Ascolta

3.3 Memoria virtuale

Mentre da un lato i registri base e limite possono essere utilizzati per creare l'astrazione degli spazi degli indirizzi, dall'altro sorge un nuovo problema: la gestione del *bloatware*, cioè del software sempre più "gonfio". Le dimensioni della memoria sono in costante aumento, ma quelle dei software aumentano ancora più velocemente. Negli anni '80 molte università utilizzavano un sistema timesharing con dozzine di utenti (più o meno soddisfatti) che lavoravano contemporaneamente su un sistema VAX da 4 MB. Oggi Microsoft raccomanda di avere almeno 2 GB per un sistema a 64 bit con Windows 10.

Per via di questi sviluppi sorge la necessità di eseguire programmi troppo grandi rispetto alla memoria, e naturalmente si desiderano sistemi in grado di supportare l'esecuzione contemporanea di più programmi, che singolarmente trovano posto nella memoria ma nell'insieme risultano più grandi della memoria disponibile. Lo swapping non è un'opzione interessante se il computer è dotato di hard disk, poiché un tipico disco SATA ha una velocità di trasferimento di picco (*peak transfer rate*) di qualche centinaio di MB/s, il che significa qualche secondo per lo scambio verso il disco di un programma di 1 GB, e altrettanto per lo scambio verso la memoria. Gli SSD sono notevolmente più veloci, ma l'overhead rimane considerevole.

Il problema dei programmi più grandi della memoria è presente fin dalle origini dell'informatica, benché in aree limitate, come le scienze e l'ingegneria (serve molta memoria per simulare l'universo, ma anche per simulare un nuovo aeroplano). Negli anni '60 fu adottata una soluzione che divideva i programmi in piccole parti chiamate **overlay**. All'avvio di un programma veniva caricato nella memoria solo il gestore degli overlay, che caricava e avviava subito l'overlay 0. Al termine, veniva indicato al gestore degli overlay di caricare l'overlay 1 sopra l'overlay 0 in memoria (se c'era spazio per farlo) oppure sovrascrivendo l'overlay 0 (in mancanza di spazio). Alcuni sistemi di overlay erano estremamente complessi, e permettevano la presenza in memoria di molti overlay contemporaneamente. Gli overlay erano mantenuti nella memoria non volatile ed erano scambiati verso la memoria e verso il disco dal gestore degli overlay.

Il lavoro di scambio degli overlay era eseguito dal sistema operativo, ma la suddivisione del programma in parti doveva essere fatta manualmente dal programmatore. Suddividere grandi programmi in piccole parti modulari richiedeva molto tempo, era noioso e passibile di errori. Pochi programmatori lo facevano bene. Ben presto qualcuno pensò a un sistema che demandasse tutto il lavoro al computer.

Il metodo che fu escogitato (Fotheringham, 1961) sarebbe diventato noto come **memoria virtuale**. L'idea alla base della memoria virtuale è che ogni programma ha un suo spazio degli indirizzi, suddiviso in parti chiamati **pagine**. Ogni pagina è un intervallo di indirizzi contigui. Queste pagine sono mappate sulla memoria fisica, ma per eseguire il programma non è indispensabile che tutte le pagine siano contemporaneamente nella memoria fisica. Quando il programma fa riferimento a una parte del suo spazio degli indirizzi che è nella memoria fisica, l'hardware esegue direttamente la mappatura necessaria. Quando il programma fa riferimento a una parte del suo spazio degli indirizzi che *non* è nella memoria fisica, il sistema operativo viene allertato, va a prelevare la parte mancante ed esegue nuovamente fallita.

In un certo senso, la memoria virtuale è la generalizzazione dell'idea dei registri base e limite. L'8088 aveva registri base separati (ma non aveva registri limite) per il testo e i dati. Con la memoria virtuale, invece di avere una rilocazione separata solo per i segmenti dati e testo, l'intero spazio degli indirizzi può essere rimappato sulla memoria fisica in unità abbastanza piccole. Implementazioni diverse della memoria virtuale adottano scelte diverse rispetto a queste unità; oggi la maggior parte dei sistemi impiega una tecnica detta paging (paginazione) che prevede unità di dimensioni fisse, ad esempio di 4 KB. Una soluzione alternativa detta segmentazione usa come unità interi segmenti di dimensione variabile. Osserveremo entrambe le soluzioni, concentrandoci però sulla paginazione (la segmentazione non è più molto usata).

La memoria virtuale funziona bene in un sistema multiprogrammazione, con vari pezzetti di molti programmi contemporaneamente in memoria. Mentre un programma è in attesa che venga letto qualche suo pezzo, la CPU può essere assegnata a un altro processo.

?

3.3.1 Paginazione

La maggior parte dei sistemi di memoria virtuale usa una tecnica chiamata **paginazione** o **paging**. Su qualsiasi computer i programmi fanno riferimento a un set di indirizzi di memoria. Quando un programma esegue un'istruzione come

```
MOV REG,1000
```

la esegue per copiare il contenuto dell'indirizzo di memoria 1000 in REG (presupponendo che il primo operando rappresenti la destinazione e il secondo l'origine). Gli indirizzi possono essere generati usando indicizzazione, registri base e vari altri modi.

Questi indirizzi generati dal programma sono chiamati **indirizzi virtuali** e formano lo **spazio degli indirizzi virtuali**. Sui computer senza memoria virtuale, l'indirizzo virtuale è posto direttamente sul bus di memoria e provoca la lettura o la scrittura della parola della memoria fisica con lo stesso indirizzo. Quando è utilizzata la memoria virtuale, gli indirizzi virtuali non vanno direttamente al bus di memoria, ma a una **MMU** (**Memory Management Unit**, unità di gestione della memoria) che mappa gli indirizzi virtuali sugli indirizzi della memoria fisica, come illustrato nella **Figura 3.8**.

