Memoria virtuale (seconda parte)

G. Lettieri

30 Marzo 2017

1 MMU_2: tabella su 4 livelli

Proviamo a calcolare quanto è grande la tabella di corrispondenza usata da MMU_1. La memoria virtuale è di 2^{48} byte e ogni pagina è grande 2^{12} byte, quindi la memoria virtuale contiene

$$\frac{2^{48}}{2^{12}} = 2^{36} = 64\,\text{Gi pagine.}$$

La tabella di corrispondenza deve avere una entrata per ognuna di queste pagine. Ogni entrata deve contenere almeno i bit P, D, A e il campo F che fornisce i bit da 12 a 51 dell'indirizzo fisico, per un totale di 43 bit, arrotondati in 6 byte. Se poi vogliamo che la dimensione di ogni entrata sia una potenza di 2, dovremo usare almeno 8 byte. In conclusione, la tabella di corrispondenza dovrà essere di

$$64\,\text{Gi} \times 8\,\text{B} = 512\,\text{GiB}.$$

Difficilmente, quindi, possiamo pensare di avere questa tabella nella memoria fisica. Per affrontare il problema notiamo le seguenti cose:

- La stragrande maggioranza dei programmi ha bisogno soltanto di una piccola frazione dei 2^{48} byte disponibili di memoria virtuale; per questi vorremmo avere una tabella contenente le sole entrate effettivamente utilizzate;
- Possiamo sperare che i programmi grandi rispettino i principi di località e dunque non abbiano bisogno di avere tutta la tabella sempre disponibile.

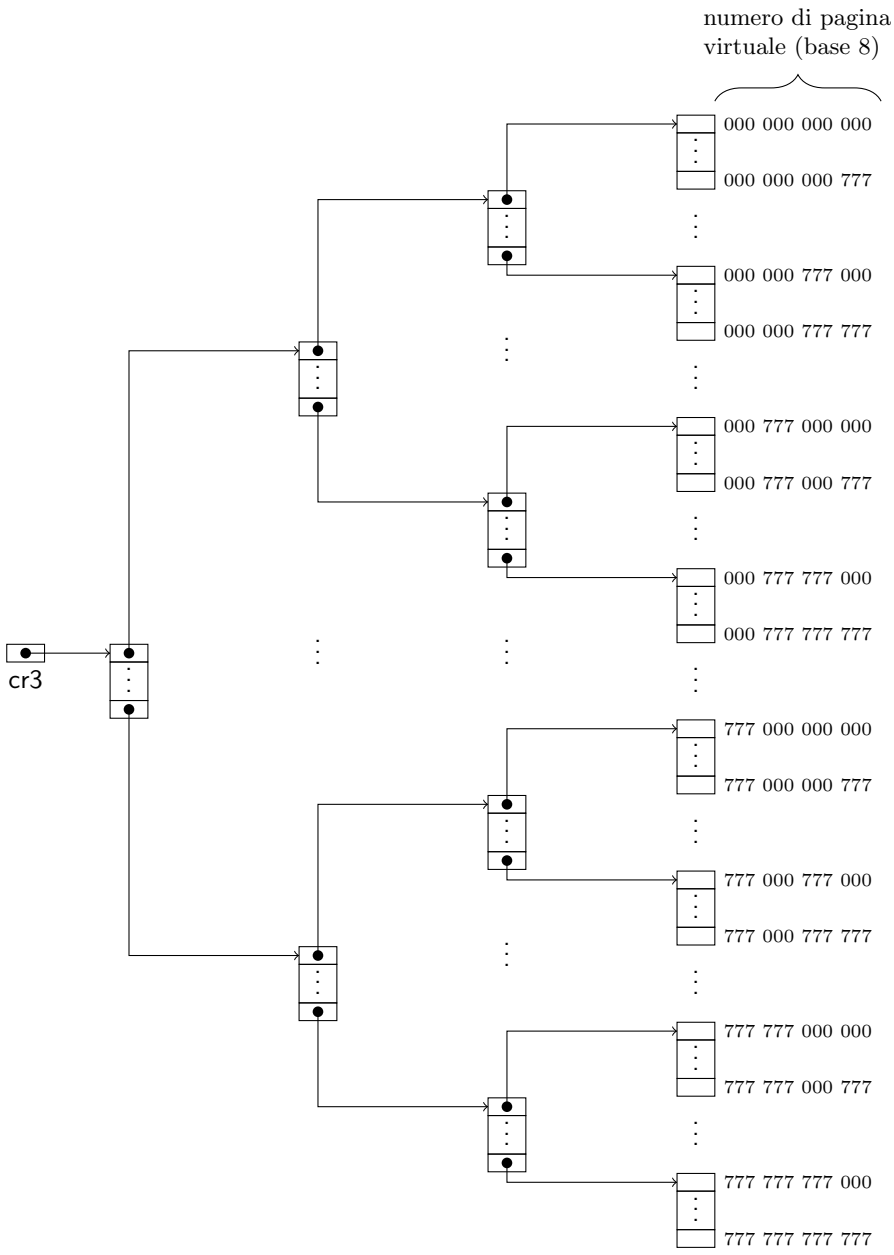
L'idea è di spezzare la tabella in parti più piccole, salvarle in memoria secondaria e caricarle in memoria fisica soltanto quando servono. Per i programmi piccoli potremmo anche evitare di creare le parti di tabella che non verranno mai richieste, risparmiando spazio anche sulla memoria secondaria. Notiamo che 512 GiB sono tanti anche per un comune hard disk, ma per il momento non affrontiamo questo ulteriore problema.

