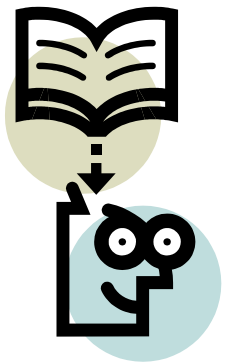
SISTEMI OPERATIVI

08.a

Gestione della memoria

Gestione della memoria

- Partizionamento
- Paginazione
- Segmentazione
- Memoria virtuale



Obiettivi

- La gestione della memoria (effettuata dal SO usando gli accorgimenti presenti nell'hardware), consiste nel:
- suddividere la memoria per consentire la presenza di più processi (presupposto indispensabile per il multitasking)
- effettuare la suddivisione della memoria in modo efficiente, per farci stare il numero maggiore possibile di processi (quanto maggiore è questo numero, tanto minore è la probabilità che la ready list sia vuota)

Funzioni di gestione

- La gestione della memoria deve garantire:
  - la rilocazione: in modo che un programma possa essere caricato ovunque in memoria
  - la protezione: del SO dai processi e tra i processi
  - la condivisione: in modo che diversi processi possano condividere codice e/o dati
  - una efficace organizzazione logica della memoria: tale da rispecchiare la struttura in moduli dei programmi
  - una efficace organizzazione fisica della memoria: gestendo con efficienza i trasferimenti da disco a memoria e viceversa

Rilocazione

- Il programmatore non sa dove il programma sarà caricato in memoria quando verrà eseguito; la posizione infatti dipende da:
  - quali e quanti moduli compongono il programma
  - quali e quanti altri programmi sono presenti in memoria nel momento in cui esso verrà caricato
- Inoltre, se la memoria occupata dal programma durante l'esecuzione dovesse servire per altri scopi
  - il programma può essere temporaneamente ricopiato su disco (swap)
  - il programma può venir successivamente collocato in una posizione diversa (rilocato) in memoria
- Gli indirizzi di memoria contenuti nel codice e nei dati di un programma sono indirizzi logici, diversi da quelli fisici

Indirizzi logici → indirizzi fisici

- Gli indirizzi logici contenuti nel codice e nei dati di un programma non individuano direttamente indirizzi fisici della memoria
- È necessaria la presenza di un meccanismo che trasformi correttamente gli indirizzi di memoria (contenuti nello spazio dei nomi) nei corrispondenti indirizzi della memoria fisica (Mapping):

$$f : N \rightarrow M$$

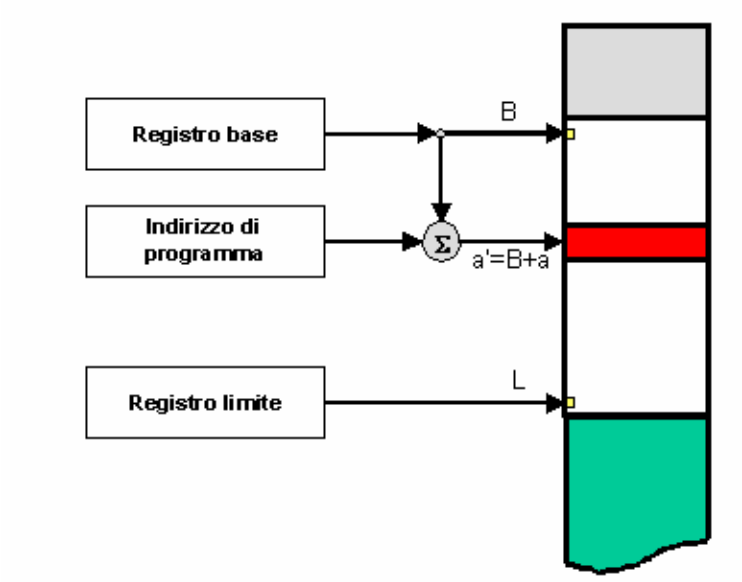
- Il meccanismo che realizza questa trasformazione degli indirizzi logici in indirizzi fisici richiede:
  - la presenza di alcune funzioni fornite dall'hardware,
  - la presenza di alcune funzioni fornite dal SO

Mapping lineare

f(a) = B + a

B indirizzo base

a indirizzo di programma



Protezione

- È necessario impedire che un processo acceda a locazioni di memoria di un altro processo (o del SO)
- La verifica non può essere fatta dal compilatore perché:
  - molti indirizzi vengono calcolati al momento della esecuzione (ad es. i metodi di indirizzamento per accedere agli elementi di un vettore)
  - i processi possono essere rilocati da una posizione di memoria ad un'altra
- La verifica va fatta al momento della esecuzione
- Le funzioni necessarie a questa verifica devono essere fornite dall'hardware.

Condivisione

- Condivisione di dati
  È necessario consentire che processi diversi accedano ad aree di memoria comuni (ad es. dati comuni di processi cooperanti che fanno parte del medesimo programma);
- Condivisione di codice
  nel caso che un insieme di subroutine sia utilizzato da più processi, conviene che di esse vi sia in memoria un'unica copia accessibile a tutti, piuttosto che averne tante copie replicate per ciascun processo.