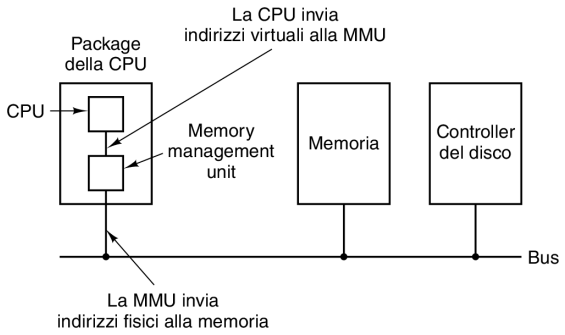

Figura 3.8 Posizione e funzione della MMU. In questo caso la MMU è illustrata come parte della CPU poiché oggigiorno di solito lo è. Dal punto di vista logico, però, potrebbe essere un chip separato, come è stata in passato.

Un esempio molto semplice del funzionamento di questa mappatura è illustrato nella **Figura 3.9**. In questo esempio abbiamo un computer che genera indirizzi di 16 bit, da 0 a 64K – 1. Questi sono gli indirizzi virtuali. Il computer però ha solo 32 KB di memoria fisica, quindi si possono scrivere programmi di 64 KB ma non è possibile caricarli in memoria nella loro interezza nella memoria ed eseguirli. Una copia completa dell'immagine del nucleo del programma, fino a 64 KB, deve quindi essere presente sul disco o SSD, in modo da poterne caricare dinamicamente delle parti secondo necessità.

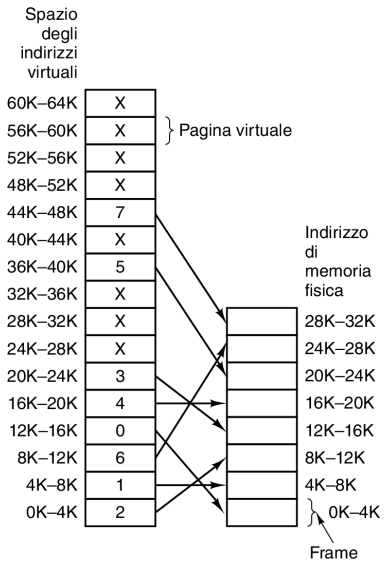

Figura 3.9 La relazione fra indirizzi virtuali e indirizzi di memoria fisica è data dalla tabella delle pagine. Ogni pagina inizia a un multiplo di 4096 e termina 4095 indirizzi più in alto, così da 4K a 8K significa in realtà 4096–8191 e da 8K a 12K significa 8192–12287.

Lo spazio degli indirizzi virtuali è suddiviso in unità di dimensione fissa, chiamate pagine. Le unità corrispondenti nella memoria fisica sono chiamate **frame** o **page frame**. Le pagine e i frame sono della stessa dimensione; in questo esempio sono di 4 KB, ma nei sistemi reali le pagine vanno da 512 byte fino a 64 KB. Con 64 KB di spazio degli indirizzi virtuali e 32 KB di memoria fisica otteniamo 16 pagine virtuali e 8 frame. I trasferimenti fra RAM e memoria non volatile sono sempre di pagine intere. Molti processori supportano più dimensioni di pagina, che possono essere mescolati e abbinati a discrezione del sistema operativo. Ad esempio, l'architettura x86-64 supporta pagine da 4 KB, 2 MB e 1-GB, e potremmo utilizzare le pagine da 4 KB per le applicazioni utente e una singola pagina da 1 GB per il kernel. In seguito vedremo perché a volte è meglio usare una singola pagina grande anziché un gran numero di pagine piccole.

La notazione della **Figura 3.9** è la seguente. L'intervallo indicato come 0K–4K significa che gli indirizzi fisici o virtuali in quella pagina vanno da 0 a 4095. L'intervallo 4K–8K si riferisce agli indirizzi da 4096 a 8191, e così via. Ogni pagina contiene esattamente 4096 indirizzi, da un multiplo di 4096 a un multiplo di 4096 meno 1 (la numerazione inizia da 0).

Quando il programma prova per esempio ad accedere all'indirizzo 0, usando l'istruzione

```
MOV REG,0
```

l'indirizzo virtuale 0 è inviato alla MMU. La MMU vede che questo indirizzo virtuale cade nella pagina 0, ovvero nell'intervallo da 0 a 4095 che, secondo la sua mappatura, è il frame 2 (da 8192 a 12287). Trasforma così l'indirizzo in 8192 ed emette sul bus l'indirizzo 8192. La memoria non sa dell'esistenza dell'MMU, vede semplicemente una richiesta di lettura o scrittura all'indirizzo 8192 e ottempera. La MMU ha di fatto mappato tutti gli indirizzi virtuali fra 0 e 4095 sugli indirizzi fisici da 8192 a 12287.

Allo stesso modo, l'istruzione

```
MOV REG,8192
```

è trasformata effettivamente in

```
MOV REG,24576
```

poiché l'indirizzo virtuale 8192 (nella pagina virtuale 6) è mappato sul 24576 (nel frame fisico 24K–28K). Come terzo esempio, l'indirizzo virtuale 20500 dista 20 byte dall'inizio della pagina virtuale 5 (indirizzi virtuali da 20480 a 24575) e la sua mappatura sull'indirizzo fisico è 12288 + 20 = 12308.

Questa capacità di mappare le 16 pagine virtuali su uno qualsiasi degli 8 frame, impostando in modo appropriato la mappa della MMU, da sola non basta a risolvere il problema dello spazio degli indirizzi virtuali più grande della memoria fisica. Poiché abbiamo solo 8 frame fisici, solo otto delle pagine virtuali della **Figura 3.9** sono mappate sulla memoria fisica. Le altre, mostrate nella figura con una X, non sono mappate. Nell'hardware, un **bit presente/assente** tiene traccia di quali pagine sono presenti fisicamente in memoria.

Che cosa accade se, per esempio, il programma fa riferimento a indirizzi non mappati usando l'istruzione

```
MOV REG,32780
```

che è il byte 12 all'interno della pagina virtuale 8 (che parte da 32768)? La MMU rileva che la pagina non è mappata (è contrassegnata da una X nella figura) e causa una trap della CPU verso il sistema operativo. Questa trap è chiamata

page fault (errore di pagina). Il sistema operativo preleva un frame poco utilizzato e ne scrive il contenuto su disco (se non è già presente). Prende poi la pagina alla quale è stato appena fatto riferimento e la pone nel frame appena liberato, cambia la mappa e riavvia l'istruzione che era in trap.

