

Sistemi Operativi

Docente: Ugo Erra
ugoerr+so@dia.unisa.it



10° LEZIONE MEMORIA VIRTUALE

*CORSO DI LAUREA TRIENNALE IN INFORMATICA
UNIVERSITA' DEGLI STUDI DELLA BASILICATA*



Sommario della lezione



- Introduzione
- Paginazione su richiesta
 - Prestazioni
- Algoritmi per la sostituzione delle pagine
- Allocazione dei frame
- Paginazione degenera (thrashing)

Introduzione - 1



- I metodi per gestione della memoria tentano di tenere quanti più processi in memoria per aumentare il grado di multiprogrammazione
- Il numero di processi che possono essere disponibili in memoria dipende dalla dimensione della RAM
- La **memoria virtuale** permette l'esecuzione di processi la cui immagine non è completamente in memoria primaria
- Quindi la memoria virtuale permette di mandare in esecuzione applicazioni la cui dimensione è più grande della memoria fisica disponibile

Introduzione - 2



- L'idea alla base della memoria virtuale è che i programmi non devono essere necessariamente caricati interamente in memoria per essere eseguiti
 - Le procedure per la gestione degli errori potrebbero essere eseguite raramente o addirittura mai
 - Strutture dati come array, liste o tabelle potrebbero essere più grandi di quanto necessario
 - Alcune procedure previste in fase di progettazione non sono mai utilizzate
 - Alcune librerie dinamiche sono caricate in memoria solo quando effettivamente usate

Scopo della memoria virtuale



- L'obiettivo della memoria virtuale è caricare in memoria parti dell'applicazione solo quando *devono essere effettivamente eseguite*

Vantaggi



- Più programmi possono essere mandati in esecuzioni anche se occupano più spazio della memoria fisica disponibile
- Il numero di programmi in esecuzione aumenta e quindi anche il grado di multiprogrammazione
 - Maggiore throughput della CPU
- I programmi vengono mandati in esecuzione più velocemente perché non è necessario caricare l'intera applicazione in memoria
 - Anche la chiusura di una applicazione risulta più veloce

Svantaggi



- A causa dell'utilizzo del disco fisso in fase di swapping la quantità di dati che transitano da RAM a disco aumenta
- La memoria virtuale è per ambienti multiprogrammati, un singolo programma richiederà più tempo per essere eseguito
- Se la memoria virtuale non è implementata accuratamente le prestazioni dell'intero sistema degradano drasticamente

Paginazione su richiesta



- L'idea di base della **paginazione su richiesta** è portare una pagina in memoria solo nel momento in cui sarà utilizzata
 - Ad esempio, un'istruzione fa riferimento ad un dato appartenente alla pagina stessa
- Se la CPU fa riferimento alla locazione di un'altra pagina e questa non è presente, allora:
 - Il Sistema Operativo sospende il processo
 - La pagina viene caricata in memoria
 - Il processo riprende l'esecuzione dal punto in cui era stato sospeso
- Durante questa procedura il processo non si accorgerà di nulla

