Memoria Virtuale

Memoria virtuale

- Vantaggi offerti dalla memoria virtuale
- Concetti di paginazione su richiesta, algoritmi di sostituzione di pagina e allocazione dei frame

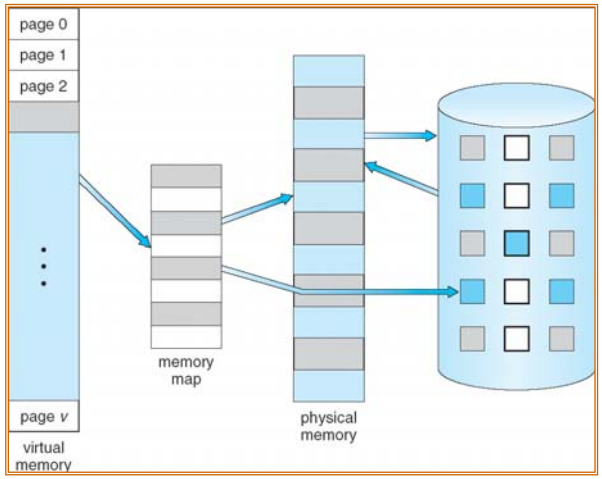
Processo in memoria

- Istruzioni in memoria prima di essere eseguite. Condizione necessaria e ragionevole…
- Codice per condizioni d'errore
- Tabelle e array sovradimensionati
- Opzioni e caratteristiche utili raramente
- Soluzione: caricamento dinamico? Parziale e a carico del programmatore

Memoria virtuale

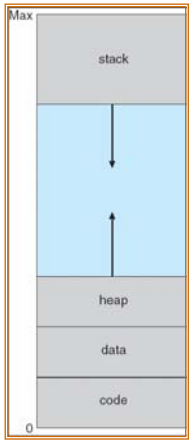
- Completa la separazione della memoria logica dalla memoria fisica.
  - Solo una parte del programma necessita di essere in memoria per l'esecuzione.
  - Lo spazio di indirizzamento logico può quindi essere più grande dello spazio di indirizzamento fisico.
  - Gli spazi di indirizzamento virtuali possono essere condivisi da diversi processi.
  - Maggiore efficienza nella creazione dei processi.
- La memoria virtuale può essere implementata attraverso:
  - Paginazione su richiesta (demand paging).
  - Segmentazione su richiesta (demand segmentation).

Memoria virtuale più grande della memoria fisica

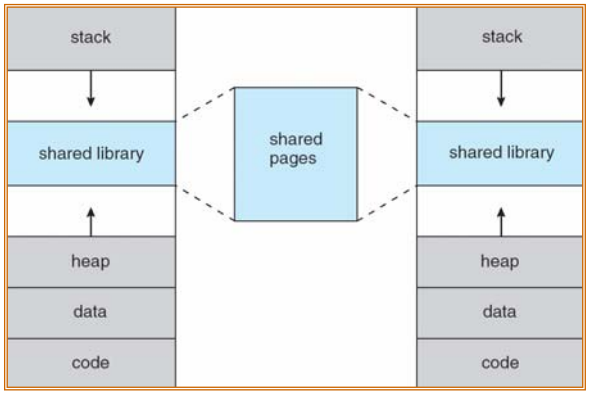


Spazio di indirizzamento virtuale

Lo spazio tra l'heap e lo stack è spazio di indirizzamento virtuale del processo, ma richiede pagine fisiche realmente esistenti solo nel caso che l'heap o lo stack crescano



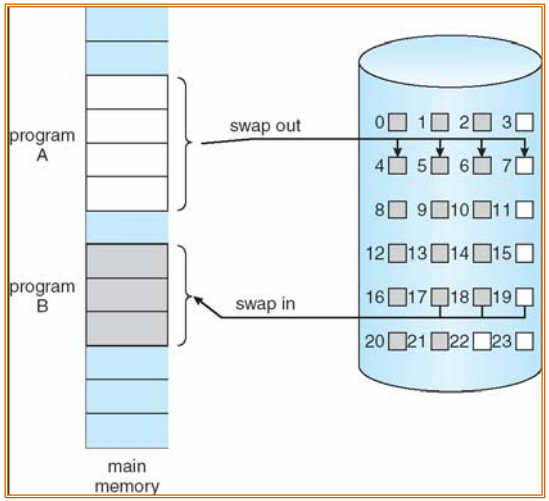
Libreria condivisa usando la memoria virtuale



Paginazione su richiesta

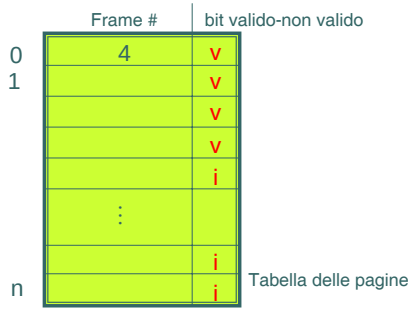
- Introdurre una pagina in memoria solo se necessario
  - Meno I/O
  - È necessaria meno memoria.
  - Risposta più veloce.
  - Più utenti.
- La pagina è necessaria ⇒ riferimento ad essa:
  - riferimento non valido ⇒ termine del processo.
  - non in memoria ⇒ portare in memoria.