Per esempio, se il sistema operativo decide di sfrattare il frame 1 dalla memoria, dovrebbe caricare la pagina virtuale 8 all'indirizzo 8192 e apportare due modifiche alla mappa della MMU: per prima cosa dovrebbe contrassegnare la prima voce della pagina virtuale come non mappata, per fare la trap di qualsiasi accesso futuro agli indirizzi virtuali da 4096 a 8191. Poi dovrebbe sostituire la X nella voce della pagina virtuale 8 con un 1 in modo che, quando l'istruzione che ha generato la trap viene rieseguita, essa mappi l'indirizzo virtuale 32780 sull'indirizzo 4108 (4096 + 12).

Guardiamo adesso all'interno della MMU per vedere come funziona e perché abbiamo scelto di usare una dimensione di pagina che sia una potenza di 2. Nella **Figura 3.10** vediamo un esempio di un indirizzo virtuale, 8196 (0010000000000100 in binario), mappato usando la mappa della MMU della **Figura 3.9**. L'indirizzo virtuale di 16 bit in ingresso è suddiviso in un numero di pagina di 4 bit e un offset di 12 bit. Con 4 bit per il numero di pagina, possiamo avere 16 pagine e con 12 bit di offset possiamo indirizzare 4096 byte per pagina.

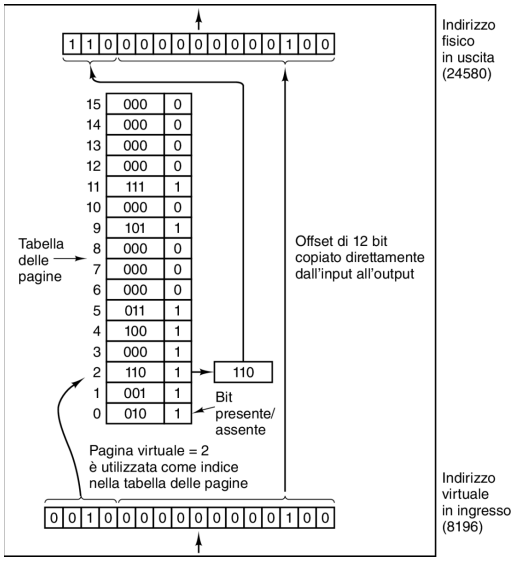

Figura 3.10 L'operazione interna della MMU con 16 pagine da 4 KB.

Il numero di pagina è usato come indice nella **tabella delle pagine** che porta al numero di frame corrispondente alla pagina virtuale. Se il bit *presente/assente* è 0, avviene una trap al sistema operativo. Se il bit è 1, il numero di frame trovato nella tabella delle pagine viene copiato nei tre bit più significativi del registro di output, insieme all'offset di 12 bit che è copiato senza modifiche dall'indirizzo virtuale in arrivo. Insieme formano un indirizzo fisico di 15 bit. Il registro di output è poi posto sul bus di memoria come indirizzo fisico di memoria.

Negli esempi abbiamo usato indirizzi a 16 bit per rendere più comprensibili testo e figure. I moderni PC usano indirizzi a 32 o a 64 bit; in teoria, un computer con indirizzi da 32 bit e pagine da 4 KB potrebbero usare esattamente lo stesso metodo trattato sopra. La tabella delle pagine dovrebbe avere 220 (1.048.576) voci. Su un computer con vari GB di RAM la cosa sarebbe fattibile. Gli indirizzi a 64 bit e le pagine da 4 KB invece richiederebbero 252 (più o meno 4,5 × 1015) voci nella tabella delle pagine, una quantità spropositata. Decisamente non fattibile, quindi sono necessarie altre tecniche che vedremo tra breve.

3.3.2 Tabelle delle pagine

In una semplice implementazione si può sintetizzare la mappatura degli indirizzi virtuali sugli indirizzi fisici come segue: l'indirizzo virtuale è diviso in un numero di pagina virtuale (i bit più significativi, o superiori) e un offset (i bit meno significativi, o inferiori). Per esempio, con un indirizzo di 16 bit e una dimensione di pagina di 4 KB, i 4 bit superiori potrebbero specificare una delle 16 pagine virtuali e i 12 bit inferiori specificherebbero l'offset (da 0 a 4095) nella pagina selezionata. È anche possibile anche una divisione con 3 o 5 o un altro numero di bit per la pagina; suddivisioni differenti implicano diverse dimensioni della pagina.

Il numero della pagina virtuale è utilizzato come un indice nella tabella delle pagine per trovare la voce per quella pagina virtuale; dalla voce della tabella delle pagine si trova il numero di frame (se esiste). Il numero di frame è allegato all'estremità più significativa dell'offset, rimpiazzando il numero di pagina virtuale, per formare un indirizzo fisico che può essere inviato alla memoria.

Quindi lo scopo della tabella delle pagine è mappare le pagine virtuali sui frame delle pagine. Da un punto di vista matematico, la tabella delle pagine è una funzione che ha come argomento il numero della pagina virtuale e come risultato il numero del frame. Usando il risultato di questa funzione, il campo della pagina virtuale in un indirizzo virtuale può essere rimpiazzato dal campo del frame, andando a formare un indirizzo di memoria fisico.

In questo capitolo ci occupiamo solamente della memoria virtuale, non della virtualizzazione, quindi non parleremo ancora di macchine virtuali. Vedremo invece nel Capitolo 7 che ogni macchina virtuale richiede la propria memoria virtuale e per questo l'organizzazione delle tabelle di pagina diventa molto più complicata, richiedendo tabelle delle pagine shadow o nidificate e altro ancora. Come vedremo, però, la paginazione e la memoria virtuale sono comunque piuttosto sofisticate anche senza queste arcane configurazioni.