Passi della paginazione su richiesta



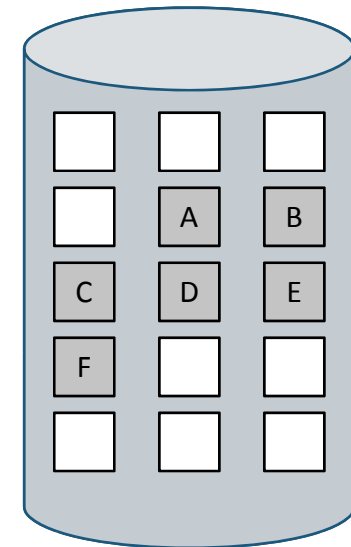
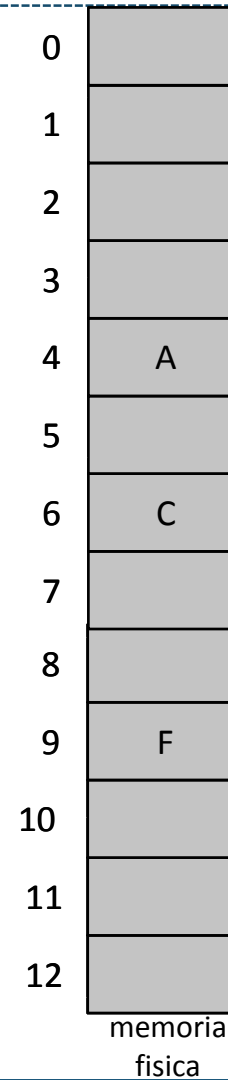
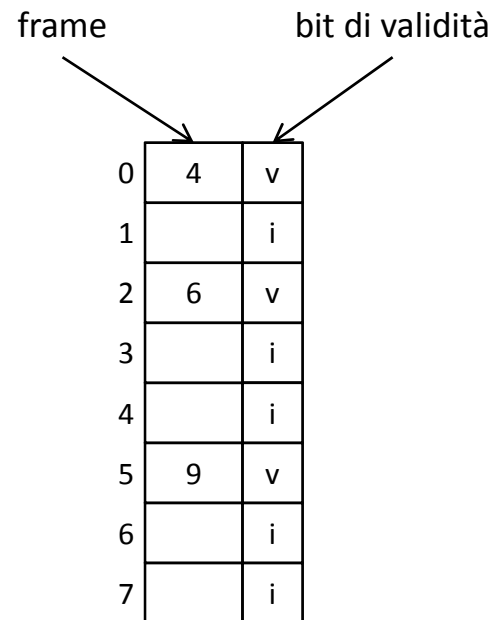
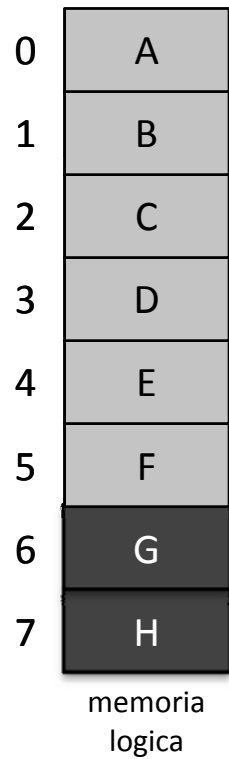
- I passi della paginazione su richiesta sono:
 - Il processo assume uno stato “**waiting for page**”
 - Un modulo del Sistema Operativo detto **paginatore** (*pager*) carica la pagina mancante dalla memoria secondaria alla memoria principale
 - Il processo in attesa può eventualmente essere prelazionato fino al termine del caricamento della pagina
 - Quando la pagina è stata caricata nella memoria principale il processo è inserito nella coda dei processi pronti
 - ✦ L'esecuzione riprenderà dall'istruzione che ha causato l'eccezione

