Sistemi Operativi

Ionut Zbirciog

21 November 2023

1 Problemi di Progettazione dei Sistemi di Paginazione

Allocazione Globale vs Locale

Allocazione Locale

Ogni processo riceve una porzione fissa della memoria. Quando avviene un page fault, l'algoritmo di sostituzione delle pagine andrà a sfrattare una delle pagine allocate al processo. Semplice da implementare, ma può portare a inefficienze se il set di lavoro aumenta o diminuisce, portando al thrashing.

Allocazione Globale

La memoria viene distribuita in modo dinamico tra i processi. Quando avviene un page fault, l'algoritmo di sostituzione delle pagine può scegliere di sfrattare anche una pagina allocata a un processo diverso. È più efficace per adattarsi alle esigenze variabili dei processi, ovvero quando la dimensione del set di lavoro varia nel tempo, ma richiede una gestione più complessa. Con l'allocazione globale, il sistema operativo deve dinamicamente assegnare e riassegnare frame ai processi. È possibile usare i bit di aging per monitorare la frequenza di accesso alle pagine, anche se non sono sufficienti contro il thrashing, poiché offrono una stima approssimativa che potrebbe non riflettere cambiamenti rapidi nel set di lavoro.

Un altro approccio è di avere un algoritmo per allocare frame ai processi assegnando un numero uguale di frame a ciascun processo (allocazione equa). Anche se sembra equo, in realtà non lo è perché significherebbe assegnare lo stesso numero di pagine ad un processo da 10KB e ad uno di 300KB. Sarebbe più opportuno allocare le pagine in proporzione della dimensione del processo (allocazione proporzinale). Questa soluzione rispecchia meglio la necessità di memoria, evitando allocazioni inadeguate.

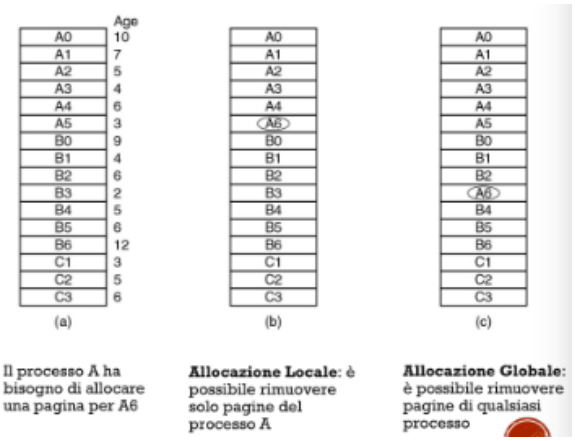

Figure 1: Allocazione statica vs Allocazione dinamica

Usando un algoritmo di allocazione dinamco è possibile avviare ogni processo con un certo numero di pagine proporzinale alla sua dimensione che dovranno essere gestite in modo dinamico in base alle esigenze del processo. Un modo per gestire l'allocazione è usare l'algoritmo PFF (Page Faul Frequency), monitora la frequenza dei page fault per regolare l'allocazione di memoria di un processo, aumentando i frame se i page fault sono troppo frequenti, diminuisce se sono rari.

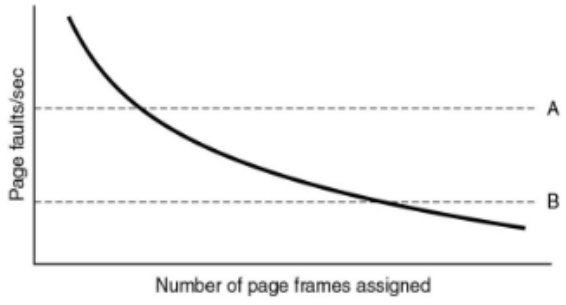

Figure 2: Numero di pagine assegnate

- A: Alta frequenza di page fault indica necessità di più frame.
- B: Bassa frequenza di page fault suggerisce che il processo ha più memoria del necessario.

Controllo del Carico

Anche con il miglior algoritmo, il thrashing può sempre verificarsi se i set di lavoro di tutti i processi eccedono la memoria disponibile.

- **Out Of Memory Killer (OOM):** processo di sistema che seleziona e termina i processi in base a un punteggio di "cattiveria" per liberare memoria. Sono selezionati di solito i processi con elevato utilizzo della memoria e minor importanza.
- **Swapping:** sposta i processi su memoria non volatile, liberando le loro pagine per altri processi, riducendo la richiesta di memoria senza interrompere i processi. Solo una parte dei processi in memoria non volatile è schedulata attivamente, aiutando a gestire meglio il carico della memoria; è utile anche per ridurre l'occupazione della memoria da processi eseguiti in background.
- Oltre a uccidere o spostare processi, si possono usare compattamento, compressione o deduplicazione.

