Gestione della memoria parte 2

Contiguos Allocation

- E' lo schema più semplice di allocazione in memoria.
- Allocazione contigua di memoria vuol dire che la memoria viene partizionata in intervalli di indirizzi dei quali uno viene assegnato al sistema operativo e gli altri a processi utente contemporaneamente attivi (significa non tutti in esecuzione parallasse, ma pronti per l'esecuzione. Lo Scheduling della cpu prevede che i processi siano in memoria e poi lui decide quali eseguire e quando).
  - Più processi condividono la RAM, ad ognuno è assegnato un intervallo di indirizzi (PARTIZIONI).
  - Una partizione al SO (high Memory), e un'altra (con un sotto-partizionamento interno) a più processi (low Memory).
- Sono necessari meccanismi di rilocazione: cioè è necessario che il processo possa non conoscere all'inizio gli indirizzi a cui sarà collocato (quindi gli indirizzi dei salti, delle chiamate a funzione), ma possa essere spostato a una parte di memoria che dipende dall'esecuzione.
  - Meccanismo (hardware o software) necessario per proteggere i processi tra di loro, e il sistema operativo da cambiamenti di codice o dati.

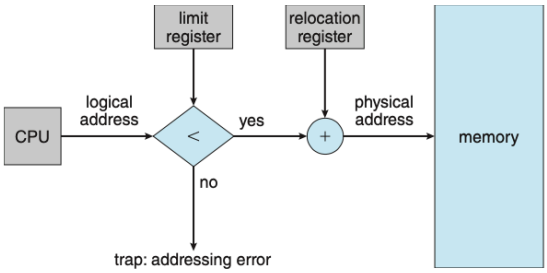


Figure 9.6 Hardware support for relocation and limit registers.

  - Meccanismo gestito dinamicamente dalla MMU implementato grazie a base register (relocation register) e limit register (ci dice quanto è grande l'intervallo di indirizzi in memoria dedicati al processo).
  - Due operazioni come in figura: verifica di correttezza (nel blocco condizionale e gestione dell'errore) + traduzione indirizzo (nel sommatore)

Variable Partition

- Processi allocati e deallocati dinamicamente. Questa fase in cui si determina il grado di multi-programmazione, cioè quanti processi possono essere contemporaneamente attivi (cioè già presi in considerazione dallo scheduling, pronti per essere eseguiti e quindi in memoria), è limitato dal numero di partizioni disponibili.

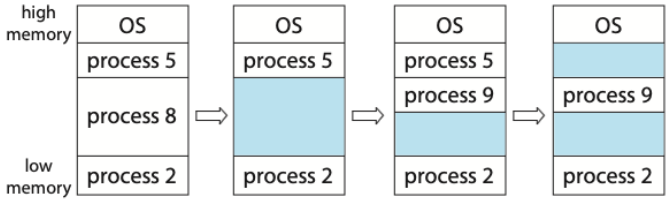


Figure 9.7 Variable partition.

- La dimensione delle partizioni è variabile (), per questioni di efficienza. Il che comporta che quando un processo rilascia una partizione viene rimpiazzato da uno che richiede meno memoria. Si crea un buco (hole), cioè blocchi di memoria disponibili.
- Il SO tiene informazioni riguardo alle partizioni allocate ed alle partizioni libere (hole)

Strategie di allocazione

- PREMESSA: per far partire un processo si deve sapere di quanta memoria ha bisogno.
- First-fit: Nel caso di più buchi in memoria prendo il primo che sia grande abbastanza per il processo.
  - Vantaggio: ottimale in termini di velocità.
  - Svantaggio: non ottimale in termini di residui di frammentazione che lascerà vuoti.
- Best-fit: viene scelta la più piccola partizione libera grande abbastanza per contenere ciocche serve.
  - Vantaggio: trova la migliore partizione, cioè lascia vuoto (e quindi inutilizzato) un residuo di partizione il più piccolo possibile.
  - Svantaggio: perde più tempo, se non ho i blocchi ordinati per grandezza.
- Worst-fit: diametralmente opposto al best-fit. Tra tutte le partizioni libere prendo quello più grande.
  - Vantaggio: in determinati casi utile, perché lascia il più grande residuo possibile (riutilizzabile??)
  - Svantaggio: perde più tempo, se non ho i blocchi ordinati per grandezza.
- A prima vista first-fit e best-fit sembrano migliori in termini di velocità e utilizzo della memoria. Tuttavia, per un'esatta valutazione è necessaria un'attività di benchmarking.

