> Invece che caricare un intero processo sulla RAM, si caricano solamente alcune pagine (solitamente quelle iniziali), sfruttando poi la località dei processi. Questo perché lo spazio di indirizzamento logico è molto più grande di quello fisico. La memoria virtuale consente ai sistemi operativi di fornire l'illusione di una quantità di memoria molto maggiore di quella fisicamente disponibile, consentendo l'esecuzione efficiente di un gran numero di processi contemporaneamente.

L'esecuzione parziale di un programma in memoria ha molti benefici:

- Un programma non è più limitato dalla memoria fisica
- Più programmi possono essere eseguiti contemporaneamente
- Meno I/O è richiesta per caricare o scambiare programmi in memoria, quindi ogni programma dell'utente viene eseguita più velocemente.

Demand Paging

Il demand paging è un sistema simile a quello di paginazione con scambi. Quando si vuole eseguire un processo, si scambia il suo posto in memoria con quello di un altro. Al posto di scambiare l'intero processo, si usa un lazy swapper, che non scambia mai una pagina in memoria a meno che quella pagina sia richiesta. Quindi nel demand paging non si usa uno "swapper" ma un "pager".

> swap out e swap in delle pagine sono più leggeri perché si stanno facendo operazioni sulle pagine e non sui processi interi

Altri fattori che rendono il demand paging più efficiente sono

- Località spaziale e temporale
- Tabella di paginazione con bit di validità
- Memoria secondaria che memorizza le pagine non presenti nella memoria principale. Questo tipo di memoria è solitamente un disco ad alta velocità detto "swap device" o "swap space".

Performance del demand paging (calcolo EAT)

$$EAT = (1 - p) \times t_{ma} + p \times \bar{t}_{pfs}$$

- p: tasso di page fault ($0 \leq p \leq 1.0$)
- t_{ma}: tempo di accesso alla memoria
- $\bar{t}_{pfs}$: page fault service time, tempo medio necessario per soddisfare una richiesta di pagina che non è attualmente residente in memoria fisica, ma deve essere caricata dalla memoria secondaria.
  Un $p = 0.001$ aumenta l'EAT di 40 volte (rispetto a un $p = 0$), un valore ottimo è $p < 10^{-6}$

Bit di validità

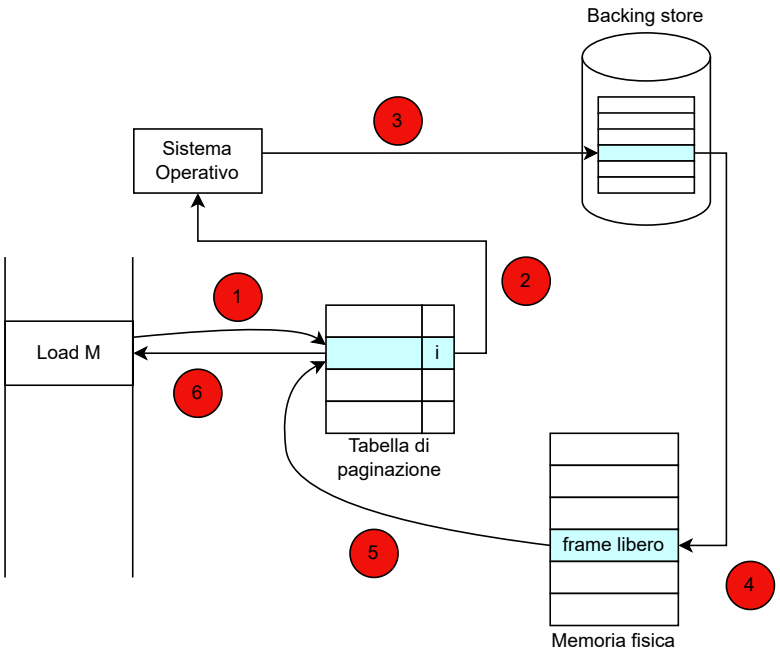
è un sistema per distinguere le pagine che sono in memoria da quelle che sono sul disco. Quando il bit è "valid", la pagina nell' entry di quel bit è sia in una posizione valida, che in memoria. Se il bit è "invalid", la pagina o non è valida, o è valida ma sul disco. Segnare che una pagina è invalida non ha alcun effetto se il processo non prova mai ad accedere a quella pagina.