Policy di Pulizia

Per garantire una certa abbondanza di pagine libere, i sistemi dispongono di un processo in background detto *paging daemon*, che è a riposo per la maggior parte del tempo ma viene risvegliato periodicamente per ispezionare lo stato della memoria. Quando i frame liberi scarseggiano, inizia a selezionare pagine da rimpiazzare utilizzando un algoritmo di sostituzione delle pagine. Un modo per implementare questa policy di pulizia è attraverso un orologio a due lancette. La lancetta anteriore (gestita dal daemon paging) avanza scrivendo le pagine sporche in memoria non volatile. La lancetta posteriore si occupa della sostituzione delle pagine.

Dimensione delle Pagine

Vantaggi Pagine Piccole

Riduzione della frammentazione interna e dell'utilizzo di memoria, poiché un programma potrebbe richiedere meno memoria con pagine piccole.

Svantaggi Pagine Piccole

Richiedono tabelle delle pagine più grandi e possono aumentare il tempo e lo spazio necessari per il trasferimento di dati e la gestione della memoria.

Alcuni sistemi operativi utilizzano pagine di diverse dimensioni per parti diverse del sistema, ad esempio, pagine più grandi per il kernel. Alcuni sistemi operativi effettuano salti mortali spostando la memoria di un processo per creare intervalli contigui adatti a una pagina grande (THP, Transparent Huge Pages).

Dimensione Ottimale e Calcolo della Dimensione Ottimale

La dimensione ottimale è determinata bilanciando la frammentazione interna e l'overhead della tabella delle pagine. Bisogna tenere conto della dimensione media del processo (s byte), della dimensione della pagina (p byte) e della dimensione di ogni voce nella tabella delle pagine (e byte).

L'overhead è dato da $\frac{s}{p} + \frac{p}{2}$, dove il primo termine (tabella delle pagine) aumenta con pagine più piccole, e il secondo termine (frammentazione interna) aumenta con pagine più grandi. L'ottimo si trova bilanciando questi due fattori. Derivando la funzione di overhead rispetto a p e ponendola uguale a zero si ottiene: $-\frac{s}{p^2} + \frac{1}{2} = 0$. Da cui si ottiene la dimensione ottimale $p = \sqrt{2se}$. Per esempio, con $s = 1$ MB e $e = 8$ byte, la dimensione ottimale è 4 KB, la dimensione comune attuale.

Istruzioni Separate e Spazi di Dati

È comune che molti utenti eseguano lo stesso programma o utilizzino le stesse librerie. Condividere pagine di memoria tra questi processi è più efficiente che mantenere copie separate. Per facilitare la condivisione, è meglio separare lo spazio degli indirizzi di un processo in:

- **I-Space:** spazio degli indirizzi delle istruzioni.
- **D-Space:** spazio degli indirizzi dei dati.

Ogni spazio degli indirizzi va da 0 a un certo massimo, tipicamente $2^{16} - 1$ o $2^{32} - 1$. Il linker deve sapere quando sono utilizzati I-Space e D-Space separatamente, poiché in questo caso i dati sono riposizionati all'indirizzo virtuale 0 anziché partire dopo il programma.

Pagine e Librerie Condivise

Condivisione delle Pagine

È possibile condividere solo le pagine di sola lettura, come il testo dei programmi e non le pagine dei dati. Processi diversi possono utilizzare la stessa tabella delle pagine per l'I-Space, ma tabelle diverse per il D-Space. Ciascun processo ha puntatori sia all'I-Space che al D-Space. Lo scheduler utilizza questi puntatori per impostare l'MMU. Questo meccanismo di condivisione delle pagine ha un grande problema quando un processo viene rimosso, può causare numerosi page fault in un altro processo che condivide le stesse pagine. In UNIX, dopo l'esecuzione di un fork, genitore e figlio condividono sia il testo che i dati, inizialmente come sola lettura. Se un processo modifica i dati, si genera una trap, e viene creata una copia della pagina modificata (*Copy on Write*). Questo metodo evita la copia di pagine che non vengono modificate.

