Scambio dei processi per diminuire il carico di memoria è simile allo scheduling a due livelli, alcuni processi sono messi nella memoria non volatile ed è usato uno scheduler a breve termine per schedulare i restanti per ottenere una numero accettabile di page fault.

Altro fattore da considerare è il grado di multiprogrammazione, quando ci sono pochi processi in memoria la CPU è inattiva, nel decidere quale processo scambiare su disco dobbiamo considerare:

- Dimensione
- Frequenza di paginazione
- Caratteristiche (I/O bound o CPU bound)

Esistono anche altre possibilità per evitare il thrashing come la **Deduplicazione**

Deduplicazione (Same Page Merging)

Si analizza la memoria e si controlla se due pagine allo lo stesso contenuto, se è vero, allora invece di mantenere i dati in due frame separati, li manteniamo solo in un uno.
Se un processo cerca di scrivere su una pagina viene generata una copia supplementare in modo tale che la scrittura non coninvolga l'altra pagina.

Si possono usare anche il **compattamento** e la **compressione** della memoria.

Policy di pulizia

> **Paging Deamon**
>
> Processo in background, in riposo la maggior parte del tempo, che ispeziona periodicamente lo stato della memoria.

Se i frame liberi sono pochi, inizia a scegliere le pagine da sfrattare tramite un algoritmo di sostituzione delle pagine. Se le pagine sono state modificate dopo il loro caricamento, vengono scritte su memoria non volatile.

> **Frame pulito**
>
> **frame di memoria fisica** che non è stato modificato dopo essere stato caricato in memoria.

I contenuti precedenti della pagina vengono ricordati, nel caso vangano richieste.
Il paging deamon garantisce che tutti i frame liberi sono puliti e quindi non devono essere scritti in memoria non volatile.

Modo per implementarlo --> Clock a due lancette, quella anteriore controllata dal deamon.

Dimensione delle pagine

E' un parametro che può essere scelto dal sistema operativo. Per trovare la migliore dimensione bisogna bilanciare alcuni fattori.

Argomentazione a favore delle pagine di piccole dimensioni:

- Qualunque segmento di testo occupa in media metà della pagina finale, l'altra metà è sprecata; Si parla di **frammentazione interna.**
- Generalmente pagine di dimensioni più grandi sprecano più spazio.

> **Un pò di matematica**
>
> n segmenti in memoria e una dimensione di pagine p, $np/2$ byte sono sprecati dalla frammentazione interna.

> **⚠ Problema delle pagine piccole**

Richiedono una tabella delle pagine estesa, il tempo di trasferimento già richiede molto tempo, inoltre se inseriamo troppe pagine la ricerca diventa complicata.
Inoltre occupano molto spazio nel TLB.

Conviene usare pagine di dimensioni grandi specialmente per migliorare le prestazioni del TLB.

Per avere un buon equilibrio il SO può scegliere di usare in certe situazioni pagine grandi (Kernel) o pagine piccole (processi utente).

Istruzioni separate e spazi dei dati

Maggior parte dei computer ha un solo spazio delii indirizzi contenente sia dati che programmi, se abbastanza grande, tutto funziona bene, ma a volte è troppo piccolo.

Una soluzione è separare lo spazio degli indirizzi delle istruzioni (**I-space**) da quello dei dati (**D-space**).

Oggi si usa ancora questa tecnica specialmente in memorie di piccole dimensioni (Chace, TLB).

Pagine condivise

E' comune che alcuni utenti utilizzino lo stesso programma o librerie, quindi è più efficiente condividere pagine di memoria tra questi processi rispetto a mantenere le copie.

Non tutti i tipi di pagine sono condivisibili:

- Pagine di sola lettura ---> SI.
- Pagine dei dati ---> NO.

I processi usano la stessa tavella delle pagine per gli I-space, ma tabelle diverse per i D-space.

Nella tabella dei processi di ogni processo ci sono due puntatori per I-space e il D-space. Quando lo scheduler sceglie il processo da eseguire, usa questi puntatori per determinare le tabelle delle pagine corrette e imposta l'MMU.

⚠ **Problemi**

La rimozione di un processo da memoria può creare numerosi page fault in un altro processo che condivide le pagine.

La condivisione dei dati è più complessa di quella del codice.

> ⓘ **In UNIX:**
>
> Dopo una fork genitore e figlio condividono sia il testo del programma che i dati solo come lettura.

Se un processo modifica i dati, si genera una trap e viene creata una copia della pagina modificata. Entrambe diventano READ/WRITE, inoltre in questo modo evitiamo di copiare pagine che non verrano mai modificate. Questa tecnica si chiama **Copy-On-Write (COW)**.

Librerie condivise

I moderni sistemi operatici utilizzano grandi librerie usate da molti processi, collegare tutte queste librerie su memoria non volatile le renderebbe ingombranti.
Per questo motivo vengono usate le **DDL (Dynamic Link Library)**.

Il problema principale è che programmi diverse potrebbero aver bisogno della stessa libreria, ma questa è posizionata in indirizzi diversi nei processi. Questo impedisce l'uso di indirizzi assoluti nelle istruzioni.

La soluzione consiste nel compilare le librerie condivise con un offset piuttosto che puntare ad indirizzi specifici, compilandole tramite indirizzi relativi.

File mappati

Librerie condivise -> Caso speciale di un concetto chiamato **memory-mapped file.**

> ⓘ **Memory-mapped file**
>
> L'idea che un processo può inviare una chiamata di sistema per mapparre un file all'interno di una porzione di un suo spazio degli indirizzi virtuali.

> ⓘ **Funzionamento**
>
> Alla mappatura, nessuna pagina viene caricata immediatamente. Sono paginate dalla memoria non volatile man mano che vengono toccate.
> Quando un processo termina o elimina la mappatura, le pagine modificate vengono riscritte sul file.

Il vantaggio è che offre un modo diverso per esguire **I/O**, permettendo di accedere al file come se fosee un grande array di caratteri in memoria.
Se più processi mappano lo stesso file possono comunicare tramite questa memoria condivisa.