Trasferimento di pagine dal disco



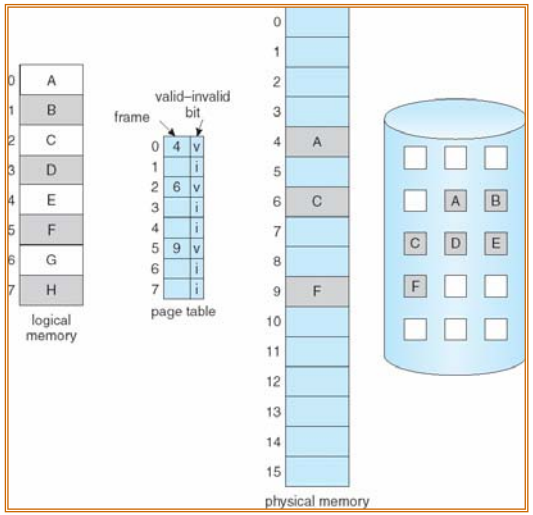
Bit di validità

- Ad ogni elemento della tabella delle pagine è associato un bit (v ⇒ in memoria, i ⇒ non in memoria).
- Inizialmente il bit valido-non valido è impostato a i per tutti gli elementi.
- Esempio di un'istantanea della tabella delle pagine.



- Durante la traduzione dell'indirizzo, se il bit è i nell'elemento della tabella delle pagine ⇒ mancanza di pagina (page fault).

Memoria e disco



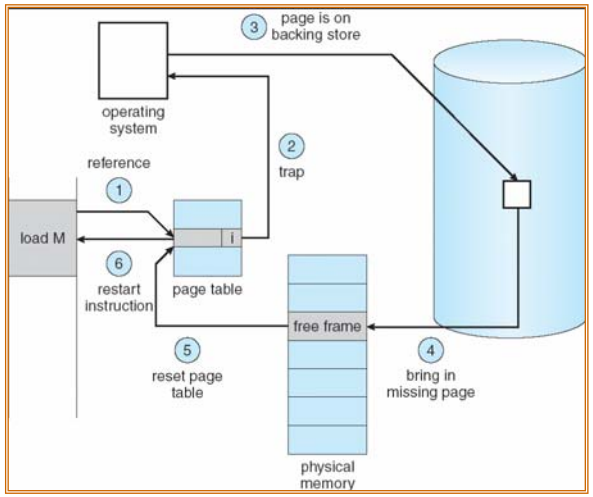
Mancanza di pagina

- Quando avviene un riferimento ad una nuova pagina, il primo riferimento provoca una trap al sistema operativo:

  page fault

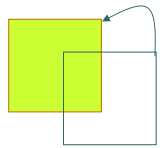
- Il sistema op. esamina una tabella interna per decidere:
  - Riferimento non valido ⇒ termine del processo.
  - Riferimento valido. Pagina non in memoria.
- Cercare un frame libero.
- Spostare la pagina nel frame.
- Modificare la tabella della pagine, validità del bit = v.
- Riavviare l'istruzione che ha causato il page fault

Passi necessari per gestire un page fault



Istruzioni parzialmente eseguite

- Riavvia l'istruzione.
- Add



- MVC (move character)

Prestazione della paginazione su richiesta

- Probabilità di page fault $0 \le p \le 1$
  - se $p = 0$ non ci sono mancanze di pagina;
  - se $p = 1$, ogni riferimento è una mancanza di pagina.

- Tempo di accesso effettivo (EAT)

  EAT = (1 – p) x accesso alla memoria
  + p (overhead page fault
  + swap out
  + swap in
  + overhead ripresa)

Gestione di un page fault: passi

1. Eccezione
2. Salvataggio registri e stato
3. Verifica interruzione dovuta a mancanza pagina
4. Determinazione locazione su disco
5. Lettura da disco – attese: coda/latenza/posizionamento
6. Allocazione CPU
7. Interruzione controller disco
8. Salvataggio registri e stato
9. Verifica interruzione dal disco
10. Aggiornamento tabelle
11. Attesa CPU nuovamente allocata
12. Recupero registri utente, stato e ripresa

Esempio di paginazione su richiesta

- Tempo di accesso alla memoria = 200 nanosecondi
- Tempo medio di gestione di un page fault = 8 millisecondi
- EAT = (1 – p) x 200 + p (8 milliseconds)
  = (1 – p) x 200 + p x 8,000,000
  = 200 + p x 7,999,800
- Se un accesso su 1000 causa un page fault, allora
  EAT = 8.2 microsecondi.

Il tempo di accesso effettivo è 40 volte superiore al tempo di accesso alla memoria centrale!

Creazione di processi

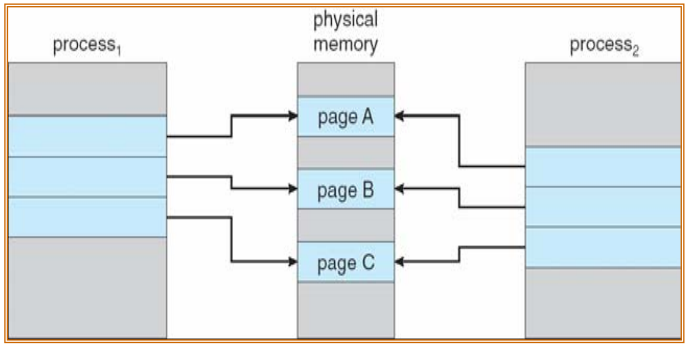
- La memoria virtuale offre altri benefici durante la creazione del processo:

  - Copia su scrittura (copy-on-write)

