Lezione 8

La gestione della memoria

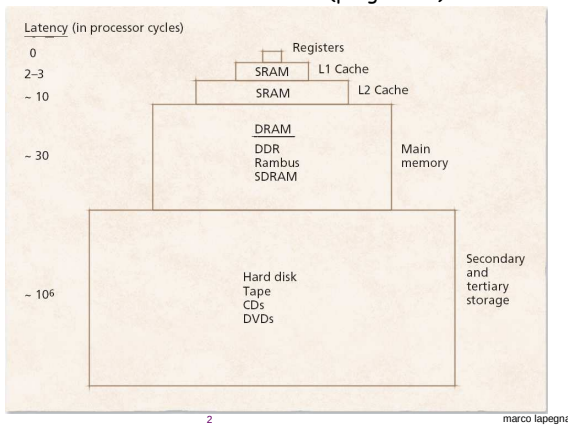
- Background
- Necessita' di un gestore della memoria
- Allocazione contigua
  - Monoutente
  - Multiutente
  - Il problema della frammentazione
- Allocazione non contigua
  - Paginazione
  - Segmentazione

I livelli di memoria

E' il supporto del calcolatore per conservare dati e istruzioni (programmi)

Veloce e cara

Lenta e economica



La memoria centrale

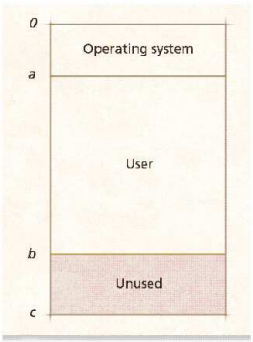
- La memoria centrale e' costituita da locazioni di memoria ad accesso casuale (RAM) individuate da indirizzi
- La CPU puo' accedere alla memoria centrale in un ordine qualunque
- la memoria centrale e' il piu' basso livello di memoria accessibille direttamente dalla CPU
- volatile: perde il contenuto in mancanza di tensione eletterica
- il Bandwidth e' la quantita' di dati che possono essere trasferiti alla CPU per unita' di tempo

Allocazione della memoria

nei primi calcolatori (anni 50), ma anche nei primi personal computer (anni 70), il calcolatore era in grado di eseguire un solo programma alla volta

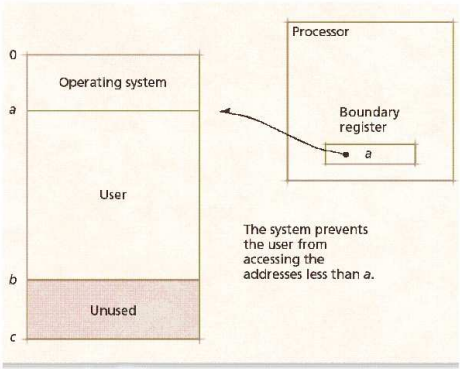
la memoria veniva condivisa tra Sistema Operativo e un programma utente

Allocazione contigua per sistemi monoutente



Protezione del sistema operaivo

- La protezione del sistema operativo viene realizzata con un registro di limitazione (Boundary register)



- Contiene l'indirizzo di inizio del programma utente
- Ogni riferimento oltre tale indirizzo e' negato
- Puo' essere definito solo in system mode
- Si puo' accedere all'area del S.O. solo attraverso chiamate di sistema che pongono il sistema in system mode

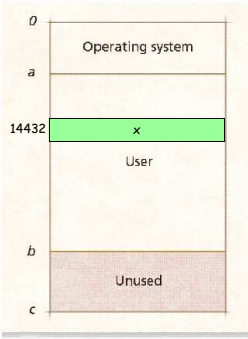
osservazione

La memoria e' un insieme sequenziale di locazioni con un proprio indirizzo.

Ogni programma risiede in una porzione ben definita della memoria,

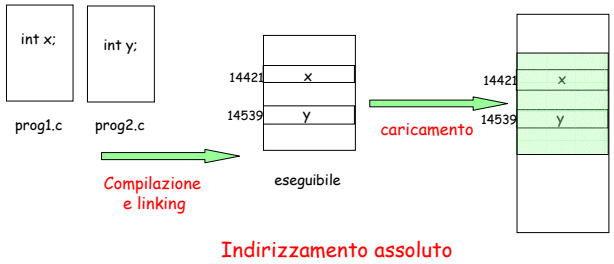
Ogni variabile del programma sara' individuata da un indirizzo

Quando vengono definiti tali indirizzi?



Generazione degli indirizzi: esempio 1

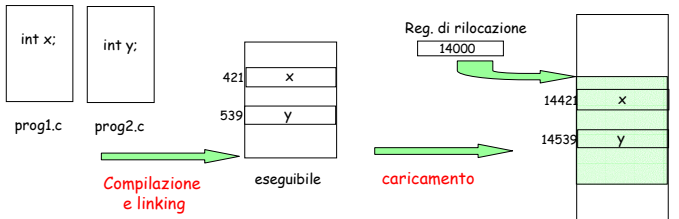
- L'associazione (*binding*) di istruzioni e dati a indirizzi di memoria può avvenire durante la fase di…
  - **Compilazione e linking**: se la posizione in memoria del processo è nota a priori, può essere generato un codice assoluto; se si volesse cambiare la locazione iniziale, è necessario ricompilare il codice;



Indirizzamento assoluto

Generazione degli indirizzi: esempio 2

- L'associazione (*binding*) di istruzioni e dati a indirizzi di memoria può avvenire durante la fase di…
  - **Esecuzione**: se il processo può essere spostato *a run-time* da un segmento di memoria all'altro, il binding viene rimandato fino al momento dell'esecuzione. È necessario un opportuno supporto hardware per mappare gli indirizzi (registri di rilocazione).



