Appunti di Informatica
La memoria virtuale

Introduzione

All'inizio dell'era dei calcolatori, le **memorie** erano piccole e care; di conseguenza, i programmatori passavano buona parte del loro tempo cercando di comprimere i propri programmi in memorie piccolissime; spesso, era perfino necessario usare un algoritmo che lavorava molto più lentamente di un altro, semplicemente perché l'altro era troppo grande per essere conservato nella memoria del calcolatore.

La soluzione tradizionale a questo problema fu, inizialmente, quella di servirsi, oltre che della **memoria principale**, anche di una **memoria secondaria**, ad esempio un disco:

- il programmatore divideva il programma in diversi "pezzi", chiamati **overlay**, ognuno dei quali poteva stare in memoria;
- per eseguire il programma, si inseriva il primo overlay e lo si eseguiva; quando era finito, si caricava l'overlay successivo e lo si eseguiva e così via.

Il problema era che, comunque, era sempre il programmatore a dover dividere il programma in overlay e a decidere dove dovesse essere tenuto ogni overlay nella memoria secondaria; il programmatore doveva anche curare gli spostamenti degli overlay tra memoria secondaria e memoria principale, senza alcun aiuto da parte del calcolatore.

E' ovvio come questa tecnica richiedesse un lavoro eccessivo e noioso da parte del programmatore, per cui, verso il 1960, fu ideato un metodo per l'*esecuzione*

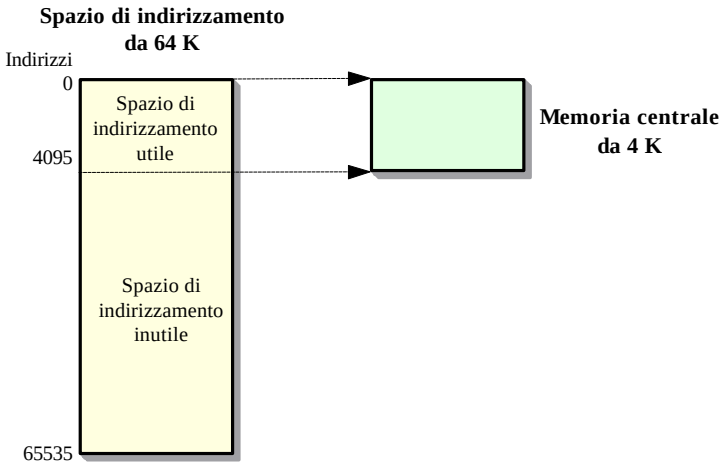
automatica della gestione degli overlay, senza che il programmatore sapesse neppure cosa stesse succedendo. Questo metodo, che ora prende il nome di **memoria virtuale**, libera dunque il programmatore da un grosso lavoro di tipo “amministrativo”. Al giorno d’oggi, praticamente tutti i microprocessori presentano sistemi più o meno sofisticati di memoria virtuale.

La paginazione

L’idea alla base del metodo della *memoria virtuale* è quella di separare due concetti: lo *spazio di indirizzamento* e le *locazioni di memoria*. Per spiegarci meglio, utilizziamo un esempio concreto.

Consideriamo un calcolatore nelle cui *istruzioni* il *campo di indirizzamento* sia di 16 bit: questo significa che un programma eseguito su questo calcolatore può indirizzare 2^{16}=65536 *parole di memoria* (che costituiscono il cosiddetto **spazio di indirizzamento**) e questo a prescindere dalla quantità di memoria a disposizione, dato che il numero di parole indirizzabili dipende appunto dal numero di bit utilizzabili per gli indirizzi e non dal numero di parole effettivamente disponibili.

Supponiamo, ad esempio, che il calcolatore disponga di una memoria di appena 4096 parole. Prima dell’invenzione della memoria virtuale, sarebbe stata necessaria una distinzione, all’interno dello spazio di indirizzamento, tra gli indirizzi al di sotto di 4096 (da 0 a 4095) e quelli uguali a superiori a 4096: i primi costituiscono lo *spazio di indirizzamento utile*, dato che corrispondono a locazioni di memoria effettivamente disponibili, mentre gli altri costituiscono uno *spazio di indirizzamento inutile*, visto che non corrispondono a locazioni disponibili.

