Paginazione su domanda

G. Lettieri

14 Maggio 2018

Usando la tecnica della paginazione abbiamo ottenuto diversi benefici per il sistema e per gli utenti, ma si può ancora migliorare qualcosa:

- gli utenti devono specificare la dimensione dei loro processi;
- la dimensione della memoria limita il numero di processi che si possono caricare e la dimensione massima di ogni processo.

È possibile superare entrambe queste limitazioni utilizzando la tecnica della *paginazione su domanda*. L'idea è di usare la memoria fisica come una cache di *pagine* invece che di processi: invece di caricare dallo swap interi processi, carichiamo solo le pagine che i processi "richiedono" accendendovi per prelevare istruzioni o leggere/scrivere operandi.

La tecnica si applica anche al caso di un sistema che esegue un solo programma, quindi concentriamoci prima su questo, per semplicità. Quello che vogliamo fare è di simulare il funzionamento di un sistema di elaborazione in cui tutta (o quasi) la memoria principale indirizzabile dal processore è disponibile al programmatore, indipendentemente dalla dimensione della memoria principale realmente installata. In questo modo il programmatore può scrivere i suoi programmi senza preoccuparsi del fatto che la memoria a disposizione potrebbe non contenerli: tramite la memoria virtuale un programma troppo grande può girare lo stesso, anche se ad una velocità ridotta. Il meccanismo è inoltre fatto in modo tale che i programmi possano automaticamente beneficiare della disponibilità di nuova memoria (per esempio, installata successivamente) senza dover essere modificati.

Nel caso dei processori Intel/AMD a 64 bit si vuole simulare il funzionamento di una macchina come quella di Fig. 1, dove si mostra solo il collegamento tra la CPU e la memoria tramite il bus degli indirizzi. Si ricordi che, per queste macchine, solo i 48 bit meno significativi di un indirizzo di 64 bit possono assumere un valore qualsiasi, mentre i 16 bit più significativi devono essere tutti uguali al bit n. 47 (contando da 0). Anche con questa limitazione gli indirizzi possibili sono $2^{48} = 256\,\text{TiB}$, ben al di là della capacità della memoria RAM comunemente disponibile su un singolo sistema.

L'idea è di simulare la macchina di Fig. 1 sostituendo la parte dentro le linee tratteggiate con un sistema che sia economicamente e tecnologicamente realizzabile ma che, visto dall'esterno, sia funzionalmente equivalente alla memoria

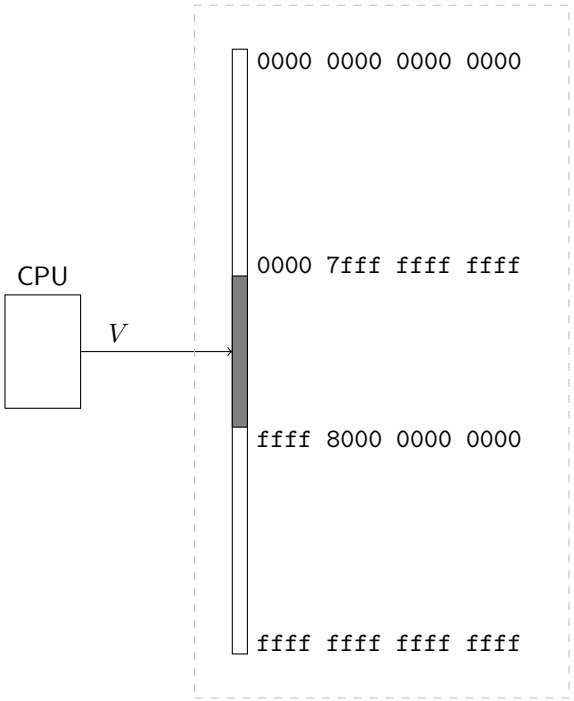

Figura 1: Macchina virtuale.

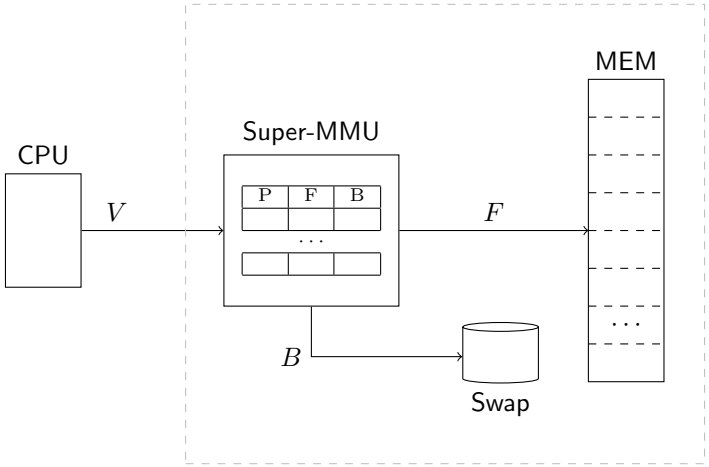

Figura 2: Implementazione della macchina di Fig. 1 con una Super-MMU.