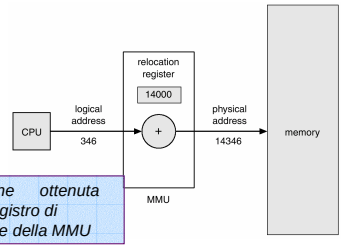
Indirizzamento rilocabile

Spazio di indirizzi logici e fisici

- Il concetto di uno *spazio di indirizzi logici* nettamente distinto dallo *spazio degli indirizzi fisici* è centrale per la gestione della memoria.
  - ***Indirizzi logici*** — generati dalla CPU; chiamati anche *indirizzi virtuali.*
  - ***Indirizzi fisici*** — indirizzi utilizzati per accedere alla memoria.
- Gli indirizzi logici corrispondono agli indirizzi fisici negli schemi di binding in fase di compilazione, mentre differiscono per il binding in fase di esecuzione.

Memory-Management Unit (MMU)

- L'operazione di far corrispondere indirizzi logici e fisici nel caso di indirizzamento rilocabile, e' un'operazione frequentissima e viene affidata, per efficienza, ad uno specifico dispositivo hardware (MMU)
- Nello schema MMU, il valore contenuto nel registro di rilocazione viene sommato ad ogni indirizzo generato dai processi utente nel momento stesso in cui l'indirizzo viene inviato alla memoria.
- Il programma utente *ragiona* in termini di indirizzi virtuali, e non è conscio della loro mappatura fisica.

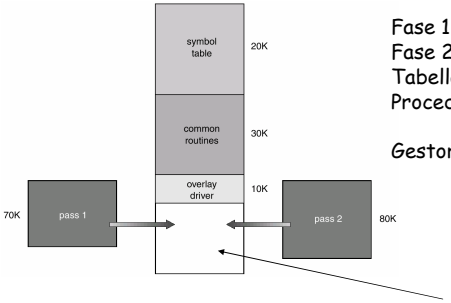


Rilocazione ottenuta tramite registro di rilocazione della MMU

Overlapping

- Un modo per avere a disposizione piu' memoria e' l'overlapping
- È richiesto quando un processo è più grande della memoria allocatagli.
- Si mantengono in memoria solo istruzioni e dati necessari in un certo istante. Quando occorrono altre istruzioni, queste vengono caricate nello spazio che era precedentemente occupato dalle istruzioni che non vengono più utilizzate.
- Viene implementato dall'utente, non viene richiesto alcun supporto speciale da parte del SO. Il progetto di software con overlay è complesso.

esempio dim memoria = 150 KB



Fase 1: 70 KB
Fase 2: 80 KB
Tabella dei simboli: 20 K
Procedure comuni: 30 KB
+
Gestore dell'overlapping 10 KB

Le fasi 1 e 2 si alternano nell'uso della memoria

- L'esecuzione è rallentata a causa dell'operazione necessaria per caricare l'overlay.

Caricamento dinamico

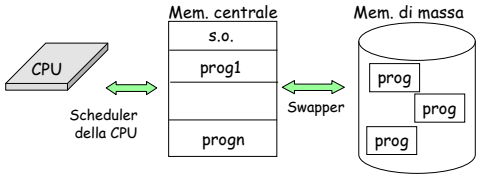
- Le fasi 1 e 2 non vengono caricate in memoria fino a quando non vengono richiamati.
- Si ha un miglior impiego della memoria: sottoprogrammi non utilizzati non vengono mai caricati.
- Utile quando si richiedono grandi quantità di codice per gestire situazioni che avvengono raramente (condizioni di eccezione).
- Al SO non è richiesto alcun supporto speciale. Il caricamento dinamico viene implementato attraverso un opportuno progetto del software o del compilatore

- Esempio: librerie dinamiche *.dll di unix

La multiprogrammazione

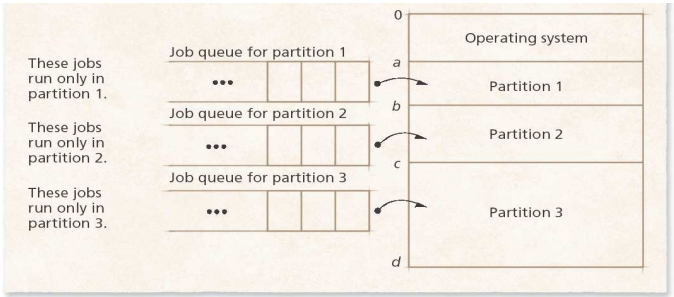
A causa delle continue interruzioni dovute all I/O, un uso efficiente della CPU impone che non solo i programmi in esecuzione, ma anche quelli che stanno per essere eseguiti, devono risiedere in memoria centrale

