CAPITOLO 3

#Sistemi_Operativi

Memoria Virtuale

La Memoria Virtuale permette al sistema di utilizzare lo spazio nel disco rigido come se fosse una parte della memoria RAM, creando così l'illusione di avere un quantitativo di memoria più grande di quella fisicamente disponibile.
Le conseguenze immediate sono la possibilità di elaborare una quantità di dati più grande dello spazio libero in RAM e di avere processi più grandi dello spazio libero in RAM.

Ogni processo ha un proprio spazio degli indirizzi virtuale , privato e isolato.
Questo rappresenta un insieme di indirizzi che un processo può usare per indirizzare la memoria.

Un processo non può leggere o scrivere nella memoria di un altro processo. Lo spazio degli indirizzi è suddiviso in pagine , dove ogni pagina è un intervallo di indirizzi contigui. Le pagine sono mappate nella memoria RAM, ma per eseguire un processo non è indispensabile che tutte le pagine siano contemporaneamente in RAM (solo quelle che utilizzerà il processo).

Se un programma accede a una parte del suo spazio di indirizzi virtuale che si trova nella memoria fisica, allora l'hardware esegue automaticamente la mappatura necessaria.
Se invece il programma tenta di accedere a una parte del suo spazio di indirizzi che non si trova in memoria fisica, allora avviene un Page Fault.
Quindi con Page Fault si intende l'evento per cui un processo tenta di accedere a una pagine di memoria che non è attualmente in RAM.

Paginazione

Il Paging è una tecnica di divisione della memoria virtuale e fisica in blocchi della stessa dimensione, detti pagine (virtuale) e frame (fisica).
Ogni processo ha una Tabella delle Pagine che mappa le pagine virtuali ai frame fisici, questa tabella contiene quindi le informazioni su quale frame fisico contiene la pagina virtuale richiesta.
La tabella si aggiorna ogni qual volta avviene un Page Fault.
(Swapping) Quando la memoria fisica è piena, il SO può spostare alcune pagine nella RAM al disco rigido in un'area chiamata file di swap o page file.

E' la MMU (Memory Management Unit) la componente hardware che si occupa di gestire l'accesso alla memoria fisica da parte dei processi in esecuzione, traducendo gli indirizzi virtuali utilizzati dai programmi nei corrispondenti indirizzi fisici della RAM.

Processo di traduzione degli indirizzi

1. Indirizzo Virtuale : Quando un processo accede in memoria, anzitutto il processore genera un indirizzo virtuale.
   Questo indirizzo è diviso in due parti: numero di pagine e offset.
2. Tabella delle Pagine : Successivamente la MMU, responsabile della traduzione degli indirizzi virtuali in indirizzi fisici, utilizza il numero di pagina dell'indirizzo virtuale come indice nella Tabella delle pagine.
3. Frame : Se la pagine è presente in memoria fisica (RAM), la MMU recupera il numero di frame corrispondente.
4. Offset : La MMU combina il numero di frame e l'offset ottenendo così l'indirizzo fisico completo, che indica la posizione esatta in memoria RAM.
5. Accesso : Può quindi avvenire l'accesso da parte del processo in memoria.
   La tabella delle pagine è organizzata come una struttura dati ad accesso diretto ed ogni voce contiene informazioni essenziali per la traduzione degli indirizzi:

- Numero del Frame
- Bit presente/assente : indica se la voce è valida o meno. E' 0 se la pagina virtuale a cui si fa riferimento non è in memoria, e quindi il suo accesso porterebbe a un Page Fault
- Bit di protezione
- Bit modificato e di riferimento (M e R): necessari per tener traccia dell'uso della pagina.
- N.B. La tabella delle Pagine si trova in RAM.

In generale, la maggior parte dei processi tende a fare molteplici riferimenti soltanto a un piccolo numero di pagine, pertanto solo una piccola parte delle voci della tabella viene letta frequentemente.

La soluzione a questo problema è il caching : si dota l'MMU di un piccolo dispositivo hardware, il TLB (Translation Lookside Buffer) che mappa gli indirizzi virtuali su quelli fisici senza passare per la Tabella delle Pagine. La TLB ha un numero ridotto di voci, quelle più utilizzate, e in questo modo si velocizza la traduzione.

