Politica di pulitura

Quando si verificano dei page fault ci sono due possibili casi : nel primo si hanno dei frame liberi da poter assegnare al processo, nel secondo non si avranno dei frame liberi. Il caso più semplice da gestire per il SO è quello dove si hanno dei frame liberi da poter assegnare a processi in sofferenza. Questo caso comporta un minore overhead e il SO stesso può mettere in atto delle strategie per favorire il numero di "casi fortunati", ovvero tenere dei frame liberi in un pool di frame liberi.

Si ha un processo **paging deamon** che si risveglia quando il pool di frame liberi scende sotto una certa soglia e controlla lo stato di occupazione globale dei frame di sistema. Il paging deamon agisce selezionando dei frame occupati e liberandoli invocando l'algoritmo di sostituzione delle pagine in maniera preventiva. Potrebbe anche pulire pagine "sporche", occupate dai dati dell'altro processo. Un processo a cui è stata sottratta una pagina, potrebbe richiederne subito dopo l'uso nel caso in cui l'algoritmo di sostituzione abbia fatto una scelta pessima. Si ha una **possibilità di ripescaggio** se il frame che si era marchiato come libero non è ancora stato sovrascritto, infatti esso viene "ripescato" e riassegnato al processo proprietario che ne ha richiesto la pagina. L'azione del paging deamon avviene senza sottrarre cicli di CPU in quanto lavora in modo asincrono e non comporta forti overhead. L'idea è di ridurre il tempo di blocco di un processo (in cui è in attesa di un frame libero) agendo in maniera preventiva sulla liberazione di un frame.

Dimensione delle pagine

La scelta della dimensione delle pagine di base è importante :

- scelta di una pagina **grande** : comporta molti benefici, ad esempio se le pagine hanno una dimensione maggiore la tabella delle pagine diventa più piccola, in quanto memorizza meno record a parità di informazioni . Un altro beneficio riguarda la gestione efficiente di traferimenti di I/O di file su disco. Nel caso di dischi meccanici si ha lo stesso costo di accesso se si accede a word contigue fisicamente. Una dimensione maggiore delle pagine forza la scrittura in memoria di informazioni della stessa pagina su word contigue (copre diversi blocchi contigui) e quindi il tempo di accesso a queste ultime è ridotto. Un ulteriore beneficio comporta che pagine più grandi portano in memoria più informazioni e quindi minimizzano il numero di page fault, infatti per dimensioni asintoticamente crescenti delle dimensioni della pagina si azzererebbero gli eventi di page fault.

- scelta di una pagina **piccola :** ci sono vantaggi nell'uso di pagine più piccole, ad esempio se si considera il working set, lavorare con pagine di dimensione minore, comporta una maggiore risoluzione nel definire il working set stesso in memoria e quindi meno memoria sprecata. Avere una pagina di dimensione minore provoca una anche una minore frammentazione interna nell'allocazione delle pagine per un processo.

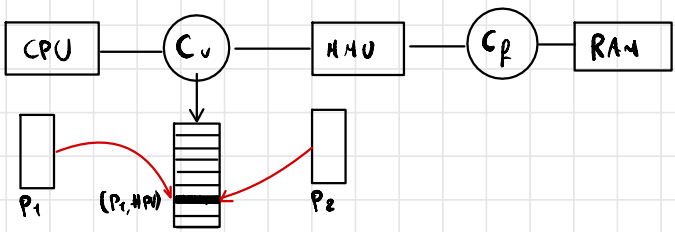
Pagine condivise

Nello spazio di indirizzamento dei processi si trova il codice, dove ogni pagina specifica permessi read-only (solo lettura). Può capitare che le stesse istruzioni del codice di diversi processi facciano riferimento alla stessa pagina di memoria. Essendo le **pagine del codice** marchiate in sola lettura, le traduzioni potranno avvenire su un'unica copia condivisa di quella pagina (copia ridondante della stessa informazione), la quale verrà riportata in memoria una sola volta e non in maniera duplicata. Ovviamente l'approccio di miglioramento della gestione nel caso di pagine condivise è perfettamente compatibile con la paginazione multilivello. Questo comporta un notevole risparmio in memoria, dato dalla condivisione delle pagine, ma funziona solo perchè le pagine del codice sono marchiate in sola lettura.

Nel caso delle **pagine dei dati** la situazione cambia, infatti esse specificano permessi sia di lettura che di scrittura. In questo caso si ha un IPC (interprocess communication) tramite segmenti di memoria condivisa. In questo caso la condivisione è voluta in quanto è corretto che la modifica venga vista dall'altro processo.

