# CAPITOLO 9: MEMORIA VIRTUALE

La memoria virtuale è una tecnica che permette di eseguire processi che possono anche non essere completamente contenuti in memoria. Il vantaggio principale offerto da questa tecnica è quello di permettere che i programmi siano più grandi della memoria fisica; inoltre la memoria virtuale astrae la memoria centrale in un vettore di memorizzazione molto grande e uniforme, separando la memoria logica, com’e vista dall'utente, da quella fisica.

Questa tecnica libera i programmatori da quel che riguarda i limiti della memoria.

La memoria virtuale permette inoltre ai processi di condividere facilmente file e spazi d’indirizzi, e fornisce un meccanismo efficiente per la creazione dei processi. La memoria virtuale è però difficile da realizzare e se è usata scorrettamente, può ridurre di molto le prestazioni del sistema.

**Introduzione**

Gli algoritmi di gestione della memoria sono necessari perchè , per l’attivazione di un processo, le istruzioni da eseguire si devono trovare all’interno della memoria fisica. Il primo metodo per far fronte a tale requisito consiste nel collocare l’intero spazio d’indirizzi logici del processo relativo in memoria fisica. Il caricamento dinamico può aiutare ad attenuare gli effetti di tale limitazione, ma richiede generalmente particolari precauzioni e un ulteriore impegno dei programmatori.

La condizione che le istruzioni debbano essere nella memoria fisica sembra tanto necessaria quanto ragionevole, ma purtroppo riduce le dimensioni dei programmi a valori strettamente correlati alle dimensioni della memoria fisica. In effetti in molti casi non è necessario avere in memoria l’intero programma;

spesso i programmi dispongono di codice per la gestione di condizioni d’errore insolite.

Poichè questi errori sono rari , anche i relativi segmenti di codice non si eseguono quasi mai.

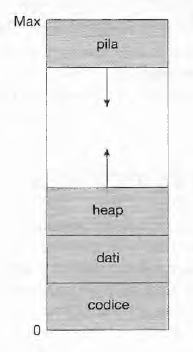
La possibilità di eseguire un programma che non si trovi completamente in memoria apporterebbe quindi vantaggi sia al sistema sia all’utente.

- La **memoria virtuale** si fonda sulla separazione della memoria logica percepita dall’utente

dalla memoria fisica. Questa separazione permette di offrire ai programmatori una memoria virtuale molto ampia, anche se la memoria fisica disponibile è più piccola.

La memoria virtuale facilita la programmazione, perchè il programmatore non deve preoccuparsi della quantità di memoria fisica disponibile, ma può concentrarsi sul problema da risolvere.

L’espressione **spazio degli indirizzi virtuali** si riferisce alla collocazione dei processi in memoria dal punto di vista logico (o virtuale). Da tale punto di vista, un processo inizia in corrispondenza di un certo indirizzo logico e si estende alla memoria contigua.



Si noti come, allo heap sia lasciato sufficiente spazio per crescere verso l’alto nello spazio di memoria, poichè esso ospita la memoria allocata dinamicamente. In modo analogo, consentiamo alla pila di svilupparsi verso il basso nella memoria, a causa di ripetute chiamate di funzione. Lo spazio vuoto ben visibile (o buco) che separa lo heap dalla pila è parte dello spazio degli indirizzi virtuali, ma richiede pagine fisiche realmente esistenti solo nel caso che lo heap o la pila crescano. Qualora contenga buchi, lo spazio degli indirizzi virtuali si definisce sparso.

Oltre a separare la memoria logica da quella fisica, la memoria virtuale offre, per due o più processi, il vantaggio di condividere i file e la memoria, mediante la condivisione delle pagine.

-Le librerie di sistema sono condivisibili da diversi processi associando l’oggetto condiviso a uno spazio degli indirizzi virtuali, procedimento detto **mappatura**.

-In maniera analoga, la memoria virtuale rende i processi in grado di condividere la memoria; due o più processi possono comunicare condividendo memoria. La memoria virtuale permette a un processo di creare

una regione di memoria condivisibile da un altro processo. I processi che condividono questa regione la considerano parte del proprio spazio degli indirizzi virtuali, malgrado le pagine fisiche siano, in realtà, condivise.

La memoria virtuale può consentire, per mezzo della chiamata di sistema fork ( ), che le pagine siano

condivise durante la creazione di un processo, così da velocizzare la generazione dei processi.

**PAGINAZIONE SU RICHIESTA**

Si consideri il caricamento in memoria di un eseguibile residente su disco. Una possibilità è quella di caricare l’intero programma nella memoria fisica al momento dell’esecuzione. Il problema, però, è che all’inizio non è detto che serva avere tutto il programma in memoria: se il programma, per esempio, fornisce all’avvio una lista di opzioni all’utente, è inutile caricare il codice per l’esecuzione di *tutte* le opzioni previste, senza tener conto di quella effettivamente scelta dall’utente.

Una strategia alternativa consiste nel caricare le pagine nel momento in cui servono realmente; si tratta di una tecnica, detta **paginazione su richiesta**, comunemente adottata dai sistemi con memoria virtuale. Secondo questo schema, le pagine sono caricate in memoria solo quando richieste durante l’esecuzione del programma: ne consegue che le pagine cui non si accede mai non sono mai caricate nella memoria fisica.

Nell’ambito della paginazione su richiesta, il modulo del sistema operativo che si occupa della sostituzione delle pagine si chiama **paginatore** (*pager*).

**Concetti fondamentali**

Quando un processo sta per essere caricato in memoria, il paginatore ipotizza quali pagine saranno usate, prima che il processo sia nuovamente scaricato dalla memoria. Anzichè caricare in memoria tutto il processo, il paginatore trasferisce in memoria solo le pagine che ritiene necessarie. In questo modo è possibile evitare il trasferimento in memoria di pagine che non sono effettivamente usate, riducendo il tempo d’avvicendamento e la quantità di memoria fisica richiesta.

Con tale schema è necessario che l’architettura disponga di un qualche meccanismo che consenta di distinguere le pagine presenti in memoria da quelle nei dischi. A tal fine è utilizzabile lo schema basato sul bit di validità (PAR. 8.5). In questo caso, però, il bit impostato come “valido” significa che la pagina corrispondente è valida ed è presente in memoria; il bit impostato come “non valido” indica che la pagina non è valida (cioè non appartiene allo spazio d’indirizzi logici del processo) oppure è valida ma è attualmente nel disco.

Occorre notare che indicare una pagina come non valida non sortisce alcun effetto se il processo non tenta mai di accedervi. Quindi, se l’ipotesi del paginatore è esatta e si caricano tutte e solo le pagine che servono effettivamente, il processo è eseguito proprio come se fossero state caricate tutte le pagine. Durante l’esecuzione, il processo accede alle pagine residenti in memoria, e l’esecuzione procede come di consueto.

Se il processo tenta l’accesso a una pagina che non era stata caricata in memoria, l’accesso a una pagina contrassegnata come non valida causa un’eccezione di pagina mancante*(page fault trap).* L’architettura di paginazione, traducendo l’indirizzo attraverso la tabella delle pagine, nota che il bit è non valido e invia un segnale di eccezione al sistema operativo; tale eccezione è dovuta a un “insuccesso” del sistema operativo nella scelta delle pagine da caricare in memoria. La procedura di gestione dell’eccezione di pagina mancante è chiara, e corrisponde ai passi seguenti:

1. Si controlla una tabella interna per questo processo; in genere tale tabella è conservata insieme al blocco di controllo di processo (PCB), allo scopo di stabilire se il riferimento fosse un accesso alla memoria valido o non valido.