Organizzazione logica

- I programmi sono costituiti da moduli, che vengono scritti e compilati separatamente
- Il modo più naturale di vedere la memoria per il programmatore non è uno spazio di indirizzi lineare (da 0 fino all'indirizzo massimo), ma costituito da spazi di indirizzi separati (uno per ciascun modulo)
- Se il SO e l'hardware consentono di gestire in forma di moduli i programmi e i dati, diventa facile:
  - associare ai diversi moduli diversi livelli di protezione (read-only, execute-only)
  - organizzare la condivisione di moduli
- La tecnica più appropriata per la gestione a moduli degli indirizzi è costituita dalla segmentazione

Organizzazione fisica

- La memoria è organizzata in due livelli:
  - memoria centrale: veloce, costosa, di estensione limitata
  - memoria secondaria (disco magnetico): più lenta, meno costosa e di estensione molto maggiore
- La memoria centrale può non essere sufficiente a contenere un programma e i suoi dati:
  - parte del programma deve stare in memoria secondaria
  - con la tecnica di overlay (non più usata) il programmatore poteva stabilire le parti del programma da caricare, a turno, nella stessa regione di memoria
  - in un sistema multiprogrammato il programmatore non sa quanto spazio di memoria è disponibile per il programma
- Trasferire programmi tra disco e memoria è compito del SO

Tecniche di gestione della memoria

- Le tecniche di gestione della memoria utilizzabili sono:

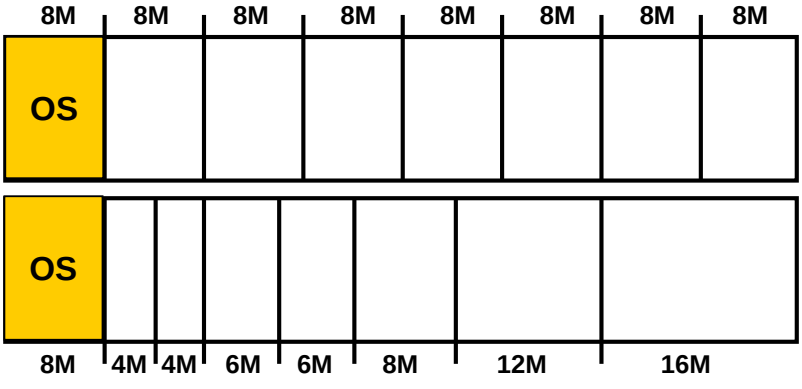
1. partizionamento (fisso e dinamico)
2. paginazione
3. segmentazione
4. paginazione con memoria virtuale
5. segmentazione con memoria virtuale

Partizionamento

- La memoria viene divisa in partizioni:
  - di dimensioni fisse (uguali oppure diverse)
  - di dimensioni variabili (partizionamento dinamico)
- Per partizioni fisse anche i processi piccoli occupano una intera partizione ⇨ frammentazione interna
- Se non ci sono partizioni libere, il SO può liberarne una trasferendo su memoria secondaria (swap) il processo relativo
- Per partizioni variabili, a ciascun processo viene assegnata una partizione delle dimensioni richieste (nessuna frammentazione interna)
- Col passar del tempo si formano in memoria delle aree di dimensioni piccole non utilizzabili ⇨ frammentazione esterna
- La frammentazione richiede che periodicamente la memoria venga compattata, con impegno di CPU

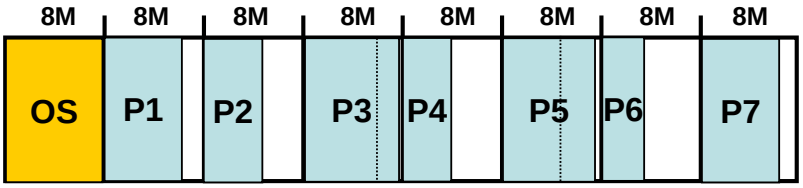
Partizionamento fisso

- La memoria viene divisa in partizioni di dimensioni fisse:
  - di dimensione uguali, oppure
  - di dimensioni diverse
- Le dimensioni delle partizioni non possono essere modificate
- Facile da realizzare, il grado di multiprogrammazione è limitato (un processo per partizione)



Partizioni fisse di dimensioni uguali

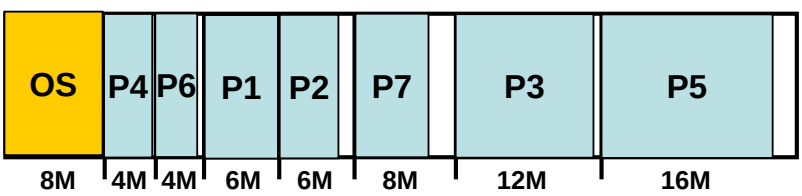
- Un processo di dimensione inferiore o uguale a quella delle partizioni può essere caricato in una qualsiasi partizione libera
  - Se non ci sono partizioni libere, il SO può liberarne una trasferendo su memoria secondaria (swap) il processo relativo
  - Se un processo è più grande di una partizione, il programma deve essere progettato con la tecnica di overlay
- Anche i processi piccoli occupano una intera partizione: ciò provoca un uso inefficiente della memoria indicato come frammentazione interna.



aree bianche = frammentazione interna (P3 e P5 con overlay)

Partizioni fisse di dimensioni diverse

- Ogni processo può essere collocato nella partizione più piccola in grado di contenerlo
  - Rispetto al caso precedente riduce la frammentazione interna (memoria inutilizzata all'interno di ciascuna partizione):
    - le partizioni piccole riducono lo spreco per processi piccoli,
    - le partizioni grandi riducono la necessità di overlay,
- Si può pensare di avere una coda di processi per ciascuna dimensione delle partizioni.