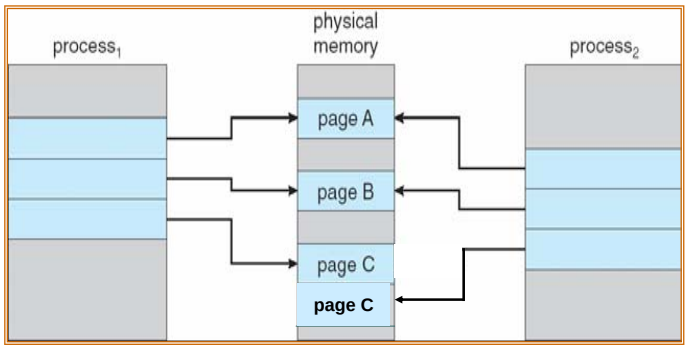
Copia su scrittura

- La copia su scrittura permette ai processi padre e figlio di *condividere* inizialmente le stesse pagine in memoria. Quando uno dei due processi scrive in una pagina condivisa, allora viene creata una copia della pagina condivisa.
- La copia su scrittura fornisce una maggiore efficienza nella creazione dei processi, poichè solo le pagine modificate vengono copiate.
- Le pagine libere sono allocate da *pool* di pagine *riempite con zero.*

Prima che $Process_2$ modifichi C



Dopo che $Process_2$ ha modificato C



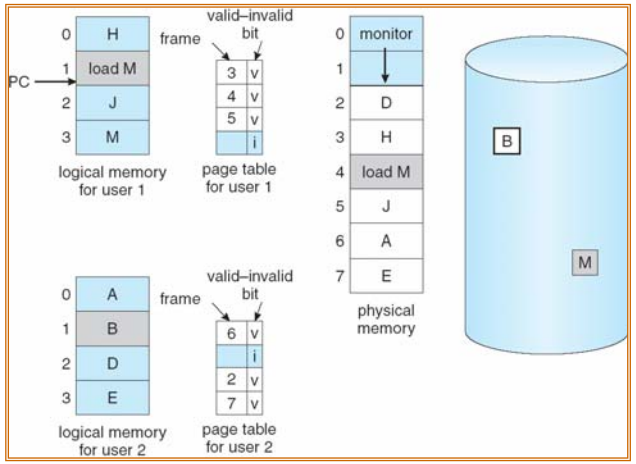
Sovrallocazione

- Cosa succede se non ci sono frame liberi?
- Sostituzione di pagina – trovare alcune pagine in memoria, ma non veramente in uso, e spostarle sul disco.
  - algoritmo
  - prestazione – l'algoritmo deve minimizzare il numero di page fault
- La stessa pagina può essere portata in memoria più volte.

Sostituzione di pagina

- Modifica della procedura di gestione di page fault per includere la sostituzione di pagina.
- Architettura: *bit di modifica* (modify bit) per ridurre il dispendio di trasferimento di pagine – solo le pagine modificate sono scritte su disco.
- La sostituzione della pagina completa la separazione fra memoria logica e memoria fisica.

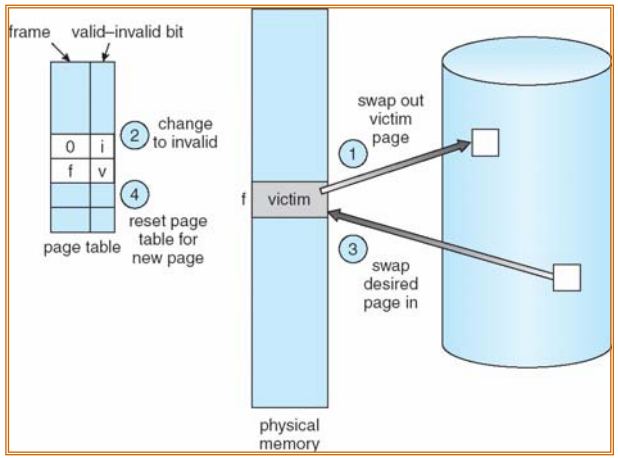
Necessità di sostituzione pagina



Sostituzione di base della pagina

1. Individuare la posizione della pagina desiderata sul disco.
2. Trovare un frame libero:
   - se c'è un frame libero, usarlo;
   - se non c'è nessun frame libero, usare l'algoritmo di sostituzione delle pagine per selezionare un frame vittima.
3. Leggere la pagina desiderata nel frame libero; aggiornare le tabelle dei frame e delle pagine.
4. Riprendere il processo.

Sostituzione dalla pagina



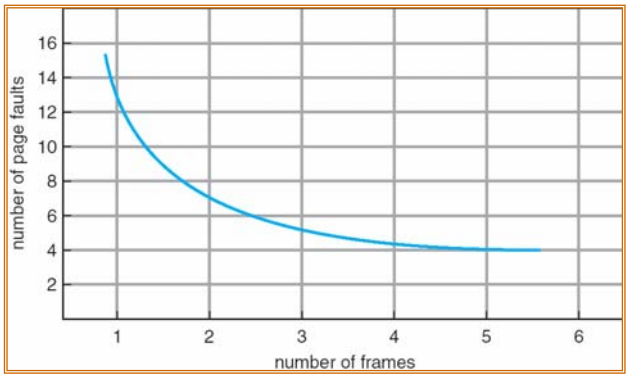
Algoritmo di sostituzione delle pagine

- Si desidera il più basso tasso possibile di page fault.
- Un algoritmo viene valutato facendolo operare su una particolare stringa di riferimenti alla memoria e calcolando il numero di page fault.
- In tutti i nostri esempi la stringa di riferimenti è:

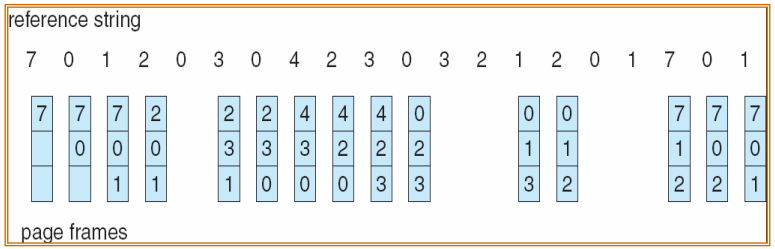
  7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1.