Lo scheduler di medio livello (swapper) decide quale programma deve competere per l'uso della CPU, e quindi deve risiedere in memoria centrale



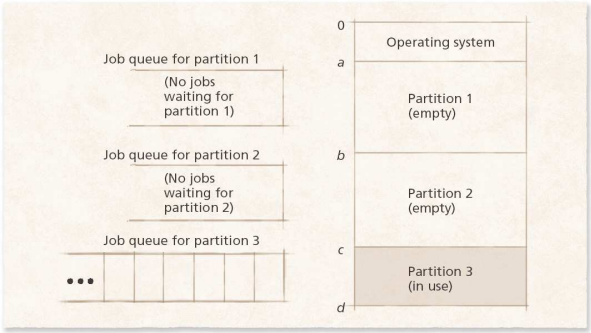
Multiprogrammazione a partizione fissa

- Ogni processo dispone di un blocco di memoria di dimensione fissata
- Differenti code di job per differenti tipi di programmi
- Piu' registri di limitazione garantiscono la protezione del s.o. e degli altri programmi
- Facilmente realizzabile con l'indirizzamento assoluto



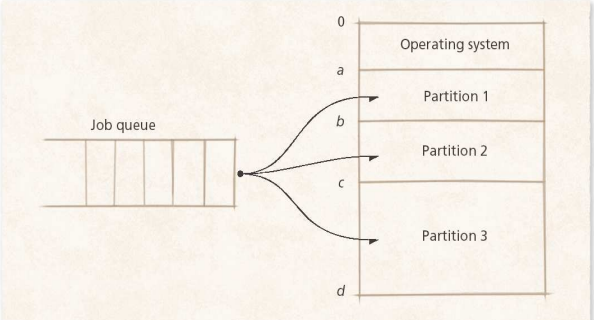
Problema

- Spreco di risorse: molte partizioni inutilizzate mentre molti processi aspettano in coda
- Es. 3 partizioni di cui 2 inutilizzate



Soluzione

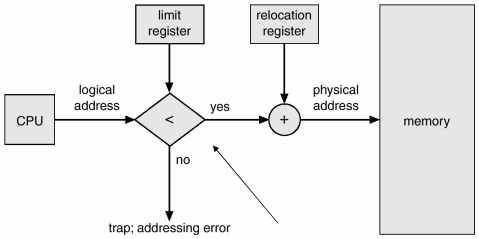
Multiprogrammazione con partizione fissa e indirizzamento rilocabile



Un job puo' essere sistemato in una qualunque partizione libera di dimensione sufficiente

Protezione

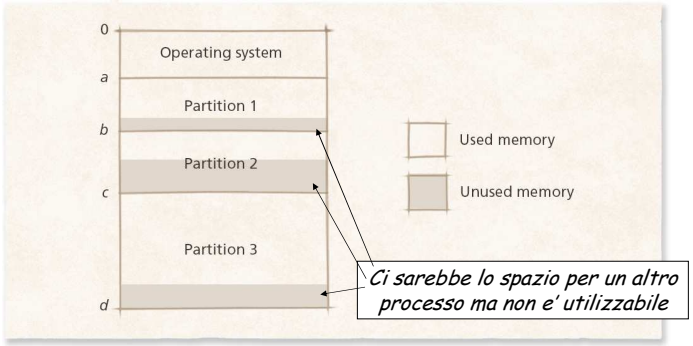
- Si puo' impiegare uno schema basato su registri base e limite
- Il registro di rilocazione (base) contiene il valore del più piccolo indirizzo fisico di memoria allocata al processo;
- il registro limite contiene l'intervallo degli indirizzi logici: ciascun indirizzo logico deve essere inferiore al valore del registro limite.



A carico del MMU

Problema della multiprogrammazione a partizioni fisse

I processi non fanno uso interamente della partizione a loro assegnata



Frammentazione interna

Soluzione

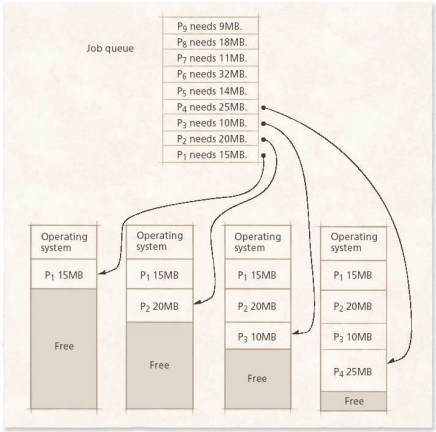
Partizioni fisse:

