Azzolini Riccardo 2019-11-28

Gestione della memoria: paginazione e memoria virtuale

1 Paginazione

L'idea della **paginazione** è dividere sia i programmi che la memoria in componenti di dimensioni non arbitrarie, ma *fisse*.

- La memoria è divisa in 2^f **page frame**, ciascuno di capacità 2^s byte. 2^{f+s} corrisponde quindi alla dimensione totale della memoria.

  Ognuno dei 2^{f+s} byte della memoria è individuato da un indirizzo fisico di $f + s$ bit, che si interpreta come composto da due parti:

  - gli f bit più significativi individuano il page frame;
  - gli s bit meno significativi individuano l'**offset** del byte nel page frame.

  Ad esempio, data una memoria da 4 GB $= 2^{32}$ byte, suddividendola in page frame da 1 kB $= 2^{10}$ byte si ottengono 2^{22} frame: $f = 22$ e $s = 10$. Un esempio di indirizzo è `00000000000000000010100000110000`, che si suddivide in:

Page frame	Offset
`0000000000000000001010`	`0000110000`
10	48

- Ogni processo viene logicamente diviso in **pagine** della stessa dimensione dei page frame. Un processo di dimensione d ha quindi $\lceil d/2^s \rceil$ pagine.

  Ognuno dei d byte dello spazio del processo è individuato da un indirizzo logico di $l + s$ bit:

  - gli l bit più significativi individuano la pagina;
  - gli s bit meno significativi individuano l'offset del byte nella pagina.

  In un sistema con memoria virtuale, in numero di pagine di un processo può essere maggiore del numero di frame della RAM, quindi si può avere $l > f$.

Con la paginazione, allocare un processo in memoria significa associare un page frame a ogni sua pagina (supponendo che debba essere caricato in RAM l'intero processo, che è vero solo nei sistemi senza memoria virtuale). Ogni pagina può essere messa in un frame qualsiasi: non è necessario che tutti i frame occupati da un processo siano adiacenti (altrimenti si avrebbe allocazione contigua, e la paginazione non avrebbe senso).

Con questa soluzione si elimina *quasi* completamente la frammentazione: l'ultimo frame di ciascun processo viene usato solo in parte, quindi rimane un minimo di spazio inutilizzabile (tanto meno quanto più piccoli sono i frame). In particolare, per un processo di dimensione d vengono allocati $2^s \cdot \lceil d/2^s \rceil - d$ byte "di troppo".

1.1 Strutture dati necessarie

- Per ogni processo, il SO mantiene una **page table**, che indica il page frame in cui è allocata ciascuna pagina di tale processo. In pratica, la page table è una sorta di array, nel quale l'indice è il numero di pagina, e il valore corrispondente è il numero del frame in cui essa è allocata.

  Per un processo con $\lceil d/2^s \rceil$ pagine, la page table contiene altrettante entry, ciascuna da f bit, e gli indici della tabella sono compresi tra 0 e $\lceil d/2^s \rceil - 1$.

  Siccome la dimensione della page table è variabile, essa non può essere contenuta direttamente nel PCB di ciascun processo (poiché è utile che tutti i PCB abbiano la stessa dimensione), e si trova invece in un'altra area di memoria, verso la quale il PCB ha un puntatore.

- Il SO deve tener traccia di quali page frame sono liberi, in modo da poterli allocare ai processi. A tale scopo, tali frame vengono solitamente organizzati in un'apposita lista, chiamata **free frame list**:
  - la dimensione della lista varia con l'allocazione e la terminazione dei processi;
  - quando viene allocato un processo, si possono semplicemente utilizzare i primi frame nella lista, senza bisogno di ricerche, dato che ciascuna pagina può essere assegnata a un frame qualsiasi.

Sia la page table di ciascun processo che la free frame list sono strutture dati di proprietà del SO, ovvero situate nella kernel area (quindi i processi non vi possono accedere).

1.2 Traduzione degli indirizzi

Per convertire un indirizzo logico (di $l + s$ bit) in un indirizzo fisico (di $f + s$ bit), la MMU:

1. estrae dall'indirizzo logico il numero di pagina, che è costituito dagli l bit più significativi;

2. ottiene il numero del frame corrispondente, usando il numero di pagina come indice per accedere alla page table;
3. nell'indirizzo, sostituisce gli l bit del numero di pagina con gli f bit del numero di frame, lasciando invece invariati gli s bit meno significativi (offset).

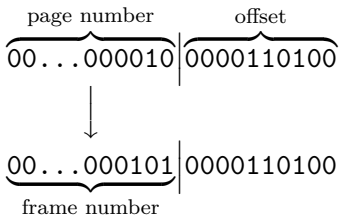
Rispetto ai calcoli che la MMU deve effettuare quando si ha allocazione non contigua con aree di dimensioni variabili, questo processo di traduzione è molto più semplice: sono sufficienti un singolo accesso alla page table (ipotizzando che essa sia situata in RAM) e uno scambio di bit.