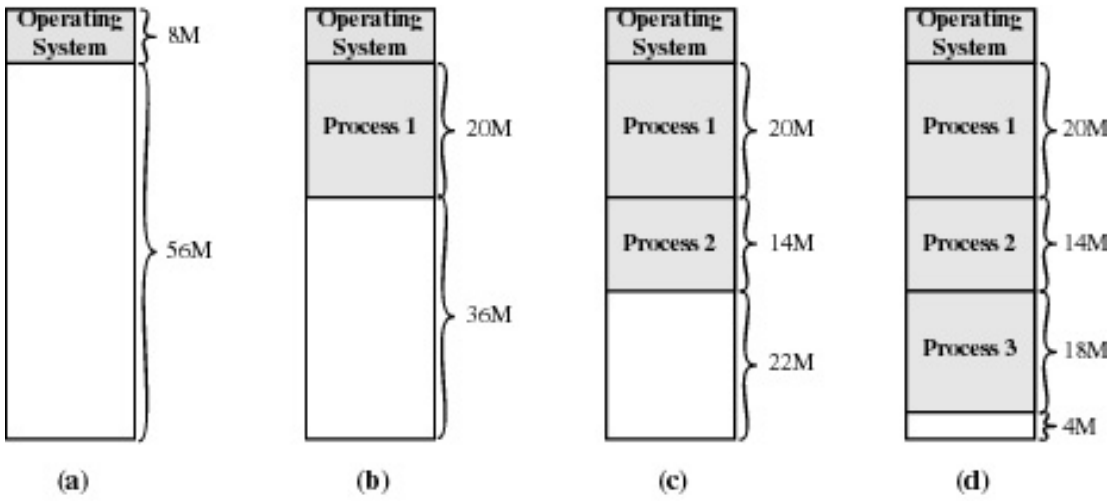
memoria da 64M con partizioni di dimensioni diverse

Partizionamento dinamico

- Il numero e le dimensioni delle partizioni sono variabili
- A ciascun processo viene assegnata una partizione delle dimensioni richieste (nessuna frammentazione interna)
- Col passar del tempo si formano in memoria delle aree di dimensioni piccole non utilizzabili (frammentazione esterna)
- Per recuperare queste aree non utilizzabili è necessario spostare i processi in memoria e compattare le aree libere in un'unica area più grande e quindi più utilizzabile
- L'operazione di compattazione richiede tempo di CPU.

Partizioni dinamiche

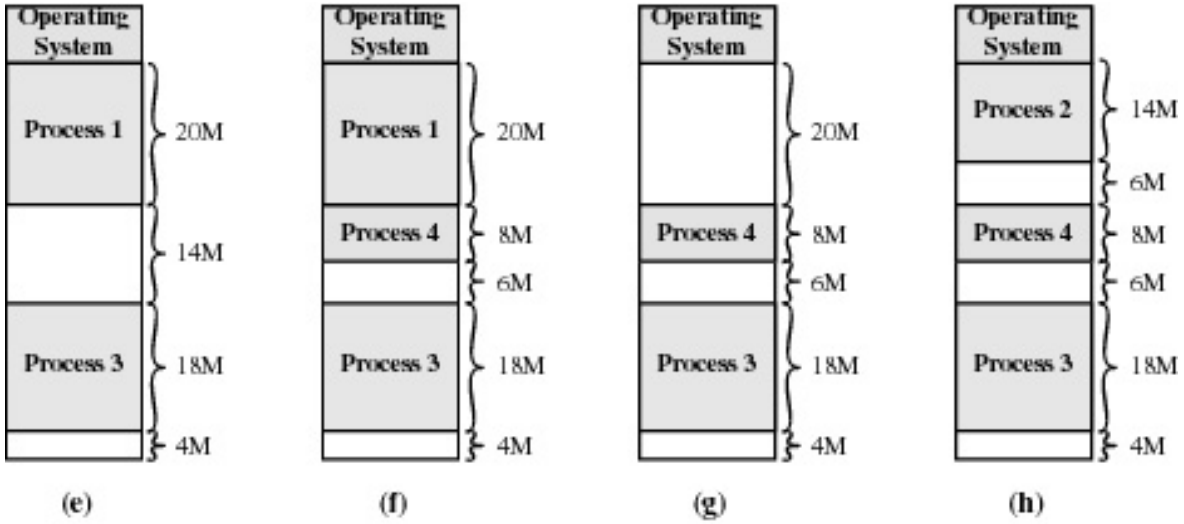
- A ciascun processo viene assegnata una partizione di dimensioni esattamente uguali a quelle necessarie
- Dopo un po' alcuni processi subiscono uno swap per caricarne altri, oppure terminano e liberano la partizione ...



