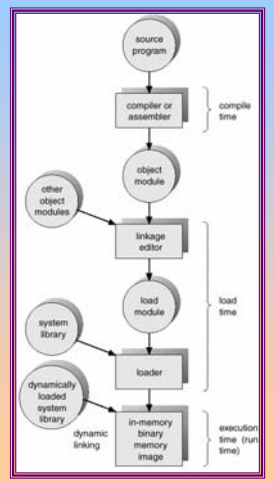
Gestione della memoria

- Gestione della memoria
  - Background
  - Swapping
  - Allocazione contigua
  - Paginazione
  - Segmentazione
- Memoria Virtuale
  - Paginazione su richiesta
  - Sostituzione delle pagine
  - Algoritmi di sostituzione delle pagine
  - Allocazione dei frame
  - Thrashing

Background

- Per poter essere eseguiti, i programmi devono essere portati in memoria e posti all'interno di un processo.
- *Coda di input* — collezione di processi su disco in attesa di essere caricati in memoria per essere eseguiti.
- I programmi utente passano attraverso stati diversi prima di essere eseguiti. L'associazione – ***binding*** – di istruzioni e dati a indirizzi di memoria può avvenire durante la fase di…
  - **Compilazione**: se la posizione in memoria del processo è nota a priori, può essere generato un codice assoluto; se la locazione iniziale cambia, è necessario ricompilare il codice;
  - **Caricamento**: se la posizione in memoria non è nota in fase di compilazione, è necessario generare codice *rilocabile*;
  - **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 (ad esempio attraverso registri *base* e *limite*).

Elaborazione multistep dei programmi utente



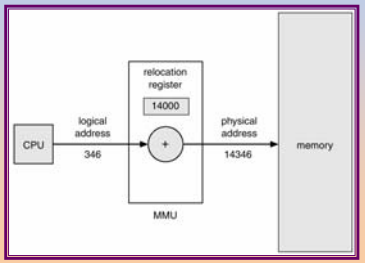
Spazio di indirizzi logici e fisici

- 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; detti altrimenti *indirizzi virtuali*.
  - ***Indirizzi fisici*** — indirizzi utilizzati nell'unità di memoria.
- Gli indirizzi logici corrispondono agli indirizzi fisici negli schemi di binding in fase di compilazione e caricamento, mentre differiscono per il binding in fase di esecuzione.

Memory–Management Unit (MMU)

- Dispositivo hardware che mappa indirizzi virtuali su indirizzi fisici.
- 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, né mai è (deve essere) conscio della loro mappatura fisica.

Rilocazione dinamica ottenuta tramite registro di rilocazione



Caricamento dinamico

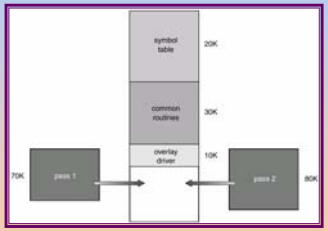
- I sottoprogrammi non vengono caricati in memoria fino a quando non vengono richiamati.
- Si ha un miglior impiego della memoria: sottoprogrammi non utilizzati non vengono mai caricati.
- Utile quando si richiedono grandi quantità di codice per gestire situazioni che avvengono raramente (condizioni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

Link dinamico delle librerie

- Il linking viene rinviato fino al momento dell'esecuzione.
- Piccole porzioni di codice (dette *stub*) vengono impiegate per localizzare la routine appropriata nella libreria residente in memoria.
- Lo stub rimpiazza se stesso con l'indirizzo della routine e la esegue.
- Il SO deve verificare che la routine si trovi nello spazio degli indirizzi (in memoria) del processo.

Overlay

- 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.
- È richiesto quando un processo è più grande della memoria allocatagli.
- L'esecuzione è rallentata a causa dell'operazione di I/O necessaria per caricare l'overlay.
- Viene implementato dall'utente, non viene richiesto alcun supporto speciale da parte del SO. Il progetto di software con overlay è complesso.

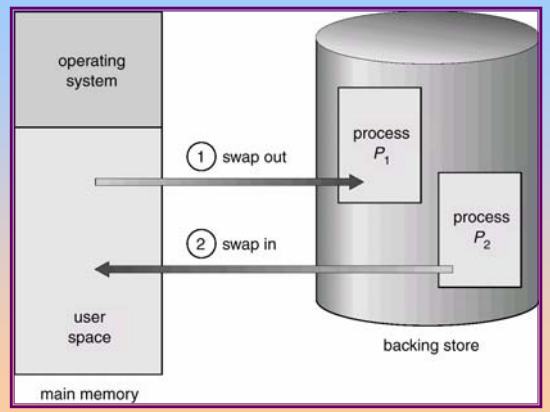


Overlay per un assemblatore a due passi

Swapping

- Un processo può venir temporaneamente riversato (***swapped***) dalla memoria centrale alla ***backing store*** da cui, in seguito, viene portato nuovamente in memoria per proseguire l'esecuzione.
- ***Backing store*** — È un disco veloce, sufficientemente capiente da accogliere copie di tutte le immagini di memoria per tutti gli utenti; deve garantire accesso diretto a tali immagini. Può essere una partizione dedicata del disco (pochi movimenti della testina).
- ***Roll out, roll in*** — È una variante dello swapping impiegata per algoritmi di scheduling basati su priorità; un processo a priorità più bassa viene riversato sulla memoria di massa, in modo tale da permettere che un processo a maggior priorità sia caricato ed eseguito.
- La maggior parte del tempo di swap è impiegata per il trasferimento di dati; il tempo totale di trasferimento è direttamente proporzionale alla quantità di memoria riversata.
- Se il binding si effettua in fase d'esecuzione, il processo sottoposto a swapping può successivamente essere riversato in uno spazio di memoria diverso.
- Molti SO attuali supportano lo swapping, ad esempio **UNIX** e **MS Windows**.