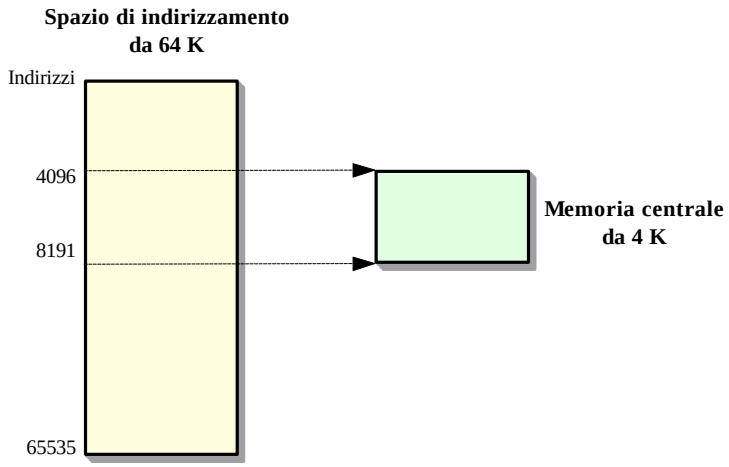


Suddivisione dello spazio di indirizzamento totale in parte “utile” e parte “inutile”

L'idea di separare lo spazio di indirizzamento e gli indirizzi di memoria è la seguente:

- avendo a disposizione 4096 parole di memoria effettiva, siamo certi che, in qualunque istante, possiamo accedere a 4096 parole di memoria;
- tuttavia, non necessariamente queste parole effettive devono corrispondente agli indirizzi compresi tra 0 e 4095: ad esempio, potremmo "imporre" al calcolatore che, da un certo momento in poi, tutte le volte che si usa l'indirizzo 4096, deve essere usata la parola 0 della memoria; quando si usa l'indirizzo 4097 dovrà essere usata la parola 1 della memoria e così via, fino all'indirizzo 8191, al quale dovrà corrispondere la parola 4095 della memoria.

Si realizza, perciò, una corrispondenza uno-ad-uno del tipo schematizzato nella figura seguente:



Corrispondenza tra indirizzi dello spazio di indirizzamento e indirizzi della memoria centrale

In termini molto semplici, possiamo affermare che è stata definita una *funzione dallo spazio di indirizzamento sugli indirizzi di memoria effettivi*: ogni indirizzo nello spazio di indirizzamento viene "tradotto" in un corrispondente indirizzo fisico, ossia indirizzo della memoria centrale effettivamente disponibile.

Supponendo dunque fissata questa particolare "funzione" (o "corrispondenza" o "associazione") tra indirizzi nello spazio di indirizzamento e indirizzi effettivi nella memoria centrale, supponiamo che un dato programma "salti" ad un indirizzo (nello

spazio di indirizzamento) compreso tra 8192 e 12287. Che succede? Se la macchina non è dotata di *memoria virtuale*, il programma provoca un errore e viene stampato un messaggio del tipo "Riferimento a memoria inesistente", dopodiché il programma termina. Il motivo è evidente: il calcolatore non ha alcuna corrispondenza tra gli indirizzi compresi tra 8192 e 12287 e gli indirizzi effettivi di memoria, per cui non sa dove trovare i dati o le istruzioni richieste dal programma e quindi non può che arrestarsi.

Introducendo, invece, la **memoria virtuale**, si avvia il seguente processo:

- il contenuto della memoria centrale viene salvato in una memoria secondaria (ad esempio un disco);
- in questa stessa memoria secondaria vengono individuate le parole da 8192 a 12287;
- le suddette parole vengono caricate nella memoria centrale;
- la *mappa degli indirizzi* (ossia la predetta funzione che associa lo spazio di indirizzamento agli indirizzi effettivi) viene modificata, al fine di far corrispondente gli indirizzi da 8192 a 12287 alle locazioni della memoria centrale da 0 a 4095;
- infine, l'esecuzione continua come se non fosse successo niente di insolito.

In pratica, si è eseguito un qualcosa di simile ad uno "spostamento automatico di overlay". Questa tecnica prende il nome di **paginazione** ed i pezzi di programma trasferiti dalla memoria secondaria in quella principale sono detti **pagine**.

In effetti, quella appena descritta è una versione particolarmente semplificata della paginazione; è invece possibile un modo più sofisticato per far corrispondere lo spazio di indirizzamento con gli indirizzi effettivi di memoria, così come vedremo più avanti.

Chiariamoci dunque sulla terminologia da adottare:

- gli indirizzi a cui il programma può fare riferimento costituiscono il cosiddetto **spazio di indirizzamento virtuale** (di 64K nel caso di indirizzi a 16 bit);
- gli indirizzi di memoria effettivamente cablati costituiscono invece lo **spazio di indirizzamento fisico** (di 4K nel nostro esempio);

- la **mappa di memoria** è infine quella funzione che lega gli indirizzi virtuali con quelli fisici.