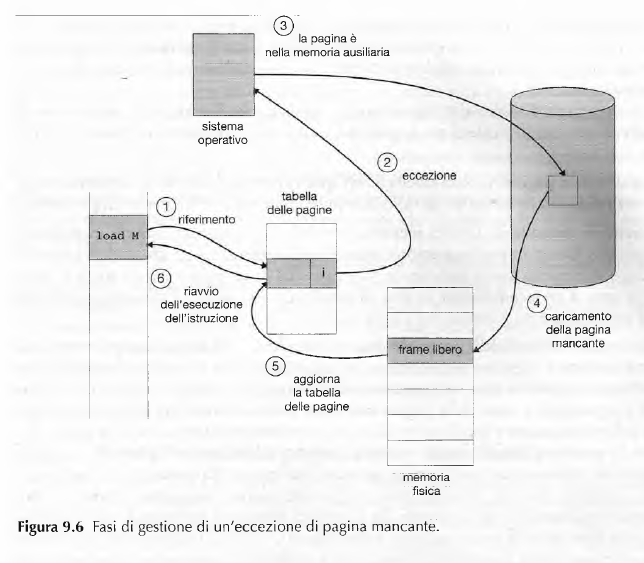
2. Se il riferimento non era valido, si termina il processo. Se era un riferimento valido, ma la pagina non era ancora stata portata in memoria, se ne effettua l’inserimento.

3. Si individua un frame libero, ad esempio prelevandone uno dalla lista dei frame liberi.

4. Si programma un’operazione sui dischi per trasferire la pagina desiderata nel frame appena assegnato.

5. Quando la lettura dal disco è completata, si modificano la tabella interna, conservata con il processo, e la tabella delle pagine per indicare che la pagina si trova attualmente in memoria.

6. Si riavvia l’istruzione interrotta dal segnale di eccezione. A questo punto il processo può accedere alla pagina come se questa fosse già presente in memoria.



E' addirittura possibile avviare l’esecuzione di un processo *senza* pagine in memoria. Quando il sistema operativo carica nel contatore di programma l’indirizzo della prima istruzione del processo, che è in una pagina non residente in memoria, il processo accusa un’assenza di pagina. Una volta portata la pagina in memoria, il processo continua l’esecuzione, subendo assenze di pagine fino a che tutte le pagine necessarie non si trovino effettivamente in memoria; a questo punto si può eseguire il processo senza ulteriori richieste.

Lo schema descritto è una **paginazione su richiesta pura**, vale a dire che una pagina non si trasferisce in memoria finchè non sia richiesta.

I meccanismi d’ausilio alla paginazione su richiesta che l’architettura del calcolatore deve offrire sono quelli richiesti per la paginazione e l’avvicendamento dei processi in memoria:

♦ tabella delle pagine: questa tabella ha la capacità di contrassegnare un elemento come non valido attraverso un bit di validità oppure un valore speciale dei bit di protezione;

♦ memoria secondaria: questa memoria conserva le pagine non presenti in memoria centrale. Generalmente la memoria secondaria è costituita da un disco ad alta velocità detto dispositivo d’avvicendamento; la sezione del disco usata a questo scopo si chiama area d’avvicendamento, o area di scambio *(swap space).*

Uno dei requisiti cruciali della paginazione su richiesta è la possibilità di rieseguire una qualunque istruzione a seguito di un’eccezione di pagina mancante o assenza di pagina *(page fault).* Avendo salvato lo stato del processo interrotto (registri, codici di condizione, contatore di programma) a causa della pagina mancante, occorrerà riavviare il processo esattamente dallo stesso punto e con lo stesso stato, eccezion fatta per la presenza della pagina desiderata in memoria.

Il sistema di paginazione si colloca tra la CPU e la memoria di un calcolatore e deve essere completamente trasparente al processo utente.

**Prestazioni della paginazione su richiesta**

La paginazione su richiesta può avere un effetto rilevante sulle prestazioni di un calcolatore.

Il motivo si può comprendere calcolando il tempo d’accesso effettivo per una memoria con paginazione su richiesta. Attualmente, nella maggior parte dei calcolatori il tempo d’accesso alla memoria, che si denota *ma,* varia da 10 a 200 nanosecondi. Finchè non si verifichino assenze di pagine, il tempo d’accesso effettivo è uguale al tempo d’accesso alla memoria. Se però si verifica un’assenza di pagina, occorre prima leggere dal disco la pagina interessata e quindi accedere alla parola della memoria desiderata.

Supponendo che *p* sia la probabilità che si verifichi un’assenza di pagina (0 < *p* < 1), è probabile che *p* sia molto vicina allo zero, cioè che ci siano solo poche assenze di pagine. Il tempo d’accesso effettivo è dato dalla seguente espressione:

*tempo d'accesso effettivo* = ( 1 — *p)* x *ma* + *p* x *tempo di gestione dell'assenza di pagina*

Per calcolare il tempo d’accesso effettivo occorre conoscere il tempo necessario alla gestione di un’assenza di pagina.

Alla presenza di un’assenza di pagina si esegue la seguente sequenza:

* segnale d’eccezione al sistema operativo
* Salvataggio dei registri utente
* Verifica che l’interruzione sai dovuta ad una mancanza di pagina
* Controlla della correttezza del riferimento alla pagina e determinazione nella locazione della pagina nel disco.
* Lettura dal disco e trasferimento nel frame libero
* Durante l’attesa, allocazione della CPU a un altro processo utente (facoltativo , caso in cui la CPU può essere assegnata a un altro processo)
* Ricezione di un’interruzione dal controllore del disco
* Salvataggio dei registri e dello stato dell’altro processo utente
* Verifica della provenienza dell’interruzione dal disco
* Aggiornamento della tabella delle pagine e di altre tabelle per segnalare che la pagina richiesta è attualmente presente in memoria
* Attesa che la CPU sia nuovamente assegnata a questo processo
* Recupero dei registri utente.

{-esempio pag 364 - 365 pdf

-pag 350 351 libro }

Per mantenere a un livello ragionevole il rallentamento dovuto alla paginazione, si può permettere meno di un’assenza di pagina ogni 399.990 accessi alla memoria. In un sistema con paginazione su richiesta, e cioè importante tenere bassa la frequenza delle assenze di pagine, altrimenti il tempo effettivo d’accesso

aumenta, rallentando molto l’esecuzione del processo.

Un altro aspetto della paginazione su richiesta è la gestione e l’uso generale dell’area d’avvicendamento. L’I/O di un disco relativo all’area d’avvicendamento è generalmente più rapido di quello relativo al file system, poichè essa è composta di blocchi assai più grandi e non s’impiegano ricerche di file e metodi d’allocazione indiretta (Capitolo 12). Perciò il sistema può migliorare l’efficienza della paginazione copiando tutta l’immagine di un file nell’area d’avvicendamento all’avvio del processo e di lì eseguire la paginazione su richiesta.

Un’altra possibilità consiste nel richiedere inizialmente le pagine al file system, ma scrivere le

pagine nell’area d’avvicendamento al momento della sostituzione. Questo metodo assicura che si leggano sempre dal file system solo le pagine necessarie, ma che tutta la paginazione successiva sia fatta dall’area d’avvicendamento.

**Copiatura su scrittura**

La generazione dei processi tramite fork( ) può inizialmente evitare la paginazione su richiesta per mezzo di una tecnica simile alla condivisione delle pagine (Paragrafo 8.4.4), che garantisce la generazione dei processi riuscendo anche a minimizzare il numero di pagine allocate al nuovo processo.