- Numero dei frame

Mancanze di pagina e numero di frame

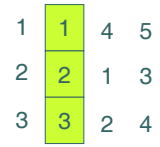


Sostituzione FIFO della pagina



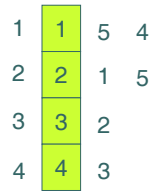
Algoritmo FIFO

- Stringa di riferimenti: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.
- 3 frame



9 mancanze di pagina

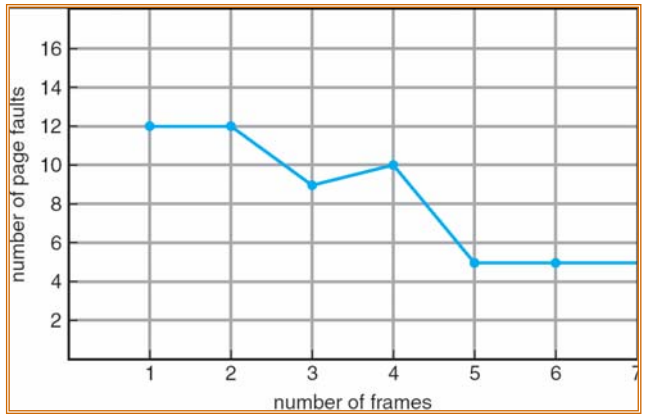
- 4 frame



10 mancanze di pagina

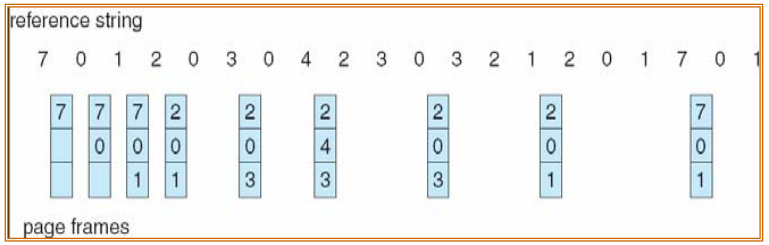
- Sostituzione FIFO – Anomalia di Belady
  - più frame ⇒ più mancanze di pagina.

FIFO: l'anomalia di Belady



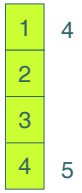
Sostituzione ottimale della pagina

Sostituisce la pagina che non sarà usata per più tempo.



Algoritmo ottimale

- Esempio con 4 frame:
  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

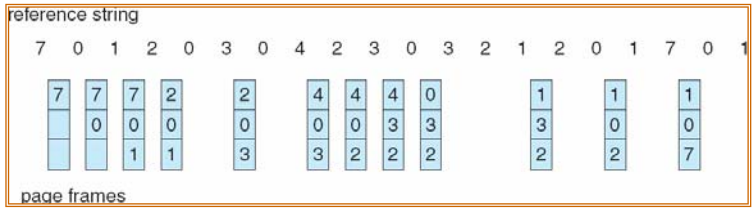


6 mancanze di pagina

- Come avere questa informazione?
- Usato per misurare quanto sono buone le prestazioni dell'algoritmo.

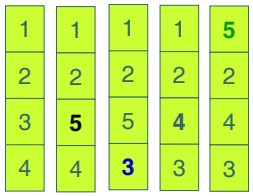
Algoritmo LRU (least recently used)

Sostituisce la pagina che non è stata usata per più tempo.



Implementazione LRU

Stringa di riferimento: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

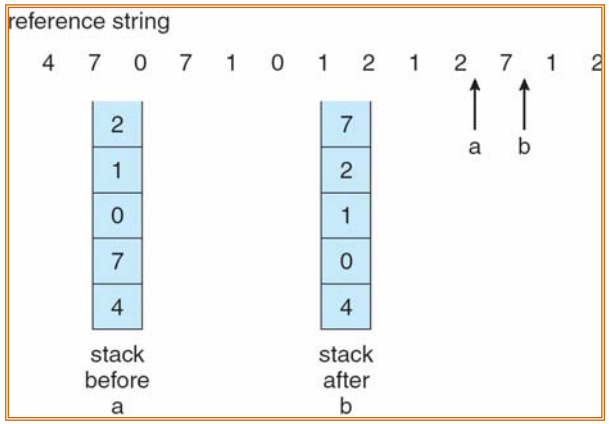


- Implementazione del contatore.
  - Ogni volta che si fa riferimento ad una pagina copia l'ora nel contatore della tabella delle pagine per quella pagina.
  - Quando una pagina deve essere sostituita si guardano i contatori.

Implementazione LRU

- Implementazione dello stack – mentenere uno stack dei numeri di pagina in una lista a doppio collegamento:
  - Riferimento ad una pagina:
    - mettere la pagina in cima allo stack;
    - richiede di cambiare al più 6 puntatori.
  - Nessuna ricerca per la sostituzione.

