Azzolini Riccardo 2019-12-12

Gestione della memoria: paginazione e memoria virtuale

1 Protezione della memoria

Se un processo prova ad accedere a un indirizzo al di fuori del suo spazio logico, viene generata una **memory protection exception**.

Ciò avviene quando un processo che ha le pagine $0, 1, \ldots, h$ tenta di accedere a una pagina $k > h$. Questo controllo è semplice e veloce se l'architettura offre un apposito registro, il **Page Table Size Register** (**PTSR**), che memorizza il numero dell'ultima pagina del processo running: per ogni accesso alla memoria, l'hardware controlla che il numero di pagina sia minore o uguale al valore del PTSR, e in caso contrario solleva un'eccezione.

Come per il PTAR, anche il valore del PTSR per ciascun processo deve essere memorizzato nel PCB corrispondente, e assegnato al registro durante il context switch.

1.1 Permessi di accesso

Il campo *prot* della page table codifica i permessi di accesso (lettura/scrittura) che un processo ha per ciascuna sua pagina. Esso può essere utile, ad esempio, per facilitare la condivisione di memoria tra processi.

Se un processo cerca di eseguire un tipo di accesso per cui non ha il permesso, la MMU deve generare una memory protection exception. Ciò richiede supporto a livello di architettura:

- a ogni accesso alla memoria, la MMU deve poter confrontare il valore di prot con il tipo di accesso effettuato;
- per ottimizzare questo confronto, il campo prot deve essere presente nel TLB;
- ecc.

2 Principio di località

Il **principio di località** afferma che un indirizzo logico generato eseguendo un'istruzione ha probabilità elevata di essere vicino agli indirizzi logici generati da altre istruzioni recenti.

Tale principio vale perché:

- gli accessi alle varie parti di una stessa struttura dati sono spesso effettuati in istruzioni ravvicinate (un esempio è l'accesso a tutte le celle di un array in un ciclo);
- le istruzioni di salto sono solitamente non più del 10–20 % del totale, quindi i processi eseguono prevalentemente istruzioni memorizzate l'una dopo l'altra.

In base a questo principio, quando è necessario effettuare lo swap out di una pagina (cioè liberare un frame), per minimizzare i page fault conviene sceglierla tra quelle visitate meno di recente: infatti, è meno probabile che la pagina scelta debba poi essere ricaricata in memoria poco dopo.

Per determinare quali pagine sono state visitate di recente si sfrutta il campo *ref* della page table. Esso è un semplice bit:

- al momento del page-in, il bit ref della pagina caricata viene impostato a 1;
- a intervalli regolari, vengono azzerati i bit ref di tutte le pagine;
- quando si accede a una pagina, il suo bit ref viene aggiornato a 1.

Quindi, se $ref = 1$ la pagina è stata visitata di recente, e allora, in base al principio di località, il frame che la contiene non è un buon candidato a essere liberato.

Nota: In aggiunta al principio di località, per la scelta del frame da liberare è meglio dare la priorità a quelli contenenti pagine con $mod = 0$, al fine di evitare il costo dell'aggiornamento dell'immagine sul disco.

3 Frame per processo e dimensione delle pagine

Se si aumenta il numero di page frame allocati a ogni processo:

- diminuisce la probabilità di avere page fault (aumento di efficienza);
- si hanno meno processi in RAM (diminuzione di efficienza).

Se, invece, il numero di page frame per processo è troppo basso, si ha un fenomeno chiamato **thrashing**:

- i page fault sono frequenti, quindi cresce il page I/O;

- si hanno tanti context switch (perché un processo che genera un page fault viene messo in waiting finché la pagina richesta non è caricata, e quindi ne viene schedulato un altro).

È quindi necessario trovare un equilibrio.

Invece, diminuendo la dimensione s delle pagine:

- si riduce la memoria sprecata (che, in media, è metà dell'ultima pagina di ciascun processo, ovvero $s/2$ byte per processo);
- a parità di memoria allocata a un processo, il numero di page fault diminuisce, perché i processi tendono a visitare più aree di memoria, ciascuna contigua, ma separate tra loro: se le pagine sono più piccole, il processo ne ha di più, quindi può tenere in RAM più aree di memoria separate;
- la page table occupa più spazio in memoria, perché:
  - il numero di entry è maggiore;
  - servono più bit per i page number e i frame number, quindi ciascuna entry è più grande;
- lo swap device perde efficienza se usa blocchi di dimensione ridotta (perché conviene effettuare trasferimenti di blocchi più grossi tra RAM e disco).

Anche qui, va quindi trovato un equilibrio.