Introduciamo quindi MMU_2, una MMU del tutto identica a MMU_1, tranne che per il formato della tabella di corrispondenza. MMU_2 intercetta tutte le

operazioni di lettura e scrittura in memoria iniziate dalla CPU e tenta di tradurre l'indirizzo virtuale in fisico. Questa volta, però, può non solo accadere che non sia presente la pagina virtuale corrispondente, ma anche che manchi proprio la parte di tabella a cui accedere. In questo caso MMU_2, in modo del tutto simile a MMU_1, solleverà una eccezione di page fault (disattivandosi fino alla fine della routine di page fault). La routine di page fault dovrà riconoscere la situazione e caricare sia la parte mancante di tabella, sia la pagina virtuale richiesta (se manca una parte di tabella, mancano anche tutte le pagine virtuali di cui quella parte si occupa). Per realizzare questa idea, ogni parte di tabella dovrà avere un suo bit P che dica (sia a MMU_2, sia alla routine di page fault) se la parte di tabella è presente o assente. L'idea è di avere una ulteriore tabella, da consultare per prima, con una entrata dedicata ad ognuna delle parti in cui abbiamo scomposto la tabella originaria.

Per semplicità di gestione, conviene spezzare la tabella di corrispondenza in parti di dimensione uguale a quella delle pagine virtuali e fisiche, che stiamo assumendo di 4 KiB. Chiamiamo ciascuna di queste parti "tabelle di livello 1". Ogni tabella di livello 1 contiene 512 entrate e si occupa quindi della traduzione di 512 pagine virtuali. Le tabelle di livello 1 saranno caricate dalla memoria di massa quando il programma ne avrà bisogno, quindi non necessariamente in ordine, e inoltre il caricamento di una potrebbe causare il rimpiazzamento di un'altra. Questo vuol dire che la tabella aggiuntiva, oltre ad avere un bit P per ogni tabella di livello 1, dovrà anche fornire l'indirizzo di memoria fisica in cui la tabella in questione è stata caricata, visto che tale indirizzo non è facilmente prevedibile. Dunque, anche le entrate della tabella aggiuntiva saranno grandi 8 byte, e la tabella occuperà 1 GiB ($2^{36}/512 = 2^{27}$ entrate, ciascuna di 8 byte). Questa dimensione, pur se più realistica, non è comunque ancora accettabile, ma possiamo riapplicare la stessa tecnica anche a questa: la spezziamo in $2^{27}/512 = 2^{18}$ parti, ciascuna grande 4 KiB, che chiamiamo "tabelle di livello 2", e introduciamo una terza tabella da consultare prima di queste. Questa terza tabella sarà ora grande 2 MiB. Per uniformità, spezziamo anche questa tabella in $2^{18}/512 = 512$ parti grandi 4 KiB che chiamiamo "tabelle di livello 3" e introduciamo un'unica "tabella di livello 4", anch'essa di 4 KiB, che contenga i bit P e gli indirizzi delle 512 tabelle di livello 3. La situazione è dunque quella di Fig. 1, in cui si illustra il caso in cui tutte le tabelle, di tutti i livelli, siano presenti.