Frammentazione esterna

- l'area liberata dallo swap-out di P2 è usata per caricare P4;
- l'area liberata dal termine di P1 è usata per lo swap-in di P2;

- Le aree recuperate non sono utilizzate appieno: si ha frammentazione esterna

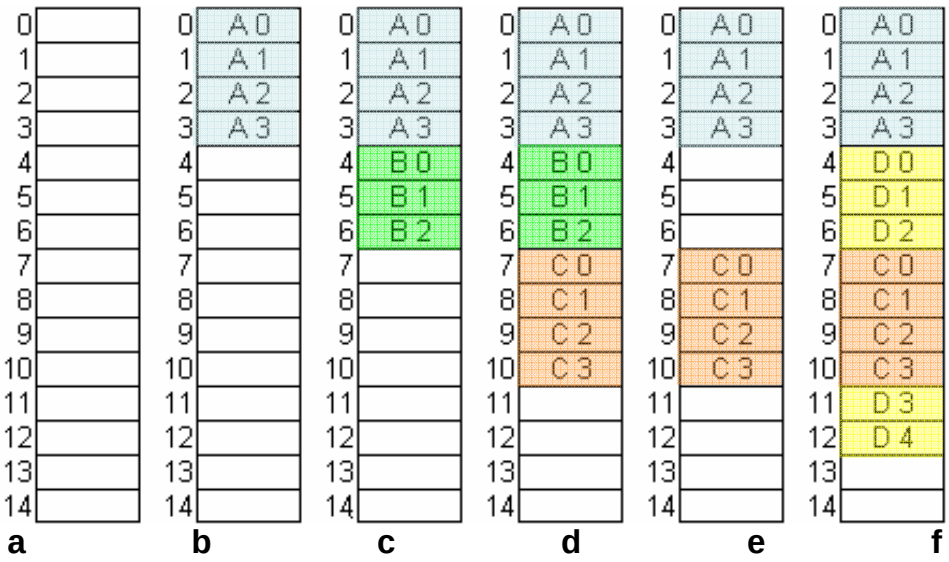


Paginazione

- La memoria fisica viene divisa in blocchi uguali, detti frame (o pagine fisiche), di dimensioni spesso piccole (es: 4KB) e uguali alla dimensione di un settore del disco
- Anche lo spazio di memoria indirizzato da ciascun processo è diviso in blocchi delle medesime dimensioni, detti pagine (o pagine logiche):
  - un indirizzo di memoria di un processo è considerato diviso in due parti: un indice di pagina e un offset nella pagina
- Il SO mantiene una page table per ciascun processo:
  - quando il processo è caricato in memoria, la sua page table indica in quale frame si trova ciascuna sua pagina
  - un indirizzo fisico è costituito da due parti: un indice di frame e un offset nel frame
  - i frame occupati da un processo possono non essere contigui.

Esempio Page frames

- Il processo A ha 4 pagine; B ne ha 3, C ne ha 4, D ne ha 5
- 3 pagine di D usano i frame liberati dallo swap-out di B



Paginazione [2]

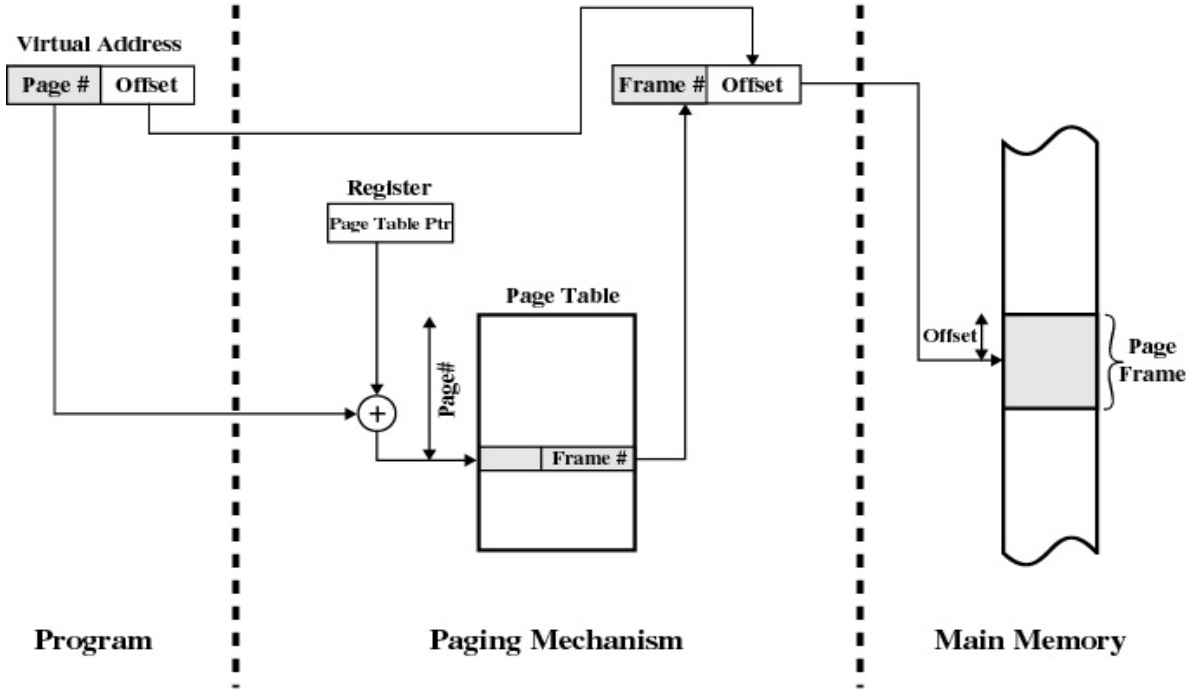
- Ad ogni processo è associata una Page Table, il cui elemento di indice i corrisponde alla i-esima pagina (dello spazio di indirizzi) del processo e contiene:
  - l'indice del frame nella memoria fisica in cui è collocata la i-esima pagina del processo,
  - alcuni bit di controllo:
    - P = pagina presente nella memoria fisica,
    - M = pagina modificata durante la permanenza in memoria fisica (se non è stata modificata non occorre ricopiarla su disco in caso di rimpiazzo),
    - bit di protezione (sola lettura, sola esecuzione, accesso riservato al SO, …)
  - elemento della page table:

Frame #	P	M			

Page table

- Le Page Table dei diversi processi hanno dimensioni diverse
  (dipende dalla dimensione del processo)
  - alcuni processi sono molto grandi e di conseguenza hanno page tabe con molti elementi
  - la page table non può essere contenuta in registri appositi, ma va collocata in memoria
  - un registro (page table pointer) punta alla page table attiva

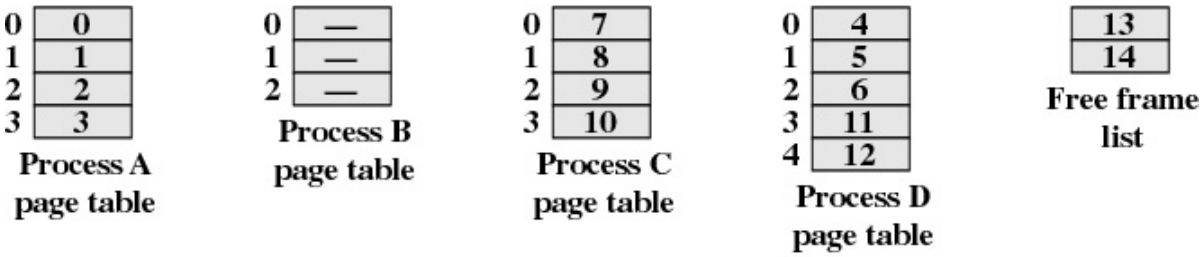
Sistema con paginazione



Traduzione di indirizzo virtuale in indirizzo fisico

Esempio: Page Tables

- Page Table dei 5 processi dell'esempio nella fase f; (indirizzi da 20-bit, pagine da 4KB):
  - le dimensioni delle pagine sono potenze di 2; ciò semplifica l'hardware di gestione della memoria: l'indice di pagina e l'offset sono forniti da due porzioni dell'indirizzo:
    - 0101 0101 1011 0100 1110 $55 / $B8E
    - primi 8 bit (0101 0101) = indice di pagina $55
    - ultimi 12 bit (1011 0100 1110) = offset $B8E