Frammentazione

- PROBLEMA: nel tempo vengono creati e distrutti processi che occupano e rilasciano partizioni di memoria di dimensione variabile, creando **alternanze di pieno/vuoto non uniformi**.
- External Fragmentation: nel senso che i buchi di memoria libera stanno fuori rispetto alla memoria allocata ai processi. Cioè quando esiste nella memoria lo spazio totale per soddisfare una richiesta di allocazione, ma questo spazio non è contiguo.
- Internal Fragmentation: quando la memoria allocata potrebbe essere leggermente più grande di quella richiesta. Quella porzione di memoria assegnata in più viene detta frammentazione interna.
- 50-percent rule: Considerando la frammentazione esterna, analisi statistiche sulla strategia first-fit ci dicono che con comportamenti ragionevolmente realistici, si può concludere che dati N blocchi allocati, 0.5*N saranno persi per frammentazione. Cioè, in maniera molto approssimativa, 1/3 della memoria potrebbe essere inutilizzabile.
- La frammentazione ha due fondamentali problemi:
  1. **PROBLEMA**: una parte della memoria è inutilizzabile perché composta da tanti piccoli frammenti che potrebbero essere troppo piccoli per soddisfare richieste di allocazione. Vuole dire che è quindi necessario compattare (compaction - deframmentazione) i frammenti.

SOLUZIONE: Cioè è opportuno spostare i processi già allocati in memoria per far sì che tutti i piccoli pezzetti di memoria libera siano posizionati insieme in un unico grande blocco contiguo. Ciò può avvenire solo se la rilocazione è dinamica (cioè fatta non solo al load del processo, ma deve essere possibile mentre il processo è in esecuzione (cioè è in stato di ready). Processo in esecuzione non significa mentre la CPU sta eseguendo, ma, ricordando lo scheduling dei processi, sappiamo che in esecuzione vuol dire quando è in memoria e può essere nei seguenti stati: new, ready, running, waiting, terminated)

2. Dalla compaction deriva un **altro PROBLEMA** di I/O: Consideriamo un processo che è in stato di waiting e sta aspettando il completamento di un I/O (cioè una struttura dati del processo è in quel momento sorgente o destinazione di una operazione di I/O). Quindi Se si fa un'operazione di rilocazione mentre è in corso un I/O la lettura o scrittura di dati potrebbe avere problemi.
   - SOLUZIONI per problema I/O:
     1. Bloccare lo spostamento mentre il processo è in waiting e coinvolto nell'operazione di I/O;
     2. Evitare il problema alla fonte: cioè fare in modo che il passaggio di una struttura dati di un processo a una struttura di I/O avvenga in un'area dati di kernel. (I cosiddetti buffer kernel)

- Il problema della frammentazione è anche un problema del backing-store (che vedremo più avanti), che riguarda dischi e/o file System, ovvero qualunque cosa in cui sia necessario allocare e deallocare aree contigue di dimensione variabile.

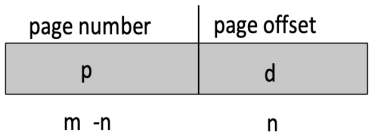
Paging

- Paginazione: vera soluzione al problema della frammentazione. Poiché compattare è costoso! Ma anche altri vantaggi.
- Lo **spazio di indirizzi fisici di un processo può essere non contiguo**, cioè il processo è allocato in maniera spezzettata nelle cosiddette frame.
  - Non ci sarà più frammentazione esterna
  - Non ci sarà più il problema delle dimensioni variabili delle partizioni allocate.
- La memoria fisica è divisa in blocchi di dimensione fissa chiamati **frames**: ogni frame ha dimensione pari a una potenza di due (da 512 B a 16MB, di solito intorno al kB). Memoria fisica (RAM) vista come un vettore di frame in cui stanno le pagine.
- La memoria logica (cioè di quanto ha bisogno un processo per essere eseguito, cioè quanto deve essere caricato in RAM) è divisa in blocchi della stessa dimensione di un frame chiamati **pages**. Memoria logica vista come vettore di pagine.
  - Lo spazio di indirizzamento logico resta contiguo. Solo che non è più mappato come prima, attraverso una somma nello spazio di indirizzamento fisico che era anch'esso contiguo (*vedi punti successivi)*.
- Con questa tecnica invece, ogni pagina verrà allocata in un frame della memoria fisica. Per fare ciò bisogna **tenere traccia dei frame liber**i.
- Per eseguire un programma di N pagine, ci vorranno N frame liberi.