Condivisione delle Librerie

I sistemi operativi condividono automaticamente tutte le pagine di testo di un programma avviato più volte. Se un processo modifica una pagina di dati condivisa, occorre applicare "copy on write". Nei sistemi operativi, esistono molte grandi librerie usate da molti processi, ad esempio, librerie di I/O e grafiche. Perciò collegare tutte queste librerie a ogni programma eseguibile lo renderebbe ancora più ingombrante di quanto già lo sia. Una tecnica è quindi usare librerie condivise, chiamate DLL (Dynamic Link Library) in Windows o file .so (Shared Objects) in UNIX.

Le librerie condivise possono essere posizionate a indirizzi diversi nei vari processi. Questo impedisce l'uso di indirizzi assoluti nelle istruzioni. Pertanto, le librerie condivise vengono compilate con indirizzi relativi anziché assoluti.

File Mappati in Memoria

I file mappati in memoria si riferiscono a una tecnica utilizzata nei sistemi operativi per consentire ai processi di accedere direttamente ai file su disco come se fossero memorizzati nella memoria principale del computer. Questa tecnica è comunemente utilizzata per ottimizzare l'accesso ai dati e migliorare le prestazioni del sistema. Quando un file viene mappato in memoria, una porzione del file viene associata a una regione di memoria virtuale. In pratica, il sistema operativo crea un collegamento tra il file su disco e una porzione della memoria virtuale del processo. Ciò consente al processo di accedere direttamente ai dati del file come se fossero presenti in memoria, senza dover leggere o scrivere direttamente dal disco ogni volta che è necessario accedere ai dati. I file mappati forniscono un modello alternativo per l'I/O. Anziché eseguire letture e scritture, si può accedere al file come a un grande array di caratteri in memoria.

L'utilizzo dei file mappati in memoria può portare a una maggiore efficienza nell'accesso ai dati, poiché il sistema operativo può gestire in modo più intelligente il caricamento e il salvataggio dei dati tra la memoria e il disco. Inoltre, i file mappati in memoria consentono di condividere dati tra processi, poiché più processi possono mappare la stessa regione di memoria associata a un file.

2 Problemi di Implementazione

Il Sistema Operativo e la Paginazione

I momenti in cui il sistema operativo deve svolgere attività correlate alla paginazione sono quattro:

Creazione del Processo

Il sistema operativo deve determinare le dimensioni iniziali del programma e dei dati, creare e inizializzare la tabella delle pagine, allocare spazio nella memoria non volatile per lo scambio, inizializzare l'area di scambio e registrare informazioni nella tabella dei processi.

Esecuzione del Processo

Il sistema operativo deve azzerrare la MMU, se necessario svuotare la TLB, rendere attiva la tabella delle pagine del processo e, opzionalmente, caricare alcune pagine in memoria per ridurre i page fault iniziali (pre-paginazione).

Gestione dei Page Fault

Il sistema operativo deve determinare l'indirizzo virtuale che ha causato il page fault, trovare la pagina necessaria nella memoria non volatile, scegliere un frame disponibile, eventualmente sfrattando pagine vecchie, caricare la pagina nel frame e ripristinare il contatore di programma.

Chiusura del Processo

Il sistema operativo deve rilasciare la tabella delle pagine, le pagine in memoria e lo spazio su memoria non volatile, gestire le pagine condivise con altri processi, rilasciandole solo dopo l'ultimo utilizzo.

Gestione dei Page Fault

1. L'hardware esegue la trap nel kernel, salvando il contatore del programma nello stack. Nella maggior parte delle macchine, alcune informazioni sullo stato dell'istruzione corrente vengono salvate in registri speciali della CPU.

2. Viene avviata una routine di servizio interrupt in codice assembly che salva i registri e altre informazioni volatili per evitare che il sistema operativo li elimini, poi chiama il gestore dei page fault.

3. Il sistema operativo determina quale pagina virtuale manca, se non disponibile nei registri hardware, recuperando la pagina analizzando l'istruzione dal program counter.

4. Il sistema controlla che l'indirizzo sia valido e che la protezione sia coerente con l'accesso. Se non lo è, viene mandato un segnale al processo o quest'ultimo viene terminato. Se l'indirizzo è valido e non è avvenuto alcun errore di protezione, il sistema verifica se c'è un frame libero. Se non vi sono frame liberi, viene eseguito un algoritmo di sostituzione delle pagine per liberarne uno.

5. Se la pagina è sporca, viene schedulata per la scrittura in memoria non volatile e il processo è sospeso consentendo l'esecuzione di un altro processo nel mentre avviene il swapping.