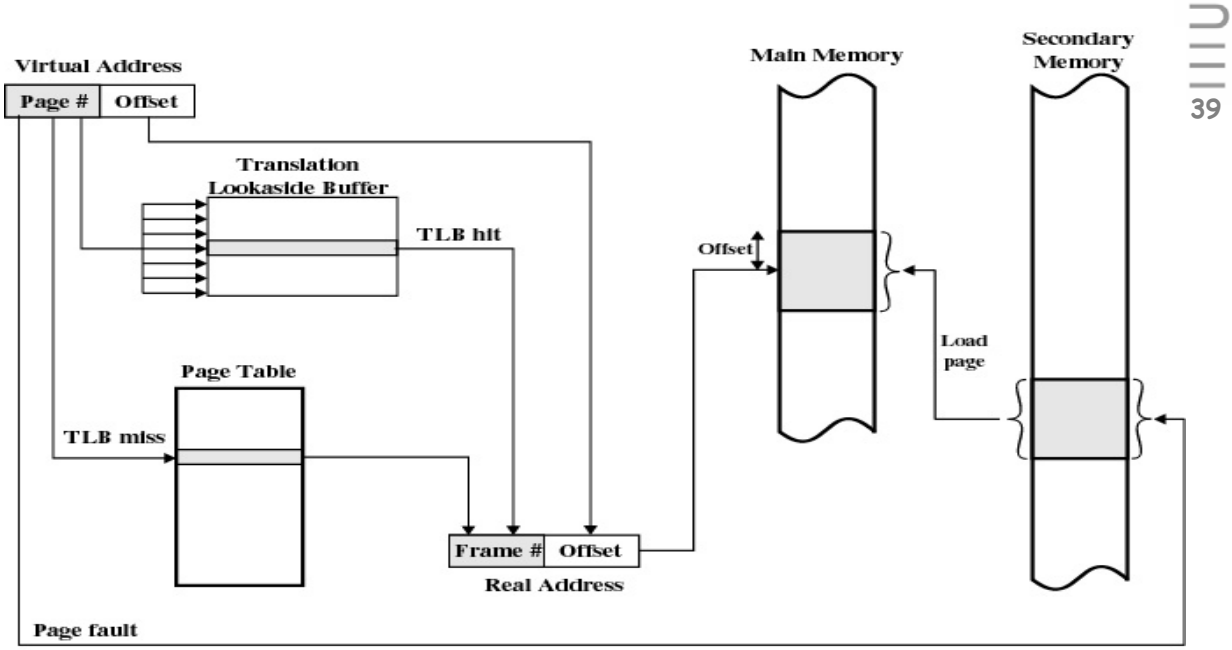
Translation lookaside buffer

- Usando la page table (tenuta in memoria) per trasformare un indirizzo virtuale in un indirizzo fisico, c'è il problema che ogni accesso alla memoria ne richiede almeno due:
  - uno per accedere alla page table;
  - uno per accedere al dato.
- ed i tempi di accesso alla memoria raddoppiano!
- Per limitare questo effetto, si può usare una cache veloce in cui tenere gli elementi della page table usati più di recente; questa cache si chiama Translation Lookaside Buffer (TLB) e ciascun suo elemento contiene:
  - un **page#** (in memoria associativa, per ricerca parallela),
  - il corrispondente **frame#** (frame che contiene la pagina)

Uso del TLB

- Dato un indirizzo virtuale (page#, offset), il page# viene cercato nella memoria associativa del TLB:
  - se trovato (hit), si estrae il corrispondente frame# e si costruisce l'indirizzo fisico (frame#, offset);
  - se non c'è (miss), si accede alla page table (indice=page#):
    - se l'elemento della page table indica che la pagina cercata è presente in un frame nella memoria fisica:
      - si costruisce l'indirizzo fisico (frame#, offset),
      - si aggiorna il TLB inserendo l'elemento trovato;
    - altrimenti si ha un page fault:
      - si porta in un frame la pagina cercata (swap in),
      - si aggiorna la page table (nuova pagina presente),
      - si costruisce l'indirizzo fisico (frame#, offset),
      - si aggiorna il TLB inserendo il nuovo elemento.

Uso del TLB – schema grafico



Translation Lookaside Buffer

Segmentazione

- I segmenti di un processo sono in genere di dimensioni diverse con un limite superiore alla lunghezza per i segmenti
- Un indirizzo è costituito da: un indice di segmento e un offset
- La segmentazione pone gli stessi problemi del partizionamento dinamico
- Ciascun processo è costituito da un insieme di segmenti (uno per ciascuno dei moduli) e da una segment table contenente, per ciascuno dei suoi segmenti, le seguenti informazioni:
  - l'indirizzo iniziale del segmento (inserito dal SO al momento in cui il segmento viene caricato in memoria)
  - la dimensione del segmento
  - bit di controllo (presente, modificato, …)
  - bit di protezione (read only, permessi, …)

Indirizzo virtuale ⇒ indirizzo fisico

- L'indirizzo virtuale definito dal codice ha due componenti:
  - un indice di segmento IS e un offset OS,
- L'indice di segmento individua un elemento della segment table da cui si estrae:
  - l'indirizzo iniziale del segmento (IIS)
  - la dimensione del segmento (DS)
- L'offset OS viene confrontato con la dimensione DS:
  - se OS > DS l'indirizzo è non valido (eccezione)
  - se OS ≤ DS l'indirizzo è valido
- L'indirizzo fisico è dato da IIS + OS (indirizzo iniziale del segmento, estratto dalla segment table, + offset, estratto dall'indirizzo virtuale).