di Fig. 1. Il sistema è molto complicato e lo studieremo per passi, introducendo delle semplificazioni che poi elimineremo via via. Come abbiamo già fatto per la paginazione semplice, esaminiamo prima il caso di una "Super-MMU" che fa tutto in hardware, poi di una MMU che ha bisogno del software di sistema, ma che usa una tabella di traduzione semplice, e infine della MMU completa, che usa l'albero di tabelle e il TLB.

1 Super-MMU

In Fig. 2 introduciamo la prima versione del sistema. La parte tratteggiata di Fig. 1 è stata sostituita con un sistema composto da una normale memoria principale (MEM), l'area di Swap e la Super-MMU. Lo scopo della Super-MMU è di fare in modo che la CPU non possa distinguere tra la situazione in Fig. 2 e quella in Fig. 1. Per ottenere il suo scopo la Super-MMU intercetta tutte le operazioni di lettura e scrittura eseguite dalla CPU, trasferisce dallo Swap in MEM la pagina che contiene l'informazione richiesta, e infine completa l'operazione della CPU. La Super-MMU fa in modo che tutto ciò appaia alla CPU come una normale operazione sulla memoria. Per una operazione di lettura, per esempio, la CPU produce l'indirizzo e si vede recapitare il contenuto della locazione indirizzata. L'unica differenza è che l'operazione avrà richiesto un tempo molto più lungo del normale, ma questo non è importante per la corretta esecuzione di molti programmi.

Abbiamo detto che usiamo MEM come una cache per lo Swap, ovvero: manteniamo una copia delle pagine caricate, nel caso venissero richieste di nuovo, e le ricopiamo nello Swap solo quando devono essere rimpiazzate per far posto a nuove pagine richieste dalla CPU (politica *write-back*). Per ogni possibile indirizzo che la CPU può emettere la Super-MMU deve essere in grado di sapere se

la pagina che lo contiene è già caricata in MEM, e dove, oppure se si trova nello Swap, e dove. Per far questo utilizziamo il bit P e aggiungiamo un campo "B" ad ogni entrata della tabella di corrispondenza usata dalla Super-MMU.

Fino ad ora avevamo usato il bit P solo per marcare pagine virtuali a cui il processo non deve accedere, o perché non fanno parte degli indirizzi che il programmatore aveva detto di voler usare o perché volevamo vietarle per altri motivi (per esempio, per intercettare la dereferenziazione di puntatori nulli). In reazione agli accessi alla pagine con P=0 ci siamo limitati a terminare forzatamente il processo corrente. Ora usiamo il bit P anche per un altro scopo: segnalare quali pagine sono state caricate (P=1) e quali si trovano ancora nello swap (P=0). Per le pagine caricate ricordiamo il numero di frame in cui le abbiamo caricate (campo F), mentre per quelle non caricate ricordiamo in quale blocco dello Swap si trovano (campo B).

È disponibile un semplice esempio che illustra le operazioni svolte dalla Super-MMU mentre il sistema esegue un semplice programma. Uno degli scopi dell'esempio è di far vedere come la memoria virtuale sia completamente *trasparente* al programmatore, che si limita a scrivere un programma per una macchina virtuale simile a quella di Fig. 1 in cui non compare alcuna MMU (Super o meno), nessuna area di Swap, nessuna suddivisione della memoria in pagine, nessuna traduzione di indirizzi. Tutte queste cose servono all'implementazione della memoria virtuale e non sono accessibili al normale programmatore.

1.1 Strutture dati aggiuntive

Nell'esempio si vede che la Super-MMU ha bisogno di almeno una struttura dati (chiamata Free nell'esempio) per sapere quali pagine fisiche sono libere o occupate. Altre strutture dati non sono necessarie, ma sono utili per migliorare le prestazioni:

- un campo **D** (Dirty), di 1 bit, da aggiungere ai campi P, F e B della tabella di corrispondenza;
- un contatore delle statistiche di accesso alle pagine, che può essere un ulteriore campo della tabella.

La Super-MMU può usare il bit D per ricordare se vi sono mai state scritture sulla pagina corrispondente (basta resettarlo quando la pagina è caricata e settarlo alla prima scrittura). In questo modo può evitare di riscrivere nello Swap le pagine che non sono state modificate mentre erano caricate in MEM, in quanto la copia che si trova nello Swap va ancora bene.