Si ricordi che la chiamata di sistema fork( ) crea un processo figlio come duplicato del genitore. Nella sua versione originale la fork ( ) creava per il figlio una copia dello spazio d’indirizzi del genitore, duplicando le pagine appartenenti al processo genitore.

Considerando che molti processi figli eseguono subito dopo la loro creazione la chiamata di sistema **exec ()**, questa operazione di copiatura risulta inutile. In alternativa, si può impiegare una tecnica nota come **copiatura su scrittura** *(copy-on-write)*, il cui funzionamento si fonda sulla condivisone iniziale delle pagine da parte dei processi genitori e dei processi figli. Le pagine condivise si contrassegnano come pagine da copiare su scrittura, a significare che, se un processo (genitore o figlio) scrive su una pagina condivisa, il sistema deve creare una copia di tale pagina.

-Si consideri ad esempio un processo figlio che cerchi di modificare una pagina contenente parti della pila; il sistema operativo considera questa una pagina da copiare su scrittura e ne crea una copia nello spazio degli indirizzi del processo figlio. Il processo figlio modifica la sua copia della pagina e non la pagina appartenente al processo genitore. E' chiaro che, adoperando la tecnica di copiatura su scrittura, si copiano soltanto le pagine modificate da uno dei due processi, mentre tutte le altre sono condivisibili dai processi genitore e figlio. Si noti inoltre che soltanto le pagine modificabili si devono contrassegnare come da copiare su

scrittura, mentre quelle che non si possono modificare (ad esempio, le pagine contenenti

codice eseguibile) sono condivisibili dai processi genitore e figlio. La tecnica di copiatura su

scrittura è piuttosto comune e si usa in diversi SO , tra i quali Windows XP , Linux e Solaris.

Quando è necessaria la duplicazione di una pagina secondo la tecnica di copiatura su scrittura, è importante capire da dove si attingerà la pagina libera necessaria. Molti sistemi operativi forniscono, per queste richieste, un gruppo *(pool)* di pagine libere, che di solito si assegnano quando la pila o il cosiddetto *heap* di un processo devono espandersi, oppure proprio per gestire pagine da copiare su scrittura. L’allocazione di queste pagine di solito avviene secondo una tecnica nota come **azzeramento su richiesta** (.*zero-fill-on demand*); prima dell’allocazione si riempiono di zeri le pagine, cancellandone in questo modo tutto il contenuto precedente.

Diverse versioni di UNIX (compreso Solaris e Linux) offrono anche una variante della chiamata di sistema fork ( ) - detta vfork ( ) (per *virtual memory fork).* La vfork ( ) offre un’alternativa all’uso della fork ( ) con copiatura su scrittura. Con la vfork ( )il processo genitore viene sospeso e il processo figlio usa lo spazio d’indirizzi del genitore. Poichè la vfork( ) non usa la copiatura su scrittura, se il processo figlio modifica qualche pagina dello spazio d’indirizzi del genitore, le pagine modificate saranno visibili al processo genitore

non appena riprenderà il controllo. Per assicurarsi che il processo figlio non modifichi lo spazio d’indirizzi del genitore è quindi necessaria molta attenzione nell’uso di vfork ( ). La chiamata di sistema vfork ( ) è adatta al caso in cui il processo figlio esegua una exec ( ) immediatamente dopo la sua creazione. Poichè non richiede alcuna copiatura delle pagine, la vfork ( ) è un metodo di creazione dei processi molto efficiente, in alcuni casi impiegato per realizzare le interfacce degli interpreti dei comandi in UNIX.

**Sostituzione delle pagine**

Se un processo di 10 pagine ne impiega effettivamente solo la metà, la paginazione su richiesta fa risparmiare l’I/O necessario per caricare le 5 pagine che non sono mai usate. Il grado di multiprogrammazione potrebbe essere aumentato eseguendo il doppio dei processi. Quindi, disponendo di 40 frame, si potrebbero eseguire 8 processi anzichè i 4 che si eseguirebbero se ciascuno di loro richiedesse 10 blocchi di memoria, 5 dei quali non sarebbero mai usati.

Aumentando il grado di multiprogrammazione, si **sovrassegna** la memoria. Eseguendo 6 processi, ciascuno dei quali è formato da 10 pagine, di cui solo 5 sono effettivamente usate, s’incrementerebbero l’utilizzo e la produttività della CPU e si risparmierebbero 10 frame. Tuttavia è possibile che ciascuno di questi processi, per un insieme particolare di dati, abbia improvvisamente necessità di impiegare tutte le 10 pagine, perciò sarebbero necessari 60 frame, mentre ne sono disponibili solo 40.

La **sovrallocazione** *(over-allocation)* si può illustrare come segue. Durante l’esecuzione di un processo utente si verifica un’assenza di pagina. Il sistema operativo determina la locazione del disco in cui risiede la pagina desiderata, ma poi scopre che la lista dei frame liberi è *vuota:* tutta la memoria è in uso;

A questo punto il sistema operativo può scegliere tra diverse possibilità:

- può terminare il processo utente ;

- può scaricare dalla memoria l'intero processo (riducendo quindi il grado di

multiprogrammazione del sistema);

A noi interessa la **sostituzione delle pagine**.

## Sostituzione di pagina

La sostituzione delle pagine segue il seguente criterio: se nessun frame è libero si può liberarne uno attualmente inutilizzato.

È possibile liberarlo scrivendo il suo contenuto nell’area d’avvicendamento e modificando la tabella delle pagine per indicare che la pagina non si trova più in memoria. Il frame liberato si può utilizzare per contenere la pagina che ha causato l’eccezione. Quindi:

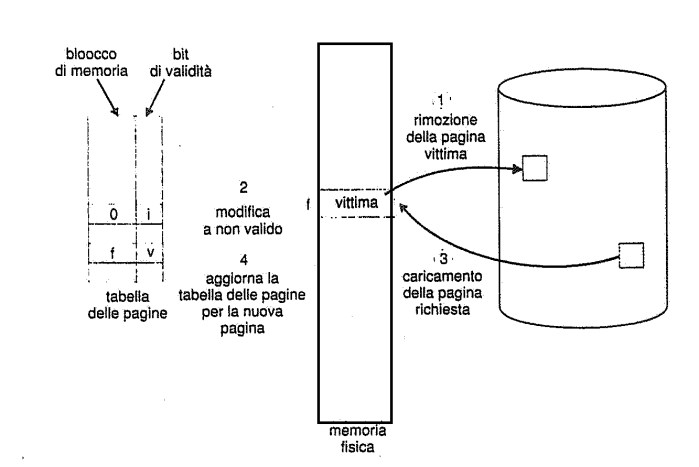
* S’individua la locazione nel disco della pagina richiesta
* Si cerca un frame libero (se esiste lo si usa altrimenti si libera scrivendo la pagina vittima nel disco)
* Si scrive la pagina richiesta nel frame appena liberato
* Si riavvia il processo utente

Dato che sono necessari 2 trasferimenti se non esiste nessun frame libero, questo sovraccarico si può ridurre usando il **bit di modifica**. L’architettura deve disporre di un bit che imposta automaticamente ogni volta che nella pagina si scrive una parola , indicando che la pagina è stata modificata. Quando si sceglie una pagina da sostituire si verifica il suo bit di modifica, se è inattivo la si può cancellare perché esiste una copia sul disco altrimenti bisogna essere prima copiata e poi sostituita. La sostituzione delle pagine è fondamentale al fine della paginazione su richiesta.