Nei nostri discorsi, diamo ovviamente per scontato che la memoria secondaria abbia abbastanza spazio per memorizzare l'intero programma ed i suoi dati.

Ogni programma viene scritto come se ci fosse abbastanza memoria centrale per l'intero spazio di indirizzamento virtuale, anche se non è così: i programmi possono quindi caricare o memorizzare una qualsiasi parola nello spazio di indirizzamento virtuale, così come possono saltare ad una qualsiasi istruzione localizzata in un posto qualsiasi dello spazio di indirizzamento virtuale, senza preoccuparsi che, in realtà, non c'è abbastanza memoria fisica. In poche parole, il programmatore può scrivere programmi senza sapere se la memoria virtuale esiste, dando comunque per scontato che il calcolatore possieda memoria sufficiente ([1]).

La paginazione dà al programmatore l'illusione di una grande memoria centrale, della stessa dimensione dello spazio di indirizzamento, anche se in pratica la memoria centrale disponibile può essere più piccola (o anche più grande) dello spazio di indirizzamento. La "simulazione" di questa grande memoria centrale, garantita dalla paginazione, è **trasparente** al programma (ed al programmatore), il quale cioè non si accorge della sua presenza (a meno di non eseguire test particolari): tutte le volte che si fa riferimento ad un indirizzo, la corrispondente parola sembra comunque sempre presente.

Meccanismi di paginazione

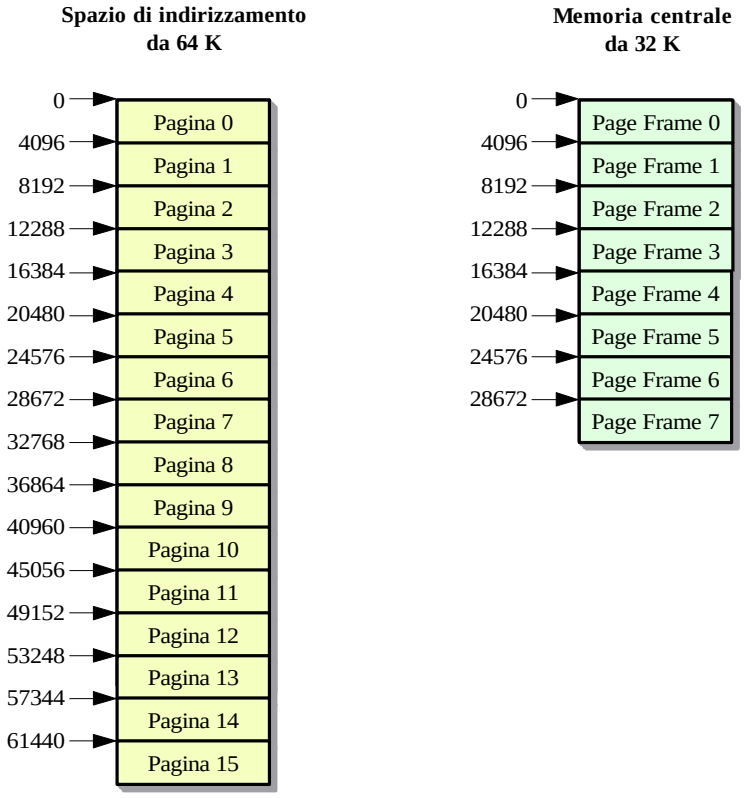
Un requisito essenziale, per una memoria virtuale, è che il programma completo da eseguire sia interamente contenuto nella memoria secondaria.

Anche se la realtà è quella per cui i "pezzi" di programma originale si trovano nella memoria principale, mentre le "copie" si trovano nella memoria secondaria, risulta comunque più semplice pensare che avvenga il viceversa, ossia che il programma nella memoria secondaria sia l'originale, mentre i pezzi portati nella memoria centrale di tanto in tanto siano delle copie. L'importante è tener presente quello che avviene nella realtà e, soprattutto, mantenere sempre aggiornato l'originale (in base all'esecuzione che se ne sta facendo): quando si operano dei

[1] Quest'ultimo concetto è fondamentale, in quanto contrapposto a quello che si usa nella cosiddetta **segmentazione**, in cui invece il programmatore deve essere a conoscenza dell'esistenza dei **segmenti**.

cambiamenti alla copia nella memoria centrale, questi devono essere riprodotti anche nell’originale ([2]).