Swapping

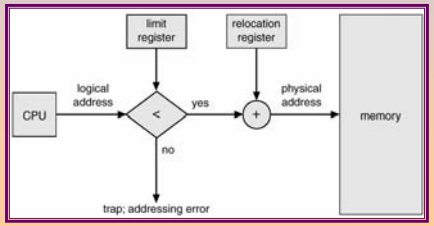


Swapping di due processi utilizzando il disco come backing store

Allocazione contigua

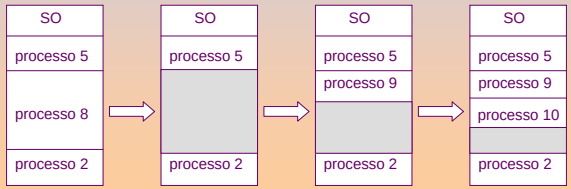
- La memoria principale viene suddivisa in due partizioni:
  - La parte residente del SO è generalmente memorizzata nella memoria bassa, insieme al *vettore degli interrupt*.
  - I processi utente sono memorizzati nella memoria alta.
- **Allocazione con partizione singola**
  - Si impiega uno schema basato su registri base e limite per proteggere programmi e dati del SO e per proteggere reciprocamente i programmi utente.
  - Il registro di rilocazione 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.

Supporto hardware ai registri base e limite.



Allocazione contigua

- **Allocazione con partizioni multiple**
  - Nei primi SO si avevano partizioni di dimensioni fisse.
  - Un *buco* (*hole*) è un blocco di memoria disponibile; nella memoria sono sparsi buchi di varie dimensioni.
  - Quando viene caricato un nuovo processo, gli viene allocato un buco grande abbastanza da contenere il processo.
  - Il SO conserva informazioni su:
    a) *Partizioni allocate*
    b) *Partizioni libere* (buchi)



Problemi di allocazione dinamica della memoria

Come soddisfare una richiesta di dimensione *n* a partire da un insieme di buchi? In ogni momento è presente un insieme di buchi di diverse dimensioni sparsi per la memoria.

- **First–fit**: Viene allocato il *primo* buco grande abbastanza.
- **Best–fit**: Viene allocato il buco *più piccolo* capace di contenere il processo. È necessario scandire tutta la lista dei buchi. Si produce il più piccolo buco residuo.
- **Worst–fit**: Viene allocato il buco *più grande*. È ancora necessario ricercare in tutta la lista. Si produce il più grande buco residuo.

First–fit e Best–fit sono mediamente migliori di Worst–fit, rispettivamente in termini di velocità e impiego di memoria.

Frammentazione

- ***Frammentazione esterna*** — È disponibile lo spazio totale per soddisfare una richiesta, ma non è contiguo.
- ***Frammentazione interna*** — La memoria allocata può essere leggermente maggiore della memoria richiesta (pochi byte di differenza). La differenza di dimensioni è memoria interna ad una partizione che non viene impiegata.
- Si può ridurre la frammentazione esterna con la compattazione
  - Si postano i contenuti della memoria per avere tutta la memoria libera contigua a formare un grande blocco.
  - La compattazione è possibile *solo* con la rilocazione dinamica e viene effettuata in fase d'esecuzione.
  - Si possono spostare tutti i processi verso un'estremità della memoria.

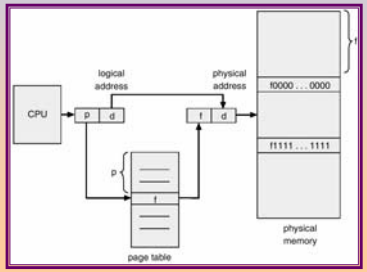
Paginazione

- 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 ***frame*** (la dimensione è una potenza di 2, compresa fra 512 e 8192 byte).
- Si divide la memoria logica in blocchi di pari dimensione detti ***pagine***.
- Si tiene traccia di tutti i frame liberi.
- Per eseguire un programma di dimensione *n* pagine, bisogna trovare *n* frame liberi prima di caricare il programma.
- Si impiega una ***tabella delle pagine*** per tradurre gli indirizzi logici negli indirizzi fisici.
- Si ha solo frammentazione interna (relativa all'ultimo frame).

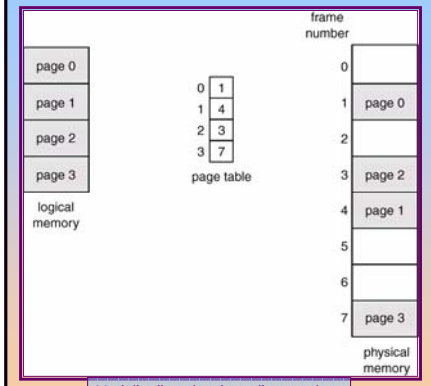
Schema di traduzione degli indirizzi

- L'indirizzo generato dalla CPU viene suddiviso in:
  - ***Numero di pagina (p)*** — impiegato come indice in una tabella di pagine che contiene l'indirizzo base di ciascuna pagina nella memoria fisica.
  - ***Offset nella pagina (d)*** — combinato con l'indirizzo base per definire l'indirizzo fisico di memoria che viene inviato all'unità di memoria.

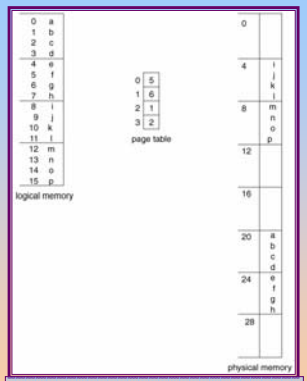
Supporto hardware alla traduzione degli indirizzi.