Funzionamento della TLB : quando un indirizzo virtuale è presentato alla MMU per la traduzione, anzitutto l'hardware verifica se il numero di pagine è presente nella TLB confrontandolo con tutte le voci:

- Se ha un riscontro valido (TLB Hit) e l'accesso non viola i bit di protezione, il frame è prelevato direttamente dalla TLB senza passare per la Tabella delle Pagine in memoria.
- Se non ha un riscontro valido (TLB Miss) la MMU accede alla Tabella delle Pagine in memoria. Viene cancellata una delle voci nella TLB per rimpiazzarla con la voce della tabella delle pagine appena trovata. Esistono due tipi di miss:

1. Soft Miss : la pagina di riferimento non è nella TLB ma in memoria.
2. Hard Miss : la pagina di riferimento non è nemmeno in memoria, è necessario accedere al disco per prelevarla.

Algoritmi di sostituzione delle pagine

Quando si verifica un page fault, per far spazio alla pagine entrante il sistema operativo deve scegliere una pagina da sfrattare (rimuovere dalla memoria).

- Algoritmo ottimale (irrealizzabile): questo algoritmo si basa sul principio teorico di sapere quando ciascuna pagina sarà necessaria in futuro. In particolare, viene etichettata ogni pagina con il numero di istruzioni che devono avvenire prima che quella pagina sia nuovamente disponibile. Nel caso di un page fault, ossia se un processo tenta di accedere a una pagina non presente in memoria RAM, allora una delle pagine in RAM deve essere rimossa per far spazio alla pagina cercata. Con questo algoritmo il numero di page faults è minimizzato poiché viene rimossa dalla RAM, ad ogni page fault, la pagina con etichetta più alta, ossia quella che non sarà necessaria per tempo più lungo. Tuttavia, poiché non è possibile per il SO sapere quando sarà effettuato l'accesso ad una pagina in memoria in futuro, questo algoritmo è irrealizzabile.

In generale sappiamo che, nei calcolatori con memoria virtuale, ad ogni partita è associato un bit R (referral) che viene impostato quando si fa riferimento alla pagina ed M (modify) quando la pagina viene scritta e quindi modificata.

- L'algoritmo NRU (Not Recently Used) si basa proprio su questi due bit. All'avvio di un processo entrambi i bit di pagina di tutte le pagine legate al processo sono impostati a 0. Periodicamente il bit R è ripulito, per contraddistinguere pagine che non hanno avuto riferimenti recentemente da quelli che ne hanno avuti. Quando avviene un Page Fault, il SO controlla tutte le pagine e le divide in 4 categorie in base agli attuali valori R e M:
  - Classe 0 : nessun riferimento e nessuna modifica.
  - Classe 1 : nessun riferimento, modificata.
  - Classe 2 : riferimento, non modificata.
  - Classe 3 : riferimento, modificata.

  A questo punto NRU rimuove una pagina a caso appartenente alla classe non vuota con numero più basso (rimuove una pagina dalla classe 0 se esiste almeno una pagina nella classe 0), in questo modo si assicura di rimuovere dalla RAM la pagina dal minor utilizzo recente.
  L'algoritmo NRU richiede solo la gestione di due bit per pagina, il che lo rende relativamente semplice e veloce da implementare. Tuttavia, poiché la scelta della pagina all'interno di una classe casuale, l'algoritmo potrebbe non sempre selezionare la pagina migliore per minimizzare i futuri page faults.