Dato lo spazio di indirizzamento virtuale, lo si divide in un certo numero di **pagine**, tutte di uguale dimensione (che deve essere una potenza di 2). Ad esempio, fino a poco tempo fa erano comuni pagine da 4096 indirizzi ciascuna. Nella figura seguente, ad esempio, è riportata la suddivisione di uno spazio di indirizzamento virtuale di 64K in 16 pagine (da 0 a 15) da 4K ciascuna e con 4096 indirizzi ciascuna:



*Suddivisione dello spazio di indirizzamento virtuale in **pagine virtuali** e della memoria centrale in **page frame***

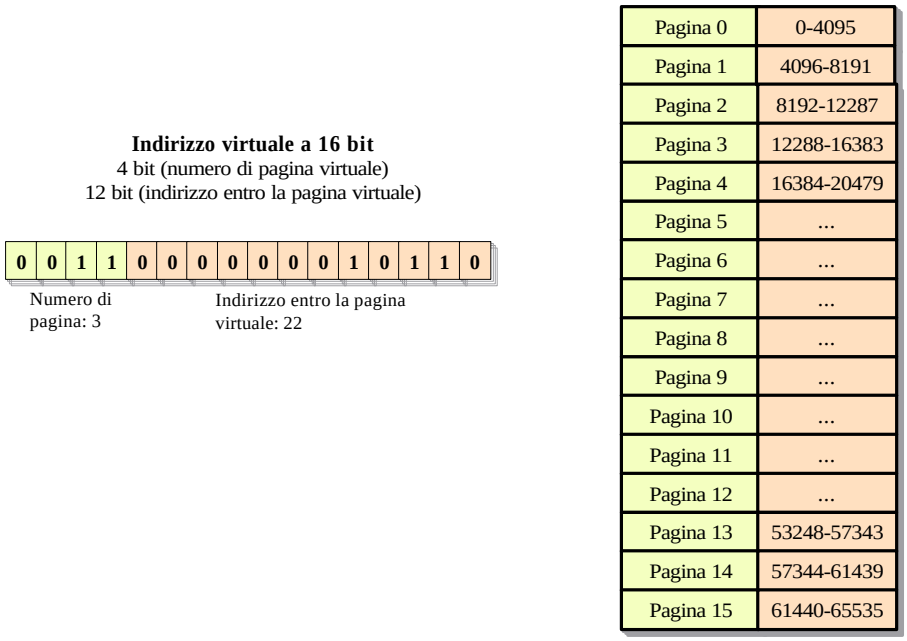
Evidentemente, anche lo spazio di indirizzamento fisico (la memoria centrale) viene diviso in modo simile, con ogni pezzo (detto **page frame**) che ha la stessa dimensione di una pagina, in modo tale che ogni pezzo di memoria centrale sia capace di contenere esattamente una pagina. In questo esempio, si considera in particolare una memoria principale da 32K, divisa perciò in 8 page frame, numerati

[2] Non è detto, comunque, che l’aggiornamento nella memoria secondaria debba essere effettuato subito.

da 0 a 7 ([3]). Nei calcolatori reali, il numero di page frame nella memoria centrale va da poche decine fino ad alcune migliaia nelle macchine più grosse.

La memoria virtuale del nostro esempio sarebbe implementata, al *livello 2* (*livello della macchina standard*), per mezzo di una **tabella delle pagine**, contenente 16 elementi (tanti quante sono le pagine). Il meccanismo della paginazione funziona allora nel modo seguente:

- quando il programma tenta di accedere alla memoria per prelevare dati o memorizzarli oppure per prelevare istruzioni o saltare, genera dapprima un indirizzo a 16 bit corrispondente ad un **indirizzo virtuale** compreso tra 0 e 65535 ([4]);
- tale indirizzo virtuale a 16 bit può essere interpretato in vari modi; in questo nostro esempio, supponiamo che esso venga visto come diviso in due parti: i primi 4 bit corrispondono al **numero di pagine virtuale** (da 0 a 15), mentre i restanti 12 bit individuano una locazione all'interno della pagina selezionata. Ad esempio, nella figura seguente si considera un indirizzo di valore 12310: separando i primi 4 bit dai restanti 12, si individua la pagina 3 e, in essa, l'indirizzo 22:



