Sistemi Operativi

Ionut Zbirciog

14 November 2023

Gestione della memoria nei sistemi operativi

La memoria principale (RAM) è una risorsa fondamentale che va gestita attentamente, anche se cresce rapidamente, i programmi crescono più velocemente. Il desiderio è quello di avere una memoria privata, grande, veloce e persistente (non volatile), ma la tecnologia di oggi ancora non permette di avere questo tipo di memoria. Nel corso degli anni si è sviluppato il concetto di gerarchia della memoria, i computer possono avere pochi megabyte di memoria molto veloce e molto costosa, qualche gigabyte di memoria abbastanza veloce ma meno costosa e volatile, e qualche terabyte di memoria lenta, poco costosa e non volatile. È compito del sistema operativo astrarre questa gerarchia. La parte del sistema operativo che gestisce la gerarchia della memoria è detto **Gestore della Memoria** i cui compiti sono: tener traccia della memoria in uso, allocare nuova memoria, e deallocare memoria

MEMORY ABSTRACTION

Memoria senza Astrazione

Il modo più semplice per astrarre la memoria è non farlo. Quindi il processo va ad utilizzare direttamente la memoria fisica. Questo tipo di "astrazione" funziona quando sulla macchina viene eseguito un solo programma. Questo modello ha fallito quando erano più di un programma as essere eseguiti sulla macchina. Inoltre, bastava far accedere il processo all'area di memoria dell sistema operativo per causare danni.

Monoprogrammazione

Anche con il modello della memoria fisica, esistono tre sottomodelli di organizzazione della memoria:

1. Il sistema operativo può trovarsi in fondo alla RAM, usato sui mainframe e sui minicomputer.
2. Il sistema operativo si trova in ROM, usato nei sistemi embedded, incastrato nella memoria ROM
3. Il sistema operativo e i drivers in RAM + ROM, usato sui primi modelli si personal computer, dove la parte del sistema nella ROM è chiamta BIOS (Basic Input Output System).

Multiprogrammazione

È però possibile eseguire più programmi contemporaneamente anche senza astrazione della memoria; il sistema operativo deve salvare l'intero contenuto della memoria in un file su memoria non volatile, quindi prelevare ed eseguire il programma successivo. Finché in memoria c'è un solo programma per volta, non ci sono conflitti. Questo concetto (swapping) verrà approfondito più avanti.

Un approccio che si ha avuto è il cosiddetto Naive Approch, ovvero caricamento di più programmi in memoria fisica consecutivamente, senza astrazione dell'indirizzo. Per esempio: Avendo due programmi da 16Kb, il programma A (0 - 16380) inizia con l'istruzione JMP 24, il programma B (16384 - 32764) inizia con l'istruzione JMP 28. Se sono caricati in sequenza, l'istruzione JMP 28 punta all'area di memoria 28, che appartiene al programma A, quindi punta ad uno spazio di memoria sbagliato, causando conflitti durante l'esecuzione e crash dei programmi.

Astrazione della Memoria

REGISTRI BASE E LIMITE

Per permettere a più applicazioni di risiedere in memoria contemporaneamente senza interferire l'una con l'altra devono essere risolti due problemi: protezione e rilocazione. Per quanto riguarda la protezione, abbiamo una soluzione primitiva usata sull'IBM 360: etichettare pezzi di memoria con una chiave di protezione e confrontare la chiave del processo in esecuzione con quella di ogni parola di memoria prelevata.

Una soluzione migliore è inventare una nuova astrazione della memoria, lo spazio degli indirizzi. Uno spazio degli indirizzi è l'insieme degli indirizzi che un processo può usare per indirizzare la memoria. Ogni processo ha il suo spazio degli indirizzi personale, indipendente da quelli degli altri processi. Più difficile è capire come dare a ogni programma il suo spazio degli indirizzi. L'indirizzo 28 di un programma è una locazione fisica in memoria diversa dall'indirizzo 28 di un altro programma.

Implementazione con registri base e limite, sono due registri presenti in molte CPU. Il registro base contiene l'indirizzo fisico di inizio di un programma in memoria. Il registro limite contiene la lunghezza del programma.

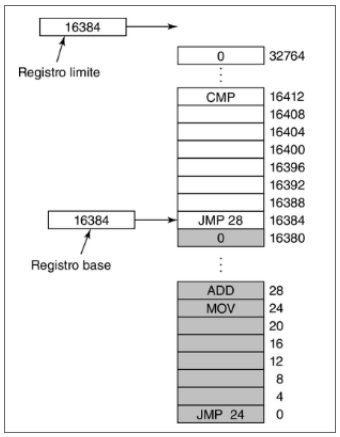

Figure 1: Registro Limite e Registro Base.

Ogni volta che un processo fa riferimento alla memoria, per prelevare un'istruzione o per leggere o scrivere una parola dati, prima di inviare l'indirizzo sul bus di memoria l'hardware della CPU aggiunge automaticamente il valore di base all'indirizzo generato tramite il processo. Contemporaneamente controlla se l'indirizzo offerto sia uguale o maggiore del valore nel registro limite, nel qual caso è generato un errore e l'accesso viene terminato.

