Memoria virtuale

Strutture hardware e di controllo

Le caratteristiche chiave delle tecniche di segmentazione e paginazione sono:

- Tutti i riferimenti di memoria all'interno di un processo sono indirizzi logici, che sono tradotti dinamicamente in indirizzi fisici a tempo d'esecuzione. Questo comportamento garantisce la rilocazione in memoria del processo.
- Un processo puo' essere diviso in un certo numero di pezzi (pagine o segmenti) che non devono essere necessariamente contigui in memoria principale.

Osservazione: non e' necessario che tutte le pagine o i segmenti di un processo siano in memoria principale per permetterne l'esecuzione.

Resident set: porzione di un processo che e' attualmente caricato in memoria principale.

Vantaggi della memoria virtuale:

- Piu' processi possono essere mantenuti nella memoria principale, e questo implica una migliore efficienza del processore (aumenta la probabilita' di trovare sempre processi in stato di ready).
- E' possibile che un processo sia piu' grande della memoria principale. Il lavoro di *overlay* e' lasciato al processore e al sistema operativo.

Trashing: fenomeno per il quale il processore impiega la maggior parte del suo tempo a gestire l'I/O da disco per caricare/scaricare le pagine dei processi, invece che eseguire istruzioni. Si cerca di evitare fenomeni di trashing pianificando la strategia di sostituzione delle pagine in memoria principale quando questa risulta al limite della capacita'.

Principio di localita': principio base seguito nella pianificazione di strategie di sostituzione di pagine della memoria principale. Afferma che i riferimenti al programma o ai dati all'interno di un processo tendono a raggrupparsi.

Affinche' si possa adoperare un meccanismo di memoia virtuale:

- Deve esserci il supporto dell'hardware per gli schemi di segmentazione/paginazione
- Deve esserci il supporto del sistema operativo, che deve creare e mantenere le opportune strutture di controllo.

Paginazione e memoria virtuale

Stesso sistema adottato per la paginazione semplice. In questo caso le entries della tabella delle pagine sono parzialmente modificate per supportare il meccanismo di memoria virtuale. Nel dettaglio:

- E' necessario un bit che specifichi se la pagina specificata e' attualmente caricata in

memoria.
- E' necessario un bit che specifichi se la pagina abbia subìto modifiche. In caso affermativo la pagina in questione, prima di essere sovrascritta, deve essere ricopiata su disco.
- Possono essere presenti altri bit che gestiscano informazioni di protezione e condivisione della pagina

Ogni processo ha la sua tabella delle pagine, di dimensione variabile. Questo comporta alcuni aspetti:

- La tabella delle pagine non puo' risiedere nel processore, al cui interno e' invece presente un registro che contiene un puntatore alla tabella delle pagine del processo in running.
- La tabella delle pagine risiede in memoria principale.
- Dal momento che la tabella delle pagine ha una dimensione proporzionale alla dimensione massima di memoria virtuale assegnabile ad ogni processo (o meglio, ha un numero di entries esattamente pari alla alla dimensione massima di memoria virtuale assegnabile ad ogni processo diviso la dimensione della pagina), essa stessa e' mantenuta in memoria virtuale, e come tale e' essa stessa sottoposta a paginazione.
- Dal punto precedente segue che, quando un processo e' in running, almeno una parte della sua tabella delle pagine deve essere in memoria, per l'esattezza l'entry relativa alla pagina correntemente sotto esecuzione.
- Per risolvere il problema della grandezza delle page tables, si puo' ricorrere a:
  - uno schema a due livelli, dove ogni entry di una directory delle pagine (definita *root page table*) punta ad una tabella delle pagine particolare[1].
  - una *inverted page table*, in cui e' presente una entry per ogni frame della memoria principale, e che e' sempre mantenuta in memoria principale. L'indicizzazione dei frames in tale tabella sfrutta una tavola hash.