Possibile struttura di un indirizzo virtuale a 16 bit

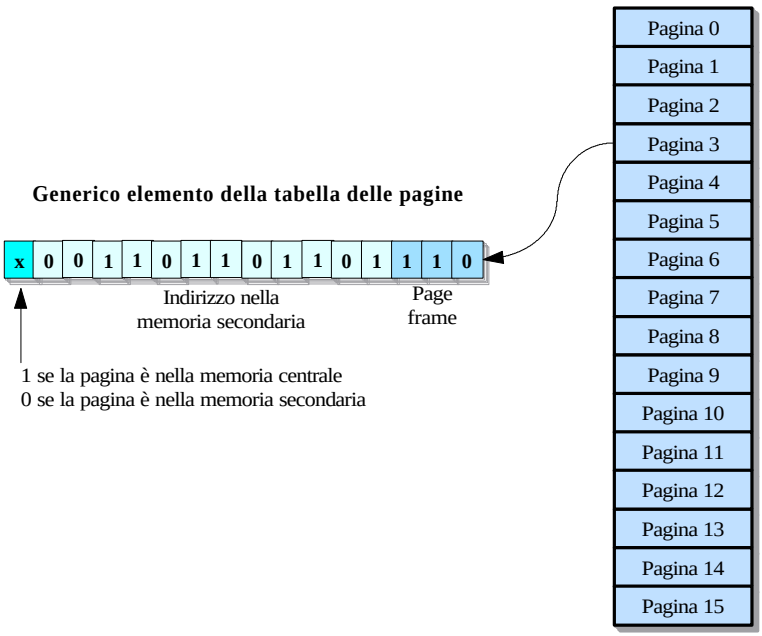
[3] Nel precedente esempio, invece, la memoria centrale era da 4 K, per cui conteneva un unico page frame.
[4] Per generare questo indirizzi si possono usare gli *indici*, l'*indirizzamento indiretto* e tutte le altre tecniche comuni.

Da notare che l'indirizzo virtuale 22, all'interno della pagina 3, corrisponde all'indirizzo fisico 12310 (=12288+22);

- quando il sistema operativo si accorge che il programma richiede la pagina virtuale 3, deve scoprire dove essa è collocata. Ci sono allora, nel nostro esempio, 9 distinte possibilità: le prime 8 sono relative alla posizione della pagina virtuale richiesta in memoria centrale (dato che quest'ultima è costituita da 8 page frame), mentre l'ultima possibilità è che la pagina virtuale non si trovi nella memoria principale, bensì nella memoria secondaria (dato che non necessariamente tutte le pagine virtuali possono stare contemporaneamente nella memoria centrale). Per capire quale possibilità risulta verificata, il sistema operativo legge il contenuto della **tabella delle pagine**: ogni elemento di tale tabella corrisponde ad una precisa pagina virtuale e ne indica l'attuale posizione;

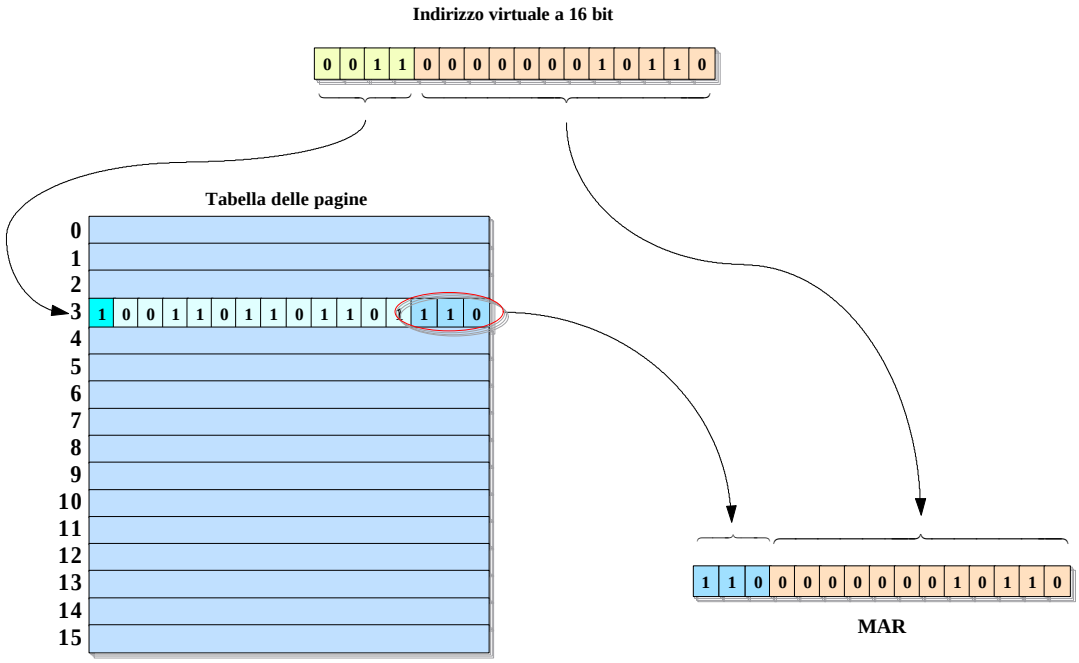
- una possibile struttura della tabella delle pagine, sempre con riferimento al nostro esempio, è riportata nella prossima figura. Si suppone che ogni elemento della tabella abbia 3 campi: il primo campo è composto da un unico bit, il quale indica se la pagina virtuale si trova in memoria principale (valore 1) oppure no (valore 0); il secondo campo (da 12 bit) fornisce l'indirizzo di memoria secondaria in cui si trova la pagina virtuale ([5]), nel caso ovviamente in cui essa non si trovi nella memoria centrale; l'ultimo campo (4 bit), invece, indica il page frame della memoria centrale in cui si trova la pagina, qualora ovviamente essa non si trovi nella memoria secondaria. E' ovvio che l'uso del secondo campo esclude quello del terzo campo e viceversa, a seconda del valore del primo campo.

