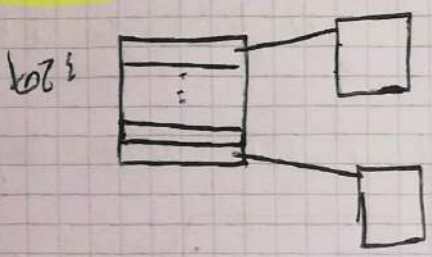
Problema di una tabella

Abbiamo una tabella più o meno grossa che vuole memoria contigua, bisogna trovare la soluzione.
Non manteniamo l'intera tabella in memoria, creiamo le tabelle multilivello che quindi fanno paginazione gerarchica.



Creiamo la page table 1 che sostanzialmente è un gruppo di tabelle (le page table 2)

Un indirizzo sarà di questa forma

PT1	PT2	Offset

I bit in base a come saranno distribuiti saranno usati per la traduzione.

Vantaggio: E' una struttura dinamica, le porzioni possono stare sparpagliate nella memoria invece che stare tutte contigue. In oltre le tabelle 2 non è detto che siano tutte in memoria
Dunque non abbiamo vincoli sulla contiguità e risparmia memoria (se ci sono 1024 gruppi non è detto siano allocati tutti)

Svantaggio: La struttura a 2 livelli potrebbe comunque addirittura 3 accessi alla memoria
La TLB è applicabile anche qui e assume valore ancora più elevato (fa risparmiare 2 accessi)
La gerarchia si può aumentare per fare ancora più livelli (ricorsione)
Il bilancio tra PT1 e PT2 può essere direttamente progettato. Potrei usare un tot di bit per PT1 e un altro tot per PT2

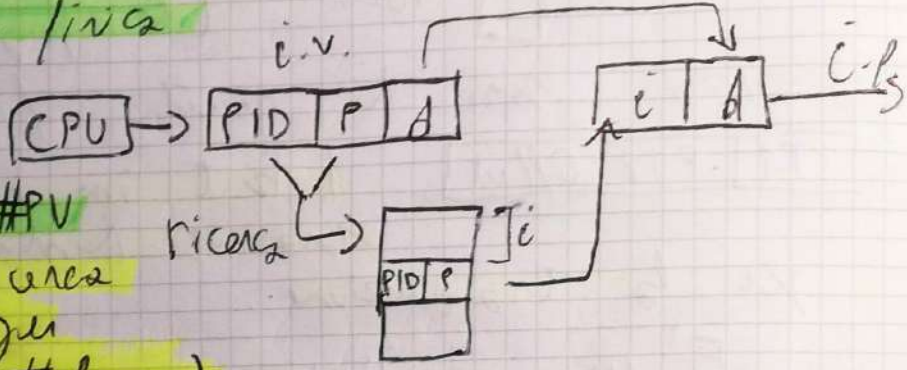
Sistema alternativo al multilivello

Tabella delle pagine invertita

Ce n'è 1 unica, non 1 per ogni processo
Abbiamo una voce per frame fisico, ogni voce ha
- id processo
- pagina virtuale

Il numero di frame non serve perché sarebbe l'indice della tabella
La tabella si comporta come se fosse una rappresentazione della memoria fisica

Traduzione

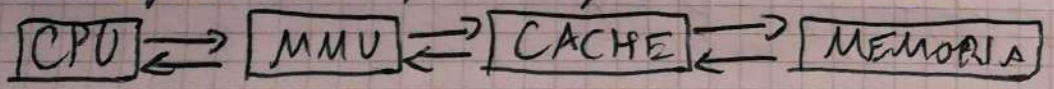


Tramite PID e #PV è fatta la ricerca nella tabella per trovare i (#frame)
che unito con d (offset) ci dà l'i.f.
Se la ricerca è senza successo c'è un page fault
La ricerca può essere lenta, si usa quindi il concetto di Hashing su PID e #PV, in più per velocizzare ancora di più usiamo la TLB

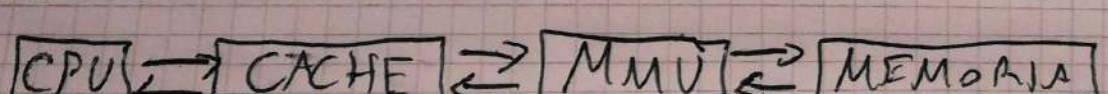
Servono ancora le tabelle classiche (per processo)?
SI, dato che la tabella invertita fa riferimento solo alle pagine in RAM, queste tabelle ci servono per gestire i page fault.