Translation lookaside buffer: cache per la tabella delle pagine, che contiene le entry della tabella delle pagine usate piu' frequentemente. Questo meccanismo permette di evitare il duplice accesso a memoria principale che, in sua assenza, avverrebbe ad ogni riferimento a memoria:

1. accedere a memoria per prelevare la entry della tabella delle pagine riferita all'indirizzo virtuale sotto esame.
2. accedere a memoria all'indirizzo specificato dopo il mapping indirizzo virtuale » indirizzo fisico.

La cache TLB viene ovviamente svuotata ad ogni cambio di processo, e si basa fortemente sul principio di localita'.

La dimensione della pagina e' una importante decisione progettuale, presa sulla base di vari fattori:

- Piu' piccola e' la dimensione della pagina, piu' piccola la frammentazione interna
- Piu' piccola e' la dimensione della pagina, piu' pagine per processo sono richieste e, di conseguenza, si avra' una tabella delle pagine con maggior numero di entries.
- In caso di programmi estesi in ambiente pesantemente multiprogrammato, grande page tables implicano maggior probabilita' di avere un doppio page fault quando si ha un riferimento a memoria (uno per mancanza della parte di page table necessaria a tradurre

1 Usato nei processori Pentium

l'indirizzo virtuale, uno per il recupero effettivo della pagina referenziata dall'indirizzo cosi' tradotto).
- Piu' grande e' la dimensione della pagina, piu' efficiente e' il suo trasferimento su disco.

La percentuale di page fault quindi e' influenzata dalla grandezza della pagina e, a grandezza fissata, diminuisce con l'aumentare del numero di frames allocati per processo.

Segmentazione e memoria virtuale

La segmentazione permette al programmatore di vedere la memoria come un insieme di segmenti o spazi di indirizzamento multipli. Questa tecnica:

- Semplifica la gestione di strutture dati che crescono.
- Permette la modifica e la ricompilazione indipendente dei programmi, senza rifare il link con tutto l'insieme dei programmi.
- Si presta alla condivisione tra processi.
- Si presta alla protezione.

La strategia implementativa e' simile a quella gia' discussa per la paginazione (una tabella dei segmenti per ogni processo, bit di modifica e di presenza).

Confronto vantaggi paginazione e segmentazione

Paginazione	***Segmentazione***
Trasparente al programmatore. Elimina il problema della frammentazione interna. Grazie alla stessa dimensione tra frame e pagina, si possono sviluppare algoritmi evoluti per la gestione della memoria.	Visibile al programmatore. Permette la gestione di strutture dinamiche. Modulare. Supporto per la condivisione e la protezione.

In un sistema che combina paginazione e segmentazione, uno spazio di indirizzo utente e' diviso in un certo numero di segmenti. Ogni segmento e' a sua volta diviso in pagine di dimensione fissa, lunghe quanto i frame della memoria principale.

Un indirizzo virtuale quindi:

- Dal punto di vista del programmatore e' visto come numero di segmento e offset di segmento.
- Dal punto di vista del sistema operativo e' visto come numero di segmento, numero di pagina e offset di pagina.

Ogni processo ha una tabella dei segmenti e, per ogni segmento, una tabella delle pagine.

Software del sistema operativo

Strategia di fetch

Determina quando una pagina deve essere caricata in memoria principale. Sono attuabili due possibili approcci:

- Paginazione a richiesta (*demand paging*): una pagina e' caricata in memoria solo quando si fa un riferimento a quella pagina. Molti page fault in fase di avvio di un processo.
- Prepaginazione (*prepaging*): oltre alla pagina in questione che ha provocato il fault, vengono caricate in memoria altre pagine. Operazione veloce se le pagine da caricare sono contigue su disco.

Strategia di posizionamento

Determina dove deve risiedere un pezzo di processo nella memoria reale. Per un sistema che usa la paginazione pure o la paginazione combinata a segmentazione, generalmente questo e' un fattore irrilevante.