Il contatore delle statistiche, invece, dovrebbe essere usato per fare scelte intelligenti al momento in cui MEM è piena e la Super-MMU deve selezionare una pagina (detta pagina *vittima*) da sovrascrivere con quella che deve essere invece caricata. L'idea è che la Super-MMU dovrebbe accumulare delle statistiche ad ogni accesso alla pagina. Quali statistiche raccogliere dipende da quale strategia la Super-MMU deve adottare per selezionare la vittima. Supponiamo che voglia selezionare la pagina che ha ricevuto meno accessi tra tutte quelle

presenti (strategia LFU, per Least Frequently Used): sarà allora sufficiente contare gli accessi (se il contatore arriva al massimo si smette di sommare). Al momento della selezione della vittima, la Super-MMU sceglierà la pagina con il valore minimo del contatore.

2 MMU_1: software di sistema

La Super-MMU svolge tutte le operazioni in hardware, mentre sappiamo che la vera MMU si limita alla sola traduzione degli indirizzi. Consideriamo ora una MMU_1 del tutto simile a quella vera, con la sola semplificazione di usare una tabella di corrispondenza invece di un albero. La MMU_1 si comporta in questo modo:

- ogni volta che la CPU inizia un nuovo accesso alla memoria, all'indirizzo V, la MMU_1 lo intercetta, lo scompone in numero di pagina virtuale e offset, usa il numero di pagina virtuale per accedere alla tabella di corrispondenza;
  - se trova P=1 traduce l'indirizzo in fisico e completa l'accesso per conto della CPU;
  - se trova P=0 dice alla CPU di sollevare una *eccezione di page fault*.

Si ricordi che, in risposta ad un fault, la CPU salva in pila l'indirizzo dell'istruzione *che stava eseguendo* (che è quella che si trova all'indirizzo V o ha tentato di accedervi) e salta all'inizio di una particolare routine, detta *routine di page fault*. Questa routine svolgerà tutte le operazioni che volevamo far svolgere alla Super-MMU. Più precisamente, la routine:

1. individua la posizione della pagina V nello Swap (supponiamo legga il valore B dal campo B);
2. sceglie una frame libero, sia F. Se non ve ne sono:
   (a) seleziona una pagina virtuale vittima, sia V', tra quelle già caricate;
   (b) ricopia V' nell'area di swap;
   (c) aggiorna la tabella di corrispondenza in modo che l'entrata relativa a V' contenga P=0;
   (d) usa come F il frame che conteneva V';
3. carica V da B in F;
4. aggiorna la tabella di corrispondenza in modo che l'entrata relativa a V contenga P=1 e il valore F nel campo F;
5. termina ritornando all'indirizzo salvato in pila alla ricezione dell'eccezione.

A questo punto la CPU riesegue l'istruzione che aveva causato il fault, riemettendo dunque l'indirizzo V, che sarà intercettato nuovamente dalla MMU_1. Questa seconda volta, però, la MMU_1 troverà P=1 nella tabella di corrispondenza e potrà completare l'accesso e far proseguire l'esecuzione del programma.

Al passo 1 la routine di page fault deve conoscere V, l'indirizzo che ha causato il fault. Introduciamo un registro speciale del processore, `cr2`, e modifichiamo la CPU in modo che, su ricezione dell'eccezione di page fault, salvi in `cr2` l'indirizzo V a cui stava tentando di accedere, prima di saltare alla routine di page fault. Aggiungiamo una nuova istruzione, `movq %cr2, %rax`, che la routine di page fault può eseguire per copiare l'indirizzo V in `rax` in modo che poi lo possa elaborare liberamente con le istruzioni già esistenti.

2.1 Descrittori di pagina virtuale e di frame

La tabella di corrispondenza è una struttura dati condivisa tra il software (le routine di inizializzazione e di page fault) e l'hardware (la MMU_1). In questi casi il formato della struttura dati è normalmente dettato dall'hardware, visto che il software si può più facilmente adattare a qualunque formato.

Nel caso di MMU_1, per avvicinarci alla MMU definitiva, prevediamo i seguenti campi per ogni entrata della tabella di corrispondenza:

- I campi **P** e **F** che già conosciamo;
- Il campo **D** introdotto in Sez. 1.1;
- Un campo **A**, di un 1 bit, che la MMU_1 pone a 1 se vi è stato un accesso (in lettura o scrittura) alla pagina corrispondente.