Uso di uno stack per registrare i riferimenti alle pagine usate più di recente



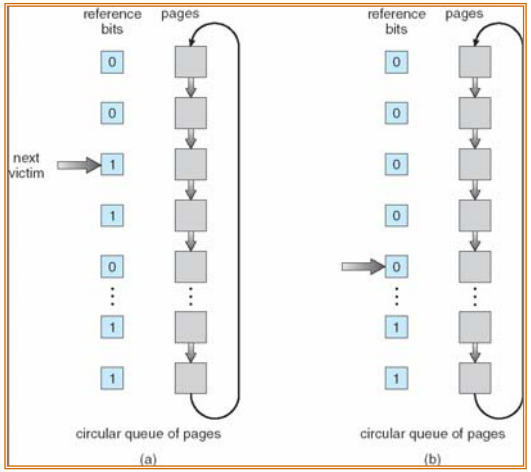
Approssimazione dell'algoritmo LRU

- Architettura: bit di riferimento.
  - Ad ogni pagina è associato un bit, inizialmente = 0.
  - Quando la pagina è referenziata il bit è impostato a 1.
  - Rimpiazzare la pagina che è a 0 (se ne esiste una).

Algoritmo della seconda chance

- Seconda possibilità.
  - Le pagine sono disposte in una lista circolare
  - Quando occorre selezionare una pagina vittima inizia la scansione della lista:
    - se una pagina ha il bit di riferimento a 1 lo si pone a 0 e si passa alla successiva (la pagina rimane in memoria);
    - altrimenti si seleziona per essere sostituita.

Algoritmo della seconda chance



Altre approssimazioni

- Raffinamento dell'algoritmo della seconda chance

  (bit riferimento, bit modifica)

- Più bit di riferimento (e.g., tenuti in un registro a 8 bit). Periodicamente, allo scadere di un intervallo temporale, il registro viene shiftato a destra di un bit, e il bit di riferimento della pagina viene copiato nel bit più significativo del registro.

Algoritmi di conteggio

- Tenere un contatore del numero di riferimenti che sono stati fatti ad ogni pagina.
- Algoritmo LFU: sostituisce la pagina con il più basso conteggio.
- Algoritmo MFU: sostituisce la pagina con il conteggio più alto.

Bufferizzazione delle pagine

- Pool di frame liberi per soddisfare le richieste velocemente
- Pagine modificate scritte sul disco periodicamente in background
- Ricerca nel pool dei frame liberi in memoria di una pagina precedentemente sostituita e nuovamente necessaria. Probabilmente ancora in memoria e non sovrascritta se il frame non è stato riallocato
- Problemi della bufferizzazione con determinate applicazioni, e.g., database etc … che possono usare un'area apposita del disco non bufferizzata del SO (raw disk)

Allocazione dei frame

- Ogni processo ha bisogno di un numero **minimo** di pagine.
- Esempio: IBM 370 – 6 pagine per gestire l'istruzione SS MOVE:
  - L'instruzione richiede 6 byte, che possono estendersi su 2 pagine.
  - 2 pagine per gestire il "from" dell'istruzione.
  - 2 pagine per gestire il "to" dell'istruzione.
- Due principali schemi di allocazione.
  - Allocazione fissa.
  - Allocazione a priorità.

Allocazione fissa

- Allocazione omogenea – per esempio, se 100 frame e 5 processi, ognuno prende 20 pagine.
- Allocazione proporzionale – si assegna la memoria disponibile ad ogni processo in base alle dimensioni di quest'ultimo.

$s_i = \text{size of process } p_i$

$S = \sum s_i$

$m = \text{total number of frames}$

$a_i = \text{allocation for } p_i = \frac{s_i}{S} \times m$

$m = 64$

$s_i = 10$

$s_2 = 127$

$a_1 = \frac{10}{137} \times 64 \approx 5$

$a_2 = \frac{127}{137} \times 64 \approx 59$

Allocazione a priorità

- Usare uno schema di allocazione proporzionale basato sulle priorità piuttosto che la dimensione.
- Se il processo P_i genera un page fault,
  - selezionare per la sostituzione uno dei suoi frame.
  - selezionare per la sostituzione un frame da un processo con un numero di priorità più basso.

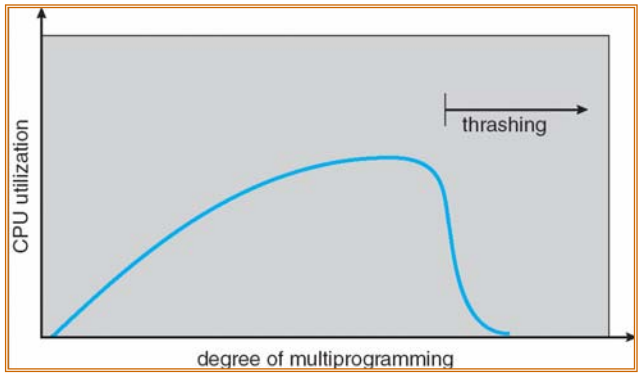
Allocazione globale e locale

- **Sostituzione globale** – permette ad un processo di selezionare un frame di sostituzione a partire dall'insieme di tutti i frame, anche se quel frame è correntemente allocato a qualche altro processo, i.e., un processo può prendere un frame da un altro.
- **Sostituzione locale** – ogni processo effettua la scelta solo nel proprio insieme di frame allocati.