Difficoltà nella gestione della cache

Il fenomeno dell'aliasing, ovvero quello per cui due indirizzi virtuali di processi differenti facciano riferimento allo stesso indirizzo fisico in memoria, crea problemi nella gestione di cache basata su indirizzi virtuali (interposta tra CPU ed MMU). Le linee di cache che coprono l'indirizzo in RAM sono identificate dalla coppia (ID, #PV), essendo cache basate su indirizzi virtuali. Queste cache quindi memorizzano l'ASID di uno dei due processi che condivide la pagina ed il problema è dato dal fatto che le cache rileveranno eventi di cache miss se è l'altro processo a chiedere di referenziare la pagina, in quanto il suo PID e il suo #PV sono differenti da quelli nella pagina in cache, nonostante sia esattamente la word richiesta. Questo è un problema di etichettatura e non è un reale page fault ed ovviamente non si verifica con le cache basate su indirizzi fisici in quanto non presentano problemi di ambiguità.



Soluzioni

Una prima soluzione potrebbe essere tenere due linee di cache differenti, ognuna con l'ID di entrambi i processi che la condividono ma qui si avrebbe un disallineamento delle informazioni condivise perchè le copie potrebbero essere modificate.

Una soluzione migliore consisterebbe nel disabilitare la cache per le pagine condivise, ma ciò non è invitante in termini di prestazioni perchè comporterebbe che quelle pagine non verranno mai portate in cache ed i processi ovviamente accederanno a queste ultime attraverso una memoria più lenta (RAM).

La soluzione effettivamente utilizzata è quella di usare cache con ricerca basata su indirizzi virtuali e tag fisici:

➢ la cache ricerca in parallelo con la TLB sulla base dell'indirizzo virtuale;
➢ per capire se si tratta di un duplicato dobbiamo aspettare che la TLB dia in output l'indirizzo fisico;

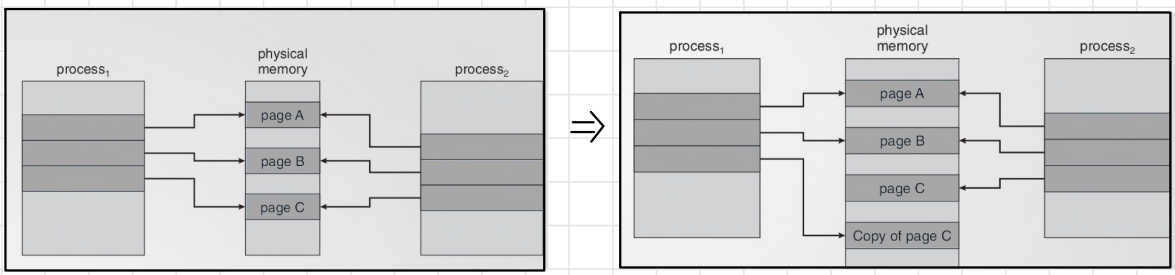
Tabella della pagine invertita

L'aliasing crea problemi anche con un altra struttura dati, infatti nella tabella delle pagine invertita si ha un problema analogo a quello della cache. La tabella delle pagine invertita è unica per tutti i processi ed il generico record viene identificato con un HashMap a partire dalla coppia (ID, #PV). Si suppone che due processi P1 e P2 condividano una pagina, magari anche in indirizzi virtuali differenti. Nella tabella delle pagine invertita verrà memorizzata la word mantenendo la coppia (ID, #PV) di uno solo dei due processi e l'altro troverebbe continui eventi page fault. Anche qui non si tratta di reali page fault ma solo di problemi di etichettatura (come nella cache) ed il SO può rilevare eventi del genere facendo delle alterazioni alla tabella delle pagine invertita in modo da correggere l'etichetta. Questa non rappresenta una soluzione permanente in quanto si riverifica lo stesso problema all'accesso dell'altro processo. Inoltre la soluzione non è applicabile in contesti multicore.

Copy-on-write

La tecnica di Copy-on-Write (CoW) è utilizzata nella gestione della memoria virtuale per ottimizzare l'allocazione di memoria tra processi che condividono pagine.

Quando un processo richiede una pagina di memoria, il sistema operativo crea una copia della pagina solo se il processo desidera modificarla. Altrimenti, se più processi richiedono la stessa pagina, il sistema operativo mantiene una sola copia della pagina in memoria fisica e la condivide tra i processi. Se uno dei processi cerca di modificarla, il sistema operativo crea una copia della pagina e la assegna al processo che ha effettuato la modifica, mentre gli altri processi continuano a condividere la copia originale.
La tecnica di CoW consente di risparmiare memoria fisica, evitando di creare copie inutili delle pagine condivise tra i processi. La tecnica di CoW, applicata al contesto della chiamata fork(), riduce il tempo di creazione dei processi, poiché non è necessario copiare l'intero spazio di indirizzamento del processo, ma solo le pagine che subiranno modifiche.



Questo meccanismo è implementato via software ed il SO si deve poter accorgere del tentativo di scrittura della pagina da parte di un processo. Le pagine condivise specificano una maschera di permessi in sola lettura, in caso di scrittura si avrà un conflitto con i permessi specificati all'MMU si genera un evento di page fault (dovuto ai permessi), il SO si occupa di valutare se l'istruzione è realmente non permessa o se invece sia lecita. In questo caso il SO stesso crea una copia della pagina e la assegna al processo, successivamente reitera l'istruzione in modo che possa essere completata con successo.