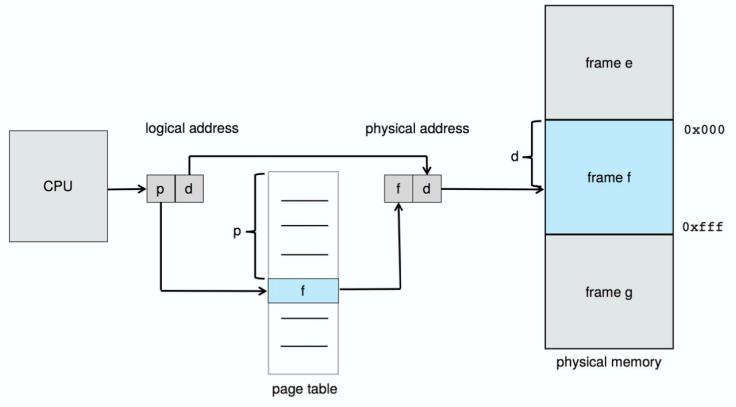
- La traduzione da logico a fisico non avverrà più con una somma, ma sarà necessaria una **page table**.
- Il problema della frammentazione interna sarà ancora rilevante. Perché come dimensione complessiva di RAM sarà sempre allocato un multiplo di un frame.

Address Translation Scheme

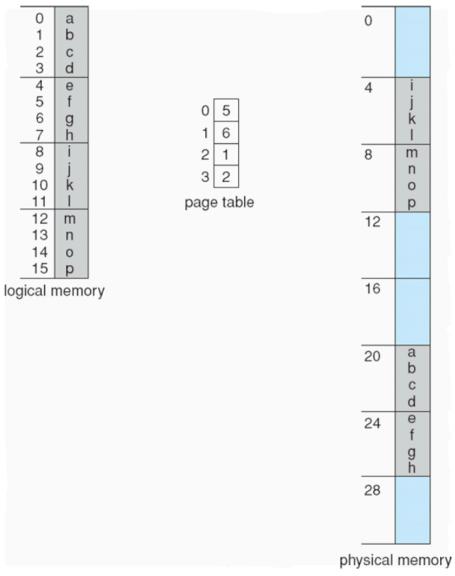
- Gli indirizzi (logici) generati dalla CPU sono divisi in:
  - **Page number (p)** : usato come indice per la **page table** che contiene il base address (indirizzo base o iniziale) di ogni frame in memoria fisica.
  - **Page offset (d)**: che combinato con il base address definisce l'indirizzo fisico che è inviato alla RAM. (d= displacement)



- For given logical address space 2^m and page size 2^n



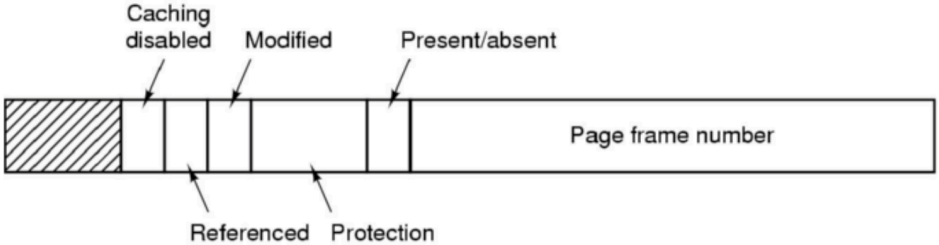
- Logical address: n = 2 and m = 4. Using a page size of 4 bytes and a physical memory of 32 bytes (8 pages)



Paging -- Calculating internal fragmentation

- Page size = 2,048 bytes
- Process size = 72,766 bytes
- 35 pages + 1,086 bytes
- Internal fragmentation of 2,048 - 1,086 = 962 bytes