Strategia di sostituzione

Riguarda la selezione di una pagina in memoria da sostituire quando una nuova pagina deve essere caricata. Piu' nel dettaglio, riguarda il criterio di scelta all'interno di un insieme di pagine segnate come sostituibili. La scelta di tale insieme di pagine prende il nome di resident set management, e verra' discusso in seguito.

Tutte le strategie di sostituzione hanno come obiettivo la sostituzione della pagina che verra' richiesta il piu' lontano possibile nel tempo (ovvero la pagina a cui ci si riferira' di meno nel prossimo futuro, la cui rimozione provocherebbe quindi un minor numero di page fault). La maggior parte delle strategie cerca di prevedere l'andamento futuro delle richieste di pagina sulla base degli andamenti passati.

Frame in blocco: frame che sono bloccati in memoria principale e che non possono essere sostituiti. Ad esempio quelli destinati ai buffer dell'I/O in memoria, a buona parte del kernel, alle strutture di controllo.

Algoritmo ottimo: seleziona per la sostituzione la pagina alla quale ci si riferira' dopo il tempo piu' lungo. Ovviamente irrealizzabile (anticausale), puo' essere usato come metro di paragone tra le strategie realizzabili.

Algoritmo least-recently-used (*LRU*): sostituisce in memoria la pagina a cui non si e' riferito per piu' tempo. Si basa sul principio di localita'. C'e' pero' una difficolta' implementativa legata all'uso di questo sistema: si richiederebbe di contrassegnare ogni pagina con un timestamp, ad ogni suo riferimento, il che risulta una operazione inutilmente costosa, anche in presenza di hardware specializzato.

Algoritmo first-in-first-out: tratta i frame allocati per le pagine di un processo come un buffer circolare e le pagine sono rimosse utilizzando lo schema round robin. Sostituisce la pagina che e' stata piu' a lungo in memoria. Non tiene conto della storia passata delle pagine.

Algoritmo dell'orologio: approssima la strategia LRU. Risoluzione del timestamp di un bit, chiamato use bit. Esso viene posto ad 1 ogni volta che una pagina viene caricata in memoria e ad ogni suo successivo riferimento. Quando si deve sostituire una pagina, i frame vengono scanditi ciclicamente. Il primo frame con use bit a 0 viene sostituito. Se si incontra un frame con use bit a 1, esso viene messo a 0. Il puntatore punta sempre al frame successivo a quello sostituito.

Algoritmo dell'orologio con bit di modifica: oltre al bit di uso si prevede anche un bit di modifica. Quando si scandiscono i frame per la sostituzione, si preferisce sempre sostituire un frame non modificato, in quanto non prevede una riscrittura della pagina su disco. Puo' compiere al piu' 4 giri del buffer:

1. Cerca un frame con u=0, m=0. Non modifica i bit.
2. In caso di fallimento cerca un frame con u=0,m=1. Durante la scansione, pone gli use bit a 0.
3. In caso di fallimento si ripete il passo 1.

Algoritmo di aging: aumenta la risoluzione dei bit per approssimare LRU nel breve periodo. Per ogni pagina si tiene un age estimator: piu' basso e' il valore dell'estimator, piu' antica e' la pagina (o meglio, piu' lontano nel tempo e' il suo ultimo riferimento). Il funzionamento e' il seguente:

1. Ad intervalli regolari, effettua la scansione dei frame, e considera gli use bit.
2. L'age estimator di ogni frame viene shiftato di una posizione a destra, e il valore del bit d'uso viene posto nella posizione piu' significativa.
3. Pone a 0 tutti gli use bit.
4. Quando si deve sostituire una pagina, si sceglie quella con piu' basso age estimator.