Proc grandi per la partizione

Proc piccoli per la partizione

Frammentazione interna

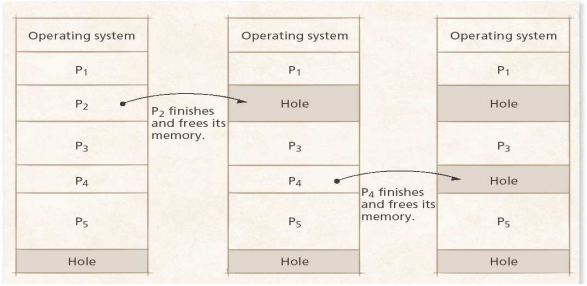
partizioni variabili



Multiprogrammazione a partizioni variabili

Con la multiprogrammazione a partizioni variabili si alloca ad un processo uno spazio della dimensione del processo stesso

non c'e' il problema della frammentazione interna, MA……

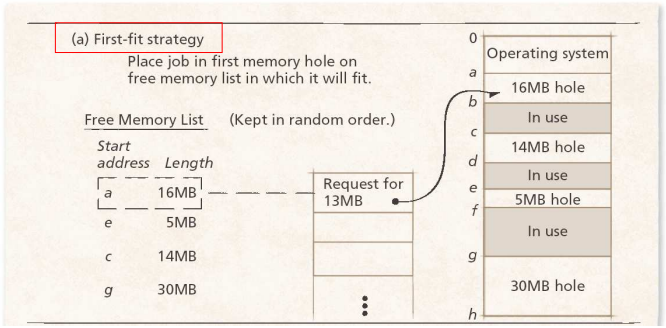


Quando i processi finiscono si creano dei buchi (holes) in memoria

FRAMMENTAZIONE ESTERNA

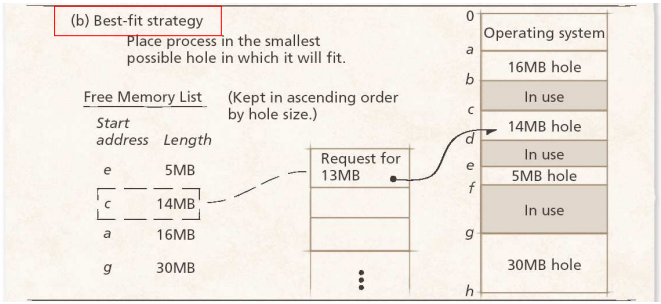
Riallocazione della memoria

- **First-fit**: ai nuovi processi viene allocato il **primo buco** grande abbastanza.
  - Facile da implementare e con basso overhead



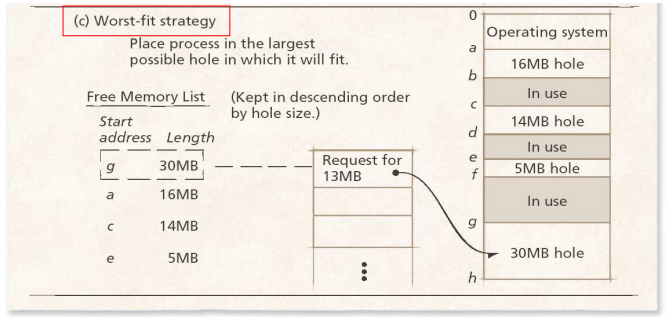
Riallocazione della memoria

- **Best-fit**: ai nuovi processi viene allocato il buco **più piccolo** capace di contenere il processo.
  - È necessario scandire tutta la lista dei buchi (maggiore overhead)
  - Si produce il più piccolo buco residuo.



Riallocazione della memoria

- **Worst-fit**: ai nuovi processi viene allocato il buco **più grande.**
  - È ancora necessario ricercare in tutta la lista.
  - Si produce il più grande buco residuo (utile per successive allocazioni)



problema

I meccanismi di riallocazione
- First fit
- Best fit
- Worst fit

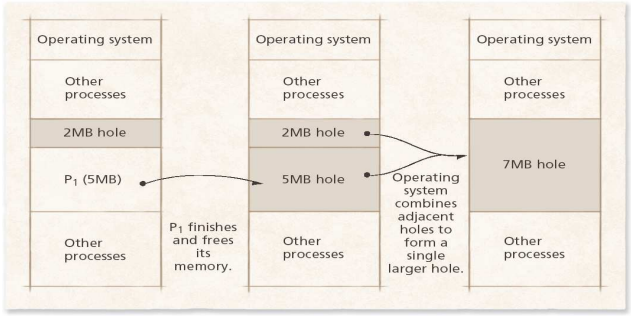
Non risolvono il problema della frammentazione esterna

ANZI, con l'andare del tempo producono buchi inutilizzati di dimensione sempre piu' piccola

Approcci alternativi

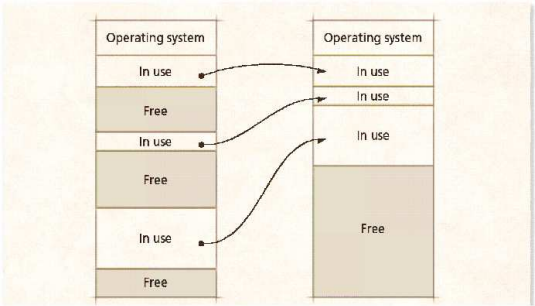
Soluzione 1: coalescenza

- Unisce blocchi adiacenti liberi in un unico blocco di dimensioni maggiori
- Applicabile raramente e non sufficiente a recuperare rapidamente sufficienti quantita' di memoria



Soluzione 2: compattamento

- Chiamata anche "garbage collection" (raccolta rifiuti)
- Unisce tutti i buchi in un singolo blocco contiguo di memoria libera e sposta tutti i blocchi occupati
- Overhead maggiore rispetto alla coalescenza



Allocazione non contigua

- Un'altra soluzione alla frammentazione esterna è ottenuta consentendo la **non contiguità degli indirizzi fisici**, permettendo così di allocare la memoria fisica ai processi ovunque essa sia disponibile.
- Si divide la memoria fisica in blocchi di dimensione fissa chiamati **blocchi** (o frame)
- Si divide il processo in blocchi della stessa dimensione chiamati **pagine**.
- Per eseguire un processo di *n* pagine, è necessario trovare *n* frame liberi prima di caricare il programma.
- Si ha solo frammentazione interna (relativa all'ultimo frame).