Si noti che ora `cr3` contiene l'indirizzo della tabella di livello 4. La traduzione da indirizzo virtuale a fisico è contenuta soltanto nelle entrate delle tabelle di livello 1, che sono le tabelle in cui abbiamo scomposto la tabella di corrispondenza di MMU_1 e che, ricomposte in ordine, ci ridanno esattamente la tabella di corrispondenza. Il formato delle entrate delle tabelle di livello 1 è mostrato in Fig. 2 e rispecchia quello dell'architettura Intel/AMD a 64 bit. Chiameremo queste entrate "descrittori di pagina virtuale", come abbiamo fatto fino ad ora, ma anche "descrittori di livello 1", essendo contenuti nelle tabelle di livello 1. Notiamo che i descrittori contengono altre informazioni oltre a quelle che già conosciamo (i bit P, A, D e il campo F). Dal momento che la MMU_2 intercetta tutte le operazioni di lettura e scrittura eseguite dal programma nel suo spazio



Liv. 4	Liv. 3	Liv. 2	Liv. 1	
1	512	256 Ki	128 Mi	Num. tabelle
4 KiB	2 MiB	1 GiB	512 GiB	Dim. tot.

Figura 1: Tabella di corrispondenza su 4 livelli.

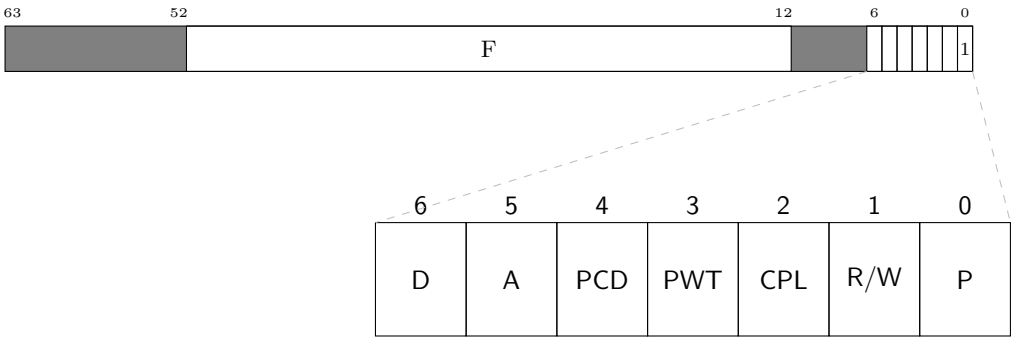


Figura 2: Descrittore di pagina virtuale (tabelle di livello 1).

di indirizzamento, si trova in una buona posizione per svolgere anche altri compiti basati sugli indirizzi. A questo scopo i descrittori di livello 1 contengono anche i seguenti bit

- Un bit **R/W** che vieta (0) o permette (1) le operazioni di scrittura all'interno della pagina virtuale;
- Un bit **CPL** che dice se la pagina "appartiene" al livello sistema o utente; una pagina che appartiene al livello sistema non può essere riferita mentre il processore si trova al livello utente[1].
- Un bit **PWT** che (se vale 1) specifica che la cache dovrà usare la politica *write-through* per tutte le scritture eseguite su questa pagina;
- Un bit **PCD** che (se vale 1) specifica che la cache dovrà essere disabilitata completamente per tutti gli accessi (letture o scritture) eseguite su questa pagina.