Per realizzare la paginazione su richiesta, è necessario risolvere 2 problemi principali: occorre sviluppare un algoritmo di allocazione dei frame e uno per la sostituzione delle pagine.

Esistono molti algoritmi di sostituzione delle pagine; probabilmente ogni sistema operativo ha il proprio schema di sostituzione. E' quindi necessario stabilire un criterio per selezionare un algoritmo di sostituzione particolare; generalmente si sceglie quello con la minima frequenza delle assenze di pagine (*page-fault rate*).

\*\*\*Sostituzione di pagina\*\*\*

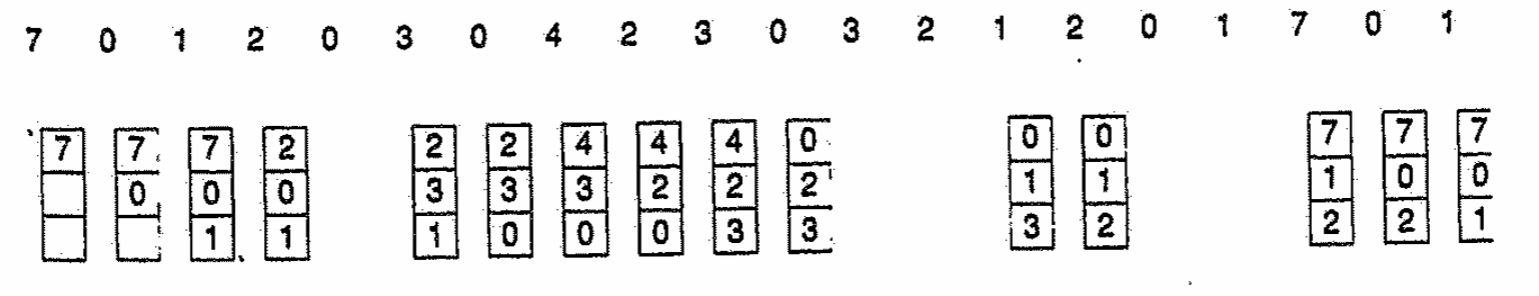


**Sostituzione delle pagine secondo l'ordine d'arrivo (fifo)**

L’algoritmo di sostituzione delle pagine più semplice è un algoritmo FIFO. Questo algoritmo associa a ogni pagina l’istante di tempo in cui quella pagina è stata portata in memoria. Se si deve sostituire una pagina, si seleziona quella presente in memoria da più tempo.

Occorre notare che non è strettamente necessario registrare l’istante in cui si carica una pagina in memoria;infatti si possono strutturare secondo una coda FIFO tutte le pagine presenti in memoria.

In questo caso si sostituisce la pagina che si trova nel primo elemento della coda. Quando si carica una pagina in memoria, la si inserisce nell’ultimo elemento della coda.

.

esempio pag 357 del libro.

L’algoritmo FIFO di sostituzione delle pagine è facile da capire e da programmare; tuttavia la sue prestazioni non sono sempre buone. La pagina sostituita potrebbe essere un modulo di inizializzazione usato molto tempo prima e che non serve più, ma potrebbe anche contenere una variabile molto usata, inizializzata precedentemente e ancora in uso.

\*\* **L' anomalia di Belady,**  *riflette il fatto che, con alcuni algoritmi di sostituzione delle pagine, la frequenza delle assenze di pagine può aumentare con l’aumentare del numero dei frame assegnati. A prima vista sembra logico supporre che fornendo più memoria a un processo le prestazioni di quest’ultimo migliorino. Si è invece notato che questo*

*presupposto non sempre è vero; l’anomalia di Belady ne è la prova.*

**Sostituzione ottimale delle pagine**

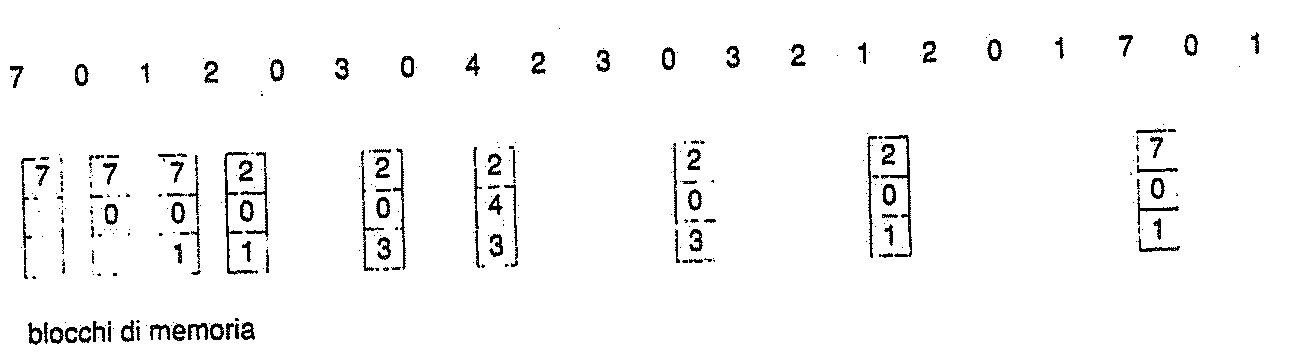
In seguito alla scoperta dell’anomalia di Belady, la ricerca si è diretta verso un **algoritmo ottimale di sostituzione delle pagine.** Tale algoritmo è quello che fra tutti gli algoritmi presentala minima frequenza di assenze di pagine e non presenta mai l’anomalia di Belady.

Questo algoritmo esiste ed è stato chiamato OPT o MIN. E' semplicemente:

*si sostituisce la pagina che non si userà per il periodo di tempo più lungo*

L’uso di questo algoritmo assicura la frequenza di assenze di pagine più bassa possibile per un numero fisso di frame.

Sfortunatamente l’algoritmo ottimale di sostituzione delle pagine è difficile da realizzare, perchè richiede la conoscenza futura della successione dei riferimenti. Una situazione analoga si è riscontrata con l’algoritmo SJF di scheduling della CPU.



esempio pag. 359 libro

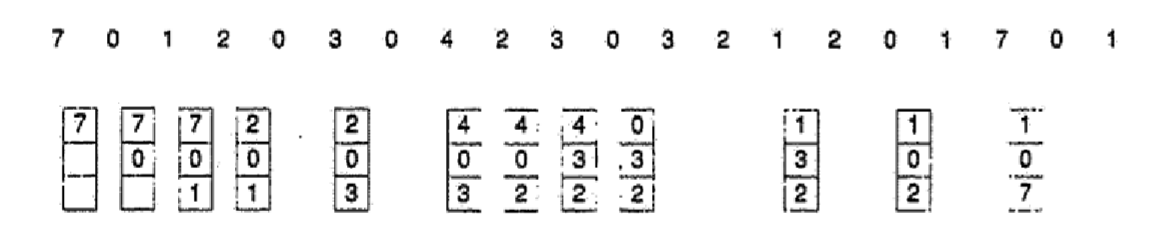
**Sostituzione delle pagine usate meno recentemente (lru) least recently used**

La distinzione fondamentale tra gli algoritmi FIFO e OPT consiste nel fatto che l’algoritmo FIFO impiega l’istante in cui una pagina è stata caricata in memoria, mentre l’algoritmo OPT impiega l’istante in cui una pagina è *usata.*

Usando come approssimazione di futuro vicino un passato recente, si sostituisce la pagina che non è stata usta per il periodo più lungo noto come LRU.