Struttura di una voce della tabella delle pagine

Passiamo ora dalla struttura della tabella delle pagine in generale ai dettagli della singola voce di una tabella delle pagine. L'esatto layout di una voce dipende molto dalla macchina, ma il tipo di informazioni presente è più o meno lo stesso per tutte le macchine. Nella **Figura 3.11** è mostrato l'esempio di una voce della tabella delle pagine. La dimensione cambia a seconda del computer, ma quella usuale in un generico computer odierno è di 64 bit. Il campo più importante è il *numero del frame* (*frame number*): l'obiettivo della mappatura delle pagine è ottenere questo valore. Se la dimensione della pagina è 4 KB (ovvero 212 byte), per il numero del frame sono necessari solo i 52 bit più significativi[1], e rimangono 12 bit per codificare altre informazioni sulla pagina. Ad esempio, il bit *Presente/assente* indica se la voce è valida e possa essere usata. Se questo bit è 0, la pagina virtuale cui appartiene la voce non è effettivamente in memoria. L'accesso alla voce della tabella delle pagine con questo bit impostato a 0 causa un page fault.

▶ 1

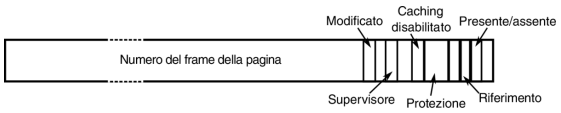

Figura 3.11 Tipica voce della tabella delle pagine.

I bit *Protezione* specificano quali tipi di accesso sono consentiti. Nella forma elementare questo campo contiene 1 bit, con 0 che significa lettura/scrittura e 1 per la sola lettura. Un'impostazione più sofisticata ha 3 bit, ogni bit per consentire lettura, scrittura ed esecuzione della pagina. Il bit *Supervisor* è in qualche modo correlato, e indica se la pagina sia accessibile soltanto al codice con privilegi, ovvero al sistema operativo (o supervisore) oppure anche ai programmi utente. Qualsiasi tentativo di accesso a una pagina supervisore da parte di un programma utente causa un page fault.

I bit *Modificato* e *Riferimento* tengono traccia dell'uso della pagina. Quando viene scritta una pagina, l'hardware imposta automaticamente il bit *Modificato*. Questo bit è valorizzato quando il sistema operativo decide di riutilizzare un frame. Se la sua pagina è stata modificata (cioè è "sporca") deve essere riscritta sulla memoria non volatile. Se non è stata modificata (cioè è "pulita") può essere abbandonata, poiché la copia su disco o SSD è ancora valida. Talvolta il bit è chiamato **dirty bit** poiché riflette lo stato della pagina.

Il bit *Riferimento* è impostato ogni qualvolta si faccia riferimento alla pagina, sia in lettura sia in scrittura. Serve per aiutare il sistema operativo a scegliere una pagina da "sfrattare" quando si verifica un page fault; le pagine inutilizzate sono più "sfrattabili" di quelle usate, e questo bit gioca un ruolo importante in molti degli algoritmi di sostituzione delle pagine che studieremo in seguito.

Infine, l'ultimo bit consente di disabilitare la cache per la pagina. Questa caratteristica è importante per le pagine che mappano sui registri dei dispositivi anziché in memoria. Se il sistema operativo si trova in un ciclo veloce in attesa che qualche dispositivo di I/O risponda a un comando appena inviato, è fondamentale che l'hardware continui a prelevare la parola dal dispositivo e non usi una vecchia copia presente nella cache. Con questo bit, l'utilizzo della cache può essere disabilitato. Questo bit non serve alle macchine che hanno uno spazio di I/O separato e che non usano l'I/O mappato in memoria.

Notate che l'indirizzo su disco (cioè del blocco su disco o SSD) utilizzato per contenere la pagina quando non è in memoria non fa parte della tabella delle pagine. La ragione è semplice: la tabella delle pagine contiene solo le informazioni necessarie *all'hardware* per tradurre un indirizzo virtuale in un indirizzo fisico. Le informazioni necessarie al sistema operativo per gestire i page fault sono conservate in tabelle software all'interno del sistema operativo. L'hardware non ne ha bisogno.

Prima di addentrarci in ulteriori argomenti di implementazione, è bene tornare a puntualizzare che fondamentalmente la memoria virtuale crea una nuova astrazione (lo spazio degli indirizzi) che è un'astrazione della memoria fisica, così come un processo è un'astrazione del processore fisico (la CPU). La memoria virtuale può essere implementata suddividendo in pagine lo spazio virtuale degli indirizzi e mappando ogni pagina su un frame di memoria fisica o tenendole (momentaneamente) non mappate. Questo paragrafo, pertanto, riguarda fondamentalmente un'astrazione creata dal sistema operativo e il modo in cui viene gestita.

È opportuno anche sottolineare che *tutti* gli accessi alla memoria eseguiti dal software usano indirizzi virtuali, e questo vale non solo per i processi utente ma anche per il sistema operativo. In altre parole, anche il kernel ha le sue mappature nelle tabelle delle pagine. Quando un processo esegue una chiamata di sistema, devono essere utilizzate le tabelle delle pagine del sistema operativo. Lo scambio di contesto (che richiede lo scambio delle tabelle delle pagine) non è economico, quindi alcuni sistemi usano un bel trucchetto e mappano semplicemente le tabelle delle pagine del sistema operativo in tutti i processi utente, ma con il bit Supervisore che indica che tali pagine sono accessibili solo al sistema operativo. Perciò, quando il processo utente cerca di accedere a una di queste pagine attiva un'eccezione. Quando però il processo utente esegue una chiamata di sistema, non c'è più bisogno di scambiare le tabelle delle pagine: tutte le tabelle delle pagine del kernel e quelle utente sono disponibili per il sistema operativo, e ciò sveltisce la chiamata di sistema. In genere, quando il sistema operativo è mappato nei processi utente viene mappato in cima allo spazio degli indirizzi, così da non interferire con i programmi utente (che iniziano a 0 o lì vicino). A volte i programmi utente iniziano a 4K anziché a 0, in modo che i riferimenti all'indirizzo 0 (che spesso sono errori) vengano catturati da una trap.

