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 restano ancora degli inconvenienti:

- 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.

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.[1]

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 $2^{48} = 256\,\mathrm{TiB}$, 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

[1] Resta inteso che, anche se gli indirizzi sono disponibili, un programma non deve necessariamente usarli tutti. Le parti non utilizzate possono essere marcate come non accessibili per intercettare eventuali errori.

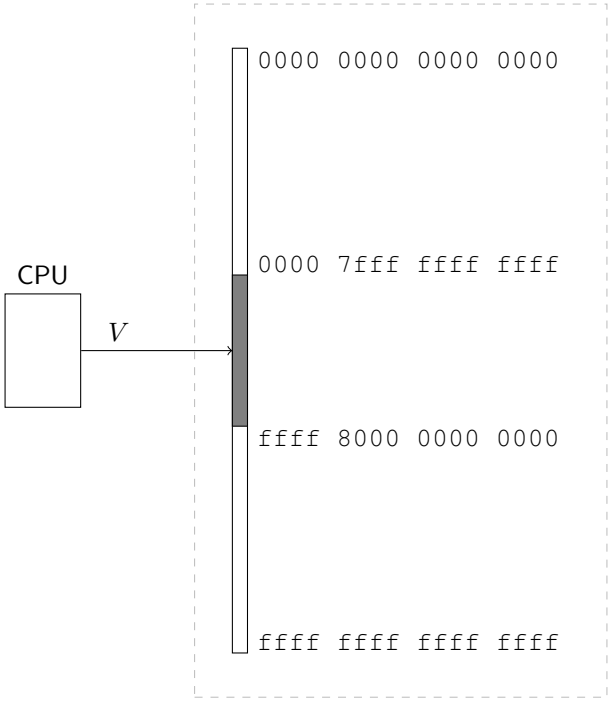

Figura 1: Macchina virtuale. Il programmatore (utente) può assumere di avere una macchina in cui tutti gli indirizzi sono potenzialmente disponibili.

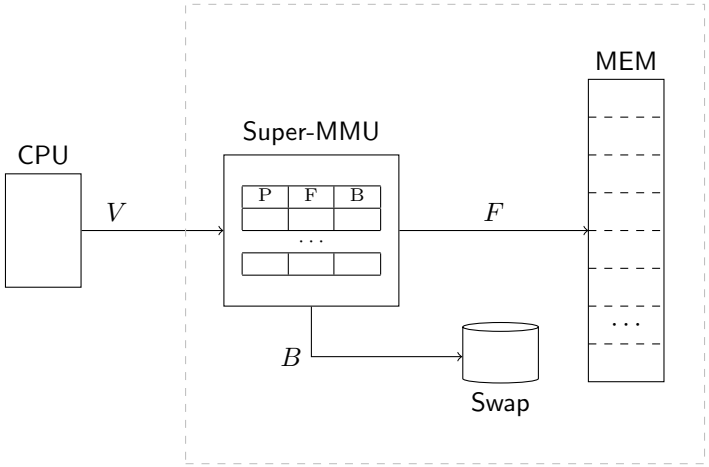

Figura 2: Implementazione della macchina di Fig. 1 con una Super-MMU.

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 (la supertabella di livello 1), 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). Possiamo continuare ad usare P=0 anche per i vecchi scopi (marcare pagine a cui il processo non deve accedere) usando, per esempio, un valore speciale per il campo B. Noi useremo B=0 per dire che la pagina non è accessibile.

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.

Ricordiamo inoltre che nel sistema sono presenti tanti processi e che, ad ogni istante, i frame di MEM possono essere occupati da pagine appartenente a processi diversi. Quando la Super-MMU deve scegliere una vittima può adottare due diverse politche di rimpiazzamento:

- *rimpiazzamento locale*, se la vittima è scelta solo tra le pagine del processo in esecuzione;
- *rimpiazzamento globale*, se la vittima è scelta tra tutte le pagine di tutti i processi.

Noi adotteremo la politica di rimpiazzamento globale.

2 MMU′: software di sistema

La Super-MMU svolge tutte le operazioni in hardware, ma alcune di queste sono semplici e altre molto complesse. Le operazioni semplici avvengono quando P=1 e si limitano alla traduzione dell'indirizzo, mentre le azioni complicate sono scatenate da P=0 e comportano anche complesse euristiche, come la scelta della pagina vittima. Le azioni complesse sono molto più facilmente realizzate in software. Sappiamo infatti che la vera MMU sa solo traddure l'indirizzo e, nel caso in cui trovi P=0, si limita a sollevare una eccezione, detta *eccezione di page fault*. Sarà la routine di sistema che parte per effetto di questa eccezione a gestire il caso P=0.