6. Una volta liberato, il frame viene usato per caricare la pagina necessaria da disco o SSD. Durante il caricamento della pagina, il processo in page fault è ancora sospeso e viene eseguito, se disponibile, un altro processo utente.

7. Quando l'interrupt del disco o dell'SSD indica che è arrivata la pagina, le tabelle delle pagine vengono aggiornate in modo da riflettere la sua posizione e il frame è contrassegnato in stato normale.

8. L'istruzione in errore è riportata allo stato che aveva all'inizio e il contatore di programma è ripristinato in modo da puntare a quell'istruzione.

9. Il processo in errore è schedulato per l'esecuzione e il sistema operativo torna alla routine (in linguaggio assembly) che lo aveva invocato.

10. La routine in questione ricarica i registri e le altre informazioni di stato e ritorna allo spazio utente per riprendere l'esecuzione da dove si era interrotta.

Bloccare le Pagine in Memoria

Un processo invia una richiesta di lettura da un file o dispositivo in un buffer nel suo spazio degli indirizzi. Mentre attende il completamento dell'I/O, può essere sospeso per permettere l'esecuzione di un altro processo.

Se il secondo processo genera un page fault, esiste il rischio che la pagina contenente il buffer di I/O venga selezionata per essere rimossa. Una soluzione a questo problema è di bloccare o "pinnare" in memoria le pagine utilizzate per l'I/O, assicurando che le operazioni possano procedere senza interruzioni. Oppure gestire l'I/O nei buffer del kernel e poi copiare i dati nelle pagine utente.

Memoria Secondaria

Il sistema operativo prevede una partizione speciale o dispositivo separato per lo scambio, come nei sistemi UNIX. Un'area del disco/SSD strutturata in maniera differente dai file system. Ogni processo ha un'area di scambio in memoria non volatile. Per affrontare la crescita dinamica di un processo, vengono riservate aree separate per testo, dati e stack. In alternativa, l'allocazione su disco/SSD avviene al momento dello scambio di ogni pagina.

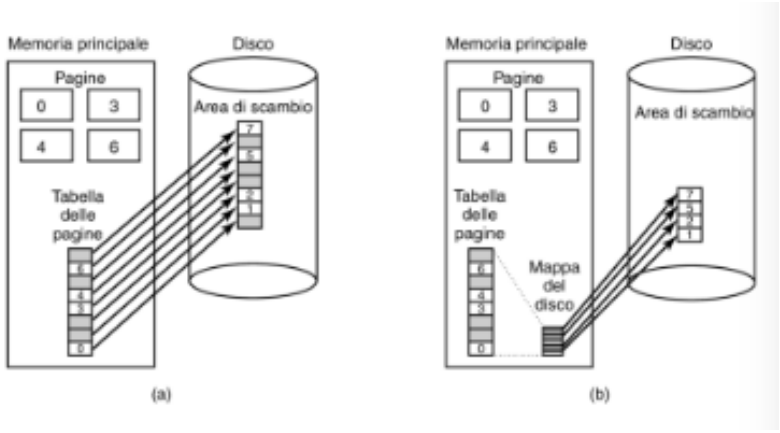

Figure 3: Struttura dell'area di scambio su disco/SSD.

3 Segmentazione

In un sistema di memoria monodimensionale, gli indirizzi virtuali vanno da 0 ad un certo massimo e sono disposti in modo lineare e contiguo. Può risultare problematica in alcuni scenari, come nella compilazione, dove ci sono diverse tabelle che crescono dinamicamente e in modo prevedibile. La crescita di una tabella può causare sovrapposizioni con altre, creando difficoltà nella gestione della memoria.

La segmentazione introduce l'idea di spazi di indirizzi virtuali multipli e indipendenti, chiamati segmenti. Ciascun segmento ha una sequenza lineare di indirizzi, iniziando da 0 fino a una massimo valore. I segmenti possono avere lunghezze diverse e la loro dimensione può cambiare durante l'esecuzione. Questa struttura consente ai segmenti di crescere o ridursi senza interferire l'uno con l'altro.

Vantaggi della Segmentazione

- **Flessibilità:** i segmenti possono crescere o ridursi in modo indipendente l'uno dall'altro.
- **Semplificazione del Linking:** se ogni processo occupa un segmento separato, il linking di procedure diventa molto più semplice.
- **Condivisione e Protezione:** la segmentazione facilità la condivisione di risorse, come librerie condivise, tra processi diversi. Offre anche la possibilità di applicare vari livelli di protezione ai segmenti.