Tutte queste informazioni sono scritte dalla routine di inizializzazione o di page fault e soltanto lette dalla MMU_2. Per esempio, la routine di inizializzazione potrebbe porre R/W=0 per tutte le pagine che contengono la sezione `.text` del programma utente da eseguire, per evitare che errori nel programma causino scritture nel suo stesso codice. La routine può porre R/W=0 anche per le pagine che contengono soltanto costanti. Durante l'esecuzione del programma, la MMU_2 solleverà un'eccezione ogni volta che il processore inizia una operazione di scrittura ad un indirizzo con associato R/W=0. La routine che gestisce l'eccezione interromperà il programma e stamperà un messaggio di errore. Analogo è il comportamento del bit CPL, anche se per il momento non ne abbiamo bisogno.

I bit PWT e PCD, invece, sono un mezzo tramite il quale le routine di inizializzazione e page fault possono indirettamente inviare ordini alla cache, sfruttando il fatto che la MMU_2 si trova sul percorso che porta dal processore

[1] In seguito vedremo più precisamente quali sono le regole di accesso.

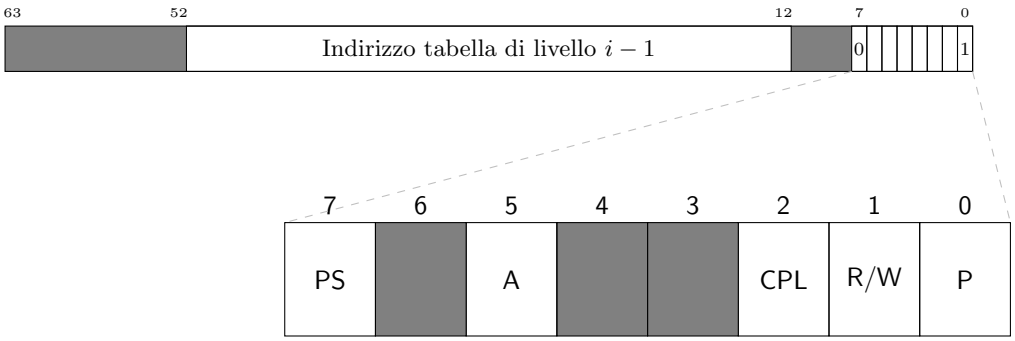


Figura 3: Descrittore di livello i, con $i = 2, 3, 4$.

alla cache (la cache si trova tra la MMU_2 e la memoria fisica). La MMU_2 si preoccuperà di inoltrare l'ordine alla cache ogni volta che traduce un indirizzo. La routine di inizializzazione porrà PCD=1 per tutte le pagine che contengono indirizzi di registri di I/O mappati in memoria, invece che locazioni di memoria. Porre PWT=1 e PCD=0 può essere utile per la parte di indirizzi relativa alla memoria video: quando il programma scrive vogliamo che la scrittura arrivi nella vera memoria video e non si fermi in cache, in modo che il controllore possa visualizzare l'informazione sul video; ma se il programma vuole leggere dalla memoria video, possiamo tranquillamente farlo leggere dalla cache.

Tutte le tabelle di livello 2, 3 e 4 hanno lo stesso formato. Ciascuna di esse contiene 512 entrate con il formato illustrato in Fig. 3. Il formato è simile a quello dei descrittori di livello 1 mostrati in Fig. 2: ci sono ancora i bit P, R/W e CPL e A, nelle stesse posizioni dei bit omonimi di Fig. 2; il campo "Indirizzo tabella di livello $i-1$" occupa la stessa posizione del campo F di Fig. 2; per il momento non ci preoccupiamo del nuovo bit PS e assumiamo che valga 0. Si noti che all'indirizzo della tabella mancano i bit da 0 a 11: la MMU_2 assume che questi bit siano 0, cioè che tutte le tabelle partano da indirizzi che sono multipli di 4 KiB (allineamento naturale). Questo vale anche per la tabella di livello 4: i 12 bit meno significativi di `cr3` devono essere tutti a 0.