Thrashing

- Se un processo non ha abbastanza pagine, il tasso di page fault è molto alto. Questo comporta:
  - basso utilizzo della CPU;
  - il sistema operativo ritiene che sia necessario aumentare il livello di multiprogrammazione;
  - un altro processo aggiunto al sistema.
- **Thrashing** ≡ un processo spende più tempo nella paginazione che nella propria esecuzione.

Thrashing



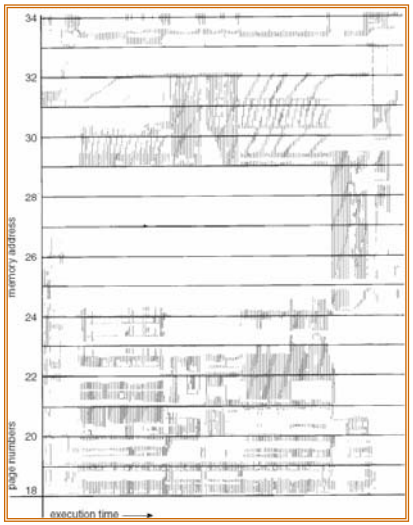
Paginazione e Thrashing

- Perchè la paginazione funziona?

  Modello di località:
  - il processo si muove da una località all'altra,
  - le località possono sovrapporsi.

- Perchè si verifica il trashing?
  Σ dimensione delle località > dimensione totale della memoria.

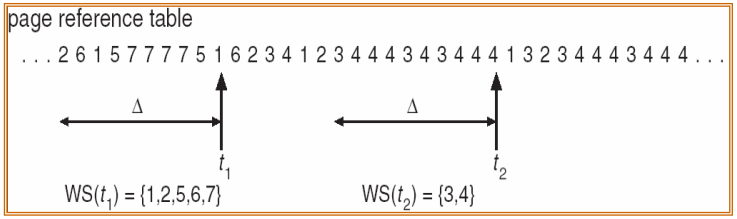
Località in una sequenza di riferimenti alla memoria



Il modello del working set

- $\Delta \equiv$ finestra di working-set $\equiv$ un numero fisso di riferimenti di pagina.
  Esempio: 10,000 istruzioni.
- WSS_i (working set del processo P_i) =
  numero totale di pagine con riferimenti nel più recente Δ (varia nel tempo);
  - se Δ è troppo piccolo non comprenderà l'intera località.
  - se Δ è troppo grande può sovrapporre parecchie località.
  - se $\Delta = \infty \Rightarrow$ il working set è l'insieme delle pagine toccate durante l'esecuzione del processo..
- $D = \Sigma\ WSS_i \equiv$ richiesta globale dei frame.
- Se $D > m \Rightarrow$ thrashing.
- Se $D >$ m, allora occorre sospendere uno dei processi.

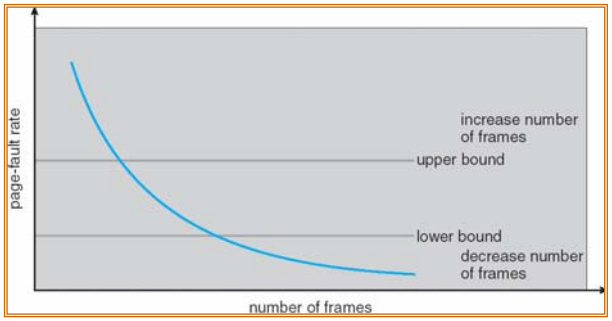
Il modello working set



Mantenere traccia del working set

- Si può approssimare il modello del working set con un interrupt a intervalli fissi di tempo generati da un temporizzatore e un bit di riferimento.
- Esempio: Δ = 10,000
  - Interrupt ogni 5000 riferimenti.
  - Tenere in memoria 2 bit per ogni pagina.
  - Ogni volta che si riceve l'interrupt del temporizzatore si copiano e si azzerano i valori del bit di riferimento per ogni pagina.
  - Se uno dei bit è uguale a 1 $\Rightarrow$ la pagina è nel working set.
- Perchè non è del tutto preciso?
- Incremento = 10 bit e interrupt ogni 1000 riferimenti.

Frequenza di page fault

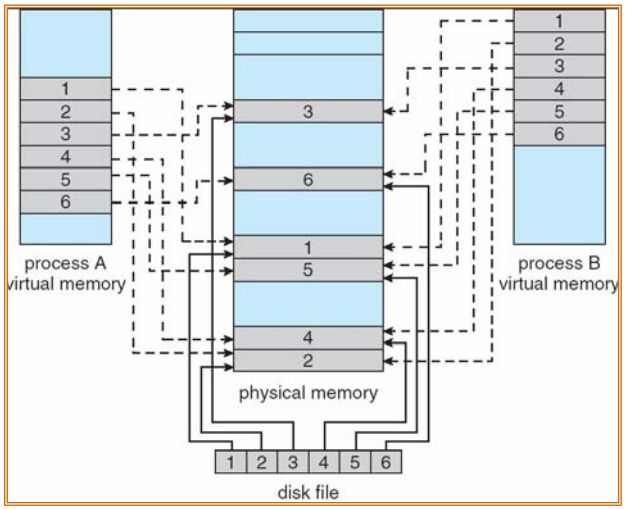


- Stabilire un tasso accettabile di page fault:
  - Se il tasso attuale è troppo basso, il processo può avere troppi frame.
  - Se il tasso attuale è troppo alto, il processo ha bisogno di più frame.

