Tabella dei frame

Il SO tiene conto dello stato di occupazione dei frame attraverso la tabella dei frame. Questa è una tabella unica che prevede una voce per ogni possibile frame ed ha quindi una dimensione direttamente proporzionale a quella della RAM. Il generico record si riferisce ad un singolo frame di memoria fisica e contiene informazioni circa lo **stato** libero/occupato di un frame e se occupato tiene conto di **quale processo** lo occupa. Questa tabella viene consultata ogni volta che viene caricato un processo in memoria per creare la relativa tabella delle pagine per quel processo.

Progettazione di una tabella delle pagine

Ogni volta che faccio una traduzione devo fare un look-up alla tabella delle pagine, questo ha un costo tangibile. Quando si progetta la tabella delle pagine si devono curare alcuni aspetti, come la velocità nella consultazione e nella dimensione.

Nei vecchi approcci, le dimensioni di queste tabelle erano ridotte e quindi si trovavano già all'interno dell'MMU, che vi faceva accesso in maniera veloce. Nell'ambito del context-switch tra processi differenti si deve tenere conto, nel riprogrammare l'MMU che si dovrebbe ricaricare la tabella delle pagine dell' altro processo e quindi questo approccio comportava un grosso overhead. Inoltre questo approccio non è scalabile alle dimensioni cui fanno riferimento le esigenze moderne.

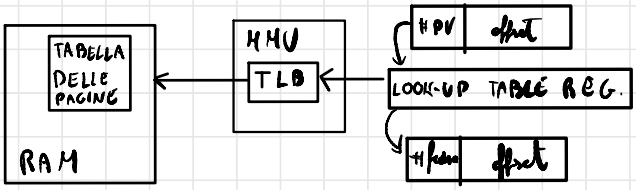
Un approccio moderno prevede che la tabella delle pagine venga caricata in RAM in un segmento di memoria contiguo che permetta di non frammentarla. Questo approccio tiene l'intera tabella in RAM e non lascia nulla nell'MMU se non un registro particolare **PTBR**(page-table based register) il quale indica all'MMU l'indirizzo base della tabella delle pagine del processo attivo, ed il SO deve mantenerlo coerente allo stato dello spazio di indirizzamento virtuale. Il PTBR rende molto veloce ed efficiente l'accesso alla tabella in caso di context-switch da un processo ad un altro. Un overhead aggiuntivo è comportato dal fatto che in una reale traduzione viene effettuato un primo accesso alla RAM per ottenere informazioni sulla tabella delle pagine corretta da consultare (attraverso il PTRB) ed un altro accesso alla RAM dovuto alla traduzione effettiva dell'indirizzo. Questo overhead dimezza la banda in quanto circa la metà degli accessi fatti avvengono per capire dove fare l'altra metà degli accessi.

Memoria associativa o TLB (translation lookaside buffer)

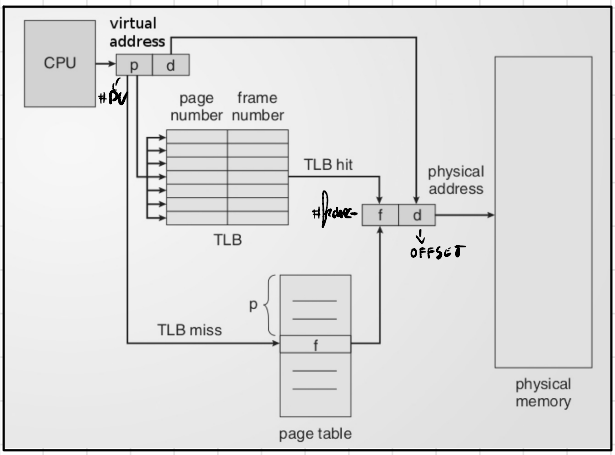
Non si è costretti a pagare un accesso aggiuntivo per ogni volta che si vuole prelevare un indirizzo dalla RAM, se si fa uso di un meccanismo di memoria associativa o TLB. La TLB è una componente integrata dell'MMU, composta da **un set di registri che consentono un accesso molto veloce** a fronte di un costo pagato dall'hardware (MMU) che effettua una ricerca parallelizzata, la quale non pesa in maniera proporzionale al numero di voci che ha la TLB.