Esempi di paginazione



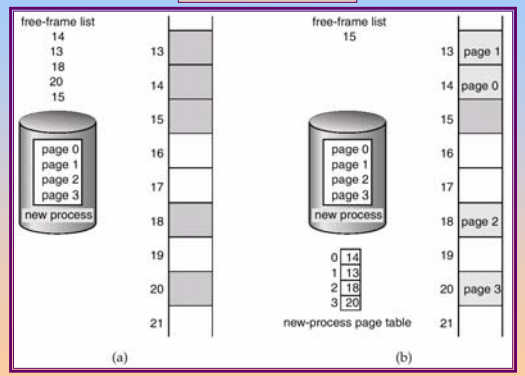
Modello di paginazione di memoria logica e memoria fisica



Esempio di paginazione per una memoria da 32 byte con pagine da 4 byte

Esempi di paginazione

Frame liberi



Prima dell'allocazione

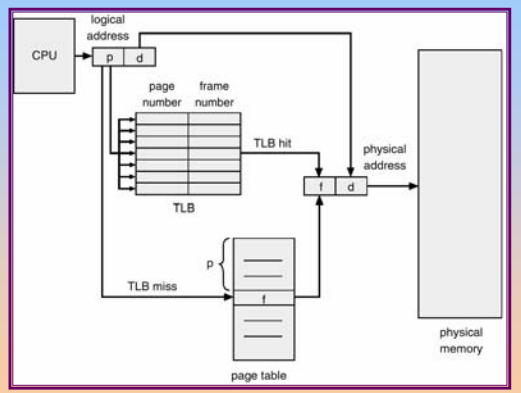
Dopo l'allocazione

Supporto alla tabella delle pagine

- La tabella delle pagine risiede in memoria centrale. Si ha una tabella per ogni processo.
- Il registro ***Page–Table Base Register*** (PTBR) punta all'inizio della tabella.
- Il registro ***Page–Table Length Register*** (PRLR) indica la dimensione della tabella.
- Con questo schema, ogni accesso a dati o istruzioni richiede di fatto due accessi alla memoria: uno per la tabella e uno per le istruzioni/dati.
- Il problema dei due accessi alla memoria può essere risolto per mezzo dei ***registri associativi*** (altrimenti detti *translation look–aside buffer*, TLB) attraverso i quali si effettua una ricerca parallela veloce su una piccola tabella.
- Traduzione dell'indirizzo (A', A'')
  - Se A' si trova nel registro associativo, si estrae il corrispondente # di frame dal registro A'';
  - Altrimenti, occore fare un riferimento di memoria alla tabella delle pagine.

# Pagina	# Frame

Hardware di paginazione con TLB



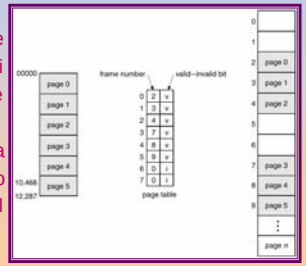
Tempo di accesso effettivo

- *Lookup associativo* = ε unità di tempo.
- Si suppone che un ciclo di memoria abbia la durata di 1 microsecondo.
- *Hit ratio* α — percentuale di volte che un numero di pagina viene reperito nei registri associativi; è correlata al numero di registri associativi.
- Tempo di accesso effettivo (EAT, *Effective Access Time*)

$$EAT = (1 + \varepsilon)\,\alpha + (2 + \varepsilon)(1 - \alpha)$$
$$= 2 + \varepsilon - \alpha$$

Protezione della memoria

- La protezione della memoria è implementata associando un *bit di protezione* a ciascun frame. Vengono impediti accessi non autorizzati (eventualmente si possono avere bit separati per lettura e scrittura).
- Un ulteriore *bit di validità* 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.



Bit di validità in una tabella delle pagine

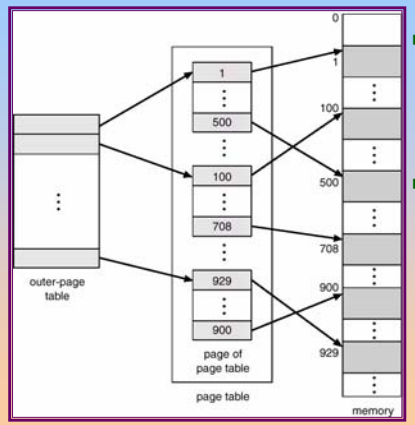
Paginazione a due livelli

- **Paginazione gerarchica** — quando lo spazio degli indirizzi logici è grande, la stessa tabella delle pagine viene paginata. Esempio semplice: **paginazione a due livelli**.
- Un indirizzo logico (in architetture a 32 bit con dimensione della pagina di 4K) viene suddiviso in:
  - un numero di pagina di 20 bit;
  - un offset all'interno della pagina di 12 bit.
- Dato che la tabella di paginazione è paginata, il numero di pagina viene ulteriormente suddiviso in:
  - un numero di pagina di 10 bit;
  - un offset di 10 bit.
- Un indirizzo logico è costituito da:

Numero di pagina		*offset*
p_1	p_2	d
10	10	12

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.

Schema di traduzione degli indirizzi

- Traduzione degli indirizzi per un'architettura a 32 bit con paginazione a due livelli.

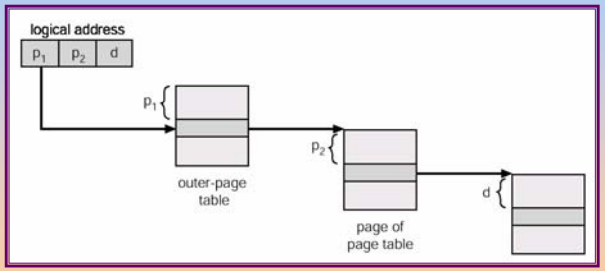
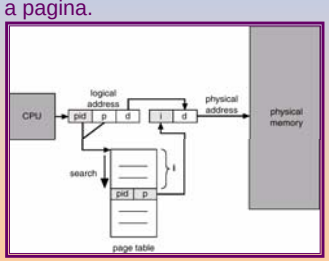


Tabella delle pagine invertita

- Un elemento della tabella per ogni frame.
- Gli elementi della tabella contengono l'indirizzo virtuale della pagina memorizzata nel dato frame, con informazioni sul processo che possiede tale pagina.
- Riduzione della memoria richiesta per memorizzare ciascuna tabella delle pagine; incremento del tempo necessario per ricercare nella tabella quando si ha un riferimento a pagina.
- Notare che è necessario ricercare su tutta la tabella! Si impiegano tabelle hash per limitare la ricerca ad uno — o al più pochi — elementi della tabella.