Chiameremo le entrate delle tabelle di livello 2 "descrittori di livello 2" o "descrittori delle tabelle di livello 1". Si noti il decremento del livello: i descrittori di livello 2 sono contenuti in tabelle di livello 2 e descrivono tabelle di livello 1. Allo stesso modo chiameremo le entrate delle tabelle di livello 3 "descrittori di livello 3" o "descrittori delle tabelle di livello 2" e, infine, chiameremo le entrate della tabella di livello 4 "descrittori di livello 4" o "descrittori delle tabelle di livello 3".

Anche se simili, i descrittori di livello 2, 3 e 4 non vanno confusi con i descrittori di livello 1: i primi descrivono tabelle, mentre quelli di livello 1 descrivono pagine. Solo i descrittori di livello 1 contengono la traduzione da indirizzo virtuale a fisico, mentre i descrittori di tabelle contengono solo dei puntatori ad altre tabelle (di livello inferiore): servono soltanto per raggiungere

i descrittori di livello 1 e trovare finalmente la traduzione.

Per eseguire una traduzione la MMU_2 deve attraversare tutto l'albero di Fig. 1, partendo dalla tabella di livello 4, puntata da `cr3`, e seguendo i puntatori fino a raggiungere una tabella di livello 1 (che è un pezzo della vecchia tabella di corrispondenza). Ogni volta che arriva ad una tabella, MMU_2 deve sapere quali delle 512 entrate consultare per poter andare avanti. La scelta è dettata dal numero di pagina virtuale (bit 12–47) dell'indirizzo virtuale che MMU_2 sta traducendo. Sulla destra di Fig. 1, sotto la scritta "numero di pagina virtuale (base 8)", abbiamo mostrato il numero di pagina virtuale della pagina di cui si occupa il descrittore di pagina subito a sinistra. È comodo esprimere i numeri di pagina virtuale in base 8, ricordando che ogni cifra in base 8 rappresenta 3 bit in base 2. Vediamo che passando dal primo descrittore (indice 0) della tabella di livello 4 si raggiungono tutti e soli i descrittori che si occupano dei numeri di pagina virtuale che cominciano con 000 (9 bit a 0). Passando dall'ultimo descrittore, invece (numero 511, o 777 in base 8), si raggiungono tutti e soli i descrittori che si occupano dei numeri di pagina virtuale che cominciano con 777 (9 bit a 1). Lo stesso vale anche per gli altri 510 descrittori della tabella di livello 4: il descrittore numero xyz (in base 8) porta a trovare la traduzione di tutti e soli i numeri di pagina virtuale che iniziano per xyz. Quindi, dato il numero di pagina virtuale, MMU_2 non deve fare altro che estrarre i primi 9 bit per sapere quale descrittore di livello 4 consultare. Fatto questo, conosce ora il puntatore alla tabella di livello 3 che le interessa. Sempre ragionando sulla Fig. 1, si può vedere che i successivi 9 bit del numero di pagina virtuale determinano ora l'entrata della tabella di livello 3 da consultare. I successivi 9 bit determinano l'entrata da seguire nella corrispondente tabella di livello 2, e gli ultimi 9 bit selezionano il descrittore di pagina nella corrispondente tabella di livello 1.

In Fig. 4 abbiamo riassunto il processo di traduzione operato dalla MMU_2, mostrando più in dettaglio le operazioni svolte da MMU_2, assumendo che tutti i bit P valgano 1. Al primo passo MMU_2 deve leggere il corretto descrittore di livello 4, che si trova in memoria fisica. Per farlo deve eseguire una operazione di lettura in memoria, ovviamente specificando l'indirizzo. La tabella di livello 4 è un vettore di descrittori, ciascuno grande 8 byte, e la MMU_2 vuole leggere il descrittore il cui indice è contenuto nei bit 39–47 di V (indice di livello 4), sia i_4. L'indirizzo a cui vuole leggere è dunque $\texttt{cr3} + i_4 \times 8$. Si noti che, per quanto detto sull'allineamento naturale delle tabelle, questa operazione non comporta una vera somma, ma solo una concatenazione di bit. Anche la moltiplicazione per 8 consiste, ovviamente, nella concatenazione con tre bit costanti pari a 0. Letto il descrittore di livello 4, MMU_2 ne estrae il campo indirizzo (bit 12–51), lo concatena con i bit 30–38 di V e con altri tre bit a 0, ottenendo così l'indirizzo del descrittore di livello 3. E così anche per i descrittori di livello 2 e 1. Arrivata al livello 1, la MMU_2 estrae il campo F (bit 12–51), lo concatena con l'offset di V e ottiene così la traduzione.