Consideriamo dunque una MMU′ del tutto simile a quella vera, con la sola semplificazione di usare una tabella di corrispondenza invece di un albero. Per quanto detto, la MMU′ si comporta in questo modo:

- ogni volta che la CPU inizia un nuovo accesso alla memoria, all'indirizzo V, la MMU′ 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. Nel caso di eccezione di page fault, la routine corrispondente è detta *routine di page fault*. Questa routine svolgerà tutte le operazioni che volevamo far svolgere alla Super-MMU.

Supponiamo che il processo in esecuzione, sia P, generi un page fault all'indirizzo virtuale V. La routine di page fault:

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' del processo P', tra quelle già caricate;
   (b) ricopia V' nell'area di swap;
   (c) aggiorna la tabella di corrispondenza di P' 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 di P 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$'$. Questa seconda volta, però, la MMU$'$ troverà P=1 nella tabella di corrispondenza e potrà completare l'accesso e far proseguire l'esecuzione del processo P.

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$'$). 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.

Ricordiamo che i descrittori di pagina virtuale, oltre ai bit P ed F (numero di frame), contengono anche i seguenti bit che ci aiutano a simulare il comportamento della Super-MMU:

- Il bit **D**, posto ad 1 ad ogni operazione di scrittura sull pagina corrispondente;;
- Il bit **A**, posto ad 1 se vi è stato un qualunque accesso (in lettura o scrittura) alla pagina corrispondente.

Si noti che manca invece il campo B della tabella della Super-MMU. I trasferimenti da e verso lo Swap sono operati esclusivamente dalla routine di page fault e non interessano l'hardware della MMU. Per quanto riguarda il bit A, 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$'$, e sono dunque informazioni che la MMU$'$ 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$'$: sono informazioni preparate dalla routine di page fault e usate dalla MMU$'$ (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$'$ 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 `ind_virtuale` che contiene il suo numero di pagina virtuale;
- un campo `processo` che contiene l'identificatore del processo a cui appartiene la pagina;
- un campo `ind_massa` 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$'$ 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 `ind_massa` 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 `ind_massa`, 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 numero di frame contenuto 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, ma è una operazione piuttosto costosa. Noi faremo l'ipotesi di effetturla soltanto quando si verifica un page fault.

I campi `processo` e `ind_virtuale`, infine, servono 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') e il processo a cui appartiene, a meno di non voler scorrere tutta la tabelle di corrispondenza di tutti i processi alla ricerca del descrittore di pagina virtuale che contiene il valore F.

3 Tabelle multilivello

Ci proponiamo ora di eliminare tutte le semplificazioni che abbiamo introdotto a livello hardware nella MMU, studiando come gestire l'albero di traduzione su più livelli e la presenza del TLB.

4 Tabella su più livelli

Dal momento che tutte le entrate delle tabelle, di ogni livello, possiedono il bit P, possiamo caricare dallo swap, su domanda, anche le tabelle. In questo modo le tabelle fungono anche da struttura dati che ci permette di trovare tutte le pagine dentro lo swap: è sufficiente conoscere il numero del blocco della tabella di livello 4. Questa conterrà inizialmente i numeri di blocco delle tabelle di livello 3 e così via fino alle pagine. All'avvio del processo possiamo caricare la sola tabella di livello 4 e cedere il controllo al processo.

Quando la routine di page fault va in esecuzione può leggere da **`cr2`** l'indirizzo che ha causato il fault, sia V, ma non sa in che punto la traduzione è stata interrotta. Per scoprirlo ripercorre il tragitto della MMU, ripetendo in software le sue azioni, fino a trovare il descrittore che contiene il bit P a zero. Il compito della routine è ora quello di caricare in memoria la pagina che contiene V e *tutte le eventuali tabelle mancanti* nel percorso di traduzione di V dal livello 4 al livello 1. Al termine della routine il processore rieseguirà l'istruzione che aveva causato il fault e questa volta la MMU riuscirà a completare la traduzione.

In pratica la routine di page fault della vera MMU deve fare le stesse azioni di quella di MMU$'$, ma deve ripeterle più volte, una volta per ogni entità mancante nel percorso di traduzione. Vedremo il codice completo quando studieremo il nucleo del sistema.