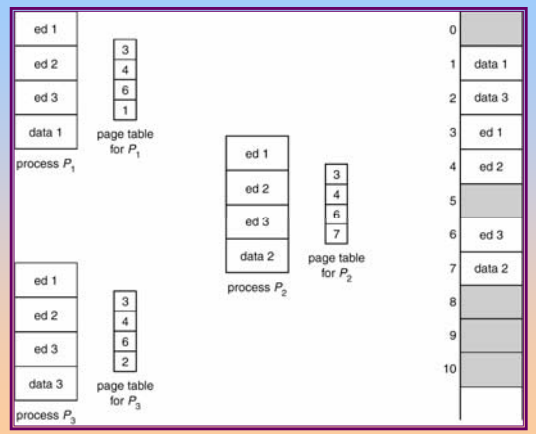


Ciascun indirizzo virtuale è formato dalla tripla:
<# processo, #pagina, offset>

Pagine condivise

- Codice condiviso
  - Una copia di codice a sola lettura (*reentrant*) viene condivisa fra processi (ad esempio: text editor, compilatori, sistemi a finestre).
  - Il codice condiviso deve apparire nella stessa locazione nello spazio degli indirizzi logici di tutti i processi.
- Codice e dati privati
  - Ciascun processo mantiene una copia separata dei dati e del codice.
  - Le pagine di codice e dati privati possono apparire ovunque nello spazio degli indirizzi logici.

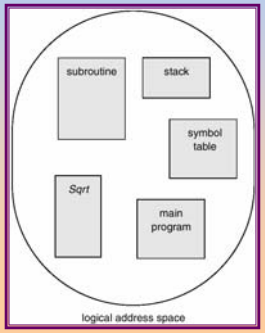
Esempio di pagine condivise



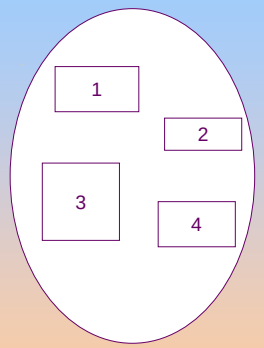
Condivisione di codice in un ambiente di paginazione:
Editor contenuto in tre pagine

Segmentazione

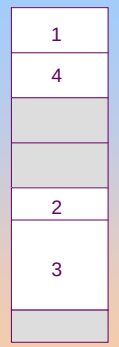
- Schema di gestione della memoria che asseconda la visione utente (l'utente generalmente non pensa alla memoria come ad un array lilneare di byte).
- Un programma è una collezione di segmenti. Un segmento è un'unità logica, come ad esempio:
  - programma principale (main),
  - procedura,
  - funzione,
  - metodo,
  - oggetto,
  - variabili logiche, variabili globali,
  - blocco common,
  - stack,
  - tabella dei simboli, matrici.



Schema logico di segmentazione



Spazio utente



Spazio fisico di memoria

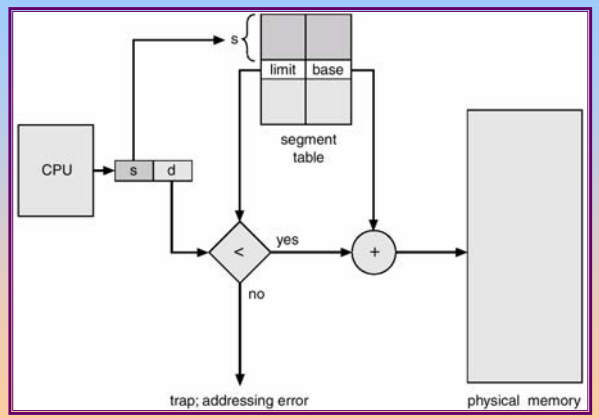
Architettura per la segmentazione

- Gli indirizzi logici sono rappresentati da coppie:
  <#segmento, offset>
- ***Tabella dei segmenti*** — mappa gli indirizzi fisici bidimensionali; ciascun elemento della tabella contiene:
  - *base* — indirizzo fisico iniziale della memoria contenente il segmento.
  - *limite* — specifica la lunghezza del segmento.
- Il registro ***Segment–Table Base Register*** (*STBR*) punta alla locazione in memoria della tabella dei segmenti.
- Il registro ***Segment–Table Length Register*** (*STLR*) indica il numero di segmenti utilizzati dal programma; un numero di segmento *s* è legale se *s* < STLR.

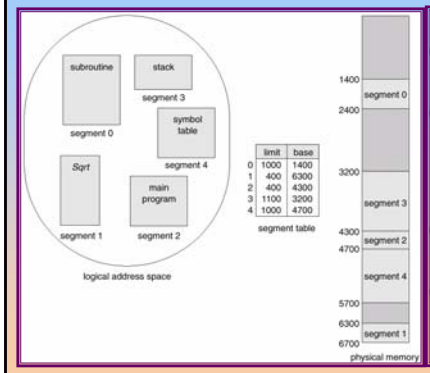
Architettura per la segmentazione

- **Rilocazione**: Dinamica, con tabella dei segmenti.
- **Condivisione**: Si ottengono segmenti condivisi, puntando allo stesso numero di segmento.
- **Allocazione**: First fit/Best fit. Possibile frammentazione esterna.
- **Protezione**: Si associa a ciascun elemento della tabella dei segmenti:
  - bit di validità = 0 ⇒ segmento illegale
  - privilegi read/write/execute
- Bit di protezione associati ai segmenti; la condivisione di codice viene implementata a livello di segmento.
- Dato che i segmenti variano di dimensione, l'allocazione della memoria è dinamica.