CACHE MEMORIA VS MEMORIA VIRTUALE

La cache della memoria può essere posta o prima o dopo tradotta la MMU

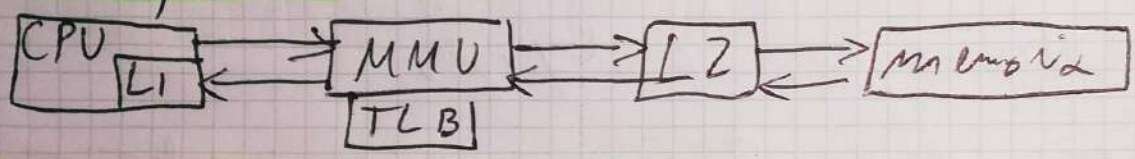


1°



2°

Nel 1° caso diciamo che i bordi sugli indirizzi fisici e appunto i numeri.
Il caching è poco efficace e l'MMU potrebbe do collo di bottiglia.
Permette però di non fare azzeramenti in caso di context-switch.
Nel 2° caso diciamo che i bordi sugli indirizzi virtuali.
L'efficacia nel caching è maggiore
Sono però costretto ad usare gli ASID per evitare gli azzeramenti. Se no una L2, vola male.
Nella pratica



Algoritmi sostituzione pagine

In caso di page fault e assenza di frame liberi dobbiamo fare una sostituzione e bisogna scegliere tramite appunto un algoritmo quale pagina vittima
La scelta è fatta con una gestione simile a quella della cache ponendoci l'obiettivo di evitare page fault in futuro il più possibile

La soluzione ottimale OPT

È una soluzione ottimale ma solo teorica poiché non implementabile.
Per implementarla dovremmo conoscere a priori quando una pagina farà page fault
Nell'algoritmo scegliere la pagina che verrà usata nel futuro più lontano.
Ci permette di avere un termine di paragone

NRU : not recently used

Usano statistiche provenienti dal bit di referenziamento e bit di modifica che sono gestiti dall'HW e azzerati a brevi dal SO
L'algoritmo intuitivamente vuole mantenere la pagina non usata di recente

L'algoritmo distingue 4 classi di pagine
- 0 : non referenziato, non modificato
- 1 : non referenziato, modificato
- 2 : referenziato, non modificato
- 3 : referenziato, modificato

L'algoritmo segue 2 passi: prima seleziona la classe non vuota di classe più bassa poi uniformemente seleziona una pagina della classe in qualche altro modo.
L'algoritmo potrebbe lavorare in modo globale oppure guardare solo le pagine del processo.
La selezione di una pagina in una classe è
- a caso
- la prima trovata

oppure tramite altri algoritmi.

FIFO

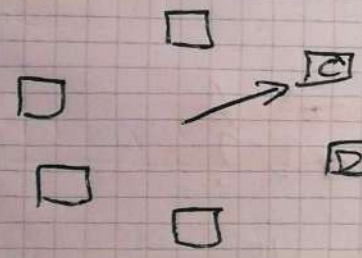
- rimuove la ~~classe~~ pagina più vecchia
- potrebbe non essere un'idea molto buona dato che una pagina potrebbe infatti essere vecchia ma allo stesso tempo essere molto usata.

Seconda Chance

Come FIFO, si butta via la pagina più vecchia che ha però il bit R a "non referenziato"
Dunque controlliamo la testa della coda, se R=0 la buttiamo via per la condizione, se R=1 la mettiamo in fondo alla coda e ripetiamo.
Di solito quando becchiamo R=1 diamo prima di mettere la pagina in fondo mettere R a 0 per evitare di ripassare la pagina troppe volte

Algoritmo clock

Implementazione della seconda chance più efficiente. Viene organizzata una coda circolare



Puntatore esterno che punta alla "testa"
Nel momento in cui C è buttata, poi, D diventa la testa.
D diventa la testa anche se C era stato esaminato e risparmiato.

ALgoritmo LRU

Idea: Probabilmente le pagine più usate di recente le useremo anche in futuro (prossimo), dunque rimuoviamo le pagine meno usate di recente
LRU: buona idea, ma è difficilmente implementabile nella sua forma rigorosa.
Ci può un supporto HW: si tiene un contatore nella CPU e altri campi relativi nella tabella delle pagine
A questo punto l'LRU sarebbe implementabile però comporta dover fare troppi accessi in RAM

Per evitare gli accessi eccessivi a lo strumento usando delle matrici di bit che tengono le informazioni sull'uso recente delle frame.

Se ho 4 frame la matrice sarà 4 × 4, avrà le righe 0, 1, 2 e 3.

Il numero di bit a 1 ~~nella~~ in una riga indica quanto è stata usata recentemente la frame relativa a quella riga

Funzionamento:

Inizialmente la matrice è piena di zeri.

Quando una frame i è usata, la riga i-esima è riempita di 1 e la colonna i-esima è riempita di 0

Quando devo rimpiazzare una pagina/frame da sostituire in un istante di tempo?

2 approcci < frame con più 0
frame con numero più alto
(i vedo la riga come un numero)

Nella pratica i 2 approcci sono equivalenti

Ci sarà sempre una riga con 3 1 (tranne all'istante 0) e mai posso avere 2 righe contemporaneamente con 3 1.

Il peso di una pagina è alleggerito quando un'altra pagina viene usata.

La matrice potrebbe essere dentro all'MMU, ma è proporzionale al numero di frame.

La matrice perciò ci evita di avere un timestamp/contatore delle pagine ed evitare accessi I/O in RAM