Supporto fornito dall'hardware

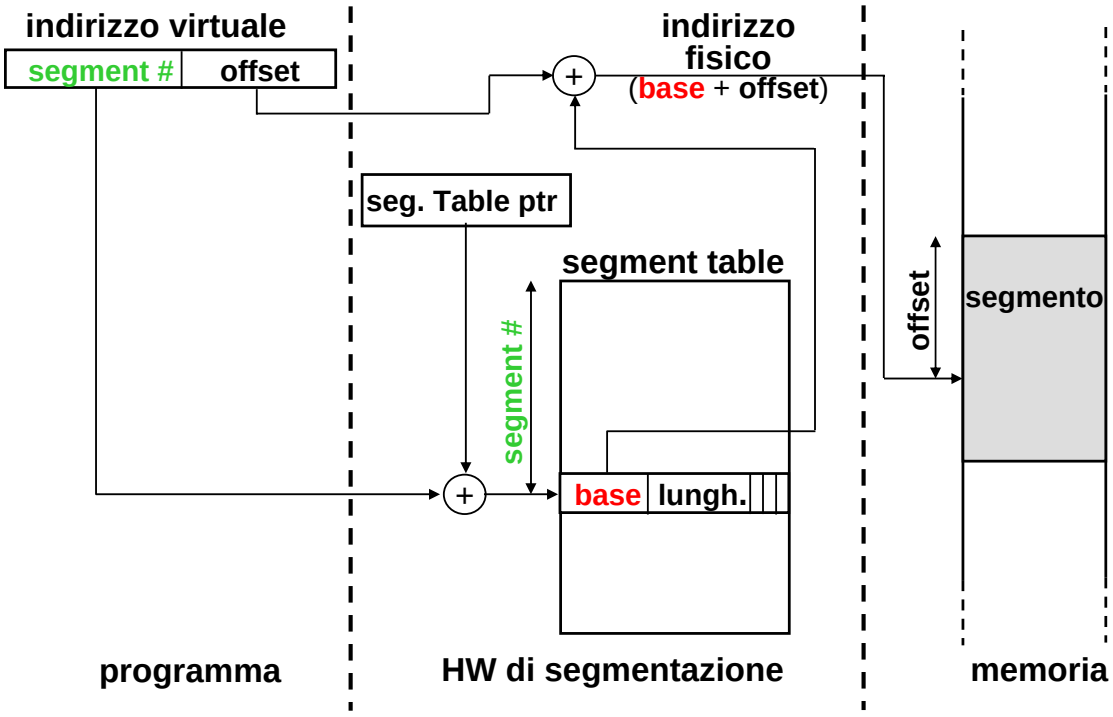
- Per realizzare un sistema di memoria virtuale (MV), è necessario che nell'hardware sia presente:
  - un meccanismo di paginazione, o
  - un meccanismo di segmentazione, oppure
  - entrambi
- Sfruttando questi meccanismi il SO può gestire i trasferimenti delle pagine e/o dei segmenti tra memoria centrale e disco
- L'organizzazione delle page table e delle segment table va rivista per tener conto delle esigenze del sistema di MV

Segmentazione [2]

- Ad ogni processo è associata una Segment Table, il cui elemento di indice i corrisponde alla i-esimo segmento del processo e contiene:
  - l'indirizzo base del segmento nella memoria fisica
  - la lunghezza del segmento
  - alcuni bit di controllo:
    - P = segmento presente nella memoria fisica,
    - M = segmento modificato durante la permanenza in memoria fisica (se non è stato modificato non occorre ricopiarla su disco in caso di rimpiazzo),
    - altri bit di protezione (sola lettura, sola esecuzione, accesso riservato al SO, …)
- Elemento della segment table:

indirizzo base	lunghezza	P	M			

Uso della Segment Table



Segmentazione - Esempio

segm. A0	segmento A1	segm. B0	segmento B1	
(codice di A)	(dati di A)	(cod. B)	(dati di B)	
$250 Byte	**$754 Byte**	**$236 Byte**	**$348 Byte**	

segment table del processo A:

indice segmento	indirizzo iniziale	lunghezza segmento	bit di contr.	bit di prot.
0	**$0**	**$250**	**p**	**rx**
1	**$250**	**$754**	**pm**	**rw**

segment table del processo B:

indice segmento	indirizzo iniziale	lunghezza segmento	bit di contr.	bit di prot.
0	**$9A4**	**$236**	**p**	**rx**
1	**$BDA**	**$348**	**pm**	**rw**

Segment table - Esempio

- Con un indirizzo virtuale costituito da 20 bit :
  - 4 bit (indice di segmento) + 16 bit (offset)
  - ogni processo può avere al massimo 16 segmenti
  - la dimensione massima di un segmento è di 64KB

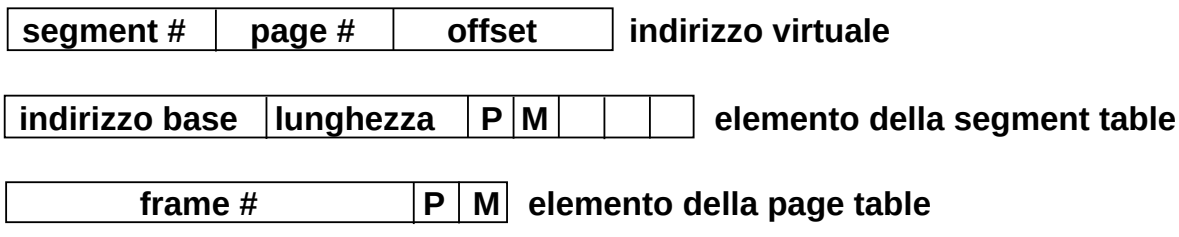
- Processo A
  - indirizzo virtuale: 0001 0000 0110 0100 1110 ($1 / $064E)
    indice di segmento = $1; offset = $064E (< $754:OK)
  - segment table del processo A:

indice segmento	indirizzo iniziale	lunghezza segmento	bit di contr.	bit di prot.
0	$0	$250	p	rx
1	$250	$754	pm	rw