[5] In particolare, vengono indicati la pista ed il settore del disco.



La tabella delle pagine contiene tanti elementi quante sono le pagine virtuali. Ogni elemento è costituito, in questo esempio, da 3 campi, rispettivamente da 1 bit (flag), 12 bit (indirizzo di memoria secondaria) e 4 bit (pagine frame nella memoria principale)

Supponiamo adesso che la pagina virtuale richiesta (la numero 3) si trovi nella memoria principale, per cui il primo bit dell'elemento corrispondente, nella tabella delle pagine, è posto ad 1. Gli ultimi 3 bit di questo stesso elemento indicano il page frame in cui è contenuta la pagina (si tratta del page frame 6 nell'ultima figura): tali 3 bit vengono allora posti nei 3 bit più a sinistra del **MAR** (*Memory Address Register*), mentre l'indirizzo all'interno della pagina virtuale, ossia i 12 bit più a destra dell'indirizzo virtuale originale, vengono posti nei 12 bit più a destra del MAR. In questo modo, viene formato un indirizzo di memoria centrale a 15 bit (adeguato perciò per una memoria fisica da 32K come quella presa in considerazione in questo esempio), così come mostrato nella figura seguente:



Formazione di un indirizzo di memoria partendo da un indirizzo virtuale

A questo punto, l’hardware può usare l’indirizzo contenuto nel MAR per prelevare la parola desiderata e porla nel registro **MBR** (*Memory Buffer Register*) oppure, viceversa, può prelevare il contenuto del registro MBR e memorizzarlo all’indirizzo contenuto nel MAR.

Possiamo ora osservare una cosa: se il sistema operativo dovesse convertire ogni indirizzo virtuale delle istruzioni di livello 3 in un indirizzo effettivo (secondo il meccanismo descritto), una macchina di livello 3 con memoria virtuale sarebbe molto più lenta rispetto a quella senza memoria virtuale, il che implicherebbe l’abbandono del concetto di memoria virtuale, proprio perché non foriero di benefici. Al contrario, per velocizzare il processo di “traduzione” degli indirizzi virtuali in indirizzi fisici, generalmente la tabella delle pagine viene memorizzata in registri speciali dell’hardware e la stessa “traduzione” viene effettuata direttamente nell’hardware. Una soluzione alternativa potrebbe essere invece quella di mantenere la tabella delle pagine nei registri veloci e di far fare la traduzione al microprogramma, per mezzo quindi di una programmazione esplicita: la maggiore o minore velocità di questa soluzione rispetto alla precedente dipende dall’architettura del livello della microprogrammazione, ma comunque essa ha il vantaggio di non richiedere circuiti speciali e modifiche dell’hardware.