Esempio:

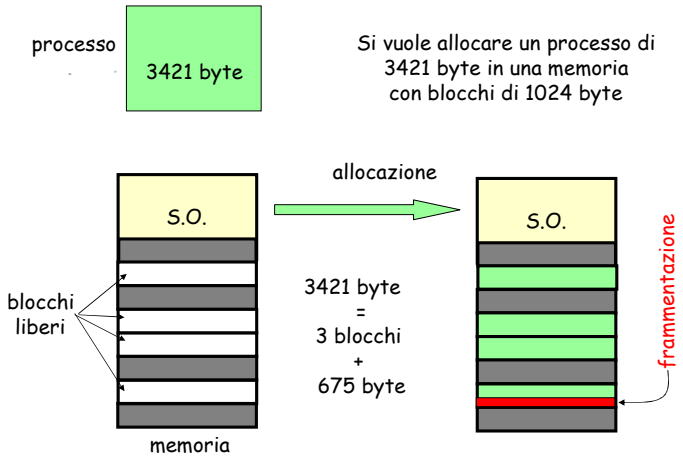
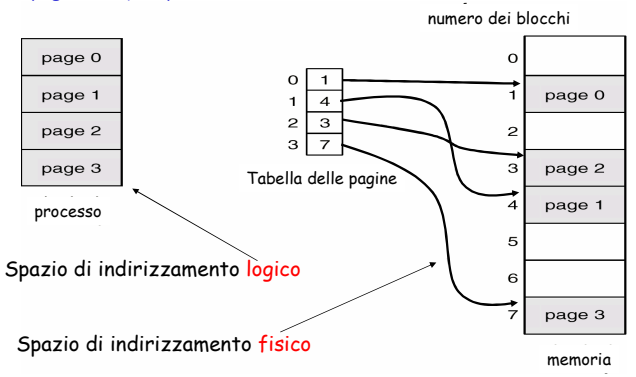
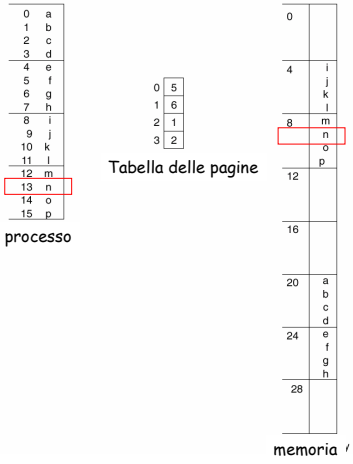


Tabella delle pagine

- Il s.o. conserva in memoria una tabella delle pagine per ogni processo che contiene gli indirizzi dei blocchi utilizzati dalle pagine di quel processo in memoria



Esempio: memoria di 32 byte e pagine di 4 byte



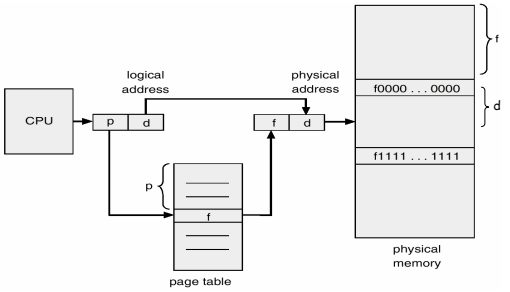
L'indirizzo logico 13 e' nella posizione 1 della pagina 3

La pagina 3 e' nel blocco di memoria 2 (indirizzo 8)

L'indirizzo logico 13 corrisponde all'indirizzo fisico 8+1 = 9

Schema di traduzione degli indirizzi nel MMU

- Ogni indirizzo generato dalla CPU (indirizzo logico) viene suddiviso in:
  - ***Numero di pagina*** **(*p*)** — impiegato come indice nella tabella delle pagine per trovare l'indirizzo del blocco
  - ***Scostamento nella pagina*** **(*d*)** — sommato con l'indirizzo del blocco per definire l'indirizzo fisico di memoria