  - Calcolo dell'indirizzo fisico:
    indirizzo fisico = $250 + $064E = $089E

Segmentazione con paginazione

- Combinando segmentazione e paginazione si combinano i vantaggi di entrambe:
  - la paginazione è trasparente al programmatore, elimina la frammentazione esterna, facilita la gestione della memoria
  - la segmentazione è visibile al programmatore, consente la modularità, la protezione e la condivisione dei segmenti
- Ad ogni processo è associata una segment table
  ad ogni segmento è associata una page table



Indirizzi virtuali ⇒ indirizzi fisici

- A partire dall'indirizzo virtuale:

segment #	page #	offset

  - il segment# individua un elemento nella Segment Table:

indirizzo base	lunghezza	P	M			

  - se offset > lunghezza: eccezione di address error;

- (page# + indirizzo base) individua elemento nella Page table:

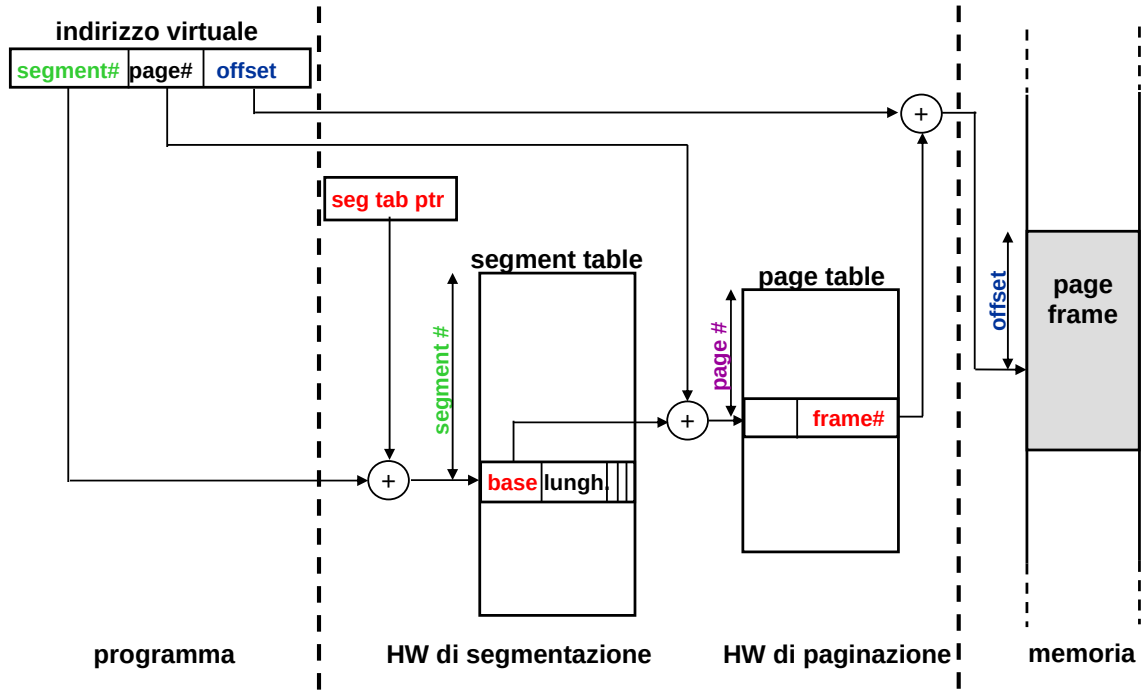
frame #	P	M

- (frame#, offset) fornisce l'indirizzo fisico:

frame #	offset

indirizzo fisico

Uso della segment e della page table



Paginazione e segmentazione

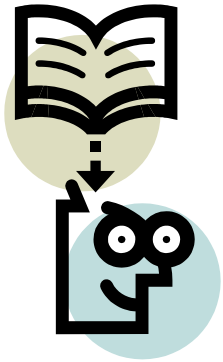
- Caratteristiche importanti:
  - gli indirizzi di memoria vengono tradotti in indirizzi fisici durante l'esecuzione, per cui i processi possono subire swap (out e in) dalla memoria e occupare aree di memoria diverse
  - i diversi blocchi (pagine o segmenti) di un processo possono essere collocati in aree non contigue della memoria
- Conseguenza:
  - non è necessario che tutti i blocchi (pagine o segmenti) di un processo siano caricati in memoria durante l'esecuzione
  - il SO carica in memoria solo alcuni blocchi (resident set).

Esecuzione di un programma

- Quando il processo P richiede un indirizzo che non si trova nel resident set (memory fault):
  - viene generato un interrupt e interviene il SO che
    - pone il processo nello stato "waiting"
    - avvia la lettura da disco del blocco richiesto (swap in)
    - rende running un altro processo Q
- Quando la lettura da disco del blocco richiesto è completata:
  - viene generato un interrupt e interviene il SO che
    - riporta il processo P nello stato "ready"
- A regime, quando la memoria fisica è tutta occupata, la lettura di un nuovo blocco in seguito a un memory fault, richiede di:
  - scegliere (secondo una politica di rimpiazzo opportuna) quale altro blocco deve essere rimpiazzato (swap out).

Fine

08.a



Gestione della memoria