Eccezione di pagina mancante

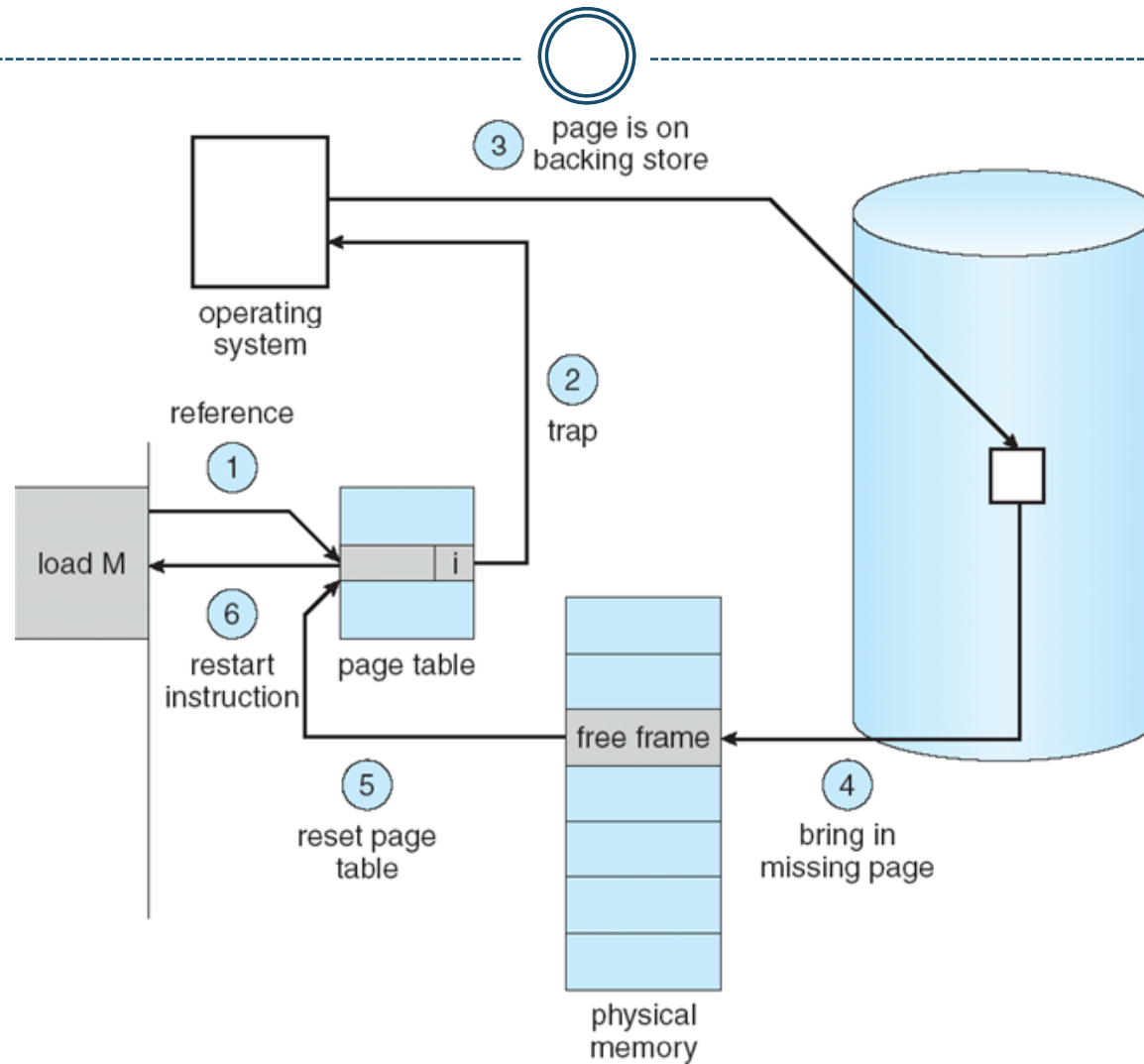


- Resta da capire in che modo una pagina non valida è caricata in memoria
- Possiamo utilizzare il **bit di validità** della pagina associato ad ogni elemento della tabella delle pagine
 - Un riferimento ad una pagina il cui bit di validità nella tabella delle pagine è a 0 causa una **eccezione di pagina mancante** (*page fault trap*)
 - ✦ Il Sistema Operativo provvederà quindi ad effettuare una paginazione su richiesta
 - Nel momento in cui la pagina è caricata in memoria la tabella della pagina è aggiornata con il bit di validità ad 1 ed il processo può ripartire

Esempio di paginazione



Fasi di gestione di un'eccezione page fault



Pure Demand Paging



- Un processo potrebbe essere mandato in esecuzione senza che sia presente in memoria nessuna delle sue pagine
- Durante l'esecuzione della prima istruzione il program counter farà riferimento all'indirizzo in cui si trova la prima istruzione causando quindi un page fault
- Questo schema è chiamato **Pure Demand Paging**
- Alternativamente il Sistema Operativo può caricare in memoria solo la pagina che contiene la prima istruzione da mandare in esecuzione