1.3 Esempio

Siano $s = 10$ (pagine da 1 K) e $f = 22$. Un processo P di dimensione 4000 byte ha 4 pagine, quindi vengono "sprecati" $4096 - 4000 = 96$ byte. Se le sue pagine vengono allocate nei page frame 3, 20, 5, e 9, la page table per P ha la forma seguente:

00...000011
00...010100
00...000101
00...001001

Come esempio di traduzione, si considera l'indirizzo logico $2100 = 2\text{ K} + 52$: esso ha offset 52 e numero di pagina 2. La MMU accede quindi alla entry numero 2 della page table (cioè alla terza riga), dalla quale ricava che il numero del frame corrispondente è 5, quindi l'indirizzo fisico risultante è $5\text{ K} + 52 = 5172$.



2 Memoria virtuale

Intuitivamente, si parla di **memoria virtuale** quando il SO crea l'illusione di avere più memoria di quella effettiva. L'obiettivo è rendere i processo indipendenti dalla capacità di memoria del sistema.

La sua implementazione si basa sull'uso del disco. Infatti, formalmente, la memoria virtuale è la gerarchia di memoria che consiste nella memoria e nel dicso, e che consente

ai processi di operare quando è presente in memoria solo una porzione del loro spazio indirizzabile.

La memoria virtuale si basa solitamente sulla paginazione[1] (*demand paging*, cioè paginazione su domanda).

3 Demand paging

Con la tecnica di **demand paging** (*paginazione su domanda*):

- si usa la paginazione, cioè la memoria è divisa in page frame e ciascun processo è suddiviso in pagine;
- l'intero spazio logico dei processi è memorizzato su un **paging device** (corrispondente allo *swap device* nei sistemi con swap), che è un disco oppure una porzione di disco;
- l'area del paging device allocata per un singolo processo è detta **swap space** di tale processo;
- quando inizia l'esecuzione di un processo, viene allocato solo un page frame, nel quale viene caricata la pagina che contiene la prima istruzione;
- quando il processo in esecuzione cerca di accedere a una pagina alla quale non è assegnato alcun page frame, tale pagina viene copiata dallo swap space a un frame libero;
- quando un processo modifica una pagina, la copia (*immagine*) di quest'ultima nello swap space diventa obsoleta;
- se servono page frame, il SO ne può liberare alcuni, aggiornando eventuali immagini obsolete sul paging device.

Quindi, con il demand paging, ciascun processo si trova sempre interamente sul disco, e alcune sue parti possono trovarsi anche in RAM.

[1] Lo swap, usato nei sistemi senza paginazione (come, ad esempio, UNIX System V), permetteva di avere complessivamente più processi di quanti ce ne stessero in RAM, ma ciascun processo doveva essere caricato interamente in memoria per operare. Di fatto, esso era quindi una forma limitata di memoria virtuale.

3.1 Concetti fondamentali

Il demand paging si basa su alcuni concetti fondamentali:

Page fault: eccezione sollevata dalla MMU quando un processo tenta di accedere a una pagina che non è caricata in nessun page frame.

Page-in: in seguito a un page fault, il SO (per la precisione, l'handler del page fault) copia la pagina richiesta dallo swap space a un frame libero; se non esistono page frame liberi, il SO deve liberarne uno.

Page-out: quando il SO libera un page frame, se la pagina associata è stata modificata dopo il suo page-in più recente (cioè se la sua immagine sul disco è obsoleta), essa deve essere copiata nello swap space.

Page replacement: consiste nel liberare un page frame e nel successivo page-in di una pagina diversa nello stesso frame.

Osservazioni:

- Il page fault è un esempio di evoluzione dell'hardware dettata dallo sviluppo dei SO.
- I page-in e page-out costituiscono il **page I/O**, che (a differenza del *program I/O*) è trasparente al programma, cioè non è controllabile dal programmatore. Infatti, con il demand paging, qualunque istruzione che (apparentemente, per come ragiona il programmatore) opera su dati situati in memoria (ad esempio, una semplice addizione tra un registro e una variabile) può in realtà comportare degli accessi al disco.
- La movimentazione delle pagine può causare un rallentamento del sistema:
  - i processi che generano page fault vengono messi in waiting (mentre viene eseguito il caricamento dal disco delle pagine richieste), quindi la loro velocità di avanzamento risulta degradata;
  - le letture/scritture su disco dovute al page I/O danno a loro volta luogo a degli I/O interrupt, che degradano anche la velocità di avanzamento di altri processi (cioè non solo di quelli a cui appartengono le pagine coinvolte).

3.2 Contenuti della page table

Per implementare il demand paging, è necessario che ciascuna entry della page table contenga più informazioni:

Validity bit: vale 1 se la pagina è caricata in un page frame, e 0 altrimenti.