Due osservazioni

1: Il numero dei blocchi e la dimensioni dei blocchi sono potenze di 2

Esempio: m= 16 bit a disposizione per gli indirizzi
m n= 8 bit per il numero di pagina p
n= 8 bit per lo scostamento d

p	d

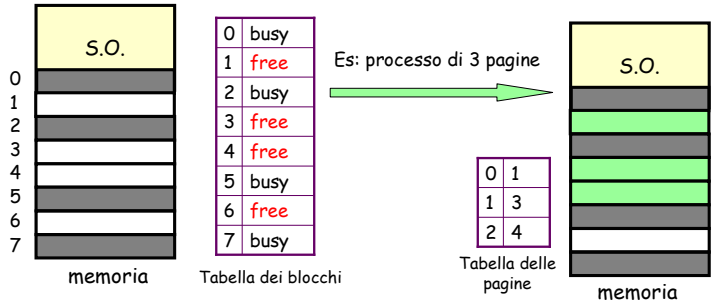
2^8 = 256 blocchi 2^8 = 256 byte per ogni blocco

In generale:
2^{m-n} blocchi di 2^n byte

2: La paginazione e' una forma di rilocazione dinamica e la protezione dei blocchi avviene mediante i valori di p e d analogamente ai registri base e limit

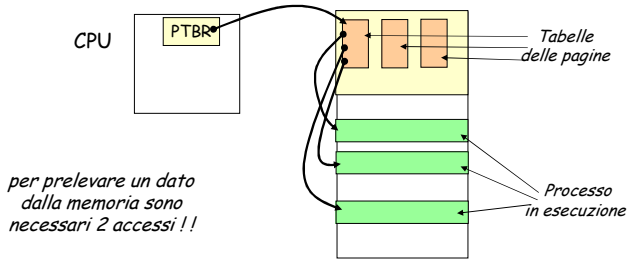
Tabella dei blocchi

- Il sistema operativo tiene traccia sull'uso dei blocchi di memoria (libero, occupato e a chi assegnato) mediante una tabella dei blocchi
- Quando un nuovo processo richiede memoria viene consultata la tabella dei blocchi e viene creata la tabella delle pagine del nuovo processo



Supporto hw per la paginazione

- Le tabella delle pagine (una per ogni processo) risiedono in memoria centrale.
- un registro della CPU identifica una tabella in memoria
  - ***Page-Table Base Register*** (PTBR) punta all'inizio della tabella.
- Al cambio di contesto il dispatcher aggiorna solo tale registro

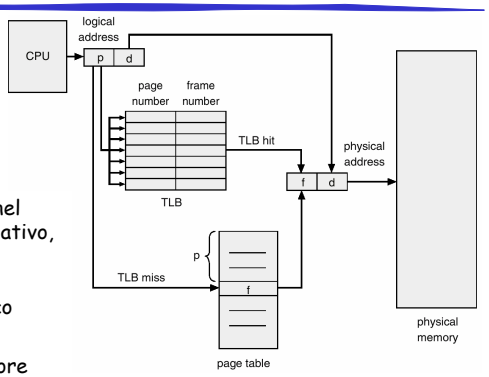


Supporto hw alla paginazione

- Il problema dei due accessi alla memoria può essere risolto con dei ***registri associativi*** (altrimenti detti *translation look-aside buffer*, TLB)
- Tali registri sono caricati dal dispatcher al momento del cambio di contesto e attraverso i quali si effettua una ricerca parallela veloce su una piccola tabella (senza scorrerla tutta)
- Tali registri , realizzati nella CPU o nella MMU, sono molto veloci ma economicamente costosi

- Vengono usati per memorizzare un sottinsieme della tabella delle pagine

Uso dei registri associativi



- Se p si trova nel registro associativo, si estrae il corrispondente numero di blocco
- Altrimenti, occore fare un riferimento in memoria alla tabella delle pagine.

Effective access time (EAT)

- Esempio
- Tempo di accesso alla memoria = 100 nsec
- Tempo di accesso alla TLB = 20 nsec

Tempo di accesso (EAT) = → 120 nsec se TLB hit; → 220 nsec se TLB miss

- Se percentuale di successi (TLB ratio) = 80%

  Tempo di accesso = 0.8 x 120 + 0.2 x 220 = 140 nsec

- In generale
  - α = tempo acc alla TLB
  - β = tempo acc alla memoria
  - ε = TLB ratio

  ⇒ EAT = ε (α+β) + (1 ε)(α+2β)

Protezione della memoria

- Un bit di validità nella tabella delle pagine viene associato ad ogni elemento della tabella delle pagine:
  - Un valore "*valido*" indica che la pagina è nello spazio degli indirizzi logici del processo, e quindi è una pagina legale.
  - Un valore "*non valido*" indica che la pagina non si trova nello spazio di indirizzi logici del processo.
- Vengono impediti accessi non autorizzati (eventualmente si possono avere bit separati per lettura e scrittura).