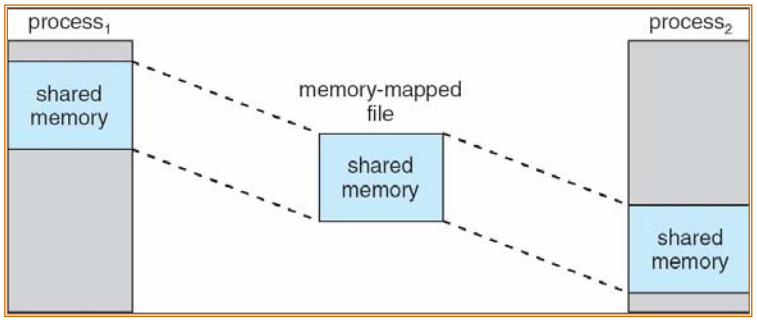
File mappati in memoria

- Un file mappato in memoria permette di trattare l' I/O sul file attraverso accessi normali alla memoria, *mappando* un blocco del disco in una pagina in memoria.
- L'accesso iniziale al file procede con la richiesta di paginazione. Una porzione del file delle dimensioni di una pagina viene letta dal file system in un frame. Le successive letture e scritture del file sono trattate come accessi ordinari alla memoria.
- Semplifica l'accesso al file, gestendo l'I/O tramite la memoria piuttosto che tramite le chiamate di sistema read() e write()
- Più processi possono mappare lo stesso file in memoria. In tal modo, le pagine possono essere condivise.

File mappati in memoria



Shared Memory in Windows



Shared Memory – API Win32

```
#include <windows.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
HANDLE hFile, hMapFile;
LPVOID mapAddress;

hFile = CreateFile("temp.txt",              // first create/open the file
                   GENERIC_READ | GENERIC_WRITE,
                   0, NULL,OPEN_ALWAYS,
                   FILE_ATTRIBUTE_NORMAL, NULL);

hMapFile = CreateFileMapping(hFile, // now obtain a mapping for it
                             NULL, PAGE_READWRITE,
                              0,0, TEXT("SharedObject"));
// now establish a mapped viewing of the file
mapAddress = MapViewOfFile(hMapFile,FILE_MAP_ALL_ACCESS,0,0,0);

// write to shared memory
sprintf(mapAddress,"%s","Shared memory message");

UnmapViewOfFile(mapAddress); // remove the file mapping

CloseHandle(hMapFile); // close all handles
CloseHandle(hFile);
}
```

Shared Memory – API Win32

```
#include <stdio.h>
#include <windows.h>

int main(int argc, char *argv[]) {
HANDLE hMapFile;
LPVOID lpMapAddress;

hMapFile = OpenFileMapping(FILE_MAP_ALL_ACCESS,         // read/write
permission
                   FALSE,                   // Do not inherit the name
                   TEXT("SharedObject")); // of the mapping object.


lpMapAddress = MapViewOfFile(hMapFile,           // handle to mapping object
                  FILE_MAP_ALL_ACCESS, // read/write permission
                  0,                   // max. object size
                  0,                   // size of hFile
                  0);                  // map entire file


printf("%s\n",lpMapAddress);

UnmapViewOfFile(lpMapAddress);
CloseHandle(hMapFile);

}
```

I/O Mappato in Memoria

- Di solito istruzioni di I/O trasferiscono dati tra i registri dei dispositivi e la memoria del sistema
- Molte architetture di computer forniscono I/O mappato in memoria (memory-mapped I/O)
- In questi casi, alcuni range di indirizzi di memoria sono messi da parte e "mappati" su registri dei dispositivi.
- Letture e scritture in queste locazioni corrispondono a letture e scritture di dati nei registri dei dispositivi
- Meccanismo appropriato per dispositivi veloci (e.g., controller del video)

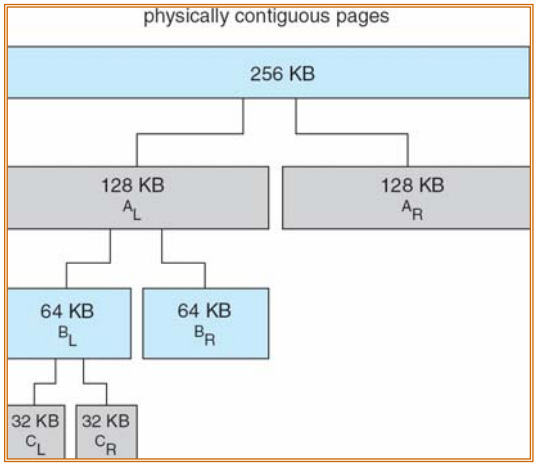
Allocazione della Memoria Kernel

- Gestita differentemente dalla memoria utente
- Spesso allocata da un pool di aree di memoria libere
  - Il kernel richiede memoria per strutture dati di varie taglie
  - Alcune aree di memoria kernel debbono essere contigue

Sistema Buddy

- Alloca memoria da un segmento di taglia fissa, contenente pagine fisicamente contigue
- Memoria allocata usando un allocatore che:
  - Soddisfa richieste in unità che sono potenze di 2.
  - Le richieste sono arrotondate per accesso alla più vicina potenza di 2.
  - Quando viene richiesto meno spazio di quanto disponibile, l'area disponibile viene divisa in due parti aventi per taglia la precedente potenza di 2.
    - Si continua fino a quando si ottiene un pezzo di taglia appropriata