4 TLB reach

Il **TLB reach** è la quantità di RAM coperta dal TLB. Esso si calcola come prodotto tra la dimensione di una pagina e il numero di entry che il TLB può contenere:

$$\text{TLB reach} = \text{page size} \cdot \text{TLB entries}$$

Siccome la dimensione delle RAM cresce rapidamente, mentre la dimensione delle memorie associative no, il rapporto tra il TLB reach e la capacità della RAM tende a diminuire. Di conseguenza, gli accessi alla memoria che sfruttano il TLB sono pochi, cioè aumenta il numero di TLB miss, e allora gli accessi alla RAM tendono a rallentare.

Il TLB reach è un altro fattore da valutare nella scelta della dimensione delle pagine (oltre a quelli elencati in precedenza): più esse sono grandi, maggiore è il TLB reach.

Inoltre, esistono delle tecniche per migliorare il TLB reach senza dover aumentare la dimensione di tutte le pagine. Una di queste è l'uso delle *superpagine*.

5 Superpagine

Una **superpagina** è come una pagina, ma ha dimensione pari a $2^n \cdot$ page size, per un n opportuno.

In un'architettura che supporta le superpagine, una TLB entry può riferirsi a una pagina o a una superpagina: se si riferisce a una superpagina, il TLB reach aumenta.

Quando ci sono più pagine contigue con accessi frequenti, il SO le promuove a una superpagina (**promotion**). Viceversa, se alcune pagine della superpagina non hanno più accessi, la superpagina viene "smembrata", separando di nuovo le singole pagine che la compongono (**demotion**).

La realizzazione di questa soluzione non è banale, perché la MMU deve effettuare la traduzione degli indirizzi per le superpagine in modo diverso rispetto a quella per le pagine.

6 Dimensione delle page table

Se le page table dei processi sono grandi, possono occupare una porzione significativa della RAM, riducendo quindi il numero di pagine che possono essere caricate in memoria.

Per risparmiare memoria, esistono due strategie:

- **Inverted Page Table** (**IPT**);
- **doppia paginazione**.

7 Inverted Page Table

L'IPT ha una entry per ogni *frame* (mentre una page table "normale" ha una entry per ogni *pagina*), che contiene:

- l'indicazione che il frame è libero, oppure
- la coppia (PID, page number), se il frame è occupato.

Il vantaggio è che la dimensione dell'IPT è fissa, perché dipende dal numero di frame, il quale è a sua volta determinato dalle dimensioni della memoria e del singolo frame. Essa non dipende quindi dal numero di processi e dal loro numero di pagine.

Lo svantaggio, però, è il costo della traduzione degli indirizzi: siccome l'indice dell'IPT è il frame number, e non il page number (che è il dato conosciuto, estratto dall'indirizzo logico), la MMU deve cercare tra tutti i frame quello che ospita la pagina richiesta. Nel

caso peggiore, quando la pagina non è caricata in memoria, si genera un page fault solo dopo una scansione completa dell'IPT.

In pratica, questo costo non è accettabile. Serve allora una soluzione per evitare la ricerca sull'intera IPT. Una possibilità è l'uso di una **funzione hash**.

7.1 IPT con funzione hash

In questa variante dell'IPT, ogni entry ha almeno 4 campi:

- lo stato libero/occupato (simile al validity bit della page table "normale");
- la coppia (PID, page number);
- il frame number, che in questa variante non corrisponde più all'indice della entry;
- un puntatore a un'altra entry, per la costruzione di una linked list.

Si fissa un numero primo a, maggiore del numero di frame, e si costruisce una **hash table** contenente a puntatori verso l'IPT.

Questa hash table sfrutta la funzione hash

$$h : x \mapsto x \bmod a$$

Dato un qualsiasi numero intero x, essa dà come risultato un altro intero $h(x) = y$, tale che $0 \leq y \leq a - 1$. y è quindi un indice valido per la hash table.

Come input della funzione h si può usare una coppia (P, p) di PID e page number: concatenando P e p, si ottiene una stringa di bit, che viene interpretata come intero e usata come argomento della funzione.

$$h(Pp) = y$$

All'interno della IPT, tutte le coppie (P, p) con lo stesso hash y (cioè per le quali $h(Pp) = y$) sono concatenate in una linked list, sfruttando il campo puntatore presente in ciascuna entry. Inoltre, il puntatore situato all'indice y della hash table punta alla prima entry di tale lista (se questa esiste, altrimenti è nullo).