La paginazione su richiesta

Nell'esempio considerato nel paragrafo precedente, abbiamo supposto che la pagina virtuale ricercata dal programma si trovasse nella memoria centrale; è chiaro che non necessariamente questo si verifica, in quanto alcune pagine virtuali potrebbero essere contenute nella memoria secondaria, a causa ad esempio dei limiti di capienza della memoria centrale. Quando la pagina virtuale richiesta non si trova nella memoria centrale, si dice che si è verificato un **page fault** (*mancanza di pagina*); in questa situazione, il sistema operativo deve compiere le seguenti operazioni:

- leggere la pagina richiesta nella memoria secondaria;
- trasferirla nella memoria principale;
- aggiornare la tabella delle pagine con la nuova posizione;
- ripetere l'esecuzione dell'istruzione che ha causato il page fault.

Avendo una macchina dotata di memoria virtuale, è possibile iniziare un programma anche se nessuna parte del programma stesso si trova nella memoria centrale: basterà infatti inizializzare la tabella delle pagine per indicare che tutte le pagine virtuali si trovano nella memoria secondaria (primo bit a 0). Quando la CPU prova a prelevare la prima istruzione del programma, ottiene subito un page fault, il che implica che la pagina contenente tale istruzione venga caricata in memoria principale e ovviamente che il corrispondente elemento della tabella delle pagine venga aggiornato. A questo punto, l'esecuzione dell'istruzione può cominciare. Se, ad esempio, essa richiede due indirizzi in pagine diverse tra loro e diverse dall'unica pagina caricata in memoria principale, si hanno altri due page fault e quindi altri due trasferimenti dalla memoria secondaria a quella principale (con relativi aggiornamenti della tabella delle pagine). L'istruzione successiva potrà eventualmente causare uno o più altri page fault e così via.

Questo modo di operare su una memoria virtuale prende il nome di **paginazione su richiesta**: le pagine vengono infatti caricate nella memoria principale solo se ne viene fatta esplicita richiesta. Chiaramente, dopo aver eseguito un certo numero di istruzioni del programma, ci saranno molte pagine virtuali caricati nella memoria principale e quindi i page fault saranno molto meno frequenti, tanto meno quanto più capiente è la memoria principale.

Politiche per la sostituzione delle pagine

Fino ad ora, abbiamo sempre supposto implicitamente che, dovendo caricare una pagina virtuale dalla memoria secondaria a quella centrale, ci sia un page frame libero disposto ad accoglierla. Generalmente, questo non succede (tutti i frame page sono occupati) ed è quindi necessario liberarne uno (cioè spostare il suo contenuto nella memoria secondaria) per ospitare la pagine virtuale richiesta dalla CPU. Bisogna allora implementare un algoritmo per la scelta del page frame da liberare, ossia quindi della pagina virtuale da "sostituire".

Il metodo più semplice è quello della **scelta casuale**, ma non si tratta decisamente del metodo migliore: infatti, se capita che venga scelta la pagina contenente la prossima istruzione da eseguire, si verificherà un altro page fault non appena la CPU cercherà di acquisire quella istruzione, con conseguente rallentamento dell'esecuzione. Di conseguenza, molti sistemi operativi tentano di individuare la pagina virtuale *meno utile* tra quelle presenti, ossia quella la cui mancanza avrebbe il minore effetto negativo sul programma in corso.

Un modo abbastanza intuitivo di far questo consiste nell'individuare la pagina che con meno probabilità sarà usata a breve. Dal punto di vista pratico, questo lo si può ottenere individuando la pagina usata meno di recente, dato che la probabilità a priori che non debba essere usata è alta. Si parla allora di metodo **Least Recently Used** (**LRB**). Generalmente, questo metodo funziona bene, ma ci sono comunque situazioni "patologiche" in cui invece fallisce miseramente.

Un altro algoritmo è quello cosiddetto **First In First Out** (**FIFO**): esso toglie la pagina caricata meno di recente, a prescindere da quando sia stata referenziata. Si procede allora nel modo seguente:

- ad ogni page frame della memoria centrale viene associato un **contatore**, memorizzato nella tabella delle pagine;
- inizialmente, tutti i contatori sono posti a 0;
- dopo ogni operazione di caricamento per page fault, il contatore di tutte le pagine già presenti in memoria viene incrementato di uno, mentre invece quello della pagina appena introdotta è posto a 0;
- in questo modo, quando è necessario scegliere la pagina da togliere, si prende quella con il contatore più alto, visto che è stata presente per il maggior numero di pagine e quindi è stata caricata prima di qualsiasi altra

pagina nella memoria, per cui si spera abbia la maggiore probabilità a priori di non essere più necessaria.