Si noti che manca il campo B, in quanto la MMU_1 non lo usa mai (i trasferimenti da e verso lo Swap sono operati esclusivamente dalla routine di page fault). Il campo A è nuovo, ma possiamo considerarlo una versione estremamente ridotta (un solo bit) del campo contatore introdotto in Sez. 1.1. I campi D ed A sono solo inizializzati dalla routine di page fault e poi scritti dalla MMU_1, e sono dunque informazioni che la MMU_1 raccoglie in modo che la routine di page fault possa utilizzarle. In particolare, al passo 2.(b), la routine di page fault evita di ricopiare V' nello Swap se vede che D vale 0. Gli altri campi sono solo letti dalla MMU_1: sono informazioni preparate dalla routine di page fault e usate dalla MMU_1 (e dalla routine stessa).

Il resto della funzionalità della Super-MMU deve essere riprodotto in software dalla routine di page fault. In particolare la routine dovrà predisporre delle strutture dati per:

1. sapere dove le pagine si trovano nell'area di swap;
2. sapere quali frame sono liberi o occupati;
3. mantenere delle statistiche più utili rispetto ai singoli bit A (come i contatori di Sez. 1.1).

Una struttura dati che permette di risolvere tutti questi problemi è un array di *descrittori di frame*, con un descrittore per ognuno dei frame disponibili. Si noti che questa struttura è utilizzata esclusivamente dalla routine di page fault: per essa non valgono le considerazioni sul formato imposto dall'hardware. Chi scrive la routine di page fault è libero di definire questa struttura dati come vuole, o anche di non definirla affatto e risolvere i problemi di cui sopra in qualche altro modo.

Per la routine di page fault associata a MMU_1 prevediamo un descrittore di frame con i seguenti campi:

- un flag **occupato**, che vale **true** se il frame contiene al momento una pagina virtuale; se il campo vale **false** il frame è libero e i campi successivi non sono significativi, altrimenti tutti i campi successivi si riferiscono alla pagina virtuale che sta occupando il frame;
- un campo **V** che contiene il suo numero di pagina virtuale;
- un campo **B** che contiene il numero del primo blocco della sua copia nello Swap;
- un campo **contatore** che contiene le statistiche dei suoi accessi.

Si noti che i descrittori di frame possono solo darci informazioni sulle pagine virtuali presenti (quelle, appunto, che in ogni momento si trovano caricate in qualche frame). È necessario però conoscere anche il valore di B per le pagine assenti, altrimenti non sarebbe possibile caricarle (passi 1 e 3 della routine di fault). Per queste si può utilmente sfruttare il fatto che la MMU_1 non usa il resto del descrittore di pagina virtuale quando trova P=0, quindi il campo B delle pagine assenti può essere memorizzato in quello spazio. Al passo 1, dunque, la routine di page fault accede all'entrata della tabella di corrispondenza relativa all'indirizzo virtuale V, vi trova ovviamente P=0 (altrimenti non ci sarebbe stato un page fault) e può leggere il valore B. Quando la pagina viene poi caricata e resa presente (passo 4), la routine di page fault copierà il valore B nel campo B del descrittore di F prima di scrivere nel campo F. Quando rende una pagina assente (passo 2.(c)) deve anche ricopiare il valore del campo B, preso dal descrittore di F, nel descrittore di pagina virtuale della vittima, in modo che la vittima possa essere ricaricata se il programma la richiede in seguito. Ricordare quale era il blocco da cui era stata caricata ogni pagina virtuale è utile nel caso in cui, quando la pagina virtuale viene scelta come vittima, si scopre che non è stata modificata (D=0 nel descrittore di pagina virtuale): allora è possibile riutilizzare la pagina che si trova già nello Swap, ma ovviamente bisogna sapere dov'è.

Il campo contatore viene aggiornato dalla routine di page fault in base ad un *campionamento* dei bit A delle pagine virtuali presenti. La routine scorre tutta la tabella di corrispondenza esaminando tutti i bit A che trova ad 1, aggiorna quindi i relativi contatori (ogni contatore si trova nel descrittore del frame in cui ciascuna pagina virtuale è contenuta, facilmente individuabile dal campo F del descrittore di pagina virtuale), e poi azzera i bit A. Azzerare i

bit A è fondamentale, altrimenti non si potrebbe mai vedere se vi sono stati nuovi accessi dall'ultima scansione. Questa scansione può essere effettuata in vari momenti, per esempio periodicamente. Un modo semplice è di effettuarla subito prima di scegliere una vittima (dunque al passo 2.(a)).

Il campo V, infine, serve alla routine di page fault per ritrovare il descrittore di pagina virtuale di V' al passo 2.(c). La selezione della vittima, infatti, passa da una scansione dei contatori, che sono però associati ai frame. Una volta trovato il contatore minimo si è individuato il *frame* F che si vuole utilizzare, ma per rimuovere la pagina virtuale in esso contenuta è necessario sapere il suo numero di pagina virtuale (V'), a meno di non voler scorrere tutta la tabella di corrispondenza alla ricerca del descrittore di pagina virtuale che contiene il valore F nel campo F.