Durante la traduzione la MMU_2 esegue anche altri compiti:

- controlla tutti i bit R/W: una operazione di scrittura è permessa solo se

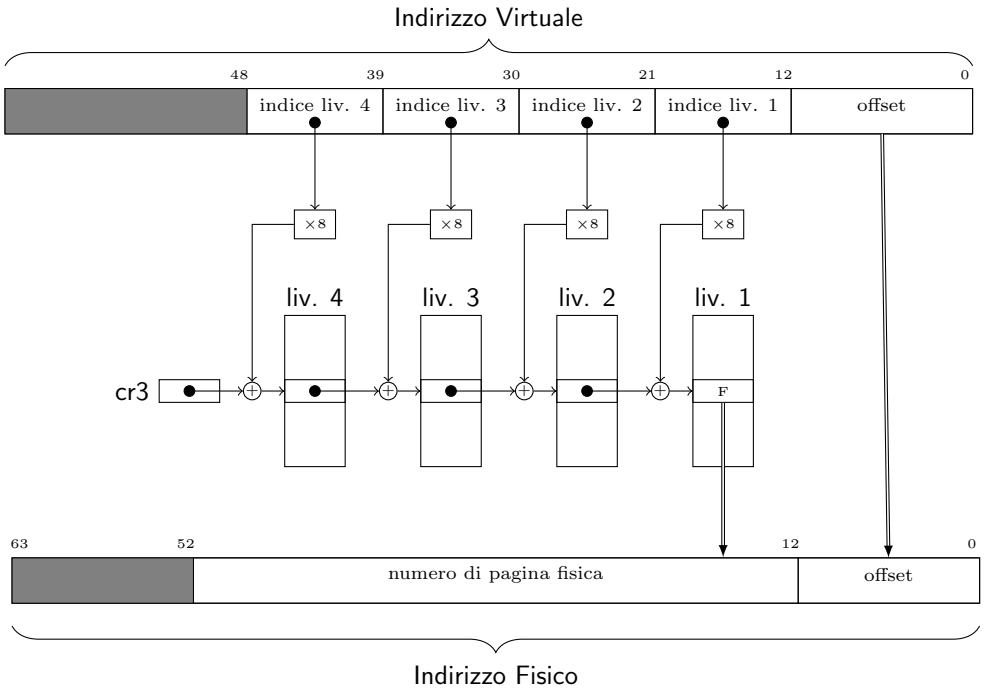


Figura 4: Traduzione da indirizzo virtuale a fisico (pagine di 4 KiB).

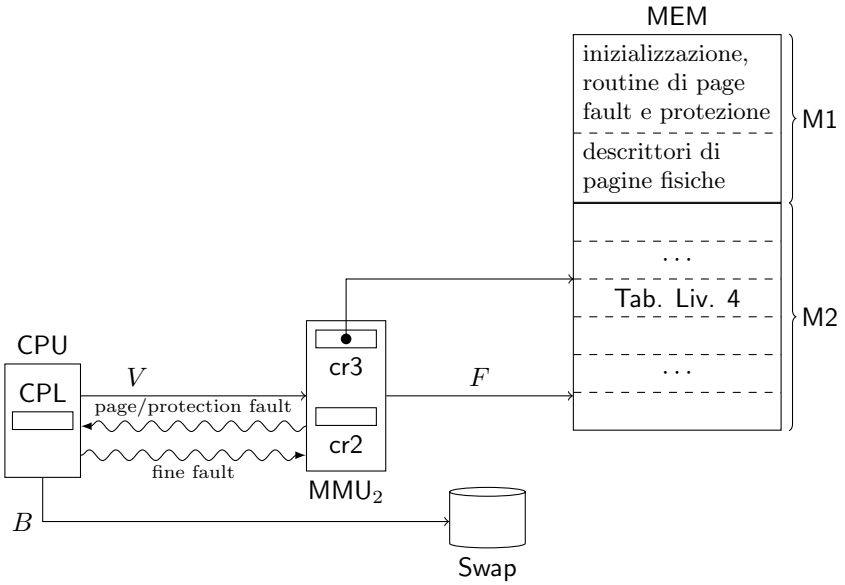


Figura 5: Implementazione della macchina virtuale con MMU_2.