La sostituzione LRU associa a ogni pagina l’istante in cui e stata usata per l’ultima volta. Quando occorre sostituire una pagina, l’algoritmo LRU sceglie quella che non è stata usata per il periodo più lungo.

Questa strategia costituisce l’algoritmo ottimale di sostituzione delle pagine con ricerca all’indietro nel tempo.



esempio pag 360 del libro

Il criterio **LRU** si usa spesso come algoritmo di sostituzione delle pagine ed èconsiderato valido. Il problema principale riguarda la realizzazione della sostituzione stessa. Un algoritmo di sostituzione delle pagine **LRU** può richiedere una notevole assistenza da parte dell’architettura del sistema di calcolo. Il problema consiste nel determinare un ordine per i frame definito secondo il momento dell’ultimo uso. Si possono realizzare le due seguenti soluzioni:

♦Contatori. Nel caso più semplice, a ogni elemento della tabella delle pagine si associa un campo del momento d’uso, e alla **CPU** si aggiunge un contatore che si incrementa a ogni riferimento alla memoria. Ogni volta che si fa un riferimento a una pagina, si copia il contenuto del registro contatore nel campo del momento d’uso nella tabella relativa a quella specifica pagina. In questo modo è sempre possibile conoscere il momento in cui è stato fatto l’ultimo riferimento a ogni pagina. Si sostituisce la pagina con il valore associato più piccolo. Questo schema implica una ricerca all’interno della tabella delle pagine per individuare la pagina usata meno recentemente **(LRU),** e una scrittura in memoria (nel campo del momento d’uso della tabella delle pagine) per ogni accesso alla memoria.

♦ Pila. Un altro metodo per la realizzazione della sostituzione delle pagine **LRU** prevede la presenza di una pila dei numeri delle pagine. Ogni volta che si fa un riferimento a una pagina, la si estrae dalla pila e la si

colloca in cima a quest’ultima. In questo modo, in cima alla pila si trova sempre la pagina usata per ultima, mentre in fondo si trova la pagina usata meno recentemente. Poichè alcuni elementi si devono estrarre dal centro della pila, la migliore realizzazione si ottiene usando una lista doppiamente concatenata, con un puntatore all’elemento iniziale e uno a quello finale. Per estrarre una pagina dalla pila e collocarla in cima, nel caso peggiore è necessario modificare sei puntatori. Ogni aggiornamento è un po’ più costoso, ma per una sostituzione non si deve compiere alcuna ricerca; il puntatore dell’elemento di coda punta al fondo della pila, vale a dire la pagina usata meno recentemente. Questo metodo è adatto soprattutto alle realizzazioni programmate (o microprogrammate) della sostituzione delle pagine **LRU.**

Nè la sostituzione ottimale nè quella **LRU** sono soggette all’anomalia di Belady. Esiste una classe di algoritmi di sostituzione delle pagine, chiamati algoritmi a pila, che non presenta l’anomalia di Belady.

**Sostituzione delle pagine per approssimazione a LRU**

Sono pochi i sistemi di calcolo che dispongono di un’architettura adatta a una vera sostituzione **LRU** delle pagine. Nei sistemi che non offrono tali caratteristiche specifiche si devono impiegare altri algoritmi di sostituzione delle pagine, ad esempio l’algoritmo **FIFO.** Molti sistemi tuttavia possono fornire un aiuto: un **bit di riferimento**.

Il **bit di riferimento** a una pagina è impostato automaticamente dall’architettura del sistema ogni volta che si fa un riferimento a quella pagina, che sia una lettura o una scrittura su qualsiasi byte della pagina.

I **bit di riferimento** sono associati a ciascun elemento della tabella delle pagine.

Inizialmente, il sistema operativo azzera tutti i bit. Quando s’inizia l’esecuzione di un processo utente, l’architettura del sistema imposta a 1 il bit associato a ciascuna pagina cui si fa riferimento.

Dopo qualche tempo è possibile stabilire quali pagine sono state usate semplicemente esaminando i **bit di riferimento**.

Non è però possibile conoscere *l'ordine* d’uso. E' questa l’informazione alla base di molti algoritmi per la sostituzione delle pagine che approssimano **LRU.**

**-Algoritmo con bit supplementari di riferimento**

Ulteriori informazioni sull’ordinamento si possono ottenere registrando i bit di riferimento a intervalli regolari. E' possibile conservare in una tabella in memoria una serie di bit per ogni pagina.

A intervalli regolari, ad esempio di 100 millisecondi, un segnale d’interruzione del timer del sistema trasferisce il controllo al sistema operativo. Questo sposta il bit di riferimento per ciascuna pagina nel bit più significativo della sequenza, traslando gli altri bit a destra di 1 bit e scartando il bit meno significativo.

Questi registri a scorrimento, ad esempio di 8 bit, contengono l’ordine d’uso delle pagine relativo agli ultimi otto periodi di tempo.

{Se il registro a scorrimento contiene la successione di bit 00000000, significa che la pagina

associata non èstata usata da otto periodi di tempo; a una pagina usata almeno una volta per ogni periodo corrisponde la successione 11111111 nel registro a scorrimento.

Una pagina cui corrisponde la successione 11000100 nel relativo registro, è stata usata più recentemente(più da poco tempo) di quanto non lo sia stata una cui èassociata la successione 01110111.}

Interpretando queste successioni di bit come interi senza segno, la pagina cui èassociato il numero minore èla pagina **LRU,** e può essere sostituita.

In ogni caso l’unicità dei numeri non ègarantita. Si possono sostituire (o scaricare dalla memoria all’area d’avvicendamento) tutte le pagine con il valore minore, oppure si può ricorrere a una selezione **FIFO.**

Il numero dei bit può ovviamente essere variato: si stabilisce secondo l’architettura disponibile per accelerarne al massimo la modifica. Nel caso limite tale numero si riduce a zero, lasciando soltanto il bit di riferimento e definendo un algoritmo noto come algoritmo di sostituzione delle pagine con **seconda chance**.

**-Algoritmo con seconda chance**

L’algoritmo di base per la sostituzione con **seconda chance** è un algoritmo di sostituzione di tipo **FIFO.**

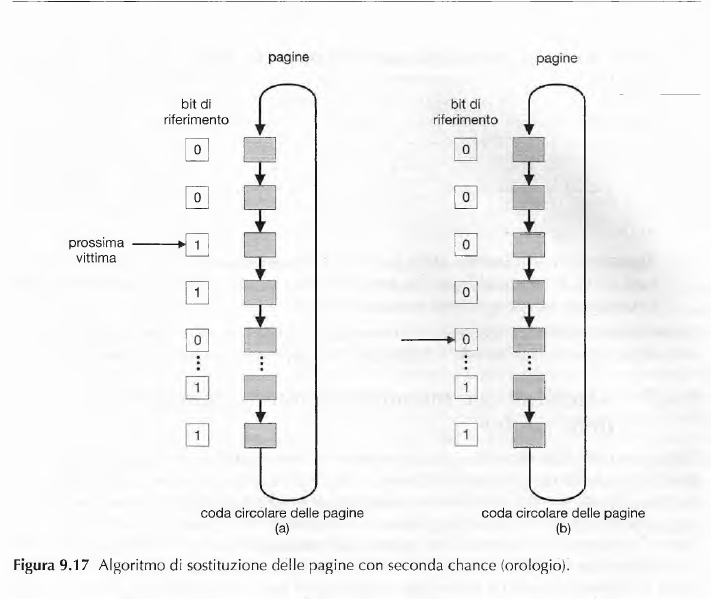
**-**Dopo aver selezionato una pagina si controlla il **bit di riferimento**, se il suo valore è 0, si sostituisce la pagina; se il **bit di riferimento** è impostato a 1, si da una **seconda chance** alla pagina e la selezione passa alla successiva pagina **FIFO.**

Quando una pagina riceve la **seconda chance**, si azzera il suo bit di riferimento e si aggiorna il suo istante d’arrivo al momento attuale. In questo modo, una pagina cui si offre una **seconda chance** non viene mai sostituita finchè tutte le altre pagine non siano state sostituite, oppure non sia stata offerta loro una **seconda chance**. Inoltre, se una pagina è usata abbastanza spesso, in modo che il suo bit di riferimento sia sempre impostato a 1, non viene mai sostituita.