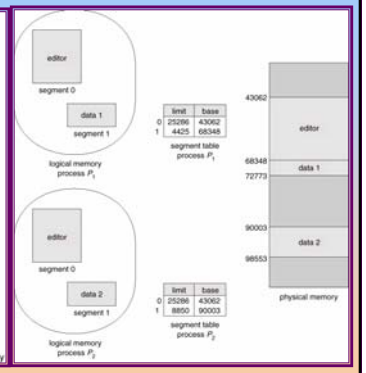
Hardware di segmentazione



Segmentazione



Esempio di segmentazione

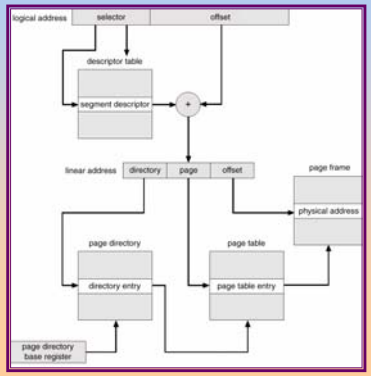


Condivisione di segmenti in un sistema con memoria segmentata

Segmentazione con paginazione — Intel 386

- L'Intel 386 impiega una segmentazione con paginazione per la gestione della memoria con uno schema di paginazione a due livelli.

Schema di traduzione degli indirizzi.



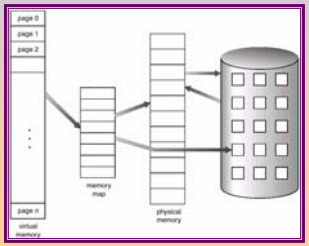
Memoria Virtuale

- Background
- Paginazione su richiesta
- Sostituzione delle pagine
- Algoritmi di sostituzione delle pagine
- Allocazione dei frame
- Thrashing

Background

- ***Memoria virtuale —*** separazione della memoria logica dell'utente dalla memoria fisica.
  - Solo parti del programma devono trovarsi in memoria per l'esecuzione.
  - Lo spazio logico degli indirizzi può essere più grande dello spazio fisico.
  - Porzioni di spazio fisico possono essere condivise da più processi.
- La memoria virtuale può essere implementata per mezzo di:
  - **Paginazione su richiesta**
  - Segmentazione su richiesta

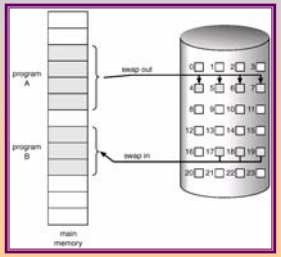
Esempio di spazio degli indirizzi virtuali più esteso dello spazio degli indirizzi fisici.



Paginazione su richiesta

- *Swapper pigro*: porta una pagina in memoria solo quando è necessaria:
  - Occorrono meno operazioni di I/O
  - Viene impiegata meno memoria
  - Si ha una risposta più veloce
  - Si possono gestire più utenti
- Richiesta di una pagina ⇒ si fa un riferimento alla pagina
  - Riferimento non valido ⇒ abort
  - Pagina non in memoria ⇒ trasferimento in memoria

Trasferimento di una memoria paginata su spazio disco contiguo



Bit di validità

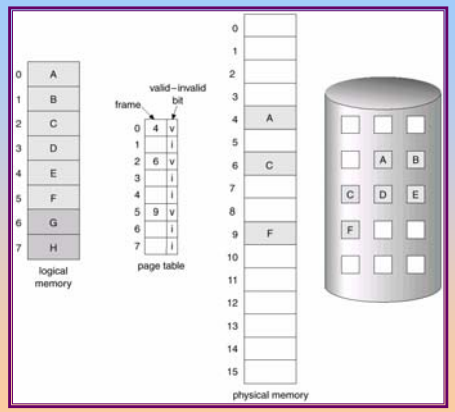
- Un bit di validità viene associato a ciascun elemento della tabella delle pagine (1 ⇒ in memoria, 0 ⇒ non in memoria).
- Inizialmente il bit di validità viene posto a 0 per tutte le pagine.
- Esempio di una configurazione della tabella delle pagine.

# Frame	Bit di validità
	1
	1
	1
	1
	0
⋮	
	0
	0

Tabella delle pagine

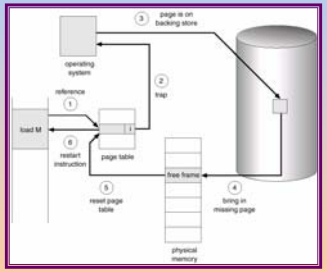
- In fase di traduzione degli indirizzi, se il bit di validità vale 0 ⇒ si ha un ***page fault***.

Istantanea di una tabella delle pagine con pagine non allocate in memoria principale



Page fault

- Se non ci sono mai stati riferimenti ad una pagina, il primo riferimento causa un ***trap*** al SO ⇒ page fault.
- Il SO consulta una tabella per decidere se si tratta di…
  - riferimento non valido ⇒ abort;
  - pagina non in memoria.
- Seleziona un frame vuoto.
- Sposta la pagina nel frame.
- Aggiorna le tabelle (bit di validità = 1).
- Viene riavviata l'istruzione che era stata interrotta.



Gestione del page fault

Cosa accade quando non ci sono frame liberi ?