tutti e 4 i bit lungo il percorso la permettono (analogamente per il bit CPL);

- pone a 1 tutti e 4 i bit A incontrati, se non lo erano già;
- in caso di scrittura, pone a 1 il bit D nella tabella di livello 1.

Si noti che la tabella di livello 4 deve essere sempre presente in memoria fisica (non c'è un bit P in `cr3` che dica alla MMU_2 che la tabella è assente), ma tutte le altre possono essere assenti. Assumeremo sempre che l'assenza di una tabella comporti anche l'assenza di tutte le tabelle del sottoalbero di Fig. 1 di cui lei è la radice, e anche l'assenza di tutte le pagine virtuali descritte da tutte le tabelle di livello 1 che si trovano nelle foglie del sottoalbero. Se uno qualunque dei bit P incontrati durante la traduzione vale 0, la MMU_2 smette di tradurre e solleva una eccezione di page fault, a cui la CPU risponde esattamente come faceva per MMU_1: salva V in `cr2`, salva in pila l'indirizzo dell'istruzione corrente, salta alla routine di page fault portandosi contemporaneamente a livello sistema. Come MMU_1, anche MMU_2 resta disattivata mentre è in esecuzione la routine di page fault. Quest'ultima, però, necessita di qualche piccolo aggiustamento rispetto a quanto visto per MMU_1.

1.1 Routine di page fault e descrittori di pagina fisica

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_2, 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_2 riuscirà a completare la traduzione.

In pratica la routine di page fault di MMU_2 deve fare le stesse azioni di quella di MMU_1, 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.

In quale zona della memoria fisica la routine di page fault carica le tabelle? Concettualmente le tabelle farebbero parte di M_1, ma visto che sono grandi quanto le pagine e, come le pagine, possono essere caricate e rimpiazzate, ci conviene caricarle in M_2 insieme alle pagine. In questo modo non dobbiamo stabilire *a priori* quanto spazio dedicare alle tabelle e quanto alle pagine e, ogni volta che si rende necessario operare un rimpiazzamento, possiamo scegliere come vittima indifferentemente una tabella o una pagina. Per uniformità allocheremo anche la tabella di livello 4 in M_2. La Fig. 5 riassume la nuova situazione.

Tutto ciò comporta che dobbiamo modificare i descrittori di pagina fisica, in quanto ora le pagine fisiche possono contenere sia tabelle (di vari livelli), sia pagine virtuali. Il descrittore di pagina fisica conterrà i seguenti campi:

- **livello**, con valori che vanno da -1 a 4: -1 indica che la pagina fisica è libera, 0 che contiene una pagina, 1–4 che contiene una tabella del corrispondente livello; i campi rimanenti sono significativi solo la pagina non è libera;
- **residente**, che può valere **true** o **false**: se true, indica che l'entità contenuta non può essere rimpiazzata (per esempio, si tratta della tabella di livello 4);
- **indirizzo virtuale**: permette di risalire al descrittore che contiene il bit P per l'entità contenuta;
- **blocco**: numero del primo 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 pagina fisica 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_2 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_1, abbiamo il problema di dover risalire dalla pagina fisica al descrittore dell'entità in essa contenuta (in modo da porre P=0). Per le pagine virtuali (livello 0) il campo "indirizzo 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 "indirizzo virtuale" del corrispondente descrittore di pagina fisica l'indirizzo che ha causato il page fault corrente.

Il campo "blocco" si utilizza in modo simile a quanto visto per MMU_1. 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 pagina fisica, e quando l'entità viene scelta come vittima deve essere ricopiato nel descrittore di tabella o pagina. 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.