Segmentazione vs Paginazione

La segmentazione suddivide la memoria in segmenti con indirizzi lineari. La segmentazione offre maggiore flessibilità e gestione delle strutture dati rispetto alla paginazione, ma può essere più complessa da implementare.

Considerazione	Paginazione	Segmentazione
Il programmatore deve sapere che questa tecnica è in uso?	NO	SI
Quanti spazi di indirizzi lineari ci sono?	1	Molti
Lo spazio degli indirizzi totale può superare la dimensione della memoria fisica?	SI	SI
Le procedure e i dati possono essere distinti e protetti separatamente?	NO	SI
Le tabelle la cui dimensione varia possono essere disposte facilmente?	NO	SI
La condivisione delle procedure fra utenti è facilitata?	NO	SI
Perché fu inventata questa tecnica?	Per avere uno spazio degli indirizzi lineare grande senza dover acquistare ulteriore memoria fisica	Per consentire a programmi e dati di essere spezzati in spazi degli indirizzi logicamente indipendenti e per facilitare la condivisione e la protezione

Figure 4: Confronto tra segmentazione e paginazione.

Implementazione della Segmentazione Pura

La maggior differenza tra segmentazione e paginazione è che i segmenti hanno dimensioni variabili e le pagine dimensioni fisse.

1. Memoria fisica con 5 segmenti.
2. Il segmento 1 è rimosso, e il segmento 7, che è più piccolo, viene messo al suo posto (ora tra 7 e 2 c'è dello spazio inutilizzato).
3. Il segmento 4 è sostituito dal segmento 5.
4. Il segmento 3 è rimpiazzato dal segmento 6. Ciò genera frammentazione esterna, dopo un po', la memoria sarà suddivisa in parti, alcune contenenti segmenti e altre spazi vuoti.
5. Può essere risolto tramite la compattazione.

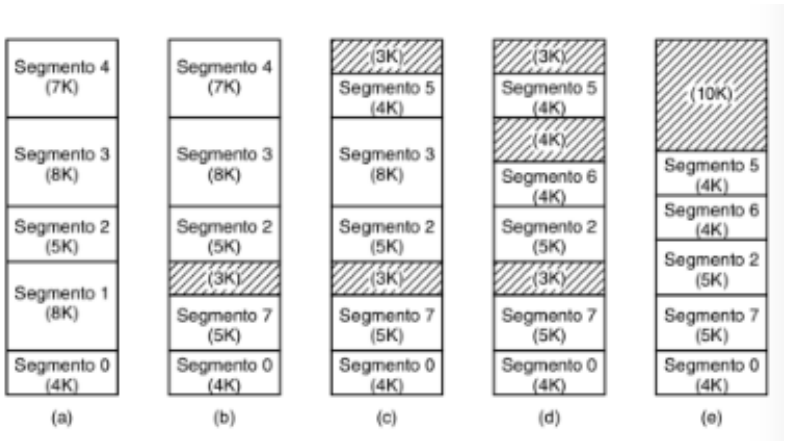

Figure 5: Esempio di Segmentazione.

Segmentazione Con Paginazione In MULTICS

In MULTICS, i segmenti venivano trattati come spazi di memoria virtuale indipendenti e paginati per gestire meglio lo spazio di memoria. C'era una tabella dei segmenti, con descrittori per ogni segmento, indicando se sono in memoria e collegamenti alle tabelle delle pagine. I descrittori avevano un puntatore al segmento, dimensione del segmento, bit di protezione e altre informazioni. Gli indirizzi erano costituiti da due parti, segmento e indirizzo del segmento, con suddivisione in numero di pagina e parola della pagina.

Conversione di un indirizzo in MULTICS

1. Il numero del segmento era usato per trovare il descrittore del segmento.
2. Si verificava se la tabella delle pagine del segmento fosse in memoria. Se lo era, veniva localizzata. Se non lo era, avveniva un errore di segmentazione (segment fault). Se c'era una violazione della protezione, si verificava un errore (trap).
3. Veniva esaminata la voce della pagina virtuale richiesta nella tabella delle pagine. Se la pagina non era in memoria veniva generato un page fault; se lo era, dalla voce della tabella delle pagine veniva estratto l'indirizzo dell'inizio della pagina nella memoria principale.
4. Si otteneva l'indirizzo nella memoria principale in cui era localizzata la parola aggiungendo l'offset all'origine della pagina.
5. Finalmente avveniva la lettura o il salvataggio.

Questo processo è stato ottimizzato con l'introduzione del TLB.