- *Sostituzione di pagina* — si trova una pagina in memoria che non venga utilizzata attualmente, e si sposta sul disco.
  - Scelta di un algoritmo di selezione;
  - Prestazioni: è richiesto un metodo che produca il minimo numero di page fault.
- La stessa pagina può essere riportata in memoria più volte.
- *Page Fault Rate:* $0 \le p \le 1.0$
  - se $p = 0$ non si hanno page fault;
  - se $p = 1$, ciascun riferimento è un fault.
- Tempo medio di accesso (EAT):

  EAT = (1 – *p*) x t[accesso alla memoria] + *p* x t[page fault]

t[swap out di pagina] + t[swap in di pagina] + overhead di restart

Esempio di paginazione su richiesta

- Tempo di accesso alla memoria = 1 μ*sec*
- Il 50% delle volte che una pagina viene rimpiazzata ha subito delle modifiche e deve essere sottoposta a swap out.
- Tempo di swap per pagina = 10 m*sec* = 10000 μ*sec*

  EAT = (1 – *p*) x 1 + *p* x (15000)

  ≈ 1 + 15000*p* (in μ*sec*)

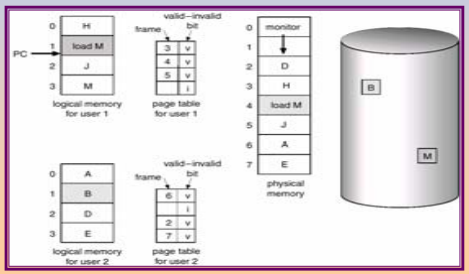
  Nota: Se un accesso su 1000 causa un page fault, EAT = 16 μ*sec* ⇒ con la paginazione su richiesta l'accesso in memoria viene rallentato di un fattore 16.
  Se si desidera un rallentamento inferiore al 10%:

  0.1>15000*p* ⇒ *p* < 0.0000067

  cioè può essere permesso meno di un page fault ogni 150000 accessi in memoria.

Sostituzione delle pagine

- Un bit di modifica (*dirty*) per ridurre il sovraccarico dei trasferimenti di pagine:
  - solo le pagine modificate vengono riscritte sul disco.
- La sostituzione delle pagine completa la separazione fra memoria logica e memoria fisica
  - una grande memoria virtuale può essere fornita ad un sistema con poca memoria fisica.

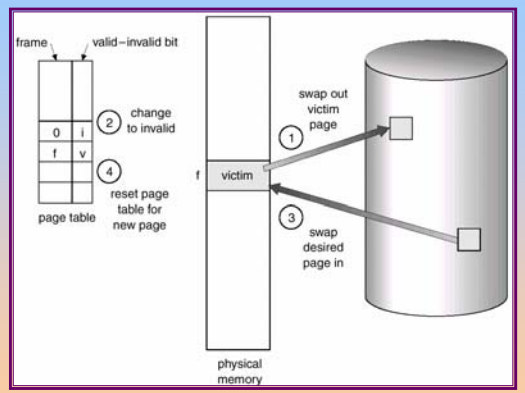


Necessità di trasferimento di una pagina

Sostituzione delle pagine

1. Individuazione della pagina richiesta su disco.
2. Individuazione di un frame libero:
   A. Se esiste un frame libero, viene utilizzato.
   B. Altrimenti viene utilizzato un algoritmo di sostituzione per selezionare un frame *vittima*.
   C. La pagina vittima viene scritta sul disco; le tabelle delle pagine e dei frame vengono modificate conformemente.
3. Lettura della pagina richiesta nel frame appena liberato; modifica delle tabelle delle pagine e dei frame.
4. Riavvio del processo utente.

Sostituzione delle pagine



Algoritmi di sostituzione delle pagine

- È richiesta la minimizzazione della frequenza di page fault.
- Si valutano gli algoritmi eseguendoli su una particolare stringa di riferimenti a memoria (*reference string*) e contando il numero di page fault su tale stringa.
- In tutti gli esempi seguenti, la reference string è

  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

Grafico del numero di page fault rispetto al numero di frame

Algoritmo First–In–First–Out (FIFO)

- Stringa dei riferimenti: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frame (3 pagine per ciascun processo possono trovarsi in memoria contemporaneamente).

1	1	4	5	
2	2	1	3	9 page fault
3	3	2	4	

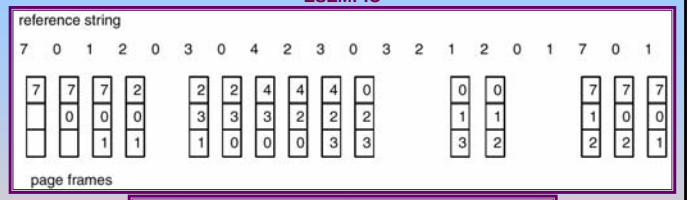
- 4 frame

1	1	5	4	
2	2	1	5	10 page fault
3	3	2		
4	4	3		

- Algoritmo FIFO — ***Anomalia di Belady***
  - ☞ più frame ⇒ più page fault

Sostituzione delle pagine FIFO

ESEMPIO

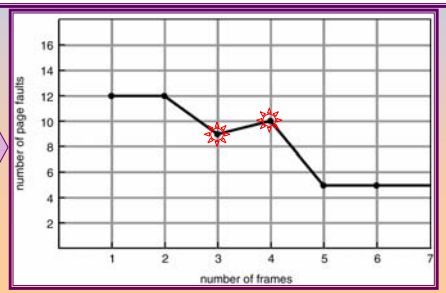


15 fault

Curva dei page fault per sostituzione FIFO

⇓

Anomalia di Belady



Algoritmo ottimo

- Sostituire la pagina che non verrà usata per il periodo di tempo più lungo.
- Esempio: 4 frame, con stringa dei riferimenti

  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

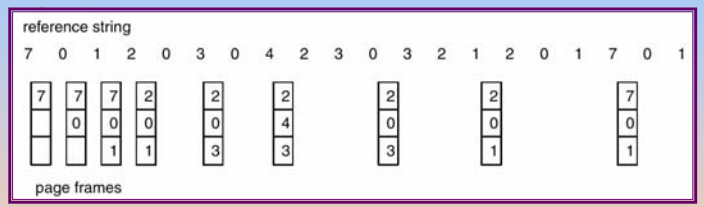
1	4
2	
3	
4	5

6 page faults

- Come si può conoscere l'identità della pagina?
- Di solo interesse teorico: viene impiegato per misurare le prestazioni (comparative) degli algoritmi con valenza pratica.