Supporto hardware



- Per implementare la memoria virtuale è necessario un supporto hardware
 - La tabella delle pagine deve contare sul bit di validità in modo da poter generare il page fault
 - Le istruzioni devono poter essere rieseguite al termine di un page fault
 - ✦ Alternativamente l'hardware della CPU deve controllare che tutti gli operandi di una istruzione siano presenti nella memoria principale prima di poter eseguire l'istruzione
- Sebbene la paginazione può essere aggiunta a qualsiasi sistema, la paginazione su richiesta deve poter contare su hardware specifico

Prestazioni della paginazione su richiesta



- Supponiamo di dover accedere alla memoria principale
- Sia ma il tempo di accesso se il dato è presente in memoria
 - Normalmente $ma \approx 100 \div 200$ nanosec
- Sia p la probabilità di un page fault
- Il **tempo d'accesso effettivo** è dato da
 $(1-p) \times ma + p \times \text{tempo di gestione dell'assenza di pagina}$

Secondo	10^{-3}
Millisecondo	1
Microsecondo	1.000
Nanosecondo	1.000.000

Prestazioni della paginazione su richiesta



- Il **tempo d'accesso effettivo** è dato da $(1-p) \times ma + p \times \text{tempo di gestione dell'assenza di pagina}$
- Il tempo di gestione dell'assenza di pagina è composto prevalentemente dai seguenti passi:
 1. Gestire il page fault ($1 \div 100 \mu \text{ sec}$)
 2. Recuperare la pagina mancante dalla memoria secondaria (≈ 8 millisecc, se non ci sono altri processi in attesa di usare la memoria secondaria)
 3. Riavviare il processo ($1 \div 100 \mu \text{ sec}$)
- I punti 1 e 3 sono trascurabili rispetto a 2

Prestazioni della paginazione su richiesta



- Supponendo $ma = 200$ nanosec (per valori espressi in nanosecondi)
- Il tempo d'accesso effettivo è
$$(1-p) \times 200 + p \times 8.000.000 = 200 + p \times 7.999.800$$
- Se $p = 0.001$ (un page fault ogni 1000 accessi), si ha che il tempo d'accesso effettivo è
$$8.199,8 \text{ (circa 8,2 microsecondi)}$$
- Il sistema rallenta di 40 volte

Prestazioni della paginazione su richiesta



- Se pretendiamo un degrado massimo del 10% allora p deve essere tale che

$$\text{tempo d'accesso effettivo} = 220 > 200 + 8 \times 10^6 \times p$$

$$20 > 8 \times 10^6 \times p$$

$$p < 2,5 \times 10^{-6}$$

- In conclusione non più di 1 page fault ogni 400.000 accessi in memoria

Prestazioni della paginazione su richiesta



- Per tenere alto il throughput dei processi bisogna cercare di tenere il numero di page fault basso
- Inoltre, bisognerebbe cercare di ottimizzare l'accesso alla memoria secondaria
 - L'accesso alla memoria secondaria non si può migliorare più di tanto in quanto le prestazioni dipendono dai produttori hardware
 - Solitamente nei sistemi operativi moderni una sezione del disco è dedicata esclusivamente alla memoria virtuale

Area di Swap



- La memoria virtuale, è in genere gestita usando una opportuna porzione del disco fisso detta **Area di Swap**
- Durante l'installazione, quest'aria del disco viene riservata esclusivamente al Sistema Operativo che la utilizzerà per la paginazione
- Questa area solitamente è gestita con meccanismi più semplici ed efficienti di quelli usati per il File System
 - Infatti non è necessario gestire le pagine dei processi come se fossero dei file
 - Inoltre si usano blocchi più grandi con allocazione solo contigua