L'**osservazione** di base è che un processo usa un gran numero di riferimenti ad un piccolo numero di pagine (principio di localizzazione del codice). La generica voce di una TLB è proprio una pagina, al pari della tabella delle pagine, in particolare tiene alcune voci della tabella delle pagine che si trova in RAM, solo che la TLB si trova nell'MMU. L'idea è che una voce usata di recente viene caricata nella TLB in modo da potervi accedere senza fare accesso in RAM e quindi si risparmia l'accesso ad una memoria più lenta.

Quando si effettua una ricerca (parallelizzata in hardware) nella TLB essa restituisce velocemente il record cercato se esso risulta presente (**TLB hit**) e mi azzera il costo di accesso alla RAM . Nel caso in cui la voce non venga trovata nella TLB si ha un evento di **TLB miss**, che viene gestito caricando nella TLB il PTBR relativo alla tabella delle pagine desiderata. Le voci della TLB vengono aggiornate in modo **LRU**(last recent used) ovvero scartano le voci utilizzate meno di recente. Quando nel TLB viene caricata la tabella delle pagine dalla RAM, ne beneficeranno pure tutti gli accessi successivi alle altre voci della stessa tabella delle pagine, rendendo più raro l'evento di cache miss rispetto a quello di cache hit.



Necessariamente i bit di protezione e il dirty bit devono essere riportati anche nei record presenti nella TLB. Non sono presenti i bit di presenza e di referenziamento, in quanto un record che si trova in TLB è un record a cui si è stato fatto accesso di recente e di conseguenza già presente in RAM.



TLB flush e ASID (adress-space identifiers)

Il context-switch deve tenere conto di come funziona la TLB. Durante il cambio di contesto devo riprogrammare l'MMU e devo aggiornare il PTBR e si necessita di effettuare il TLB flush, ovvero si deve svuotare la TLB per garantire che il nuovo processo non faccia accesso ad indirizzi fisici di un altro processo, anche se questo causerebbe molti eventi di cache miss per gli accessi futuri. Per fissare le idee si considera che il nuovo processo faccia accesso all' indirizzo virtuale 1000, in TLB si trova già un indirizzo 1000 ma facente parte dello spazio di indirizzamento del processo precedente e che quindi verrebbe tradotto nello stesso indirizzo fisico .

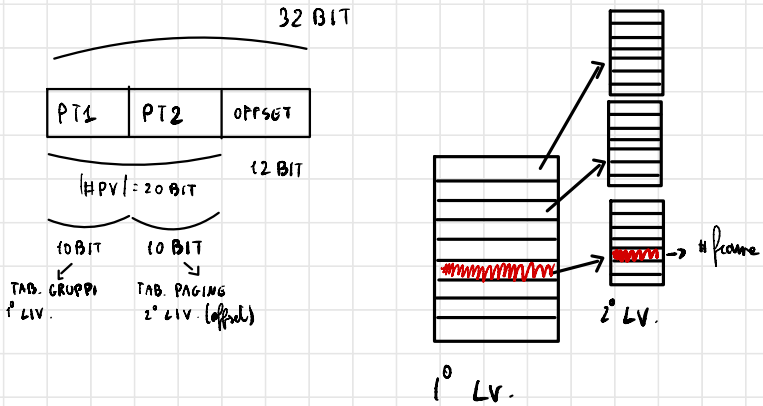
Per evitare di dover svuotare la TLB ad ogni context-switch (che causerebbe troppi eventi di cache miss) si può pensare di arricchire le voci della TLB con un campo address-space identifiers che tiene conto dello spazio di indirizzamento (per esempio l'ID del processo) a cui appartiene la determinata pagina nella TLB. Il PID viene utilizzato come chiave di ricerca insieme al numero della pagina (PID, #PV) nella TLB in modo da evitare che un processo faccia accesso ad indirizzi dedicati ad altri processi.

Tabelle delle pagine multilivello (paginazione gerarchica)

Dopo aver discusso metodologie per rendere più veloce l'accesso alle tabelle delle pagine, rimane il problema delle **dimensioni**. Si necessita, come già detto, di trovare uno spazio contiguo in RAM per allocare le tabelle delle pagine. Nell'ipotesi di uno spazio di indirizzamento virtuali a 64bit si hanno 2^52 pagine da 4KB e quindi memorizzare le intere tabelle in RAM sarebbe un'idea improbabile da realizzare.