Sostituzione delle pagine ottima

ESEMPIO



9 fault

Algoritmo Least–Recently–Used (LRU)

- Stringa dei riferimenti:

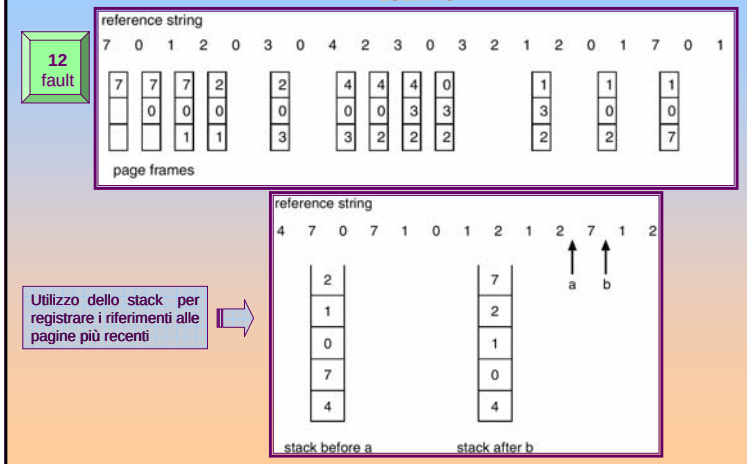
  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1	5	
2		
3	5	4
4	3	

- Implementazione con contatore
  - Ciascuna pagina ha un contatore; ogni volta che si fa riferimento alla pagina si copia l'orologio nel contatore.
  - Quando si deve rimuovere una pagina, si analizzano i contatori per scegliere quale pagina cambiare.
- Implementazione con stack — si tiene uno stack di numeri di pagina in forma di lista doppiamente concatenata:
  - Pagina referenziata:
    - Si sposta in cima allo stack
    - È necessario al più aggiornare 6 puntatori
  - Non è necessario fare ricerche per la scelta.

Sostituzione delle pagine LRU

ESEMPIO

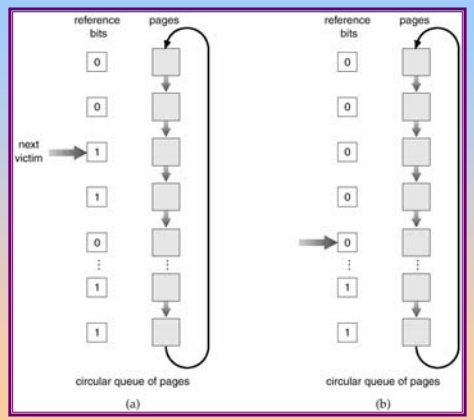


Utilizzo dello stack per registrare i riferimenti alle pagine più recenti

Algoritmi di approssimazione a LRU

- Bit di riferimento
  - A ciascuna pagina si associa un bit, inizialmente = 0.
  - Quando si fa riferimento alla pagina si pone il bit a 1.
  - Si rimpiazza la pagina il cui bit=0 (se esiste). Tuttavia non si conosce l'ordine di accesso alle pagine.
- Seconda Chance
  - È necessario un bit di riferimento.
  - Quando la pagina riceve una seconda chance il bit di riferimento viene azzerato ed il tempo di arrivo aggiornato al tempo attuale.
  - Se la pagina da rimpiazzare (in ordine di clock) ha il bit di riferimento = 1, allora:
    - Si pone il bit di riferimento a 0.
    - Si lascia la pagina in memoria.
    - Si rimpiazza la pagina successiva (in ordine di clock), in base alle stesse regole.

Algoritmo Seconda Chance



Algoritmi con conteggio

- Si tiene un contatore del numero di riferimenti che sono stati fatti a ciascuna pagina.
- Algoritmo LFU (*Least Frequently Used*): si rimpiazza la pagina col valore più basso del contatore.
- Algoritmo MFU (*Most Frequently Used*): è basato sulla assunzione che la pagina col valore più basso del contatore è stata spostata recentemente in memoria e quindi deve ancora essere impiegata.
- L'implementazione è molto costosa; le prestazioni (rispetto all'algoritmo ottimo) scadenti.

Allocazione dei frame

- Ciascun processo richiede un numero minimo di pagine.
- Si hanno due schemi principali di allocazione.
  - Allocazione fissa.
  - Allocazione con priorità.

Allocazione fissa

- Stessa allocazione per ogni processo
  - Esempio: se si dispone di 100 frame e 5 processi, si assegnano 20 pagine a ciascun processo.
- Allocazione proporzionale
  - si allocano frame sulla base della dimensione del processo.

– s_i = size of process p_i
– $S = \sum s_i$
– m = total number of frames
– a_i = allocation for $p_i = \frac{s_i}{S} \times m$

ESEMPIO

$m = 64$
$s_i = 10$
$s_2 = 127$
$a_1 = \frac{10}{137} \times 64 \approx 5$
$a_2 = \frac{127}{137} \times 64 \approx 59$

Allocazione a priorità

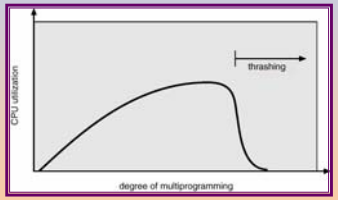
- SI impiega uno schema di allocazione proporzionale impiegando delle priorità piuttosto che la dimensione.
- Se il processo P_i genera un page fault,
  - Si seleziona per la sostituzione uno dei suoi frame;
  - Si seleziona per la sostituzione un frame di un processo con numero di priorità più basso.

Sostituzione globale e locale

- Sostituzione **globale** — il processo seleziona il frame da sostituire dall'insieme di tutti I frame; il processo può selezionare un frame di un diverso processo.
  - ⇒ Un processo non può controllare la propria frequenza di page fault.
- Sostituzione **locale** — ciascun processo seleziona i frame solo dal proprio insieme di frame allocati.
  - ⇒ Non rende dispobinibili a processi che ne facciano richiesta pagine di altri processi scarsamente utilizzate.
- La sostituzione globale garantisce maggior throughput.
  - ⇒ Più utilizzata nei SO più diffusi.