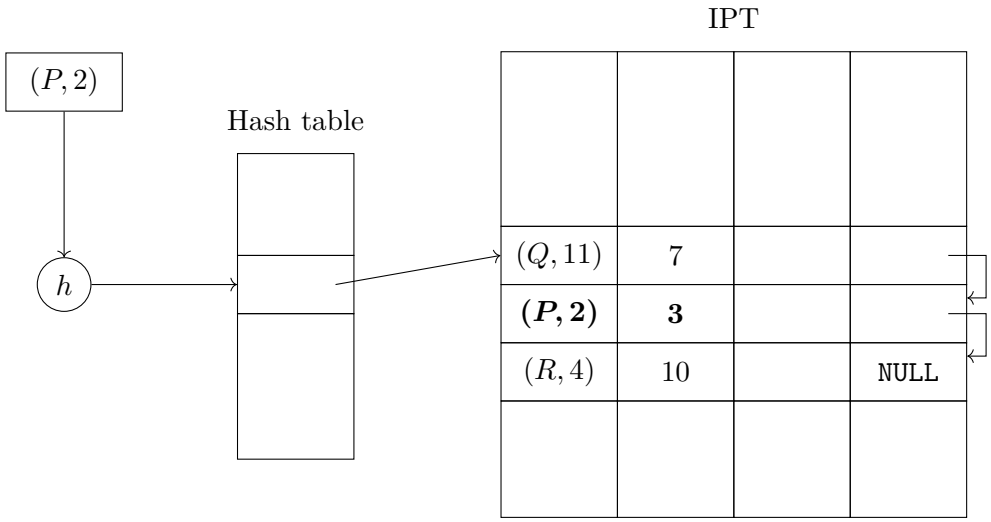
Quando un processo con PID P cerca di accedere alla sua pagina p:

1. viene calcolato l'hash $h(Pp) = y$;
2. si accede all'indice y della hash table;
3. se il puntatore individuato al punto 2 non è nullo, si accede tramite di esso a una delle linked list formate dalle entry della IPT (altrimenti, se il puntatore è nullo, significa che la pagina cercata non è in RAM, e allora si ha un page fault);
4. si esegue una scansione della lista:

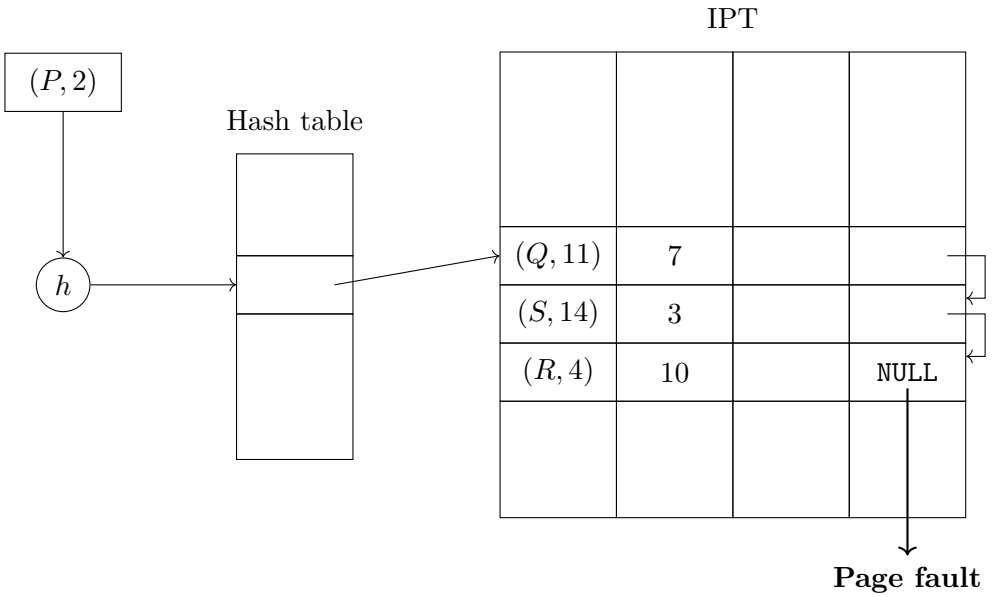
- se si trova la entry corrispondente alla coppia (P, p), la ricerca termina, e si usa il frame number appena ottenuto per costruire l'indirizzo fisico;
- se, invece, si raggiunge la fine della lista senza trovare la entry cercata, significa che la pagina corrispondente non è caricata in memoria, e viene quindi generato un page fault.

Con questa soluzione, anche nel caso peggiore devono essere scandite solo le entry della IPT che appartengono a una delle linked list, cioè che hanno lo stesso hash. Inoltre, come caso particolare, se il puntatore nella hash table è nullo si ha immediatamente un page fault, senza neanche bisogno di accedere alla IPT.