Uso dell'area di swap



- All'avvio del processo la sua immagine è copiata interamente nell'area di swap:
 - Il tempo di start up aumenta
 - Abbiamo bisogno di aree di swap grandi
- Anticipare la copia delle pagine nell'aria di swap migliora il tempo di gestione in caso di page fault
 - Non sarà necessario caricare le pagine mancanti dal file system
- Altrimenti le pagine andrebbero prelevate dal file eseguibile accedendo direttamente al file system
 - Utile se dobbiamo limitare l'area di swap
 - L'avvio dei processi è più veloce
 - L'esecuzione è rallentata dagli accessi al file system

Uso dell'area di swap



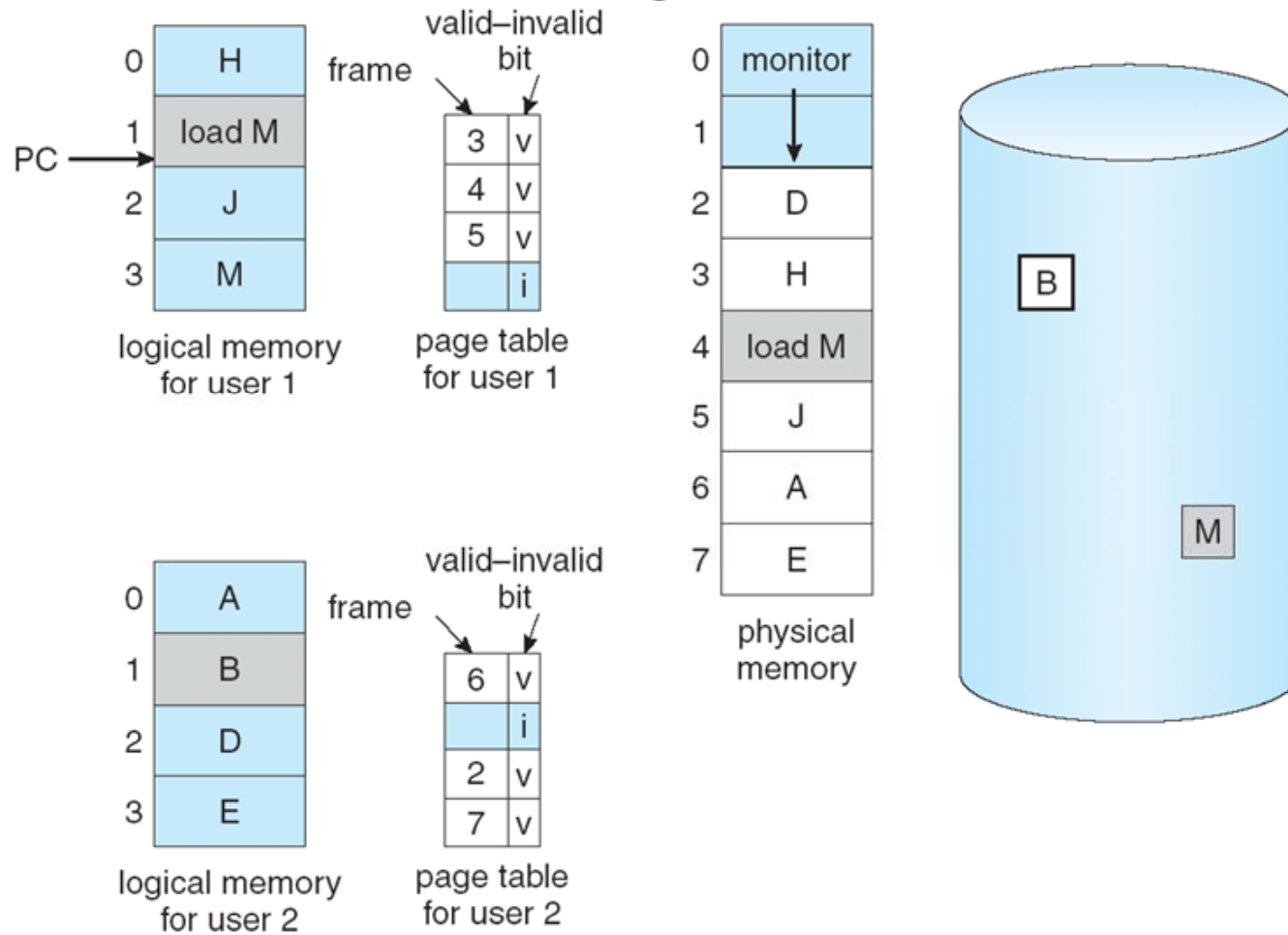
- Indipendentemente da come l'aria di swap sarà gestita nel momento dell'esecuzione di un processo...
- L'area di swap viene adoperata dal sistema operativo come area transitoria in cui tenere le pagine momentaneamente tolte dalla memoria principale
- Quindi se una pagina in memoria principale non è adoperata viene momentaneamente scaricata nell'area di swap
- Infatti, la memoria virtuale permette di:
 - Eseguire programmi più grandi della MP
 - Avere contemporaneamente in esecuzione programmi che, assieme, occupano più spazio della memoria principale disponibile