3.3.3 Velocizzare la paginazione

Dopo aver visto le basi della memoria virtuale e della paginazione è il momento di analizzarne in maggior dettaglio le possibili implementazioni. In ogni sistema di paginazione devono essere affrontate due questioni principali:

1. la mappatura dall'indirizzo virtuale all'indirizzo fisico deve essere veloce;
2. anche se lo spazio virtuale degli indirizzi è enorme, la tabella delle pagine non deve essere troppo grande.

Il primo punto deriva del fatto che la mappatura virtuale-fisica deve avvenire a ogni riferimento alla memoria. Alla fin fine, tutte le istruzioni vengono dalla memoria e molte di esse fanno riferimento a operandi anch'essi in memoria, quindi è necessario fare per ogni istruzione uno, due e talvolta più riferimenti alla tabella delle pagine per ogni istruzione. Se l'esecuzione di un'istruzione impiega, diciamo, 1 ns, la ricerca nella tabella delle pagine deve essere fatta in meno di 0,2 ns per evitare che la mappatura diventi un collo di bottiglia importante.

Il secondo punto deriva dal fatto che tutti i computer moderni usano indirizzi virtuali di almeno 32 bit, e i 64 bit stanno diventando la norma su desktop e portatili. Anche se un processore moderno usa solo 48 dei 64 bit per l'indirizzamento, con una dimensione della pagina di 4 KB uno spazio di indirizzi di 48 bit ha 64 miliardi di pagine. Con 64 miliardi di pagine nello spazio virtuale degli indirizzi, la tabella delle pagine deve avere 64 miliardi di voci di 64 bit ciascuna. È opinione diffusa che usare centinaia di gigabyte solo per memorizzare la tabella delle pagine sia un tantino eccessivo. E ricordiamo che ciascun processo ha bisogno della propria tabella delle pagine (perché ha il suo spazio degli indirizzi personale).

La necessità di una mappatura veloce delle pagine per i grandi spazi degli indirizzi è un vincolo molto importante per la costruzione dei computer Lo schema più semplice (quanto meno dal punto di vista concettuale) è avere un'unica tabella delle pagine che consiste di un array di registri hardware veloci, con una sola voce per ogni pagina virtuale, indicizzata per numero di pagina virtuale, come illustrato nella **Figura 3.10**. All'avvio di un processo, il sistema operativo carica i registri con la tabella delle pagine del processo, presa da una copia che tiene in memoria. Durante l'esecuzione del processo non sono necessari altri riferimenti alla memoria per la tabella delle pagine. Il vantaggio di questo metodo è che è semplice e non richiede riferimenti alla memoria durante la mappatura. Lo svantaggio è che è insopportabilmente dispendioso se la tabella delle pagine è estesa; nella maggior parte dei casi non è praticabile. Un altro è che dover caricare l'intera tabella delle pagine a ogni cambio di contesto compromette le prestazioni.

In una soluzione diametralmente opposta, la tabella delle pagine può essere interamente caricata nella memoria principale; a quel punto all'hardware serve un singolo registro che punti all'inizio della tabella delle pagine. Questo design consente di cambiare la mappatura virtuale-fisica a ogni cambio di contesto ricaricando un solo registro. Lo svantaggio è che naturalmente richiede uno o più riferimenti di memoria per la lettura della tabella delle pagine durante l'esecuzione di ogni istruzione, rendendola molto lenta.

Translation lookaside buffer

Esaminiamo alcuni schemi largamente implementati per velocizzare la paginazione e per gestire grandi spazi degli indirizzi virtuali, partendo dalla prima delle due esigenze. Il punto di partenza della maggior parte delle tecniche di ottimizzazione è che la tabella delle pagine sta nella memoria. Questa scelta progettuale ha potenzialmente un impatto enorme sulle prestazioni. Consideriamo per esempio un'istruzione di 1 byte che copia un registro in un altro. In assenza di paginazione, questa istruzione fa un solo riferimento alla memoria, per prelevare l'istruzione. Con la paginazione, per accedere alla tabella delle pagine è necessario almeno un altro riferimento alla memoria. Poiché il limite della velocità di esecuzione è generalmente dato dalla velocità con cui la CPU può prelevare le istruzioni e i dati dalla memoria, dover fare due riferimenti alla memoria per ogni riferimento alla memoria riduce le prestazioni della metà. A queste condizioni, nessuno userebbe la paginazione.

I progettisti di computer conoscono questo problema da anni, e hanno escogitato una soluzione basata sull'osservazione che la maggior parte dei programmi tende a fare un gran numero di riferimenti a un piccolo numero di pagine e non il contrario. Dunque solo una piccola parte delle voci della tabella delle pagine viene letta frequentemente; il resto è poco utilizzato.

La soluzione è dotare i computer di un piccolo dispositivo hardware per mappare gli indirizzi virtuali sugli indirizzi fisici senza passare dalla tabella delle pagine. Il dispositivo, chiamato **TLB** (**Translation Lookaside Buffer**) o qualche volta anche **memoria associativa**, è illustrato nella **Figura 3.12**. Si trova abitualmente all'interno della MMU e consiste di un numero ridotto di voci, otto in questo caso, ma raramente più di 256. Ciascuna voce contiene informazioni riguardo una pagina, tra cui il numero di pagina virtuale, un bit impostato quando la pagina viene modificata, il codice di protezione (i permessi di lettura, scrittura ed esecuzione) e il frame fisico in cui si trova la pagina. Questi campi hanno una corrispondenza uno-a-uno con i campi nella tabella delle pagine, eccetto che per il numero di pagina virtuale, che non è necessario nella tabella delle pagine. Un altro bit indica se la voce è valida (cioè in uso) o no.

Valido	*Pagina virtuale*	*Modificato*	*Protezione*	*Frame*
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

Figura 3.12 Un TLB per velocizzare la paginazione.

Un esempio che potrebbe generare il TLB della **Figura 3.12** è un processo in un ciclo che percorre le pagine virtuali 19, 20 e 21, così che queste voci del TLB abbiano codici di protezione per la lettura e l'esecuzione. I dati principali attualmente usati (diciamo, un array in elaborazione), sono sulle pagine 129 e 130. La pagina 140 contiene gli indici usati nei calcoli dell'array. Infine, sulle pagine 860 e 861 c'è lo stack.