- Algoritmo FIFO (First-In, First-Out): FIFO è un algoritmo di paginazione che elimina la pagina più vecchia in memoria. Il sistema operativo rimuove la pagina in testa alla lista (la più vecchia) durante un page fault, aggiungendo la nuova pagina in coda.
- Seconda Chance : una semplice modifica a FIFO che evita il problema di gettare una pagina usata di frequente consiste nel controllare il bit R della pagina più vecchia. Se è 0, la pagina è vecchia e inutilizzata e viene così sostituita immediatamente. Se R è 1, il bit viene azzerato, la pagina è posta in fondo all'elenco e il momento in cui è stata caricata in memoria viene aggiornata per farla sembrare appena arrivata.
- Clock (funzionamento) : lista circolare dei frame di pagina con un puntatore simile ad una lancetta di un orologio per identificare la pagina più vecchia. Quando avviene un page fault, la pagina indicata dalla lancetta viene controllata. Se il suo bit R è 0 viene sfrattata, la nuova pagina viene inserita al suo posto nell'orologio e la lancetta viene spostata in avanti di una posizione. Se R è 1, viene azzerato e la lancetta passa alla pagina successiva. Questo processo è ripetuto finché viene trovata una pagina con R=0. E' più efficiente rispetto a seconda chance e FIFO.
- LRU (Least Recently Used) : L'idea dell'algoritmo è di sfrattare la pagina rimasta inutilizzata per più tempo. Per implementare tale algoritmo è necessario tenere in memoria una lista concatenata di tutte le pagine, con quelle più usate in testa e quelle meno usate in coda.
- Demand Paging : rappresenta una tecnica di gestione della memoria che carica le pagine in memoria solo quando sono necessarie, cioè su richiesta.
- Si parla invece di Working Set quando si fa riferimento all'insieme di pagine di memoria che il processo utilizza attivamente durante la sua esecuzione. Molti sistemi di paginazione cercano di tenere traccia del lavoro di ciascun processo e di accertarsi che sia in memoria prima di consentirne l'esecuzione. Questo approccio è detto Working Set Model ed è stato progettato per ridurre la frequenza di Page Faults.
- Il Thrashing rappresenta una condizione critica nei SO in cui il sistema è occupato principalmente nello scambio di pagine tra memoria principale (RAM) e memoria di massa (disco rigido o SSD). Ciò avviene a causa di Page Fault frequenti e quindi di un Working Set dalla dimensione maggiore rispetto allo spazio disponibile in memoria.

Progettazione dei sistemi di Paginazione

Esistono due principali approcci:

- Allocazione Locale : ogni processo riceve una porzione fissa della memoria fisica disponibile, che viene divisa in pagine.
  In caso di Page Fault (cioè il processo tenta di accedere ad una pagina non presente in memoria), l'algoritmo di sostituzione delle pagine sceglie una delle pagine già allocate da quel processo per essere sostituita. Quindi il processo può sostituire solo le proprie pagine, e non quelle degli altri processi.

Si ha il vantaggio che è facile da implementare e garantisce che ogni processo riceva una porzione specifica in memoria, ma lo svantaggio che può portare a una sotto-utilizzazione della memoria se un processo non utilizza le pagine che gli sono state allocata.

- Allocazione Globale: la memoria viene gestita in modo dinamico e condivisa tra tutti i processi.
  In caso di Page Fault quindi l'algoritmo può scegliere di sfrattare una pagina di un qualsiasi processo, e non solo di quello che ha generato il Page Fault.
  Si ha un uso ben più efficiente della memoria rispetto alla allocazione locale, ma complesso da implementare (da gestire il meccanismo per scegliere il processo che cederà la pagina).

Un modo per gestire l'allocazione è l'uso dell'algoritmo PFF (Page Fault Frequency).
Questo algoritmo monitora la frequenza con cui un processo genera Page Fault. Se avviene frequentemente, significa che quel processo ha maggiormente bisogno di memoria e quindi il SO gli assegnerà più frame. Al contrario, in caso di Page Fault più scarsi, il numero di frame assegnati al processo sarà gradualmente ridotto.

Dimensione delle pagine : con pagine più piccole, si ha una riduzione della Frammentazione interna dell'uso della memoria (ossia quando la memoria utilizzata da un processo è leggermente superiore rispetto a quella necessaria, portando a sprechi).
Ciò è vantaggioso per programmi che richiedono l' utilizzo di meno memoria rispetto alla dimensione di una pagina grande, poiché la frammentazione è ridotta così come lo spreco di memoria.
Tuttavia pagine più piccole comportano una Tabella delle Pagine più grande con più voci per identificarle tutte, e quindi un overhead di traduzione maggiore.
La dimensione delle pagine ottimale si ottiene quindi bilanciando la riduzione della frammentazione interna e l'incremento dell'overhead della tabella delle pagine (pagine più grandi = minor overhead ma maggior frammentazione e viceversa).

Condivisione delle pagine : è comune che i processi condividano pagine di memoria, soprattutto quando i tratta di codice eseguibile (che tipicamente non è modificabile e quindi sicuro da condividere).
Questa tecnica è efficiente perché evita la duplicazione del codice in memoria, permettendo a più processi di eseguire lo stesso programma utilizzando la stessa memoria.
Per fare ciò, lo spazio degli indirizzi di un processo è diviso in:

- I-Space (Instruction Space): è lo spazio riservato alle istruzioni del programma, ossia codice eseguibile. E' tipicamente condiviso solo se non modificabile.
- D-Space (Data Space): è lo spazio degli indirizzi riservato ai dati del programma. A differenza dell'I-Space, il D-Space non è solitamente condiviso a meno che non si implementi il meccanismo specifico di COW (Copy-on-Write), per cui non appena un

processo vuole scrivere sui dati del programma condiviso viene creata una copia lasciando inalterato lo spazio originario per gli altri processi.