Problema

- quando lo spazio degli indirizzi logici è grande, lo schema descritto diventa inefficiente
- Esempio indirizzamento a 32 bit (es. Pentium)
  - Numero di pagina 20 bit
  - Scostamento 12 bit

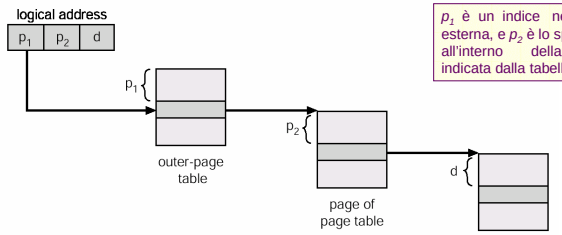
2^{20} blocchi 2^{12} = 4 KB per ogni blocco

- Un'unica tabella di 4MB per le pagine (ma i blocchi sono di 4KB)

Paginazione gerarchica (a livelli)

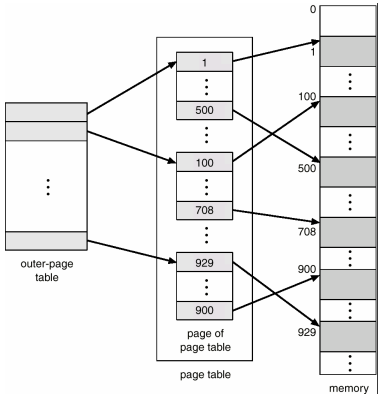
- Una soluzione e' "paginare" la tabella delle pagine.
- il numero di pagina viene ulteriormente suddiviso in:
  - un numero di pagina di p_1 bit;
  - un offset di p_2 bit.
- Un indirizzo logico è costituito da:

Numero di pagina		*offset*
p_1	p_2	d



p_1 è un indice nella tabella esterna, e p_2 è lo spostamento all'interno della pagina indicata dalla tabella esterna.

Schema di tabella delle pagine a due livelli

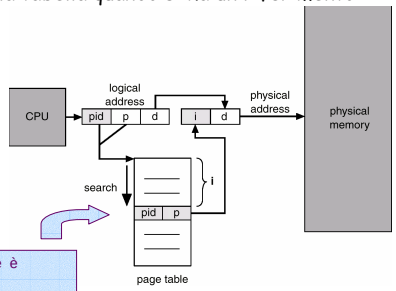


- Dato che ciascun livello viene memorizzato come una tabella separata in memoria, la traduzione di un indirizzo logico in uno fisico può richiedere **tre accessi alla memoria.**
- Anche se il tempo richiesto per un accesso alla memoria è triplicato, la **presenza di cache** consente di mantenere prestazioni ragionevoli.
- Per archietture a 64bit servirebbero **ulteriori livelli**

Possibile soluzione: tabella delle pagine invertita

- Un elemento della tabella per ogni blocco di memoria.
- Gli elementi della tabella contengono l'indirizzo virtuale della pagina memorizzata nel dato frame, con informazioni sul processo che possiede tale pagina.
- Una sola tabella delle pagine, ma incremento del tempo necessario per ricercare nella tabella quando si ha un riferimento a pagina.
- Notare che è necessario ricercare su tutta la tabella!

Ricerche veloci tipo hash in sottoinsiemi memorizzati in registri associativi

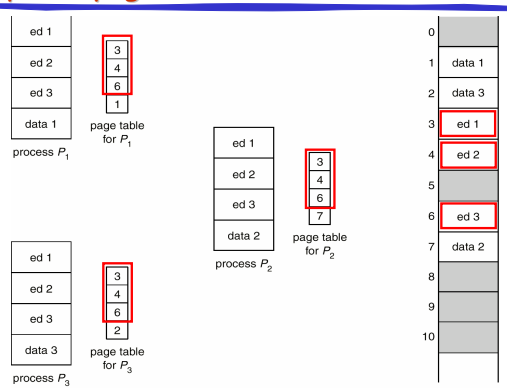


Ciascun indirizzo virtuale è formato dalla tripla:
<# processo, #pagina, offset>

Osservazione: pagine condivise

- Un ulteriore vantaggio della paginazione e' la condivisione del codice
- Esempio
  - Una copia di codice a sola lettura (ad esempio: text editor, compilatori, sistemi a finestre) viene condivisa fra processi con notevole risparmio di memoria
  - Il codice condiviso deve apparire nella stessa locazione nello spazio degli indirizzi logici di tutti i processi.
- Codice condiviso e dati privati
  - Ciascun processo mantiene una copia separata dei dati
  - Le pagine di codice e dati privati possono apparire ovunque nello spazio degli indirizzi logici.

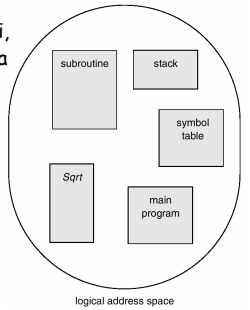
Esempio di pagine condivise



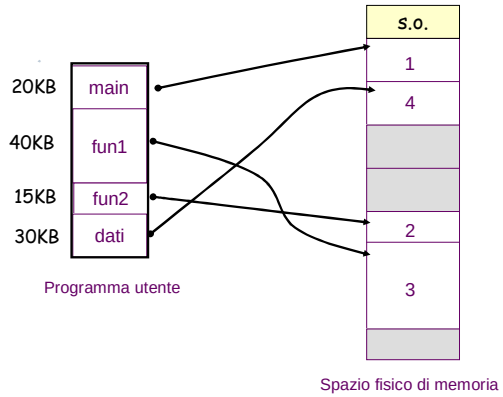
Condivisione di codice in un ambiente di paginazione: Editor contenuto in tre blocchi (3, 4 e 6)