Sostituzione delle pagine

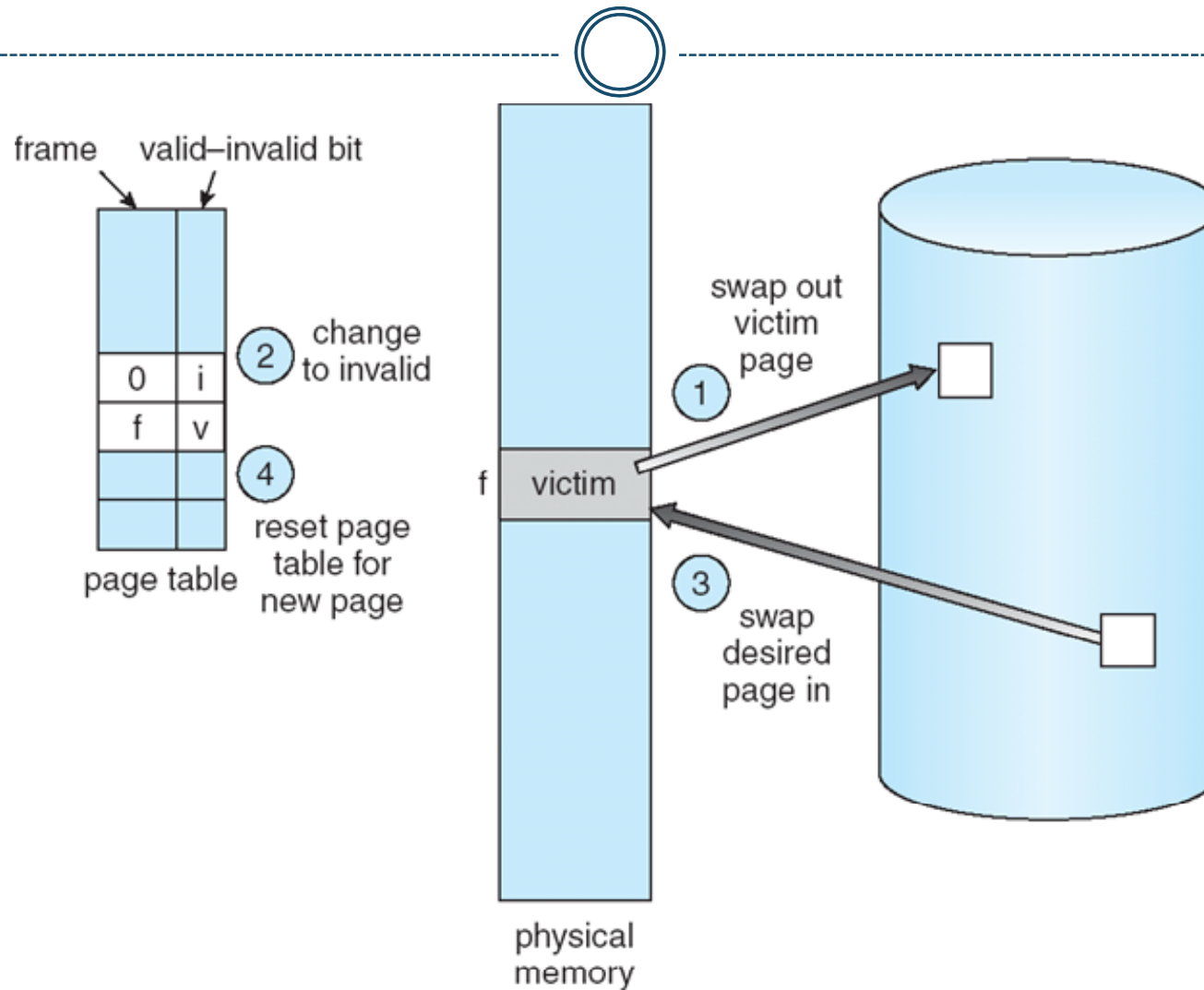


- Un page fault significa che una determinata pagina non è presente in memoria principale
- Poiché la memoria virtuale è più grande della memoria fisica potrebbe accadere che non esiste un frame libero dove caricare la pagina mancante
- Il Sistema Operativo deve scegliere una **pagina vittima** dalla memoria principale da rimuovere
 - La pagina scelta sarà salvata nell'aria di swap
 - Il frame liberato sarà utilizzato dalla pagina richiesta

Necessità di sostituzione di pagine



Sostituzione di una pagina



Dirty bit



- Il tempo di gestione del page fault raddoppia se oltre a caricare la nuova pagina salviamo anche la pagina vittima
- Ma se la pagina vittima non è stata modificata, allora abbiamo già una copia nell'area di swap e quindi è inutile salvarla
- Un **dirty bit** associato ad ogni entry della PT ci dice se la pagina relativa è stata modificata
- Così dobbiamo salvare nella memoria secondaria solo le pagine vittime che hanno il dirty bit impostato ad 1

Schema di base



- L'operazione di sostituzione delle pagine è essenziale per permettere l'esecuzione di programmi più grandi della memoria principale
- I problemi da affrontare sono:
 - In che modo selezioniamo la pagina vittima da rimpiazzare?
 - Quanti frame allocare ad ogni processo?