-Un metodo per realizzare l’algoritmo con **seconda chance**, detto anche a orologio *(clock),* è basato sull’uso di una coda circolare, in cui un puntatore indica qual è la prima pagina da sostituire. Quando serve un frame, si fa avanzare il puntatore finchè non si trovi in corrispondenza di una pagina con il bit di riferimento 0; a ogni passo si azzera il bit di riferimento appena esaminato. Una volta trovata una pagina “vittima”, la si sostituisce e si inserisce la nuova pagina nella coda circolare nella posizione corrispondente.

-Si noti che nel caso peggiore, quando tutti i bit sono impostati a 1, il puntatore percorre un ciclo su tutta la coda, dando a ogni pagina una **seconda chance**.

Prima di selezionare la pagina da sostituire, azzera tutti i bit di riferimento. Se tutti i bit sono a 1, la sostituzione con **seconda chance** si riduce a una sostituzione **FIFO.**



**-Algoritmo con seconda chance migliorato**

L’algoritmo con **seconda chance** descritto precedentemente si può migliorare considerando i bit di riferimento e di modifica come una coppia ordinata, con cui si possono ottenere le seguenti quattro classi:

1. (0, 0) ne' recentemente usato ne' modificato (migliore pagina da sostituire);

2. (0, 1) non usato recentemente, ma modificato (la pagina non cosi buona poichè prima di essere sostituita deve essere scritta in memoria secondaria);

3. (1,0) usato recentemente ma non modificato (probabilmente la pagina sarà presto ancora usata);

4. (1,1) usato recentemente e modificato (probabilmente la pagina sarà presto ancora usata e dovrà essere scritta in memoria secondaria prima di essere sostituita).

Ogni pagina rientra in una di queste quattro classi. Alla richiesta di una sostituzione di pagina, si usa lo stesso schema impiegato nell’algoritmo a orologio, ma anzichè controllare se la pagina puntata ha il bit di riferimento impostato a 1, si esaminano le classi cui la pagina appartiene e si sostituisce la prima pagina che si trova nella classe minima non vuota.

-Si noti che la coda circolare deve essere scandita più volte prima di trovare una pagina da sostituire.

-La differenza principale tra questo algoritmo e il più semplice algoritmo a orologio è che nel primo si dà la preferenza alle pagine modificate, al fine di ridurre il numero di l/O richiesti.

**Sostituzione delle pagine basata su conteggio**

Esistono molti altri algoritmi che si possono usare per la sostituzione delle pagine. Ad esempio, si potrebbe usare un contatore del numero dei riferimenti fatti a ciascuna pagina, e sviluppare i due seguenti schemi.

**♦ Algoritmo di sostituzione delle pagine meno frequentemente usate** *(least frequently used*, **LFU);** richiede che si sostituisca la pagina con il conteggio più basso. La ragione di questa scelta è che una pagina usata attivamente deve avere un conteggio di riferimento alto.

-Il punto debole di questo algoritmo è rappresentato dai casi in cui una pagina è usata molto intensamente durante la fase iniziale di un processo, ma poi non viene più usata.

Poichè è stata usata intensamente il suo conteggio è alto, quindi rimane in memoria anche se non è più necessaria. Una soluzione può essere quella di spostare i conteggi a destra di un bit a intervalli regolari, formando un conteggio per l'uso medio con esponente decrescente.

**♦ Algoritmo di sostituzione delle pagine piu frequentemente usate** *(most frequently used*, **MFU);** è basato sul fatto che, probabilmente, la pagina con il contatore più bassoè stata appena inserita e non è stata ancora usata.

Le sostituzioni MFU e LFU non sono molto comuni, poichè la realizzazione di questi algoritmi è abbastanza onerosa e comunque non approssimano bene la sostituzione OPT.

**Algoritmi con memorizzazione transitoria delle pagine**

Oltre a uno specifico algoritmo, per la sostituzione delle pagine si usano spesso anche altre procedure; ad esempio, i sistemi hanno generalmente un gruppo di frame liberi *(pool offree* *frames).*

Quando si verifica un' assenza di pagina, si sceglie innanzitutto un frame vittima, ma prima che sia scritta in memoria secondaria, si trasferisce la pagina richiesta in un frame libero del gruppo.

Questa procedura permette al processo di ricominciare al più presto, senza attendere che la pagina vittima sia scritta in memoria secondaria.

Quando nel seguito si scrive la vittima in memoria secondaria, si aggiunge il suo frame al gruppo dei frame liberi.

Quest’idea si può estendere conservando una lista delle pagine modificate: ogniqualvolta il dispositivo di paginazione è inattivo, si sceglie una pagina modificata, la si scrive nel disco e si reimposta il suo bit di modifica.

-Questo schema aumenta la probabilità che, al momento della selezione per la sostituzione, la pagina non abbia subito modifiche e non debba essere scritta in memoria secondaria.

Anche un’altra modifica prevede l’uso di un gruppo di frame liberi ma, in questo caso, per ricordare quale pagina era contenuta in ciascun frame. Poichè quando si scrive il contenuto di un frame in un disco tale contenuto non cambia, se è necessaria, la vecchia pagina è ancora utilizzabile direttamente dal gruppo dei frame liberi, prima che sia riusato quel frame. In questo caso non è necessario alcun I/O.

Se si verifica un’assenza di pagina occorre controllare se la pagina richiesta si trova nel gruppo dei frame liberi; se non c’è si deve individuare un frame libero e trasferirvi la pagina.

**Applicazioni e sostituzione della pagina**

In alcuni casi le applicazioni che accedono ai dati tramite la memoria virtuale del sistema operativo non conseguono prestazioni migliori di quelle che il sistema, senza impiegare alcun buffer, potrebbe offrire. Alcuni SO permettono a certi programmi di utilizzare una partizione del disco come un array sequenziale di blocchi logici senza ricorrere alle strutture del file system.

Un simile array è anche detto **disco di basso livello** e il relativo I/O è denominato di basso livello.

Il **disco di basso livello** salta tutti i servizi del file system(come la paginazione su richiesta dei file in ingresso e in uscita, i lock dei file, il prefetching, l’allocazione dello spazio, i nomi dei file e le directory).

Si noti come, sebbene alcune applicazioni siano più efficienti nel gestire i propri servizi specifici di memorizzazione sul disco di basso livello, quasi tutte hanno una resa migliore quando operano con i servizi regolari del file system.

**Allocazione dei frame**

Come esempio, è possibile considerare un caso in cui 93 frame liberi si debbano assegnare a due processi.

Il caso più semplice di memoria virtuale è il sistema con utente singolo. Si consideri un sistema monoutente che disponga di 128 KB di memoria, con pagine di 1 KB. Complessivamente sono presenti 128 frame.

Il sistema operativo può occupare 35 KB, lasciando 93 frame per il processo utente. In condizioni di paginazione su richiesta pura, tutti i 93 blocchi di memoria sono inizialmente posti nella lista dei frame liberi. Quando comincia l’esecuzione, il processo utente genera una sequenza di eccezioni di pagine mancanti.