Analizziamo adesso come funziona il TLB. Quando un indirizzo virtuale viene presentato alla MMU per la traduzione, l'hardware prima guarda se il suo numero di pagina virtuale è presente nel TLB confrontandolo simultaneamente (cioè in parallelo) con tutte le voci. Questa operazione richiede un hardware specializzato, presente in tutte le MMU dotate di TLB. Se trova un riscontro valido e l'accesso non viola i bit di protezione, il frame è prelevato direttamente dal TLB, senza andare alla tabella delle pagine in memoria. Se il numero di pagina virtuale è presente nel TLB, ma l'istruzione prova a scrivere su una pagina di sola lettura, si genera un errore di protezione.

È interessante ciò che accade quando il numero di pagina virtuale non è nel TLB. La MMU rileva il **TLB miss** (assenza dal TLB) ed esegue una normale ricerca sulla tabella delle pagine. Quindi sfratta una delle voci dal TLB e la rimpiazza con la voce della tabella delle pagine appena trovata, così se quella pagina viene riutilizzata a breve, la seconda volta si avrà una **TLB hit** (presenza nel TLB) invece di un TLB miss. Quando

una voce è eliminata dal TLB, il bit Modificato viene copiato di nuovo nella voce della tabella delle pagine nella memoria. Gli altri valori sono già lì, eccetto il bit Riferimento. Quando il TLB viene caricato dalla tabella delle pagine, tutti i campi vengono presi dalla memoria.

Se il sistema operativo vuole modificare i bit nella voce della tabella delle pagine (ad esempio, per rendere scrivibile una pagina in sola lettura), lo farà modificandolo in memoria. Per accertarsi che la successiva operazione di scrittura sulla pagina riesca, però, deve anche eliminare dal TLB la voce corrispondente con i vecchi bit di autorizzazione.

Gestione software del TLB

Finora abbiamo supposto che ogni macchina con la memoria virtuale paginata abbia delle tabelle delle pagine riconosciute dall'hardware, più un TLB. In questo schema, la gestione del TLB e il trattamento degli errori del TLB sono interamente svolti dall'hardware della MMU. Le trap al sistema operativo avvengono solo quando una pagina non è in memoria.

Questo presupposto vale per molte CPU, ma molte macchine RISC moderne, tra cui SPARC, MIPS e l'oggi defunto HP PA, forniscono supporto per la gestione delle pagine tramite software. Su queste macchine, le voci del TLB sono caricate esplicitamente dal sistema operativo. Quando si verifica un TLB miss, invece di far andare la MMU alla tabella delle pagine per cercare e prelevare il necessario riferimento alla pagina, viene semplicemente generato un errore di TLB e il problema passa al sistema operativo. Il sistema deve trovare la pagina, rimuovere una voce dal TLB, inserirne una nuova e riavviare l'istruzione in errore. Naturalmente tutto questo deve avvenire con una manciata di istruzioni, poiché i TLB miss sono molto più frequenti dei page fault, ed è importante capire il perché.

Il punto chiave è che di solito in memoria ci sono migliaia di pagine, quindi i page fault sono rari, ma i TLB normalmente contengono solo 64 voci, quindi i TLB miss accadono in continuazione. I produttori di hardware potrebbero ridurre il numero di TLB miss aumentando la dimensione del TLB, ma è un'operazione costosa e la superficie occupata da un TLB maggiorato toglierebbe spazio ad altri elementi importanti, come le cache. La progettazione dei chip è piena di compromessi.

È abbastanza sorprendente che, se il TLB è mediamente grande (diciamo 64 voci) per ridurre la frequenza di miss, la gestione software del TLB si rivela di un'efficienza accettabile. In questo caso, il guadagno principale è una MMU molto più semplice, che libera una quantità considerevole di spazio sul chip della CPU per le cache e altri elementi che possono migliorare le prestazioni.

È fondamentale capire la differenza fra due tipi di miss. Un **soft miss** avviene quando la pagina di riferimento non è nel TLB ma è nella memoria. In questo caso basta aggiornare il TLB, non serve alcun I/O da disco o SSD. Solitamente per trattare un soft miss ci vogliono 10-20 istruzioni macchina che possono essere completate in pochi nanosecondi. Per contro, un **hard miss** avviene quando la pagina richiesta non è in memoria (e ovviamente nemmeno nel TLB). Per prelevare la pagina serve un accesso al disco o all'SSD, nell'ordine dei millisecondi a seconda del tipo di memoria non volatile. Un hard miss è milioni di volte più lento di un soft miss. Cercare la mappatura nella gerarchia delle tabelle delle pagine è un'operazione che prende il nome di **page table walk** (passeggiata per la tabella delle pagine).

La situazione, in realtà, è ancora peggiore. Un miss non è solamente soft o hard; alcuni miss sono leggermente più soft (o leggermente più hard) rispetto ad altri. Si supponga, per esempio, che la page walk non trovi la pagina all'interno della tabella delle pagine del processo e il programma incorra in un errore di pagina. Le possibilità sono tre. Primo, la pagina potrebbe essere in memoria, ma non nella tabella delle pagine di questo processo, magari perché è stata caricata dalla memoria non volatile da parte di un altro processo. In questo caso non è necessario accedere nuovamente alla memoria non volatile, ma è sufficiente mappare la pagina in maniera appropriata nella tabella delle pagine. Questo è un miss abbastanza soft, chiamato anche **minor page fault** (errore di pagina non grave). Secondo, un **major page fault** (grave) si può verificare se la pagina dev'essere caricata dalla memoria non volatile. Terzo, è possibile che il programma abbia semplicemente avuto accesso a un indirizzo non valido e non sia necessario aggiungere alcuna mappatura al TLB. In questo caso, il sistema operativo solitamente termina il programma con un **segmentation fault** (errore di segmentazione): solamente in questo caso è stato il programma a comportarsi in modo sbagliato. Tutti gli altri casi vengono automaticamente corretti dall'hardware e/o dal sistema operativo, pur al costo di una certa perdita di prestazioni.