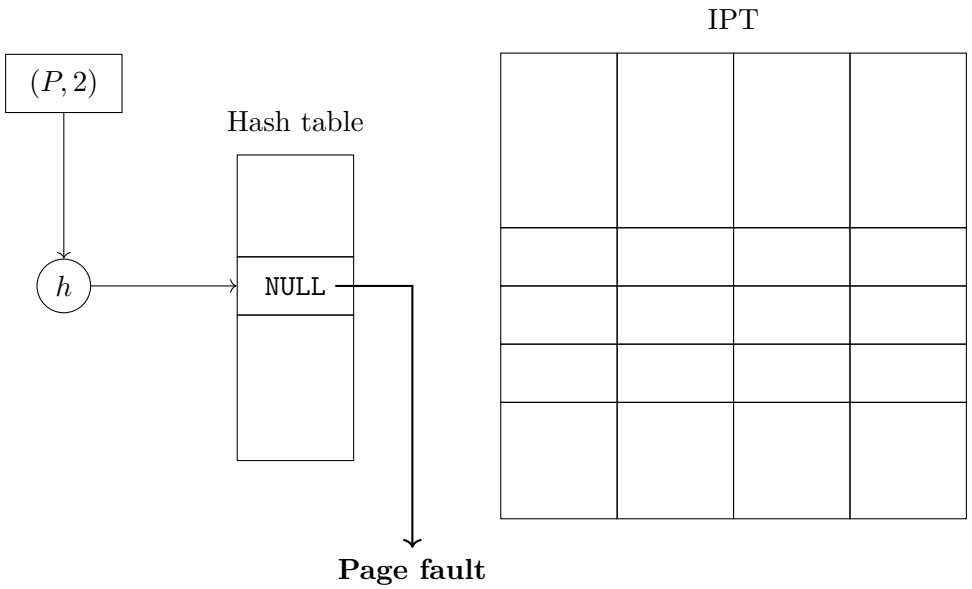
7.1.1 Esempi



In questo caso, la pagina richiesta corrisponde alla seconda entry nella linked list, ed è caricata in memoria nel frame numero 3.



In questo caso, sono presenti coppie (PID, page number) con lo stesso hash di $(P, 2)$, ma non c'è una entry per quest'ultima coppia (cioè la pagina corrispondente non è in memoria): quando la ricerca raggiunge l'ultima entry della lista, cioè trova il puntatore nullo, termina sollevando un page fault.



In questo caso, l'IPT non contiene alcuna coppia con hash corrispondente a quello di $(P, 2)$, quindi il puntatore nella hash table è nullo, e si ha immediatamente un page fault.

8 Doppia paginazione

Idea: Per evitare di avere in memoria page table di grandi dimensioni, si realizza una struttura a due livelli, formata da page table di alto livello (*higher level page table*) che contengono gli indirizzi di page table di basso livello (*lower level page table*). Le lower level page table sono paginate, quindi è possibile effettuarne lo swap out per liberare spazio in RAM, e riportarle in memoria solo quando servono per tradurre degli indirizzi.

Siano 2^f il numero di frame e 2^s la dimensione di ciascun frame: un indirizzo è costituito da $f + s$ bit. Se una entry della page table occupa 2^e byte, allora una pagina della page table contiene 2^{s-e} entry. Gli indirizzi assumono quindi il seguente significato:

- $f - (s - e)$ bit più significativi: indice della entry nella higher level page table;
- $s - e$ bit centrali: indice della entry nella lower level page table;
- s bit meno significativi: offset nella pagina.

Con la doppia paginazione, la traduzione di un indirizzo avviene mediante l'accesso a due page table:

1. si accede alla entry della higher level page table specificata dai bit più significativi dell'indirizzo logico, per ottenere l'indirizzo della lower level page table;
2. all'interno della lower level page table situata all'indirizzo appena ottenuto, si accede alla entry specificata dai bit centrali dell'indirizzo logico, che contiene il numero del frame in cui è caricata la pagina richiesta (se *validity bit* $= 1$, altrimenti viene generato un page fault).

Questa soluzione può essere generalizzata a $n > 2$ livelli di paginazione.

Il vantaggio della doppia paginazione è che le page table dei livelli inferiori al primo possono non essere sempre in memoria, ma ciò può comportare anche dei page fault in più: infatti, una o più delle page table necessarie per tradurre un certo indirizzo possono non essere caricate in RAM al momento della traduzione. Nel caso peggiore, con n livelli di paginazione, un singolo accesso alla memoria può generare n page fault, se è necessario caricare le page table di tutti gli $n - 1$ livelli inferiori e la pagina a cui si vuole accedere non è in memoria.