Vantaggi: Offre a ciascun processo uno spazio di indirizzi protetto e separato

Svantaggi: Necessità di eseguire somme e confronti ad ogni accesso alla memoria, il che può essere lento.

SWAPPING

La strategia più semplice, chiamata swapping (scambio) dei processi, consiste nel prelevare ciascun processo nella sua totalità, eseguirlo per un certo tempo, quindi porlo nuovamente nella memoria non volatile. I processi inattivi vengono archiviati per la maggior parte su memoria non volatile, così non occupano memoria mentre non sono in esecuzione.

L'altra strategia, chiamata memoria virtuale, consente ai programmi di essere eseguiti anche quando sono solo parzialmente nella memoria principale.

Quando lo swapping crea più spazi vuoti nella memoria (frammentazione della memoria), è possibile combinarli tutti in un unico spazio vuoto spostando tutti i processi il più in basso possibile. Questa tecnica è conosciuta come memory compaction, di solito non viene attuata perché richiede troppo tempo alla CPU.

Un punto che vale la pena sottolineare è quanta memoria dovrebbe essere allocata per un processo quando viene creato o quando viene riportato in memoria dal disco mediante lo swapping. Se i processi sono creati con una dimensione fissa che non cambia mai, l'allocazione è semplice: il sistema operativo alloca esattamente il necessario, né più né meno.

Se invece i segmenti dei dati dei processi possono crescere, appena il processo prova a crescere sorge un problema.

Una soluzione a questo problema è di allocare memoria extra durante lo swapping o lo spostamento dei processi. Se un processo non può crescere nella memoria e l'area di swap del disco o dell'SSD è piena, il processo deve essere sospeso finché non sia liberato dello spazio (oppure terminato).

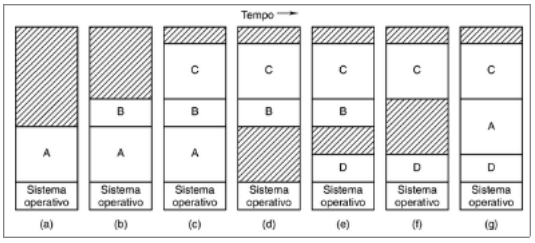

Figure 2: Funzionamento di un sistema di swapping.

Il funzionamento di un sistema di swapping è illustrato nella figura sopra. All'inizio solo il processo A è in memoria. Poi i processi B e C vengono creati o prelevati dalla memoria non volatile. Successivamente viene attuato lo swapping su memoria non volatile di A. Quindi arriva D ed esce B. Alla fine A ritorna, e poiché ora è in una posizione diversa, i suoi indirizzi devono

essere rilocati dal software al momento dello swapping o (più probabilmente) dall'hardware durante l'esecuzione del programma. In questo caso i registri base e limite così funzionerebbero bene.

GESTIONE DELLA MEMORIA LIBERA

Quando la memoria è assegnata dinamicamente, il sistema operativo deve gestirla. In termini generali, ci sono due modi per tener traccia dell'utilizzo della memoria: bitmap e liste.

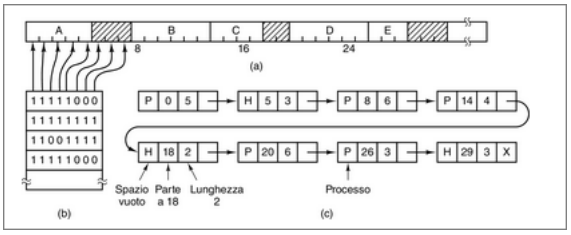

Figure 3: Bitmap e Liste nella gestione della memoria.

BITMAP

La memoria, con una bitmap, è divisa in unità di allocazione che possono essere piccole come qualche parola o grandi come vari kilobyte. A ogni unità di allocazione corrisponde un bit della bitmap, che è 0 se l'unità è libera e 1 se è utilizzata (o viceversa). La dimensione della bitmap dipende solo dalla dimensione della memoria e dalla dimensione dell'unità di allocazione, quindi una bitmap fornisce un modo semplice per tener traccia delle parole di memoria in una quantità fissa di memoria. Il problema principale è che, se si stabilisce di portare un processo di k unità in memoria, il gestore della memoria deve cercare nella bitmap per trovare una serie di k bit 0 consecutivi nella mappa. Cercare in una bitmap una serie di una certa lunghezza è un'operazione lenta.

LISTE

Un altro sistema per tenere traccia della memoria è mantenere una lista concatenata di segmenti di memoria allocati e liberi, in cui un segmento contiene un processo oppure è uno spazio vuoto fra due processi. Ogni voce della lista specifica uno spazio vuoto (H) o un processo (P), l'indirizzo da cui parte, la lunghezza e il puntatore alla voce successiva. Nella realtà viene spesso usata una doppia linked list, poiché rende più facile gestire lo spazio libero. Può controllare facilmente se il precedente spazio è libero. Può regolare facilmente i puntatori.

SCHEMI DI ALLOCAZIONE DELLA MEMORIA

Quando processi e spazi vuoti sono tenuti su una lista ordinata per indirizzo, è possibile usare vari algoritmi per allocare la memoria per un processo creato (o già esistente e scambiato da disco o SSD).