Thrashing

- Se un processo non ha abbastanza pagine, la frequenza di page fault è molto alta:
  - basso impiego della CPU;
  - il SO crede di dover aumentare il grado di multiprogrammazione;
  - si aggiunge un altro processo al sistema!
- **Thrashing** ≡ un processo è costantemente occupato a spostare pagine dal disco alla memoria e viceversa.
- Perché la paginazione funziona?
  Modello di località:
  - Una località è un insieme di pagine che vengono usate attivamente insieme.
  - Il processo passa da una località ad un'altra.
  - Le località possono essere sovrapposte.
- Perché avviene il thrashing?
  dim. località > dim. memoria (disponibile per il processo).

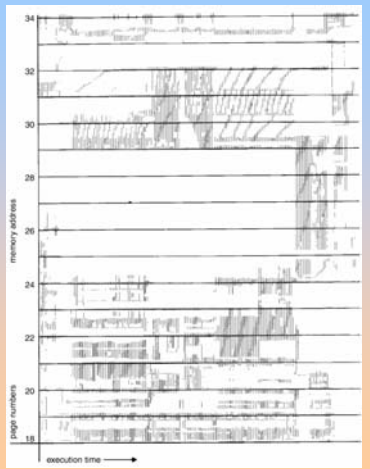


Località in una sequenza di riferimenti a memoria

- Le località hanno una connotazione spazio–temporale.

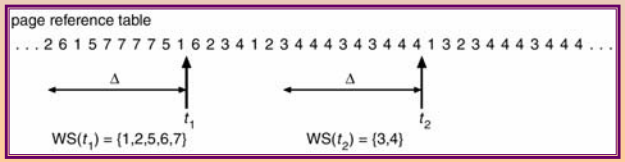
Esempio

- Quando viene chiamata una subroutine, si definisce una nuova località: vengono fatti riferimenti alle sue istruzioni, alle sue variabili locali ed a un sottoinsieme delle variabili globali.
- Quando la subroutine termina, il processo lascia la località corrispondente.
- Le località sono definite dalla struttura del programma e dalle strutture dati relative.



Modello Working–Set

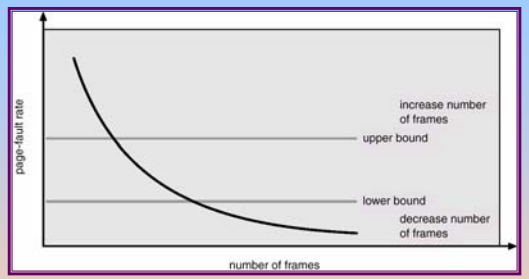
- Δ ≡ finestra di working–set ≡ un numero fisso di riferimenti a pagina; esempio: 10.000 istruzioni.
- WSS_i (working–set del processo P_i) = numero di pagine referenziate nel più recente Δ (varia col tempo):
  - se Δ è troppo piccolo non comprende tutta la località.
  - se Δ è troppo grande comprenderà più località.
  - se $\Delta = \infty \Rightarrow$ comprende l'intero programma.
- $D = \Sigma\ WSS_i$ ≡ numero totale di frame richiesti
- se $D > m$ (numero totale dei frame) $\Rightarrow$ Thrashing
- Politica: se $D > m$, occorre sospendere un processo.



Modello Working–Set

- Problema: la finestra del working–set è una finestra in movimento, con riferimenti che entrano ed escono dal working–set.
- Si approssima con un interrupt del timer e un bit di riferimento
- Esempio: Δ = 10.000
  - Il timer emette un interrupt ogni 5000 unità di tempo.
  - Si tengono in memoria 2 bit per ogni pagina.
  - Quando si ha un interrupt del timer, si copiano i valori di tutti i bit di riferimento e si pongono a 0.
  - Se uno dei bit in memoria è 1 $\Rightarrow$ pagina nel working-set.
- Questo approccio non è completamente accurato.
- Miglioramento: 10 bit e interrupt ogni 1000 unità di tempo.

Frequenza di page fault



- Si stabilisce una frequenza di page fault "accettabile".
  - Se la frequenza effettiva è troppo bassa, il processo rilascia dei frame.
  - Se la frequenza è troppo elevata, il processo acquisisce nuovi frame.

Altre considerazioni

- **Prepaginazione**: Si portano in memoria tutte le pagine richieste in una volta sola. Ad esempio si può memorizzare il working–set al momento della sospensione per I/O per poi riprendere tutte le pagine che gli appartengono.
- Criteri per la determinazione della **dimensione delle pagine**:
  - frammentazione
  - località
    - ⇒ pagine di piccole dimensioni
  - dimensione della tabella delle pagine
  - sovraccarico (overhead) di I/O
    - ⇒ pagine di grandi dimensioni

Altre considerazioni

- **Struttura dei programmi**
  - **A[ ] [ ] = new int[1024] [1024];**
  - Ciascuna riga viene memorizzata in una pagina
  - Programma 1

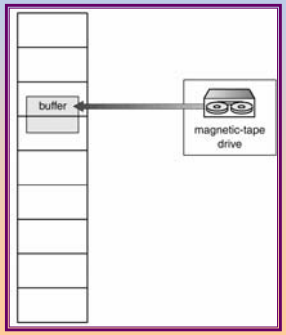
```
for (j = 0; j < A.length; j++)
    for (i = 0; i < A.length; i++)
        A[i,j] = 0;
```

1024 x 1024 page fault

  - Programma 2

```
for (i = 0; i < A.length; i++)
    for (j = 0; j < A.length; j++)
        A[i,j] = 0;
```

1024 page fault

Altre considerazioni

- **I/O Interlock** — Talvolta, occorre permettere ad alcune pagine di rimanere *bloccate* in memoria.
- Esempio: Le pagine utilizzate per copiare un file da un device di I/O devono essere bloccate, affinché non possano essere selezionate come vittime da un algoritmo di sostituzione.