Le prime 93 pagine assenti ricevono i frame liberi dalla lista. Una volta esaurita quest’ultima, per stabilire quale tra le 93 pagine presenti in memoria si debba sostituire con la novantaquattresima, si può usare un algoritmo di sostituzione delle pagine. Terminato il processo, si reinseriscono i 93 frame nella lista dei frame liberi.

Questa strategia è semplice, ma può subire molte variazioni.

Si può richiedere che il sistema operativo assegni tutto lo spazio richiesto dalle proprie strutture dati attingendo dalla lista dei frame liberi. Quando questo spazio è inutilizzato dal sistema operativo può essere sfruttato per la paginazione utente.

**Un’altra variante** prevede di riservare sempre tre frame liberi, in modo che quando si verifica un’assenza di pagina sia disponibile un frame libero in cui trasferire la pagina richiesta. Mentre ha luogo il trasferimento della pagina, si può fare una sostituzione, la pagina coinvolta viene poi scritta nel disco mentre il processo utente continua l’esecuzione.

Sono possibili anche altre varianti, ma la strategia di base è chiara: **al processo utente si assegna qualsiasi frame libero**.

**Numero minimo di frame**

Le strategie di allocazione sono soggette a parecchi vincoli. Non si possono assegnare più frame di quanti siano disponibili. Inoltre è necessario assegnare almeno un numero minimo di frame. Una delle ragioni per allocare sempre un numero minimo di frame è legata alle prestazioni. Al decrescere del numero dei frame allocati a ciascun processo aumenta la frequenza di mancanza di pagina con conseguente ritardo dell’esecuzione dei processi. Quando si verifica un'assenza di pagina prima che sia stata completata l’esecuzione quest’ultima deve essere riavviata. I frame disponibili devono essere in numero sufficiente per contenere tutte le pagine cui ogni singola istruzione può far riferimento. Il numero minimo dei frame per ciascun processo è definito dall’architettura del calcolatore mentre il numero massimo è definito dalla quantità di memoria fisica disponibile.

**Algoritmi di allocazione**

Il modo più semplice per suddividere *m* frame tra *n* processi è quello per cui a ciascuno si da una parte uguale, *m/n* frame.

Dati 93 frame e cinque processi, ogni processo riceve 18 frame. I tre frame lasciati liberi si potrebbero usare come gruppo dei frame liberi. Questo schema è chiamato **allocazione uniforme.**

Un' alternativa consiste nel riconoscere che diversi processi hanno bisogno di quantità di memoria diverse. Si consideri un sistema con frame di 1 KB. Se un piccolo processo utente di 10 KB e una base di dati interattiva di 127 KB sono gli unici due processi in esecuzione su un sistema con 62 frame liberi, non ha senso allocare a ciascun processo 31 frame. Al processo utente non ne servono più di 10, quindi gli altri 21 sarebbero semplicemente sprecati.

Per risolvere questo problema è possibile ricorrere all’**allocazione proporzionale,** secondo cui la memoria disponibile si assegna a ciascun processo secondo la propria dimensione.

Si supponga che si sia la dimensione di memoria virtuale per il processo pi. Si definisce la seguente quantità:

S=∑(si)

Quindi se il numero totale di frame disponibili è m al processo pi si assegnano ai frame, dove ai è approssimativamente:

ai=si/S\*m

Naturalmente è necessario scegliere ciascun ai in modo che sia un intero maggiore del numero minimo di frame richiesti dalla struttura della serie di istruzioni di macchina e in modo che la somma di tutti gli ainon sia maggiore di *m.*

Usando **l’allocazione proporzionale**, per suddividere 62 frame tra due processi, uno di 10 e uno di 127 pagine, si assegnano rispettivamente 4 e 57 frame, infatti:

10/137 x 62 ~ 4

127/137 x 62 ~ 57.

In questo modo entrambi i processi condividono i frame disponibili secondo le rispettive necessità, e non in modo uniforme.

Sia **nell’allocazione uniforme** che in quella **proporzionale**,l’allocazione a ogni processo può variare rispetto al livello di multiprogrammazione. In entrambe le allocazioni i processi sono trattati a bassa priorità, una soluzione prevede uno schema di allocazione proporzionale in cui il rapporto dei frame non dipende dalle dimensioni relative ai processi ma dalla priorità degli stessi oppure una combinazione di dimensioni e priorità.

**Allocazione globale e allocazione locale**

Nei casi in cui vi siano più processi in competizione per i frame, gli algoritmi di sostituzione delle pagine si possono classificare in due categorie generali: **sostituzione globale** e **sostituzione locale.**

**-**La **sostituzione globale** permette che per un processo si scelga un frame per la sostituzione dall’insieme di tutti i frame, anche se quel frame è correntemente allocato a un altro processo; un processo può dunque sottrarre un frame a un altro processo.

-La **sostituzione locale** richiede invece che per ogni processo si scelga un frame solo dal proprio insieme di frame.

Con la strategia di **sostituzione locale**, il numero di blocchi di memoria assegnati a un processo non cambia. Con la **sostituzione globale**, invece, può accadere che per un certo processo si selezionino solo frame allocati ad altri processi, aumentando cosi il numero di frame assegnati a quel processo, purchè per altri non si scelgano per la sostituzione i *propri* frame.

L’algoritmo di **sostituzione globale** risente di un **problema**: un processo non può controllare la propria frequenza di assenze di pagine *(page-fault rate),* infatti l’insieme di pagine che si trova in memoria per un processo non dipende solo dal comportamento di paginazione di quel processo, ma anche dal comportamento di paginazione di altri processi.

Con l’algoritmo di **sostituzione locale** questo problema non si presenta. Infatti l’insieme di pagine in memoria per un processo subisce l’effetto del comportamento di paginazione di quel solo processo.

Generalmente, la **sostituzione globale** genera una **maggiore produttività del sistema**, e perciò è il metodo più usato.

**Accesso non uniforme alla memoria**

Spesso in sistemi informatici con processori multipli, un certo processore può accedere in alcune regioni della memoria più rapidamente rispetto ad altre. Tali differenze sono causate dalla modalità di interconnessione tra processori e memoria all’interno del sistema. Un sistema è costituito da diverse schede madri ognuna contenenti più processori e una parte della memoria . Le schede sono connesse in vari modi e i processori di una particolare scheda possono accedere alla memoria della scheda stessa in meno tempo rispetto ad altre schede del sistema.

I sistemi nei quali i tempi d’accesso alla memoria variano in modo significativo sono generalmente detti **sistemi con accesso non uniforme alla memoria** e senza eccezioni sono più lenti dei sistemi nei quali memoria e processori risiedono sulla stessa scheda madre.

**Paginazione degenere (thrashing)**

Se il numero dei frame allocati a un processo con priorità bassa diviene inferiore al numero minimo richiesto dall’architettura del calcolatore, occorre sospendere l’esecuzione del processo, e quindi togliere le pagine restanti, liberando tutti i frame allocati.

Questa operazione introduce un livello intermedio di scheduling per la gestione dell’entrata e dell’uscita dei processi in memoria centrale.

Infatti, si consideri un qualsiasi processo che non disponga di un numero di frame “sufficiente”.