Un processo ha bisogno di una pagina che non è in memoria centrale

L'accesso a una pagina invalida innesca una page fault (solitamente prodotti prima dell'esecuzione del processo). Dato che il SO non è stato in grado di portare la pagina richiesta in memoria si attiva una trap. Per gestire questa fault bisogna:

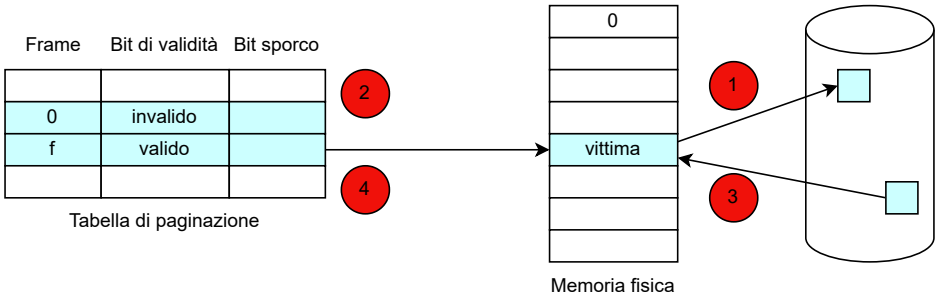
1. Controllare la tabella di paginazione per verificare che si sia fatto riferimento ad un indirizzo di memoria valido o invalido.
2. Se il riferimento è invalido si termina il processo. Se è valido viene mandato alla paginazione (?)
3. Si trova un frame libero
4. Si pianifica un'operazione di lettura del disco per leggere la pagina richiesta nel nuovo frame libero. (operazione più costosa)
5. Quando l'operazione di lettura del disco è completa, si modifica la tabella interna associata al processo e si indica alla tabella di paginazione che la pagina ora è in memoria

6. Si fa ripartire l'istruzione interrotta dalla trap.



Sostituzione di pagina

> Durante l'esecuzione di un processo avviene una page fault e il SO stabilisce dove la pagina richiesta è sul disco ma vede che non ci sono frame liberi: tutta la memoria è in uso e non si può terminare il processo. Un alternativa è **sostituire un processo**, liberando tutti i suoi frame e riducendo il livello di multiprogrammazione.



Si usano algoritmi per produrre frame liberi quando non ce ne sono. Si modifica il sistema di page-fault per includere la sostituzione di pagina:

1. Si trova la posizione della pagina richiesta sul disco
2. Si cerca un frame libero:
    1. Se c'è un frame libero si usa
    2. Se non c'è un frame libero si una un algoritmo di sostituzione di pagina per scegliere il frame "vittima"
    3. Si scrive il frame vittima sul disco e conseguentemente si cambiano le tabelle di pagina e di frame.
3. Si legge la pagina richiesta nel nuovo frame libero e si cambiano le tabelle di pagina e di frame.
4. Continua il processo da dove era capitata la page fault.
   Se non ci sono frame liberi, sono richiesti due page transfer: questo raddoppia il tempo del sistema page-fault e di conseguenza aumenta l'EAT. Si può mitigare questo problema usando un **bit sporco**.
   Ogni pagina o frame ha un bit sporco associato a livello hardware. Un bit a 1 indica che la pagina è stata modifica da quando è stata letta dal disco, il che significa che bisogna riscriverla sul disco. Se il dirty bit è a 0, la pagina non è stata modificata da quando è stata letta in memoria, e in questo caso non dobbiamo aggiornare la pagina in memoria dato che che non ha subito operazioni di scrittura.
   Per implementare il demand paging bisogna avere un algoritmo di allocazione di frame e un algoritmo di sostituzione di pagine. L'algoritmo di page replacement migliore è quello con il minor tasso ti page fault.

Il test di valutazione di un algoritmo consiste nell'eseguirlo su una particolare sequenza di riferimenti di memoria e calcolare il numero di page fault. Questa sequenza è chiamata stringa di riferimento. Per determinare il numero di page fault di una stringa di riferimento e algoritmo di sostituzione bisogna anche sapere il numero di frame disponibili.