Segmentazione

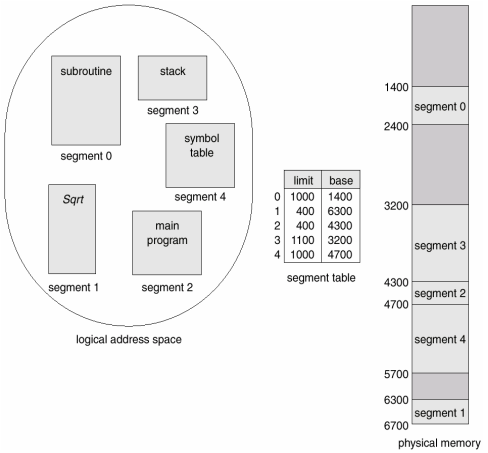
- La paginazione alloca la memoria in blocchi ' di dimensioni fissate (i frame)
- La segmentazione alloca la memoria in blocchi di dimensioni variabili (i segmenti) , assecondando la visione utente
- Un programma è una collezione di segmenti, dove ogni segmento contiene un'unita' logica del programma
  - programma principale (main),
  - procedura,
  - funzione,
  - variabili locali, variabili globali,
  - blocco common,
  - stack,
  - tabella dei simboli, matrici.



Schema logico di segmentazione

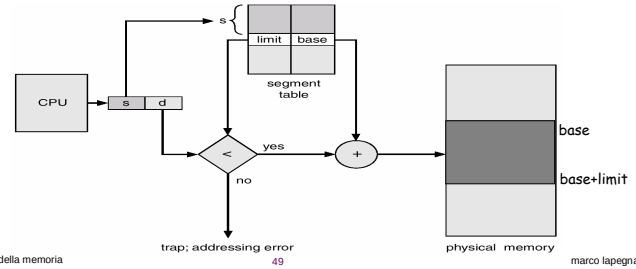


esempio



Architettura per la segmentazione

- Gli indirizzi logici sono rappresentati da
  - s = Numero del segmento
  - d = Spiazzamento all'interno del segmento
- **Tabella dei segmenti** — mappa gli indirizzi fisici bidimensionali; ciascun elemento della tabella contiene:
  - *base* — indirizzo fisico iniziale del segmento.
  - *limite* — specifica la lunghezza del segmento.

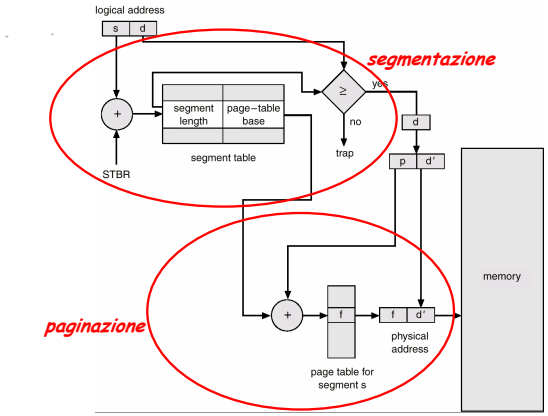


Segmentazione / paginazione

- E' possibile la **condivisione**: analogamente alla paginazione si ottengono segmenti condivisi, puntando allo stesso numero di segmento.
- **Protezione**: analoga alla paginazione
- E' possibile **combinare paginazione e segmentazione** (es.: Intel 80386, Motorola 68000)
- Dato che i segmenti variano di dimensione, **l'allocazione della memoria è dinamica.**

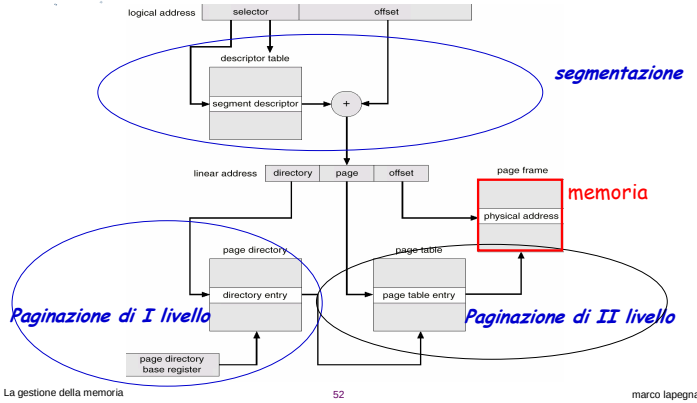
Esempio: MULTICS

- Il sistema MULTICS pagina(va) i segmenti.



Esempio: Intel 80386

- impiega una segmentazione con paginazione per la gestione della memoria con uno schema a due livelli: ogni segmento e' paginato



sommario

- allocazione contigua
  - sistema dedicato monoutente
    - partizioni fisse con indirizzamento assoluto
  - sistema multiprogrammato
    - partizioni fisse
      - indirizzamento assoluto
      - indirizzamento rilocabile
    - partizioni variabili
- allocazione non contigua
  - paginazione
  - segmentazione
  - segmentazione con paginazione

Le strategie di gestione analizzate differiscono per :
- l'architettura che usano,
- le prestazioni che producono,
- la frammentazione che introducono,
- la rilocazione che impiegano,
- l'eventuale utilizzo dello swapping,
- la possibilità di condivisione