1. **First Fit:** Il gestore della memoria scorre la lista dei segmenti finché non trova uno spazio vuoto abbastanza grande. Lo spazio viene suddiviso in due parti, una per il processo e l'altra

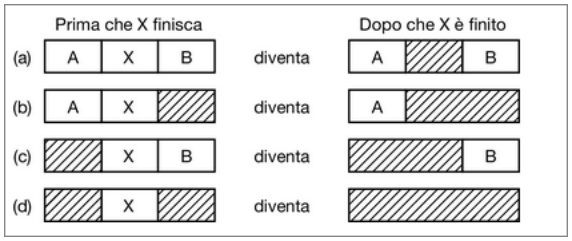

Figure 4: Struttura di una lista nella gestione della memoria.

per la memoria inutilizzata, tranne nel caso statisticamente improbabile che lo spazio coincida perfettamente. First fit cerca il meno possibile, quindi è un algoritmo veloce.

2. **Next Fit:** È una variazione di first fit; opera allo stesso modo, ma tiene traccia dei luoghi nei quali ha trovato uno spazio adatto. La volta seguente che viene chiamato per cercare uno spazio, cerca nella lista a partire dal punto dove era rimasto l'ultima volta, invece di partire dall'inizio come fa sempre first fit. Più lento di First Fit.

3. **Best Fit:** Cerca all'interno della lista dall'inizio alla fine, prendendo lo spazio più piccolo che sia comunque adatto. Anziché occupare solo in parte uno spazio grande che potrebbe essere necessario in seguito, best fit prova a cercare lo spazio della dimensione più vicina a quanto richiesto, per abbinare al meglio le richieste e gli spazi disponibili.

4. **Worst Fit:** Si prende sempre lo spazio disponibile più grande e lascia uno spazio di risulta abbastanza grande da essere utile.

5. **Quick Fit:** Mantiene liste divise per alcune delle dimensioni richieste più comunemente. Per esempio potrebbe esserci una tabella con n voci, di cui la prima è un puntatore alla testa di una lista di spazi di 4 KB, la seconda è un puntatore alla lista degli spazi da 8 KB, la terza a quelli da 12 KB e così via. Gli spazi da 21 KB potrebbero essere insieme a quelli da 20 KB oppure in una lista speciale degli spazi "strani". Con quick fit la ricerca di uno spazio della dimensione richiesta è molto veloce, ma presenta lo stesso svantaggio di tutti gli schemi ordinati per dimensione: quando un processo termina o viene scambiato su disco, trovare i suoi vicini per vedere se sia possibile un'unione è dispendioso. Se non viene eseguita l'unione, la memoria si frammenta rapidamente in un gran numero di piccoli spazi non adatti ad alcun processo.

6. **Buddy Allocation:** Migliora la performance di coalescenza del Quick Fit

BUDDY ALLOCATION

Funzionamento: La memoria inizia come un singolo pezzo contiguo. Ad ogni richiesta, la memoria viene divisa secondo una potenza di 2. I blocchi di memoria contigui vengono uniti quando rilasciati.

Questo algoritmo porta a una considerevole frammentazione interna, perché se è necessario un pezzo costituito da 65 pagine si dovrà chiedere e ottenere un pezzo da 128 pagine.

Per risolvere questo problema, Linux dispone di un secondo allocatore di memoria, l'allocatore a slab, che prende i pezzi usando l'algoritmo buddy ma poi da questi ritaglia gli slab (unità più piccole)

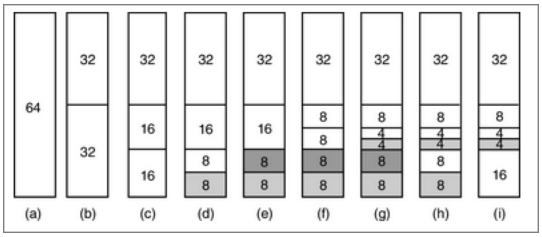

Figure 5: Esempio di funzionamento del buddy allocation.

e li gestisce separatamente. Lo slab allocator è usato per gestire l'allocazione di memoria in heap, di oggetti come array, linked list, struct, ecc... Il kernel spesso ha bisogno di creare e distruggere piccoli oggetti di dimensioni e tipi specifici. Senza ottimizzazione, questa operazione potrebbe portare a una significativa frammentazione della memoria. Nello slab allocation, la memoria è divisa in blocchi chiamati "slabs". Questi slab vengono ulteriormente suddivisi in chunk di dimensioni uniformi, adeguati per ospitare un oggetto di un certo tipo. Un slab può essere in uno dei seguenti stati: pieno (tutti i chunk pieni), parzialmente pieno (alcuni chunk sono pieni) o vuoto (tutti i chunk sono vuoti.). Quando un oggetto viene deallocato, non viene immediatamente restituito al sistema come memoria libera. Viene mantenuto nella cache in modo che, se viene richiesta un'altra istanza dello stesso tipo di oggetto, possa essere rapidamente riallocato senza l'overhead di inizializzazione.