Algoritmo di sostituzione FIFO

L'algoritmo più semplice di sostituzione delle pagine è il FIFO, che associa ad ogni pagina il tempo col quale una pagina è stata portata in memoria. Quando una pagina deve essere sostituita, viene scelta quella più vecchia. Questo algoritmo però ha 2 problemi:

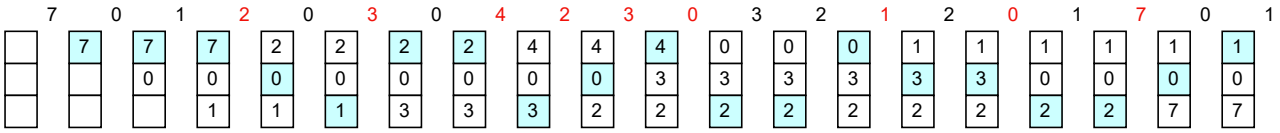
1. Assume che le pagine vecchie siano quelle meno utilizzate
2. Anomalia di Belady: il tasso di page fault aumenta se si aumenta il numero di frame disponibili.

Algoritmo di sostituzione OPT

Un algoritmo ottimale non presenta l'anomalia di Belady e presenta il minor tasso di page fault possibile. L'idea di base di questo algoritmo è **sostituire la pagina che non sarà usata per il maggior periodo di tempo**. Tuttavia questo algoritmo non può essere utilizzato in quanto necessita di vedere gli elementi della reference string futuri.

Algoritmo di sostituzione LRU

Usando il passato recente come approssimazione del futuro prossimo, si può sostituire la pagina che non è stata utilizzata per il periodo di tempo maggiore (least recently used). Per implementare questo algoritmo si associa ad ogni entry un contatore che aumenta ad ogni ciclo di clock.



Esempio di LRU con stringa di riferimento 70120304230321201701

Algoritmo di sostituzione LRU-efficiente

Dato che pochi computer sono progettati per eseguire una LRU, si usa un'alternativa simile che usa un **bit di riferimento**. Questo viene impostato a 1 ogni volta che viene fatto riferimento alla pagina (scrittura o lettura). I bit di riferimento sono associati ad ogni entry della tabella di paginazione.

1. Tutti i bit vengono impostati a 0
2. Durante l'esecuzione del processo il bit associato ad ogni pagina è impostato a 1 dal hardware.
3. Si può determinare quali pagine sono state utilizzate e quali no esaminando i bit di riferimento (anche se non si sa l'ordine di utilizzo).
   In alternativa si possono usare 8 bit (1 byte) che tiene traccia degli ultimi 8 periodi di tempo per quella pagina, dove 00000000 significa che la pagina non è mai stata utilizzata negli ultimi 8 periodi. Se si tratta questo byte come un intero, quello con il valore piccolo è la pagina che non è stata utilizzata da più tempo e può dunque essere sostituita.

Algoritmo di sostituzione seconda chance

Questo algoritmo è simile al FIFO, ma usa un bit di riferimento. Se questo è 0 si sostituisce la pagina, altrimenti si da un'altra chance alla pagina di proseguire e selezionare un'altra pagina. Questa struttura è implementata con una lista circolare.

Allocazione di frame

Quanti frame si allocano a un singolo processo? Si usano algoritmo di allocazione:

- equal allocation: vengono assentati li stessi frame a tutti i processi
- proportional allocation: a un processo vengono allocati frame in base al peso che ha rispetto a tutti i processi in esecuzione.

$$a_i = \frac{s_i}{S} \times m$$

- a_i: numero di frame allocati a p_i
- p_i: processo
- s_i: dimensione della memoria virtuale per il processo p_i
- m: numero di frame disponibili

Allocazione globale e allocazione locale

L'allocazione globale permette a un processo di scegliere una frame di sostituzione dall'insieme di tutti i frame, anche se è già allocata in un altro processo. Un problema di questo tipo di sostituzione è che non può controllare il suo tasso di page fault.

La sostituzione locale richiede che ogni processo sia scelto solamente dal suo insieme di frame allocati. Questo approccio permette a un processo con priorità elevata di aumentare il suo numero di frame al costo di un processo di bassa priorità.

Trashing

Cosa succede alla memoria virtuale al variare del livello di multiprogrammazione?
Con trashing si intende quando un processo sta continuamente facendo swapping di pagine, mettendo la CPU in attesa. Quindi il SO si accorge che il processore è usato poco e aumenta il livello di multiprogrammazione togliendo frame a processi esistenti. C'è un monitoraggio dei page fault per ogni processo, il SO sa che su un certo arco di tempo un processo va bene se produce un numero di page fault molto ridotto.
Se il SO si accorge che un processo produce un page fault molto basso gli si può togliere
qualche frame.