Page buffering: generamente accoppiato con l'algoritmo FIFO. I frame sostituibili vengono collocati in due liste:

- lista delle pagine libere, se la pagina non e' stata modificata
- lista delle pagine modificate in caso contrario

La pagina non e' fisicamente spostata nella memoria principale: la sua entry nella tabella delle pagine e' trasferita nella lista delle pagine libere o nella lista delle pagine modificate. Per migliorare le prestazioni, il sistema cerca di mantenere sempre un piccolo numero di pagine libere. Quando una pagina deve essere caricata, viene assegnata al primo frame della lista delle pagine libere, sostituendo la pagina ivi residente. Quando una pagina non modificata sta per essere sostituita, in realta' rimane in memoria. Quindi se il processo fa di nuovo riferimento alla pagina in questione, questa pagina in realta' e' ancora in memoria, ed il suo caricamento e' in pratica immediato. Inoltre,la lista delle pagine modificate e' scritta su disco tutta insieme: in tal modo si riduce il numero di accessi al disco.

Gestione del resident set

Il sistema deve decidere quanta memoria principale allocare per un particolare processo. Sono possibili due strategie:

- *Allocazione fissa*, assegna al processo un numero fissato di pagine per l'esecuzione, deciso al momento del primo caricamento, e determinato in base al tipo di processo o alle indicazioni del programmatore o del sistemista.
- *Allocazione variabile*, consente di variare il numero di frame allocati per un processo

durante la sua esecuzione (ad esempio sulla base della frequenza di page fault: con frequenze elevate si aumenta il resident set del processo; con basse frequenze, lo si diminuisce).

L'allocazione variabile, benche' piu' potente, ha un overhead maggiore per il sistema, che deve valutare costantemente i processi attivi, e richiede specializzazione dall'hardware.

Correlata alla strategia di allocazione e' la nozione di ambito di sostituzione, che puo' essere di due tipi:

- *Sostituzione locale*, sceglie la pagina da sostituire tra quelle residenti in memoria del processo che ha causato il fault. E' ovviamente l'unica scelta per l'allocazione fissa, ma puo' essere adottata anche dalla allocazione variabile.
- *Sostituzione globale*, considera tutte le pagine non bloccate in memoria principale come candidate alla sostituzione. Puo' sicuramente essere adottata da una politica di allocazione variabile.

Allocazione fissa, ambito locale.

Presenta un duplice svantaggio:

- Se l'allocazione e' troppo piccola, allora ci saranno numerosi fault di pagina, con conseguente rallentamento dovuto all'I/O su disco.
- Se troppo grande, ci saranno pochi processi in memoria, con un utilizzo quindi meno efficiente del processore.

Allocazione variabile, ambito globale.

Il sistema operativo mantiene una lista di frame liberi. Quando avviene un page fault:

1. Un frame libero viene aggiunto al resident set del processo causante il frame, e la pagina richiesta viene caricata.
2. Se non ci sono frame liberi, ne viene scelto uno globalmente tra quelli non bloccati, sulla base delle precedenti considerazioni. Conseguenza e' che il resident set di un processo sara' diminuito. Ma non c'e' un criterio per stabilire quale processo perdera' una pagina.

Si possono limitare gli svantaggi introducendo un sistema di page buffering.

Allocazione variabile, ambito locale.

Si compone dei seguenti passi:

1. Quando si carica in memoria un nuovo processo, gli vengono allocati alcuni frame, il cui numero dipende da vari fattori (tipo di processo, configurazione della macchina, richieste del programma, carico corrente...).
2. Quando avviene un page fault si seleziona la pagina da sostituire tra quelle del resident set del processo che ha causato il fault.
3. Periodicamente si rivaluta l'allocazione fornita al processo e la si aumenta o decrementa per migliorare le prestazioni globali.

Il criterio di rivalutazione dell'allocazione per ciascun processo e la periodicita' della rivalutazione sono i punti cardine di questa tipologia di allocazione.