3.3.4 Tabelle delle pagine per grandi memorie

I TLB possono essere usati per velocizzare la traduzione dell'indirizzo virtuale nell'indirizzo fisico rispetto allo schema originale della tabella delle pagine in memoria, ma la velocità non è l'unico problema da affrontare. Un altro, ad esempio, è la gestione degli spazi degli indirizzi virtuali molto grandi.

Tabelle delle pagine multilivello

Consideriamo innanzitutto le **tabelle delle pagine multilivello**. Un semplice esempio è illustrato nella **Figura 3.13**. Nella **Figura 3.13**(a) abbiamo un indirizzo virtuale a 32 bit partizionato in un campo *PT1* a 10 bit, un campo *PT2* a 10 bit e un campo *Offset* a 12 bit. Poiché gli offset sono 12 bit, le pagine sono da 4 KB e ce ne sono in totale 2^{20}. Il segreto del metodo della tabella delle pagine multilivello sta nell'evitare di mantenere tutte le tabelle delle pagine in memoria per tutto il tempo. In particolare quelle non necessarie dovrebbero essere messe da parte. Supponiamo che un processo abbia bisogno di 12 MB, i 4 MB alla base per il testo del programma, i successivi 4 MB per i dati e i 4 MB in cima per lo stack. In mezzo, fra la cima dei dati e la base dello stack c'è un gigantesco spazio vuoto inutilizzato.

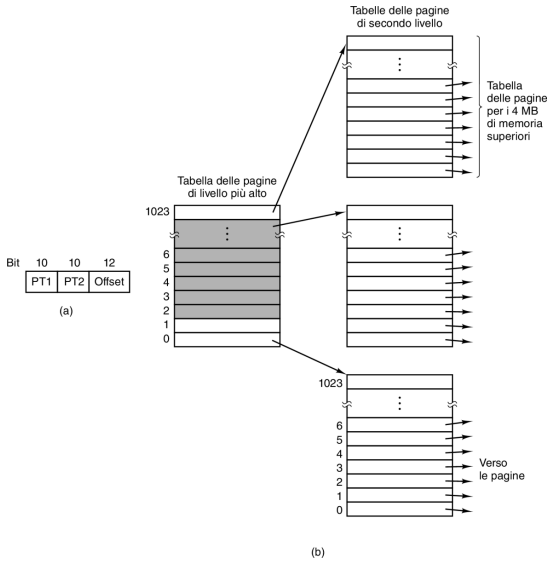

Figura 3.13 (a) Un indirizzo a 32 bit con due campi per le tabelle delle pagine. (b) Tabelle delle pagine a due livelli.

Nella **Figura 3.13**(b) vediamo come funziona la tabella delle pagine a due livelli. Sulla sinistra abbiamo la tabella delle pagine di livello più alto, con 1024 voci, corrispondenti ai 10 bit del campo *PT1*. Quando viene presentato un indirizzo virtuale alla MMU, prima estrae il campo *PT1* e usa questo valore come indice nella tabella delle pagine di livello più alto. Ognuna di queste 1024 (o 210) voci rappresenta 4 MB (o 222 byte), poiché l'intero spazio virtuale degli indirizzi di 4 GB (cioè 32 bit) è stato spezzettato in parti di 4096 (o 212) byte.

La voce localizzata tramite l'indicizzazione nella tabella delle pagine di livello superiore produce l'indirizzo o il numero di frame di una tabella delle pagine di secondo livello. La voce 0 della tabella delle pagine di livello superiore punta alla tabella delle pagine per il testo del programma, la voce 1 punta alla tabella delle pagine per i dati e la voce 1023 punta alla tabella delle pagine per lo stack. Le altre voci (in grigio) sono inutilizzate. Il campo *PT2* adesso è usato come un indice nella tabella delle pagine di secondo livello selezionata per cercare il numero di frame della pagina stessa.

Come esempio consideriamo l'indirizzo virtuale a 32 bit 0x00403004 (4.206.596 in decimale), corrispondente a 4.206.596 – 4MB = 12.292 byte nell'area dati. Questo indirizzo virtuale corrisponde a *PT1* = 1, *PT2* = 2 e *Offset* = 4. La MMU usa prima *PT1* per indicizzare nella tabella delle pagine di livello più alto e ottiene la voce 1, corrispondente agli indirizzi da 4 MB a 8 MB – 1. Poi usa *PT2* per indicizzare nella tabella delle pagine di secondo livello appena trovata ed estrae la voce 3, corrispondente agli indirizzi da 12288 a 16383 dentro il suo pezzo di 4 MB (cioè gli indirizzi assoluti da 4.206.592 a 4.210.687). Questa voce contiene il numero di frame della pagina contenente l'indirizzo virtuale 0x00403004. Se quella pagina non è in memoria, il bit *Presente/assente* nella voce della tabella delle pagine sarà zero, provocando un page fault. Se la pagina è nella memoria, il numero di frame preso dalla tabella delle pagine di secondo livello è combinato con l'offset (4) per costruire l'indirizzo fisico. Questo indirizzo viene posto nel bus e inviato alla memoria.

La cosa interessante da notare riguardo alla **Figura 3.13** è che, sebbene lo spazio degli indirizzi contenga più di un milione di pagine, sono effettivamente necessarie solo quattro tabelle delle pagine: la tabella di livello più alto e le tabelle di secondo livello da 0 a 4 MB (per il testo del programma), da 4 MB a 8 MB (per i dati) e i 4 MB in alto (per lo stack). I bit *Presente/Assente* in 1021 voci della tabella delle pagine di livello più alto sono impostati a 0, forzando un page fault nel caso qualcuno provasse mai ad accedervi. Se accadesse, il sistema operativo noterebbe che il processo sta provando a riferirsi a memoria che non gli compete e

intraprenderebbe un'azione adeguata, come mandargli un segnale o terminarlo. In questo esempio abbiamo scelto numeri interi di varie dimensioni e abbiamo preso *PT1* uguale a *PT2*, ma nella pratica sono ovviamente possibili anche altri valori.