Frame number: il numero del page frame associato alla pagina; è significativo solo se *validity bit* = 1.

Disk address: indirizzo della pagina sul paging device.

Modified bit (mod): vale 1 se la pagina è stata modificata dopo l'ultimo page-in, cioè se la copia nello swap space è obsoleta (e, pertanto, sarà necessario il page-out quando il frame verrà liberato), altrimenti vale 0.

Reference info (ref): informazione sugli accessi recenti alla pagina, che è utile per scegliere quali pagine liberare.

Protection info (prot): permessi per accedere alla pagina in lettura/scrittura.

3.3 Traduzione degli indirizzi

La procedura di traduzione da indirizzo logico a indirizzo fisico, effettuata dalla MMU, è simile a quella che si ha con la paginazione "normale" (senza memoria virtuale), ma in più deve gestire la possibilità che la pagina richiesta non sia in memoria, e debba quindi essere caricata dal paging device:

1. Gli l bit più significativi determinano la entry della page table da considerare.
2. Se *validity bit* = 0, viene sollevato un page fault:
   a) il page fault handler usa il valore del campo *disk address* per individuare il blocco del paging device da leggere, e accede alla free frame list per assegnare un frame libero alla pagina;
   b) dopo aver copiato la pagina dal paging device nel page frame, il page fault handler imposta il *validity bit* a 1, e aggiorna il campo *frame number*.

   Dopo la gestione del page fault, si riprende l'esecuzione della procedura di traduzione (dal punto 3).
3. Se *validity bit* = 1, l'indirizzo fisico viene determinato concatenando gli f bit del *frame number* con gli s bit meno significativi dell'indirizzo logico (cioè l'offset).

4 Traduzione con uso del TLB

Se la MMU effettuasse la traduzione degli indirizzi usando direttamente la page table, per ogni accesso alla memoria servirebbero in realtà due cicli di memoria (quando la pagina richiesta è già caricata in un frame):

1. il primo ciclo per leggere una entry della page table;
2. il secondo ciclo per l'accesso vero e proprio.

In questo modo, la traduzione avrebbe quindi un costo elevato.

Per risparmiare uno dei due cicli, la MMU può usare un dispositivo hardware chiamato **Translation Look-aside Buffer** (**TLB**). Esso è una **memoria associativa** contenente coppie (page number, frame number): per accedervi, si specifica un page number (al posto di un indirizzo, come invece avviene per le memorie "normali"), e, se esso è presente in una delle coppie, l'hardware restituisce il frame number corrispondente.

Le memorie associative sono molto veloci, ma anche costose, e quindi piccole. Perciò, il TLB contiene le coppie (page number, frame number) solo per alcune delle pagine caricate in RAM.

Per effettuare la traduzione, dato un indirizzo logico (composto da page number e offset), la MMU accede al TLB:

- Se nel TLB è presente la coppia (page number, frame number) per il page number considerato, la MMU utilizza il frame number così ricavato per costruire direttamente l'indirizzo fisico.
- Se, invece, tale coppia non è presente nel TLB, si ha un **TLB miss**, e la MMU deve accedere alla page table in memoria per effettuare la traduzione (come descritto in precedenza).

  In questo caso, la coppia (page number, frame number) ricavata dalla page table viene aggiunta al TLB, sostituendo eventualmente una coppia esistente, perché è molto probabile che la stessa pagina venga richiesta nuovamente in futuro.

Complessivamente, la traduzione di un indirizzo prevede tre casi:

1. Viene usata la entry del TLB dedicata alla pagina, se esiste.
2. Viene sfruttata la entry relativa alla pagina nella page table, se tale entry ha *validity bit* $= 1$ e il caso 1 non è applicabile. Si aggiorna poi il TLB, inserendo una coppia corrispondente a questa pagina.
3. Si genera un page fault, se i casi 1 e 2 non sono applicabili. Dopo aver caricato in RAM la pagina, vengono aggiornati la page table e il TLB.

Osservazione: Quando avviene un context switch, il TLB deve essere azzerato, per evitare che il nuovo processo schedulato acceda alla memoria del processo che ha perso il processore. Di conseguenza, il processo schedulato avrà il TLB vuoto: questo è un altro elemento che contribuisce all'overhead dei context switch.

5 Page Table Address Register

La MMU deve conoscere l'indirizzo di memoria della page table del processo corrente. A tale scopo, l'architettura può mettere a disposizione, per esempio, un apposito registro, chiamato **Page Table Address Register** (**PTAR**):

- il valore da assegnare a esso è memorizzato nel PCB di ciascun processo, e l'assegnamento viene effettuato dal SO durante il context switch;
- quando la MMU deve accedere alla page table, il valore del PTAR viene sommato al numero di pagina (moltiplicato per la dimensione di una singola entry della tabella) per ottenere l'indirizzo della entry richiesta.