Ricordiamoci che abbiamo deciso di usare la memoria M_2 per contenere sia le pagine che le tabelle. Vediamo ora la motivazione di questa scelta: sia le pagine che le tabelle competono per gli stessi frame. Il caricamento di una pagina o di una tabella può causare il rimpiazzamento di una o più pagine o tabelle.

La presenza delle tabelle in M2 ci impone di modificare i descrittori di frame, in quanto i frame possono ora contenere sia tabelle (di vari livelli), sia pagine virtuali. Il descrittore di frame conterrà i seguenti campi:

- `livello`, con valori che vanno da -1 a 4: -1 indica che il frame è libero, 0 che contiene una pagina, 1–4 che contiene una tabella del corrispondente livello; i campi rimanenti sono significativi solo se il frame non è libero;
- `residente`, che può valere **`true`** o **`false`**: se true, indica che l'entità contenuta non può essere rimpiazzata;
- `ind_virtuale`: permette di risalire al descrittore che contiene il bit P per l'entità contenuta;
- `ind_massa`: numero del blocco nell'area di swap da cui è stata caricata l'entità contenuta;

- `contatore`: statistiche degli accessi all'entità contenuta.

Per aggiornare i contatori la routine di page fault deve consultare (e poi azzerare) i bit A di tutte le pagine e tabelle caricate. Tali bit sono contenuti a loro volta nelle tabelle e queste sono sparpagliate in M_2, ma è possibile ritrovarle tutte scorrendo i descrittori di frame e considerando solo quelli in cui il campo livello è maggiore di 0. Anche in questo caso vedremo il codice completo quando studieremo il nucleo.

Al momento di scegliere una vittima per il rimpiazzamento, dobbiamo stare attenti a non scegliere mai una tabella che abbia ancora qualche entrata con P=1, altrimenti renderemmo irraggiungibili tutte le entità a cui sta puntando (e quelle a cui queste puntano, etc.). Notiamo che il contatore delle statistiche di una tabella sarà sempre maggiore o uguale dei contatori di tutte le entità a cui essa punta (ogni volta che la MMU mette a 1 un bit A in un descrittore, lo mette a 1 anche nella catena di descrittori incontrati fino a quel punto). Se scegliamo come vittima sempre l'entità che ha il contatore minimo abbiamo dunque un solo caso critico: una tabella che ha il valore del contatore uguale ad almeno una delle entità a cui essa punta (questo può accadere se la tabella punta ad una sola entità). Per risolvere anche questo caso è sufficiente che, tra due contatori che hanno lo stesso valore, si scelga sempre quello il cui corrispondente campo livello è minore.

Una volta scelta una vittima, la routine di page fault deve renderla non presente. Come per la MMU$'$, abbiamo il problema di dover risalire dal numero di frame al descrittore dell'entità in essa contenuta (in modo da porre P=0). Per le pagine virtuali (livello 0) il campo `ind_virtuale` continua ad avere lo stesso significato. Per le tabelle (livelli 1 e 4) abbiamo bisogno del numero di pagina virtuale di una qualunque delle pagine virtuali la cui traduzione passa dalla tabella in questione: è sufficiente che la routine di page fault, nel momento in cui carica una tabella, memorizzi nel campo `ind_virtuale` del corrispondente descrittore di frame l'indirizzo che ha causato il page fault corrente.

Il campo `ind_massa` si utilizza in modo simile a quanto visto per MMU$'$. Anche qui conviene usare i descrittori che hanno P=0 per memorizzare il blocco di swap che contiene l'entità assente. Ora la cosa è particolarmente conveniente, in quanto la routine di page fault deve già accedere ai vari descrittori per scoprire quali di essi ha il bit P a 0: nel momento in cui lo trova ha anche subito a disposizione il blocco da cui caricare l'entità non presente. Una volta caricata l'entità il blocco deve essere copiato nel corrispondente descrittore di frame. Si noti cosa accade alle tabelle: quando sono caricate sono vuote (tutti i bit P sono 0) e tutti i descrittori contengono i blocchi delle entità puntate nell'area di swap. Mentre si trovano in memoria fisica saranno modificate varie volte, per rendere presenti o assenti le entità di livello inferiore, ma quando sono selezionate come vittime sono sicuramente ritornate vuote, e ora tutti i descrittori puntano nuovamente agli stessi blocchi a cui puntavano quando la tabella era stata caricata. Quindi, sotto le ipotesi di funzionamento fin qui esaminate, non è mai necessario ricopiare nell'area di swap una tabella vittima.