L'idea è quella di raggruppare ogni gruppo di pagine nella tabella delle pagine in gruppi da 1024 pagine. Con un milione di pagine, raggruppando in gruppi da 1024 pagine , si ottengono 1024 gruppi da 2^10 (1024) pagine. La struttura viene organizzata come un albero di due livelli , dove la radice è la tabella dei gruppi ed ogni voce di questa tabella è un gruppo che ha come figlio una tabella di secondo livello che contiene tutte le pagine di quel determinato gruppo

Dato il numero di pagina virtuale si vuole trovare la relativa voce : si suddivide il numero di pagina virtuale (numerati da 0 a 2^10 -1) in due parti PT1 e PT2, in modo da ottenere l'identificativo del gruppo e la relativa pagina all'interno del gruppo. Viene diviso il numero di pagina virtuale per la dimensione (fissa) dei gruppi e si ottiene un quoziente PT1 che rappresenta l'identificativo del gruppo, ovvero l'indirizzo a partire dal quale si sviluppa la tabella di primo livello ed un resto PT2 (altro offset) che permette di trovare l'identificativo dell'indirizzo a partire dal quale si sviluppa la tabella di secondo livello desiderata, per ottenere il numero di frame dell'indirizzo fisico al quale si va a sommare l'altro offset.



L'implementazione dell'albero può avvenire pure su più livelli ma si tiene conto che avere più livelli implica un maggiorn numero di accessi. Il numero dei livelli nei calcolatori reali si aggira attorno ai 4 o 5.

Il grosso vantaggio di questo modello multilivello per rappresentare l'informazione è data dal fatto che non tutte le foglie dell'albero sono piene , ovvero si può pensare che una foglia dell'albero rappresenti l'heap, uno il codice, uno lo stack ma tutte le altre locazioni di memoria della RAM non sono allocate (vedono il loro bit di validità a 0) e quindi non sono utilizzabili concretamente dai processi, per tanto non vengono rappresentate. Questo rappresenta un grosso risparmio in memoria.

Tabella a pagine invertita

Un'approccio differente prevede la presenza di una struttura differente da quella della tabella delle pagine. L'MMU deve essere a conoscenza di come è organizzata la struttura e quindi la scelta del modo di organizzare i dati non è a carico del SO ma dell'hardware. Un'approccio a tabella invertita prevede che si ha con un unica tabella lo spazio di indirizzamento, ovvero una sorta di generalizzazione della tabella dei frame. Si ha un'unica tabella con una voce per ogni frame fisico e non più una per ogni processo. L'idea è di mettere in ogni record informazioni che riguardano non solo lo stato di allocazione del frame ma anche altre informazioni come il PID (ASID) del processo per il quale ho allocato il frame, specifico anche il numero di pagina virtuale e di quale è il numero di frame. La ricerca all'interno della tabella invertita avviene ricercando il PID del processo e il numero di pagina (PID, #PV) nella struttura e se non si ha riscontro si ha un page fault, ovvero nessun frame ospita quella pagina. Se trovo un riscontro riesco facilmente a fare la traduzione.

Con una descrizione di questo tipo il punto carente sono le prestazioni del sistema, ovvero si è costretti a fare una ricerca lineare. La soluzione è utilizzare una **tabella Hash** per abbattere il costo di ricerca (diventa costante) . Si utilizza la coppia (PID, #PV) come chiave di ricerca e la funzione Hash restituisce in tempo costante l'identificativo dello slot che contiene il numero dello slot nella tabella invertita. Questo approccio rimane pienamente compatibile con la TLB.

La dimensione della tabella invertita è direttamente proporzionale alla memoria fisica del dispositivo (avendo una voce per ogni frame) ed è stata pensata per le architetture a 64bit , per le quali si necessita di essere slegati dalla dimensione dell'astrazione dello spazio virtuale. I record della tabella non contengono il bit di validità in quanto le pagine allocate in memoria RAM sono le uniche a risultare presenti nella tabella invertita. Questa tabella è unica e supplisce alla necessità di traduzione nel limite in cui non si verifichi un evento di page fault. Nel caso di page fault si necessita di avere a disposizione le n tabelle delle pagine, le quali verranno memorizzate sul disco e quindi si necessita di un accesso lento al disco ma solo in caso di page fault.