Strategia del working set

Tempo virtuale: si consideri una sequenza di riferimenti a memoria generati da un processo P. Siano r(1), r(2), ... tali accessi. r(i) e' la pagina che contiene l'i-esimo indirizzo referenziato da P. t=1,2,3 ... e' chiamato tempo virtuale per il processo P.

Working set: definito per un processo a un certo istante di tempo virtuale t con parametro fissato Δ come insieme di pagine di quel processo a cui ci si e' riferiti nelle ultime Δ unita' di tempo virtuale.
Maggiore e' la finestra Δ , maggiore o uguale e' il working set. Inoltre, ad ogni istante t, la cardinalita' del working set e' compresa nell'intervallo chiuso [1,min(Δ ,N)], con N numero delle pagine che compone l'immagine del processo.

A livello ideale, sarebbe conviente porre il resident set pari al working set. Quindi una strategia di gestione del resident set puo' essere verosimilmente la seguente:

1. Controllare il working set di ogni processo
2. Rimuovere periodicamente dal resident set di un processo quelle pagine che non sono nel suo working set
3. Un processo puo' essere eseguito solo se il suo working set e' nella memoria principale (cioe' se il suo resident set contiene il suo working set)

Problemi correlati:

- Sia la dimensione del working set sia i frame che vi appartengono cambiano nel tempo.
- Non e' pratico stimare la dimensione del working set per ogni processo. Uso di timestamp.
- Il valore ottimale di Δ e' sconosciuto ed e' soggetto a variazioni.

Un modo per avere una stima indiretta del working set (o almeno, sulla sua dimensione) e' quello di considerare la frequenza di fault di pagina di un processo. Infatti se il resident set di un processo e' maggiore del suo working set, si avra' una bassa percentuale di faults, e viceversa. Si puo' sfruttare questa idea tramite l'uso di una frequenza di soglia P:

- Se PFF (*Page Faults Frequency*) e' maggiore di P, si aumenta il resident set.
- Viceversa, si diminuisce il resident set.

Il PFF viene stimato nel modo seguente:

1. Si mantiene un contatore t dei riferimenti a memoria.
2. Ad ogni page fault, si stima il nuovo PFF:
   - $PFF_{now} = \alpha \cdot \frac{1}{t - t_1} + (1-\alpha) \cdot PFF_{prev}$
3. Si decide l'azione da intraprendere.

Questa strategia migliora con l'adozione del page buffering. D'altro canto, essa peggiora durante i periodi di transizione di localita'. In questa fase avvengono numerosi pagefaults che portano il processo ad aumentare il proprio resident set a scapito di altri processi, che possono anche venire

sospesi e spostati su disco.

Strategia di cleaning

Determina quando una pagina modificata deve essere scritta in memoria secondaria.

Due alternative:

- Cleaning a richiesta (*demand cleaning*), in cui una pagina viene scritta su disco solo quando deve essere sostituita.
- *Precleaning*, in cui le pagine sono scritte a gruppi prima della loro selezione per la sostituzione.

L'approccio migliore usa il page buffering: le pagine vengono divise in due gruppi, modificate e non modificate. Quelle modificate vengono scritte in batch, quelle non modificate vengono richiamate se riaccedute, perse se sovrascritte.

Controllo del carico

Determina il numero di processi che sara' presente in memoria principale, a livello di multiprogrammazione. Con pochi processi, aumenta la probabilita' che siano in blocco, e si perde molto tempo a trasferirli su disco. Inoltre l'uso del processore diventa inefficiente. Se sono troppi, si rischia il trashing.

Se il livello di multiprogrammazione deve essere ridotto, uno o piu' processi residenti debbono essere trasferiti su disco. Le possibilita' sono di trasferire su disco:

- Il processo con la priorita' piu' bassa.
- Il processo che provoca fault.
- L'ultimo processo attivato.
- Il processo con il piu' piccolo resident set.
- Il processo piu' grande.
- Il processo con la piu' grande finestra di esecuzione rimanente.