Anche se tecnicamente si può ridurre al valore minimo il numero dei frame allocati, esiste un certo (in generale grande) numero di pagine in **uso attivo**. Se non dispone di questo numero di frame, il processo accusa immediatamente un’assenza di pagina. A questo punto si deve sostituire qualche pagina; ma, poichè tutte le sue pagine sono in **uso attivo**, si deve sostituire una pagina che sarà immediatamente necessaria, e di conseguenza si verificano subito parecchie assenze di pagine.

Il processo continua a subire assenze di pagine, facendo sostituire pagine che saranno immediatamente trattate come assenti e dovranno essere riprese.

Questa intensa quanto degenere paginazione (nota come *thrashing)* si verifica quando si spende più tempo per la paginazione che per l’esecuzione dei processi.

**Cause della paginazione degenere**

La degenerazione dell’attività di paginazione causa parecchi problemi di prestazioni. Si consideri il seguente scenario, basato sul comportamento effettivo dei primi sistemi di paginazione.

II sistema operativo vigila sull’utilizzo della CPU. Se questo è basso, aumenta il grado di multiprogrammazione introducendo un nuovo processo. Si usa un algoritmo di sostituzione delle pagine globale, che sostituisce le pagine senza tener conto del processo al quale appartengono.

{Per ora si ipotizzi che un processo entri in una nuova fase d’esecuzione e richieda più frame; se ciò si verifica si ha una serie di assenze di pagine, cui segue la sottrazione di nuove pagine ad altri processi. Questi processi hanno però bisogno di quelle pagine e quindi subiscono anch'essi delle assenze di pagine, con conseguente sottrazione di pagine ad altri processi. Per effettuare il caricamento e lo scaricamento delle pagine per questi processi si deve usare il dispositivo di paginazione. Mentre si mettono i processi in coda per il dispositivo di paginazione, la coda dei processi pronti per l’esecuzione si svuota, quindi l’utilizzo della CPU diminuisce.

Lo scheduler della CPU rileva questa riduzione dell’utilizzo della CPU e *aumenta* il grado di multiprogrammazione. Si tenta di avviare il nuovo processo sottraendo pagine ai processi in esecuzione, causando ulteriori assenze di pagine e allungando la coda per il dispositivo di paginazione. L’utilizzo della CPU scende ulteriormente, e lo scheduler della CPU tenta di aumentare ancora il grado di multiprogrammazione.

L’attività di paginazione è degenerata in una situazione patologica che fa precipitare la produttività del sistema. La frequenza delle assenze di pagine aumenta in modo impressionante, e di conseguenza aumenta il tempo effettivo d’accesso alla memoria. I processi non svolgono alcun lavoro, poichè si sta spendendo tutto il tempo nell’attività di paginazione. }

Se l'attività di paginazione degenera (e fa crollare l’utilizzo della CPU), per aumentare l’utilizzo della CPU occorre *ridurre* il grado di multiprogrammazione.

Gli effetti di questa situazione si possono limitare usando un algoritmo di **sostituzione locale**, o un **algoritmo di sostituzione per priorità**.

-Con la **sostituzione locale**, se un processo ricade nell’attività di paginazione degenere, non può sottrarre frame a un altro processo e quindi provocarne a sua volta la degenerazione. Le pagine si sostituiscono tenendo conto del processo di cui fanno parte.

-Tuttavia, se i processi la cui attività di paginazione degenera rimangono nella coda d’attesa del dispositivo di paginazione per la maggior parte del tempo,il tempo di servizio medio di un’eccezione di pagina mancante aumenta a causa dell’allungamento medio della coda d’attesa del dispositivo di paginazione.

Di conseguenza, il tempo effettivo d’accesso al dispositivo di paginazione aumenta anche per gli altri processi.

-Per evitare il verificarsi di queste situazioni, occorre fornire a un processo tutti i frame di cui necessita.

Per cercare di sapere quanti frame “servano” a un processo si impiegano diverse tecniche.

Il modello dell'insieme di lavoro (*working-set*), comincia osservando quanti siano i frame che un processo sta effettivamente usando. Questo metodo definisce il modello di località d’esecuzione del processo.

Il modello di località stabilisce che un processo, durante la sua esecuzione, si sposta di località in località. Una località è un insieme di pagine usate attivamente.

Generalmente un programma è formato di parecchie località diverse, che sono sovrapponibili.

**Modello dell’insieme di lavoro**

Il **modello dell'insieme di lavoro** (*working-set model*) è basato sull’ipotesi di località. Questo modello usa un parametro Δ*,* per definire la **finestra dell’insieme di** **lavoro.** L’idea consiste nell’esaminare i più recenti Δriferimenti alle pagine. L’insieme di pagine nei più recenti Δriferimenti è l*’insieme di lavoro* .

La precisione dell’insieme di lavoro dipende dalla scelta del valore di Δ*.*

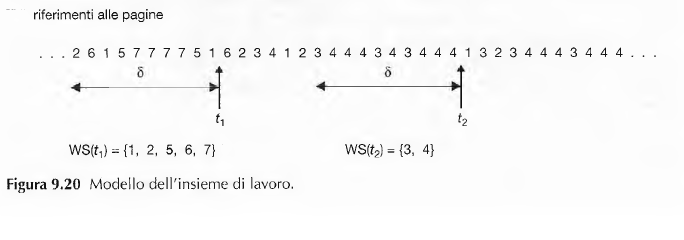
*-Se* Δ *è* troppo piccolo non include l’intera località, se è troppo grande può sovrapporre più località.

Al limite, *se* Δ è infinito l’insieme di lavoro coincide con l’insieme di pagine cui il processo fa riferimentodurante la sua esecuzione.

Calcolando la dimensione WSSi per ciascun processo pi si può determinare la richiesta totale di frame:

D=∑(WSSi)

Ogni processo usa attivamente le pagine del proprio insieme di lavoro. Il SO controlla l’insieme di lavoro di ogni processo e gli assegna un numero di frame sufficiente rispetto alle dimensioni del suo insieme di lavoro. Questa strategia impedisce la paginazione degenere mantenendo il grado di multiprogrammazione più alto possibile, quindi ottimizza l’utilizzo della CPU. Poichè la finestra dell’insieme di lavoro è dinamica la difficoltà è tenere traccia degli elementi che compongono l’insieme di lavoro stesso.

**Frequenza delle assenze di pagine**

Il modello dell'insieme di lavoro riscuote un discreto successo, e la sua conoscenza può servire per la prepaginazione , ma appare un modo alquanto goffo per controllare la degenerazione della paginazione.

La strategia basata sulla frequenza delle assenze di pagine *(page fault frequency,* PFF) è più diretta.

-Il problema specifico è la **prevenzione della paginazione degenere**. La frequenza delle assenze di pagine in tale situazione è alta, ed è proprio questa che si deve controllare.

Se la frequenza delle assenze di pagine è eccessiva, significa che il processo necessita di più frame.

Analogamente, se la frequenza delle assenze di pagine è molto bassa, il processo potrebbe disporre di troppi frame.

-Si può fissare un limite inferiore e un limite superiore per la frequenza desiderata delle assenze di pagine.

Se la frequenza effettiva delle assenze di pagine per un processo oltrepassa il limite superiore, occorre allocare a quel processo un altro frame; se la frequenza scende sotto il limite inferiore, si sottrae un frame a quel processo.

Quindi, per prevenire la paginazione degenere, si può misurare e controllare direttamente la frequenza delle assenze di pagine.

Come nel caso dell'insieme di lavoro, può essere necessaria la sospensione di un processo , se la frequenza delle assenze di pagine aumenta e non ci sono frame disponibili, occorre selezionare un processo e sospenderlo. I frame liberati si distribuiscono ai processi con elevate frequenze di assenze di pagine.