Facciamo osservare che, se la pagina che deve essere "sfrattata" dalla memoria principale non è stata modificata da quando è stata letta (cosa molto probabile se contiene parte del programma invece di dati), non è necessario riscriverla nella memoria secondaria, poiché ne esiste già una copia fedele e quindi è inutile "perdere tempo". Se invece essa è stata modificata, allora la copia nella memoria secondaria non è fedele e quindi bisogna necessariamente riscriverla.

Il modo più semplice per indicare se una pagina è stata modificata o meno nel tempo in cui è rimasta nella memoria principale, è quello di associarle un bit nella tabella delle pagine: il bit viene inizializzato a 0 quando la pagina viene caricata e viene posto ad 1 in corrispondenza della prima modifica. Esaminando questo bit, il sistema operativo può dunque capire se ci sono state modifiche (si parla di **pagina sporca**) o meno (si parla di **pagina pulita**) ed agire di conseguenza.

E' chiaramente auspicabile mantenere un alto rapporto tra pagine pulite e pagine sporche, in modo da minimizzare il numero di riscritture nella memoria secondaria. Quasi tutti i calcolatori sono comunque in grado di copiare le pagine dalla memoria centrale alla secondaria mentre la CPU sta lavorando, usando ad esempio il **DMA** (*Accesso Diretto alla Memoria*) o le apposite **unità di canale**. Addirittura, alcuni sistemi operativi approfittano di questo parallelismo tutte le volte che il disco è libero: viene individuata una pagina sporca, preferibilmente con alta probabilità di essere scritta a breve (perché vecchia), e ne viene imposta la copia sul disco, nonostante non ci sia stata ancora la necessità di togliere la suddetta pagina dalla memoria centrale. Naturalmente, può capitare che quella stessa pagina venga "risporcata" immediatamente dopo il procedimento di copiatura o addirittura durante: in questa situazione, se è vero che la "copia anticipata" è stata inutile, è anche vero che il costo di tale operazione è stato piccolo, dato che il disco era comunque inutilizzato e la CPU era comunque in grado di lavorare dopo aver imposto la copia. Le scritture su disco fatte con l'intenzione di "pulire" le pagine sporche sono chiamate **scritture false**.

La dimensione delle pagine e la frammentazione

Quando il programma utente ed i suoi dati riempiono totalmente un certo numero di pagine, non c'è spazio sprecato quando essi si trovano nella memoria centrale. Viceversa, se essi non riempiono esattamente tutte le pagine, c'è senz'altro dello spazio inutilizzato nell'ultima pagina. Questo "problema" va sotto il nome di **frammentazione**.

Supponiamo che la dimensione di pagina sia di N parole: si può subito dire che la quantità media di spazio sprecato nell'ultima pagina è di N/2 parole; per ridurre tale quantità, bisogna evidentemente ridurre N, ossia fare le pagine più piccole. Ma fare pagine piccole significa fare più pagine e quindi anche ingrandire la tabella delle pagine: se tale tabella è gestita dall'hardware, il suo ingrandimento comporta un numero maggiore di registri per la sua memorizzazione e dei relativi circuiti, ossia quindi un costo maggiore del calcolatore. Non solo, ma sarà anche necessario più tempo per caricare o salvare tali registri tutte le volte che il programma viene fermato e poi fatto ripartire.

Ancora, l'uso di pagine piccole rende inefficiente l'uso di memorie secondarie con lunghi tempi di accesso, come tipicamente i dischi: infatti, dato che bisogna aspettare 10 ms o più per accedere alla prima parola da trasferire e iniziare il trasferimento, è preferibile poter trasferire grossi blocchi di informazioni, dato che il *tempo di trasferimento* risulta generalmente minore della somma del *tempo di ricerca* delle parole e del *tempo di rotazione* del disco. Al contrario, se la memoria secondaria non presenta ritardo rotazionale (ad esempio una *memoria a nuclei*), il tempo di trasferimento è proporzionale solo alla dimensione del blocco e quindi sarebbe opportuno avere pagine piccole.

Tutte queste considerazioni mostrano che è sempre necessario trovare un compromesso, sulla base delle proprie esigenze, per scegliere una dimensione delle pagine ottimale. Una nota legge dell'informazione (**legge di Amdahl**) suggerisce che *è sempre opportuno ottimizzare il caso più frequente*: la scelta delle dimensioni delle pagine andrà perciò fatta in modo da ottimizzare le situazioni che si presentano con maggior frequenza, anche eventualmente a scapito di quelle meno frequenti.

Autore: **Sandro Petrizzelli** (sandry@iol.it)
sito personale: http://users.iol.it/sandry