Con la condivisione delle pagine il problema principale è che se un processo che condivide pagine di memoria viene terminato, potrebbero avvenire Page Fault negli altri processi che ancora necessitavano delle sue pagine condivise.

File mappati in memoria : si tratta di una tecnica che consente ai processi di accedere ai file su disco come se in realtà fossero in RAM.
Ciò è possibile mappando il contenuto del file su una regione della memoria virtuale del processo.
In pratica, quando un file viene mappato in memoria, il SO crea un'associazione tra file su disco e una specifica area della memoria virtuale del processo. In questo modo quando il processo legge o scrive in quella data regione di memoria, in realtà lo sta facendo sul disco.
Con questa tecnica è inoltre possibile condividere tra più processi la stessa regione di memoria associata a un file.

SO e Paginazione

Quando il SO deve gestire la paginazione:

- Creazione del processo : Il SO determina anzitutto la dimensione inziale del programma e dei dati, inizializzando la tabella delle pagine che mappa gli indirizzi virtuali con quelli fisici. Alloca poi uno spazio sulla memoria non volatile per lo swapping (usato se lo spazio disponibile in RAM è insufficiente), l'area viene inizializzata e le informazioni del processo salvate nella tabella dei processi PCB.
- Esecuzione del processo : il SO configura l'MMU, se necessario svuota la TLB e rende attiva la tabella delle pagine del processo.
- Gestione dei Page-Fault : in caso di Page Fault il SO deve individuare l'indirizzo virtuale che l'ha causato, trovare la pagina necessaria nella memoria non volatile (spazio di swap), scegliere un frame disponibile in RAM (eventualmente sfrattando un'altra pagina), inserirvi la pagina richiesta e ripristinare il PC.
- Chiusura del processo : il SO libera la tabella delle pagine del processo, dealloca le pagine in RAM e libera lo spazio di swap riservato al processo per lasciare queste risorse agli altri.

Gestione dei Page Fault

1. Interruzione: Quando si verifica un Page Fault, l'hardware genera una trap (interruzione) nel kernel che porta il controllo al kernel del SO. Il PC viene salvato nello stack del processo per poter riprendere successivamente l'esecuzione.
2. Salvataggio di contesto: L'ISR salva lo stato dei registri e altre informazioni critiche per preservare lo stato del processo.

3. Il SO determina quale pagina virtuale è stata richiesta e ha causato il Page Fault.
4. Validazione: il SO verifica se l'indirizzo di memoria richiesto è valido e se l'accesso è permesso, in tal caso cerca un frame libero in RAM per caricare la pagina.
5. Pagina sporca: Se il frame da liberare contiene una pagina sporca (cioè pagina modificata ma non ancora salvata su disco), il SO programma la scrittura della pagina sul disco per evitare la perdita di dati. Nell'arco di questo periodo il processo è sospeso.
6. Una volta liberato il frame la pagina è caricata dal disco in RAM.
7. Viene poi aggiornata la tabella delle pagine.
8. Ripristino dell'istruzione: l'istruzione che ha causato Page Fault viene riportata allo stato iniziale, pronto per essere eseguito ora che la pagina è in memoria.
9. Riprese dell'esecuzione: il processo che ha causato il Page Fault viene schedulato per riprendere l'esecuzione.
10. Ripristino del contesto: L'ISR ricarica i registri e le altre informazioni di stato precedentemente salvate per garantire che il processo riprenda correttamente l'esecuzione.

Segmentazione

Si tratta di un'altra tecnica di gestione della memoria.
Questa suddivide la memoria in segmenti logici distinti. Ogni segmento rappresenta un'unità di lavoro come una funzione, un modulo di dati o un insieme di istruzioni.
A differenza della paginazione, dove la memoria è suddivisa in pagine, nel caso della segmentazione i segmenti possono assumere lunghezze variabili.