Il sistema della tabella delle pagine a due livelli della **Figura 3.13** può essere esteso a tre, quattro o più livelli. I livelli aggiuntivi danno maggior flessibilità. Per esempio, il processore Intel a 32 bit 80386 (lanciato nel 1985) era in grado di indirizzare 4 GB di memoria, utilizzando una tabella di pagine a due livelli consistente di una **directory delle pagine** le cui voci puntavano a tabelle di pagina che, a loro volta, puntavano agli effettivi frame da 4 KB. Sia la directory delle pagine sia le tabelle delle pagine contenevano ognuna 1024 voci, arrivando a un totale di $2^{10} \times 2^{10} \times 2^{12} = 2^{32}$ bit indirizzabili, come desiderato.

Dieci anni dopo, il Pentium Pro introdusse un altro livello: la **tabella dei puntatori alla directory delle pagine**. Inoltre, ogni voce di ciascun livello della gerarchia delle tabelle delle pagine fu aumentata da 32 a 64 bit, così che potesse indirizzare la memoria al di sopra del limite dei 4 GB. Poiché conteneva solamente 4 voci nella tabella dei puntatori alla directory delle pagine, 512 in ogni directory delle pagine e 512 in ogni tabella delle pagine, la quantità totale di memoria che era in grado di indirizzare era comunque limitata a un massimo di 4 GB. Quando alla famiglia dei processori x86 fu aggiunto un completo supporto ai 64 bit (originariamente da parte di AMD), il livello aggiuntivo avrebbe potuto essere chiamato "puntatore alla tabella dei puntatori della directory delle pagine", o qualcosa di altrettanto orribile, il che sarebbe stato perfettamente in linea con il modo in cui i produttori di chip tendono ad assegnare nomi alle cose. Per fortuna, non fu così. L'alternativa scelta, "**mappa delle pagine di quarto livello**", per quanto non sia esattamente poetica, è almeno breve e chiara. In ogni modo, questi processori oggi utilizzano tutte le 512 voci di ciascuna tabella, arrivando a una quantità di memoria indirizzabile pari a $2^9 \times 2^9 \times 2^9 \times 2^9 \times 2^{12} = 2^{48}$ byte. Si poteva aggiungere anche un altro livello, ma probabilmente i produttori hanno pensato che, almeno per il momento, 256 TB di memoria indirizzabile sarebbero stati sufficienti.

Sbagliavano. Alcuni nuovi processori supportano un quinto livello per estendere la dimensione degli indirizzi fino a 57 bit. Con uno spazio degli indirizzi del genere è possibile indirizzare fino a 128 petabyte. È un numero enorme. Consente di mappare file giganteschi. Ovviamente, un numero di livelli così alto rende ancor più costose le page table walk.

Tabelle invertite delle pagine

Un'alternativa al continuo aumento dei livelli della gerarchia della paginazione è nota con il termine **tabelle invertite delle pagine**. Queste tabelle sono state utilizzate per la prima volta in processori come PowerPC, UltraSPARC e Itanium (a volte noto come "Itanic", visto che non fu il grande successo che Intel si aspettava). Ha seguito il destino di Amazon Fire Phone, Apple Newton, AT&T Picture Phone, videoregistratore Betamax, auto DeLorean, Ford Edsel e Windows Vista.

Le tabelle invertite delle pagine, però, sono ancora tra noi. In questo design c'è una sola voce per frame nella memoria reale, anziché una voce per pagina dello spazio virtuale degli indirizzi. Per esempio, con indirizzi virtuali a 64 bit, una pagina da 4 KB e 16 GB di RAM, una tabella delle pagine invertite richiede solo 4.194.304 voci. La voce tiene traccia di ciò (processo, pagina virtuale) che si trova nel frame.

Sebbene le tabelle invertite delle pagine risparmino una gran quantità di spazio, presentano un grosso svantaggio quando lo spazio virtuale degli indirizzi è molto superiore rispetto alla memoria fisica: la traduzione da virtuale a fisica diventa molto, molto difficile. Quando il processo *n* fa riferimento alla pagina virtuale *p,* l'hardware non può più trovare la pagina fisica usando *p* come indice nella tabella delle pagine. Deve cercare invece la voce (*n*, *p*) nell'intera tabella invertita delle pagine. Per di più questa ricerca deve essere eseguita per ogni riferimento alla memoria e non solo in caso di page fault. Fare una ricerca su una tabella di 256K per ogni riferimento alla memoria non è certamente il modo per rendere fulminea una macchina.

La soluzione è data dall'impiego del TLB. Se il TLB può contenere tutte le pagine più usate, la traduzione può avvenire velocemente come con le normali tabelle delle pagine. Nel caso di un TLB miss, però, deve essere eseguita una ricerca software nella tabella invertita delle pagine. Un modo possibile di realizzare questa ricerca è di avere una tabella di hash sull'indirizzo virtuale. Tutte le pagine virtuali attualmente in memoria che hanno lo stesso valore di hash sono concatenate, come illustrato nella **Figura 3.14**. Se la tabella di hash ha tanti elementi quante sono le pagine fisiche della macchina, la catena media avrà una sola voce, velocizzando incredibilmente la mappatura. Una volta trovato il numero di frame, la nuova coppia (virtuale, fisico) è inserita nel TLB.

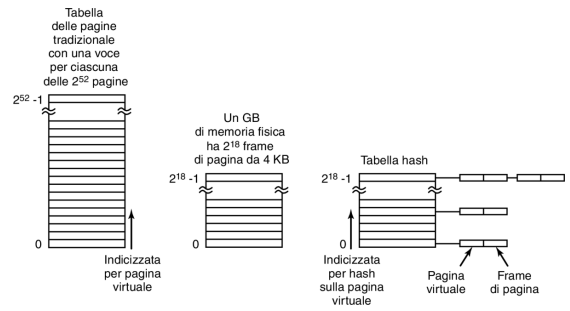

Figura 3.14 Confronto fra una tabella delle pagine tradizionale e una tabella invertita delle pagine.

Le tabelle invertite delle pagine sono utilizzate nelle macchine a 64 bit poiché, anche con una dimensione della pagina molto grande, il numero di voci della tabella delle pagine è enorme. Per esempio, con pagine di 4 MB e indirizzi virtuali di 64 bit sono necessarie 2^{42} voci di tabella delle pagine.