- Se di un processo si conosce esattamente la dimensione di cui ha bisogno per l'esecuzione, si sa anche qual è la sua frammentazione interna. Cioè lo spazio inutilizzato dell'ultima pagina. (FRAMMENTAZIONE solo nell'ultima pagina)
- Worst case: frammentazione interna uguale a 1 frame - 1 byte. (Cioè l'ultimo frame è stato allocato solo perché serviva 1 byte)
- Best case: frammentazione interna uguale a 0.
- Average case: frammentazione interna = 0,5 frame
- Dal punto di vista della frammentazione (interna) è meglio avere frame e pagine piccole, poiché nel caso medio si spreca mezzo frame. Quindi più piccolo è, meglio è.
- Dal punto di vista della dimensione della page table, se diminuisce la dimensione del frame (e quindi della pagina perché sono uguali) aumenta la dimensione della page table.
- STRUTTURA DI UNA ENTRY DELLA PAGE TABLE
  - page frame number: info più importante.
  - Present/absent:
  - Protection: significa una certa pagina può essere solo in lettura (o scrittura o entrambi)
  - Caching disabled:
  - Modified: questa pagina è stata modificata rispetto all'ultima volta in cui è stata caricata
  - Referenced:



Implementation of Page Table

- La Page table è tenuta nella memoria principale (Quindi STA in RAM), perché è troppo grande per stare in CPU. Invece nell'allocazione contigua tutto il lavoro di traduzione era fatta all'interno della MMU, quindi all'interno della CPU
  - Nella CPU ci stanno dei registri che dicono come accedere a un vettore, cioè puntatore all'inizio del vettore e dimensione del vettore.
  - **Page-table base register (PTBR)**: in cui c'è l'indirizzo di RAM in cui inizia la page table
  - **Page-table length register (PTLR)**: che indica la dimensione della page table di un processo
- Questo significa che per passare da logico a fisico, e quindi per fare una lettura e scrittura in RAM, bisogna fare prima un'altra lettura in RAM. La page table dovrà stare in un'area di memoria non paginata, cioè in una struttura dati del kernel.
- PROBLEMA: Per cui, il fatto che ogni accesso a dati/istruzioni richieda due accessi, ci sarà un rallentamento della memoria di un fattore due.
- SOLUZIONE: uso un **fast-lookup hardware speciale (si sfrutta l'idea del caching)** che si trova dentro la CPU. Tale schema è chiamato **Translation look-aside bufferà** (**TLBs** o anche chiamata **memoria associativa**).
  - Si fa in modo che la traduzione logico-fisica prima di arrivare alla page table in ram, passi dal TLBs che "sta in CPU", e che permette di evitare se ci va bene l'accesso alla page table.

Translation Look-Aside Buffer

- E' un vettore compatibile con quello che può stare dentro una CPU, in cui non ci staranno molte corrispondenze pagine-frame, ma ce ne staranno quanto basta per riuscire a fare in modo che quasi sempre vada bene. (Da 64 a 1024 entries)
- Ogni riga della TLB rappresenta un match *pagina logica-frame fisico.* La dimensione della TLB sarà compatibile con la numerosità complessiva dei registri (cioè alcune decine)
- Si prova a usare la TLB, se non funziona si va alla page table in RAM
  - Il tutto funziona supponendo che gli accessi in memoria avvengano non a caso, ma con una certa ripetitività.
  - Ogni riga ha un comparatore hardware interno per evitare la scansione lineare, che è troppo costosa. Tutto questo fatto come in un circuito combinatorio, cioè non si sprecano colpi di clock, è fatto tutto in pochissimo tempo.

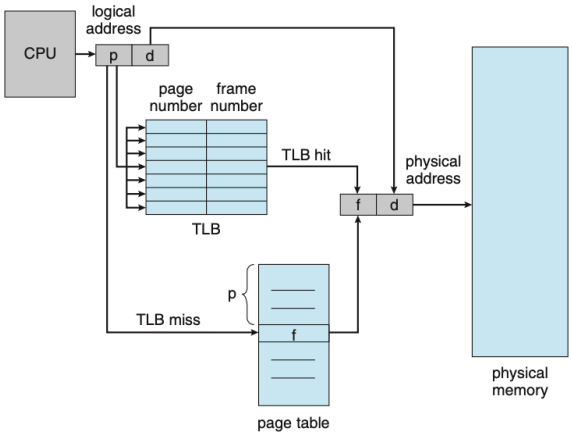


Figure 9.12 Paging hardware with TLB.