Zero-fill-on-demand

Questa è una tecnica che riguarda la creazione di nuove pagine. Il principio di base del Zero-fill-on-demand (ZFOD) è che le nuove pagine di memoria sono vuote e allocate solo su richiesta del processo.

La tecnica di Copy-on-Write viene spesso utilizzata in combinazione con Zero-Fill-on-Demand nel seguente modo : le pagine in sola lettura possono essere condivise tra processi utilizzando una "read-only static zero page" che contiene solo zeri. In questo modo, le pagine non inizializzate possono essere allocate come copie della "static zero page" e condivise tra i processi, evitando di dover allocare nuove pagine vuote per ogni processo.

Il sistema operativo mantiene un **pool di pagine vuote** (piene di zeri) per anticipare l'allocazione efficiente dei frame per richieste future da parte di altri processi. Le pagine vengono azzerate preventivamente in quanto è un overhead che verrà comunque pagato, allora il SO si occupa di svuotare preventivamente le pagine attraverso un demone che di occupa di mantenere il numero di pagine vuote nel pool al di sopra di una certa soglia.

Librerie condivise

Grandi libreri sono comunemente usate in maniera condivisa da differenti processi, ed il loro utilizzo favorisce il riuso del codice. Il collegamento tra il codice sorgente contenuto nelle librerie ed il codice scritto dal programmatore può avvenire in modi differenti :

- **Linking statico** : la compilazione della libreria produce un file oggetto che viene incorporato direttamente nel file eseguibile del processo. Questo approccio ha il vantaggio della semplicità nell'uso e nella portabilità, ma il file eseguibile risultante può essere molto più grande di quello necessario, in quanto include tutto il codice della libreria, anche quello che non viene effettivamente utilizzato dal processo.

- **Linking dinamico** : viene previsto il collegamento a run-time di librerie condivise. Il SO carica in RAM la liberia utilizzata ed informa il processo che l'ha richiesta di dove si trovano le pagine della liberia richieste contenenti le funzioni richieste dal codice all'interno dello spazio di indirizzamento del codice stesso. Questo corrisponde a caricare le parti della libreria utilizzate nel codice, all'interno dello spazio di indirizzamento del processo. Il caricamento in questo caso avviene "on-demand" del processo e consente ai vari processi di attingere dalla liberia condivisa in RAM ed il notevole risparmio è dato dall'avere un solo una copia della liberia in RAM, nelle cui pagine è specificata una maschera dei permessi in sola lettura.

File mappati

Un approccio moderno per interagire coi file prevede che questi vengano mappati nello spazio di indirizzamento del processo (virtualmente illimitato) tramite un intervento del SO. Con la chiamata di sistema **MMAP**, è possibile mappare una porzione del file in memoria, in modo che le pagine di memoria puntino direttamente alle pagine del file sul disco. In questo modo, il processo può accedere alle pagine del file come se fossero in memoria, anche se in realtà **non sono presenti nella RAM**. Quando il processo accede a una pagina del file che non è ancora in memoria, si verifica un page fault, che causa il caricamento della pagina nel frame di memoria corrispondente.

La condivisione del file tra più processi è efficiente in quanto vengono caricate in memoria solo le pagine del file utilizzate da un qualche processo e non tutto il file. Quando un processo modifica una pagina del file, il dirty bit della pagina viene aggiornato, indicando che la pagina è stata modificata e quando il processo termina o la pagina in memoria viene scartata dall'algoritmo di sostituzione delle pagine, la copia sul disco verrà aggiornata tramite la chiamata **MSYNC** che sincronizza le modifiche con i file su disco. In questo modo, l'approccio a file mappati consente di effettuare operazioni sia in lettura che in scrittura sui file in modo efficiente e sicuro, senza la necessità di caricare l'intero file in memoria.

Un utilizzo dei file mappati è quella di predisporre la creazione di un processo durante la chiamata **exec** , nella quale il codice si trovano sul disco e viene mappato nello spazio di indirizzamento.

Allocazione della memoria per il kernel

Il SO necessita di gestire efficientemente il proprio spazio di memoria, in particolare dello spazio che esso neccita per la gestione delle strutture dati che utilizza (esempio i PCB dei processi, semafori ecc..)esso viene allocato in uno spazio di memoria contigua in RAM. Un approccio Linux è quello **slub allocator**. Una **cache** è una sequenza di pagine contigue dette **slub** che possono essere piene, vuote o parzialmente piene. Una cache è specializzata per una data dimensione (ad esempio per gli oggetti da 3KB), ovvero idoneo alla gestione degli oggetti (strutture dati) aventi una dimensione specifica. Quando si allocano degli slub in memoria fisica per una cache si deve tenere conto di allocarne blocchi che sono multipli sia della dimensione degli oggetti (3KB) e sia della dimensione delle pagine col fine di non frammentare la memoria. Questo approccio apporta dei vantaggi come l'inesistenza di frammentazione e l'efficienza nella gestione della memoria.