In un sistema segmentato infatti un indirizzo di memoria è composto da un numero di segmenti e un offset all'interno di esso. Ciò consente al sistema di gestire ogni segmento in modo indipendente, comportando maggiore efficienza e flessibilità.
I vantaggi della segmentazione sono:

- Flessibilità: i segmenti possono crescere o ridursi in modo indipendente uno dall'altro.
- Semplificazione del Linking: il Linking (ossia il processo di combinazione dei vari moduli compilati in un unico eseguibile) diventa più semplice dal momento che ogni processo lavora in un segmento separato e pertanto non è necessario riassegnare gli indirizzi di memoria all'interno di ogni modulo (cosa che avveniva invece col paging e blocchi di indirizzi contigui).
- Condivisione e Protezione: la segmentazione facilita la condivisione e protezione delle risorse tra processi (due processi possono ad esempio condividere un segmento di codice comune senza interferire con i dati l'uno dell'altro).

Gestione della memoria libera

Esistono due modi per tener traccia del fatto che una porzione di memoria sia utilizzata o meno : BitMap e Liste.

- BitMap: la memoria è divisa in molteplici unità di allocazione, ad ognuna di esse corrisponde un bit delle bitmap. Se è 0 l'unità è libera, altrimenti è già occupata. La dimensione della BitMap dipenderà quindi solo dalla quantità di memoria totale e dalla dimensione delle unità di allocazione (minore l'unità, maggiore la bitmap perché servono più bit). Quando si cercano di allocare k unità di memoria, allora è necessario trovare una sequenza di k bit consecutivi a 0 nella bitmap, che rappresenta k unità di memoria libere. Questa ricerca può essere lenta se la bitmap è grande e dispersa.
- Liste : in questo approccio la memoria è rappresentata come una lista concatenata di segmenti, dove ogni segmento può essere libero o occupato da un processo. Ogni segmento contiene la informazioni sull'indirizzo di partenza, la lunghezza del segmento e un puntatore al segmento successivo della lista.

Frammentazione interna

Si verifica quando viene allocato un blocco di memoria che è più grande di quanto richiesto dal processo. La memoria in eccesso all'interno del blocco allocato rimane inutilizzata e sprecata.

Frammentazione esterna

Si verifica quando la memoria libera è suddivisa in tanti piccoli blocchi non contigui, rendendo difficile o impossibile allocare memoria ai processi, anche se la somma totale di memoria libera sarebbe sufficiente.

Schemi di allocazione della memoria

Associati tendenzialmente a liste:

- First Fit : il gestore della memoria scorre i segmenti finché non trova uno spazio di allocazione abbastanza grande.
- Next Fit : Simile al First Fit, ma invece di ricominciare la ricerca dall'inizio della lista riprende da dove si era fermato l'ultima volta. Ciò potrebbe distribuire meglio la frammentazione, ma potrebbe non essere molto efficiente se la lista è lunga.
- Best Fit : cerca il blocco di memoria libero più piccolo che sia comunque sufficientemente grande per soddisfare la richiesta. Riduce la frammentazione rispetto al First Fit, ma potrebbe essere più lento dal momento che scorre tutta la lista.
- Worst Fit : opposto al Best Fit, l'idea è di lasciare grandi blocchi di memoria liberi, ma può portare a una maggiore frammentazione.
- Buddy Allocation: la memoria viene considerata come un unico grande blocco che è una potenza di 2. Quando un processo richiede memoria, il sistema ricerca un blocco di

dimensioni appropriate. Se non è disponibile, allora il sistema cerca un blocco più grande e lo divide in due blocchi uguali (buddies). La divisione continuerà finché il blocco non sarà della dimensione appropriata per il processo. I blocchi della stessa dimensione sono detti buddies poiché, quando il processo lascerà libero il blocco, questi potranno riunirsi in un blocco più grande. La buddy allocation riduce la frammentazione esterna, ma in caso un processo non richiede quantità di memoria pari a potenza di 2 provocherà delle frammentazioni interne.

Es. (Buddy Allocation):
Richiesta da un processo di 18 KB:

- Se il sistema ha un blocco da 64 KB disponibile, ma nessun blocco da 32 KB (la dimensione più vicina a 18 KB che è una potenza di 2),divide il blocco da 64 KB in due blocchi da 32 KB.
- Quindi prende uno dei blocchi da 32 KB e lo divide ulteriormente in due blocchi da 16 KB ciascuno.
- Alla fine, allocherà il blocco da 32 KB al processo, lasciando un blocco di 32 KB libero per altre allocazioni.