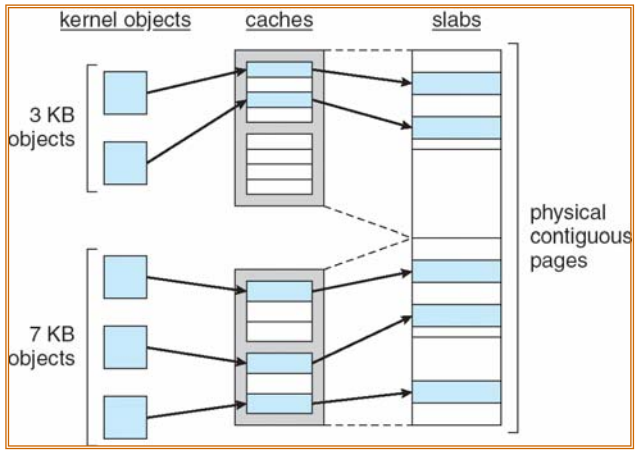
Sistema Buddy



Sistema Slab

- Strategia alternativa
- Uno **slab** è una o più pagine fisicamente contigue
- Una **cache** consiste di uno o più slab
- Una cache singola viene usata per ciascuna struttura dati del kernel
  - Ciascuna cache viene riempita con **oggetti** – istanze della struttura dati
- Quando la cache viene creata, viene riempita con oggetti marcati **free**
- Quando le strutture vengono memorizzate, gli oggetti divengono **used**
- Se uno slab è pieno di oggetti **used**, il prossimo oggetto viene allocato da uno slab vuoto
  - Se non ci sono slab vuoti, un nuovo slab viene allocato
- Benefici: mancanza di frammentazione, risposta veloce alle richieste di memoria

Sistema Slab



Altre considerazioni – Prepaginazione

- Prepaginazione
  - Per ridurre il numero di page fault che si verificano all'avvio di un processo
  - Prepagina tutti o alcune delle pagine di cui il processo avrà bisogno, prima che vengano referenziate
  - Ma se le pagine prepaginate non vengono usate, c'è spreco di memoria e I/O
  - Si assuma che *s* pagine sono prepaginate e *a* di esse vengono usate
    - *É il costo di s * a page fault > o < del costo di prepaginazione di s * (1- a) pagine inutili?*
    - *Se a* è vicino a zero ⇒ la prepaginazione perde

Altre considerazioni – Taglia della pagina

- La selezione della taglia della pagina deve tener conto di:
  - Frammentazione
  - Taglia della tabella
  - Overhead di I/O
  - Località

Altre considerazioni - TLB

- **Estensione della TLB** – quantità di memoria accessibile dalla TLB.
- Estensione della TLB = (dimensione TLB) X (dimensione pagina).
- Idealmente il working set di un processo è memorizzato nella TLB. Altrimenti c'è un numero elevato di page fault.

Estensione della TLB

- **Incremento della dimensione della pagina**. Può portare ad un incremento della frammentazione poichè non tutte le applicazioni richiedono una dimensione grande di pagina.
- **Fornire pagine di diverse taglie**. Permette alle applicazioni che necessitano di pagine di maggiori dimensioni l'opportunità di usarle senza che aumenti la frammentazione.

Altre considerazioni - Programmi

- Struttura del programma:
  - **int A[][] = new int[1024][1024];**
  - Ogni riga è memorizzata in una pagina.
  - Programma 1

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

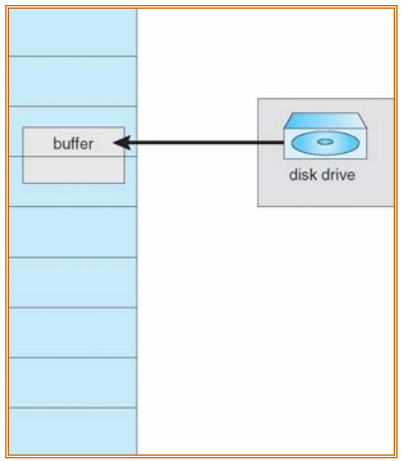
    1024 x 1024 mancanze di pagina.

  - Programma 2

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

    1024 mancanze di pagina.

Ragione per cui i frame usati per operazioni di I/O devono essere in memoria.



Esempi di sistemi operativi

- Windows XP.
- Solaris.

Windows XP

- Usa la paginazione su richiesta con **clustering**. Gestisce i page fault caricando non solo la pagina su cui è avvenuto il page fault ma anche pagine vicine.
- Ad ogni processo è assegnato un **working set minimo** ed un **working set massimo**.
- Il working set minimo è il numero minimo di pagine garantite al processo.
- Ad un processo possono essere assegnate tante pagine quanto il suo working set massimo.
- Quando la quantità di memoria libera scende sotto la soglia, l'**automatic working set trimming** ristabilisce il valore sopra la soglia.
- Il working set trimming rimuove pagine dai processi che hanno pagine in eccesso rispetto al loro working set minimo.

Solaris

- Mantiene una lista di pagine libere da assegnare ai processi che falliscono.
- **Lotsfree** – parametro associato alla lista delle pagine libere.
- La paginazione è eseguita attraverso il processo di *pageout*.
- Il pageout esamina le pagine con un algoritmo dell'orologio modificato.
- **Scanrate** è il tasso di esplorazione delle pagine. Varia da **slowscan** a **fastscan**.
- La frequenza di invocazione di pageout dipende dalla quantità di memoria libera disponibile.

Esplorazione delle pagine in Solaris