Il campo `residente` serve a gestire il caso particolare delle tabelle di livello 4, che non devono essere mai rimpiazzate, ma può essere usato anche in tutti gli altri casi in cui ci dovesse servire di marcare qualche pagina o tabella come non rimpiazzabile.

5 Il TLB

Notiamo ora cosa comporta il fatto che, se il descrittore che sta cercando si trova già nel TLB, la MMU non percorre l'albero delle tabelle:

1. che succede quando la routine di page fault rimpiazza una pagina vittima?
2. la MMU non aggiornerà i bit A nei descrittori dopo la prima volta che ha usato un descrittore, almeno fino a quando quel descrittore resta nel TLB;
3. supponiamo che un descrittore venga caricato a causa di una operazione di lettura e abbia in quel momento il bit D a zero; poi il processore esegue una operazione di scrittura all'interno della stessa pagina e la MMU trova la traduzione nel TLB. Che accade al bit D nel descrittore in memoria?

Nel caso 1 la routine di page fault deve invalidare (usando l'istruzione `invlpg`) la traduzione della pagina vittima dopo aver posto il bit P a zero nel suo descrittore di livello 1. Se non lo facesse, e il TLB avesse una copia del descrittore caricata precedentemente, la MMU continuerebbe a tradurre l'indirizzo virtuale della pagina rimossa ottenendo l'indirizzo del frame che, dopo il rimpiazzamento, contiene ora una pagina virtuale completamente diversa. Si noti che questa invalidazione può essere risparmiata se il TLB deve essere invalidato per altri motivi prima che il processo possa accedere alla pagina. Per esempio, se la pagina scelta come vittima appartiene ad un processo diverso da quello in esecuzione, tale processo potrà accedervi solo dopo un cambio di processo, che comporta l'invalidazione di tutto il TLB.

Consideriamo il caso 2. Il problema, in questo caso, è che il mancato aggiornamento dei bit A per tutti i descrittori che si trovano nel TLB falsa le statistiche sugli accessi calcolate dalla routine di page fault. Quel che è peggio, proprio le pagine il cui descrittore si trova nel TLB, e che sono quindi probabilmente quelle usate più di recente, vedrebbero i loro contatori non aggiornati. Per rimediare a questo problema si richiede un intervento della routine che aggiorna le statistiche: dopo aver letto e azzerato tutti i bit A deve invalidare l'intero TLB. In questo modo gli accessi successivi costringeranno la MMU a percorrere la catena e rimettere a 1 i bit A delle pagine a cui il programma sta accedendo. Per invalidare tutto il TLB basta eseguire la coppia di istruzioni **`movq %cr3, %rax; movq %rax, %cr3`**. La seconda, infatti, invalida il TLB anche se il contenuto di **`cr3`** non cambia.

Consideriamo infine il caso 3. Il problema, in questo caso, è che la routine di page fault deve sapere se ci sono state scritture su una pagina perché, se la pagina è scelta come vittima, deve sapere se è necessario ricopiarla nell'area di swap. La routine di page fault ricava questa informazione dal bit D nei

descrittori che si trovano in memoria (ricordiamo che non ha modo di sapere cosa ci sia dentro il TLB), e dunque il valore di tale bit deve essere corretto. Questo problema è risolto dalla MMU, che agisce nel seguente modo:

1. quando il descrittore cercato non è nel TLB si comporta come già detto (percorre tutta la catena e poi carica il descrittore trovato nel TLB); questa operazione porta anche il valore corrente del bit D nel TLB;
2. se invece il descrittore cercato è nel TLB e
   (a) l'accesso è in scrittura e il bit D vale 0, si comporta come se il descrittore *non* fosse nel TLB;
   (b) negli altri casi (lettura, oppure scrittura con D=1) usa il descrittore nel TLB.

Il caso interessante è il 2.(a). Se in questo caso la MMU si limitasse ad usare l'informazione nel TLB senza andare ad aggiornare il descrittore in memoria, la routine di page fault potrebbe non sapere mai che c'è stata una scrittura su quella pagina. Invece la MMU percorre tutta la catena e aggiorna il bit D. Alla fine ricopia anche il descrittore nel TLB, e questa volta avrà il bit D=1: in questo modo (caso 2.(b)) questa costosa operazione viene eseguita soltanto per la prima scrittura sulla pagina.