- Il tipo di soluzione adottato per questi problemi influenza moltissimo l'efficienza dei S.O.

Algoritmi di sostituzione delle pagine



- Un buon **algoritmo di sostituzione delle pagine** deve cercare di minimizzare il numero di page fault, infatti
 - Dopo aver selezionato una pagina vittima questa viene immediatamente riferita da un altro processo e quindi deve essere ricaricata nella memoria principale quindi il lavoro è stato sprecato
 - Se l'algoritmo seleziona una pagina vittima che non verrà più usata da alcun processo allora non sarà più necessaria ricaricarla nella memoria principale
- Un algoritmo di sostituzione si valuta eseguendolo su una particolare successione di riferimenti alla memoria

Sequenze dei riferimenti - 1



- Il modo con cui è possibile valutare diversi algoritmi di sostituzione delle pagine è attraverso le **sequenze dei riferimenti** che rappresentano la successione dei riferimenti in memoria principale
- Due sono i modi per generare queste successioni:
 - In modo casuale, oppure
 - Dall'esecuzione di programmi reali
- Notare che non ci interessa l'indirizzo preciso generato dalla CPU, ma solo la pagina che contiene quell'indirizzo

Sequenze dei riferimenti - 2



- Ad esempio, la successione dei riferimenti potrebbe essere:

10, 7, 4, 5, 6, 1, 10, 4, ...

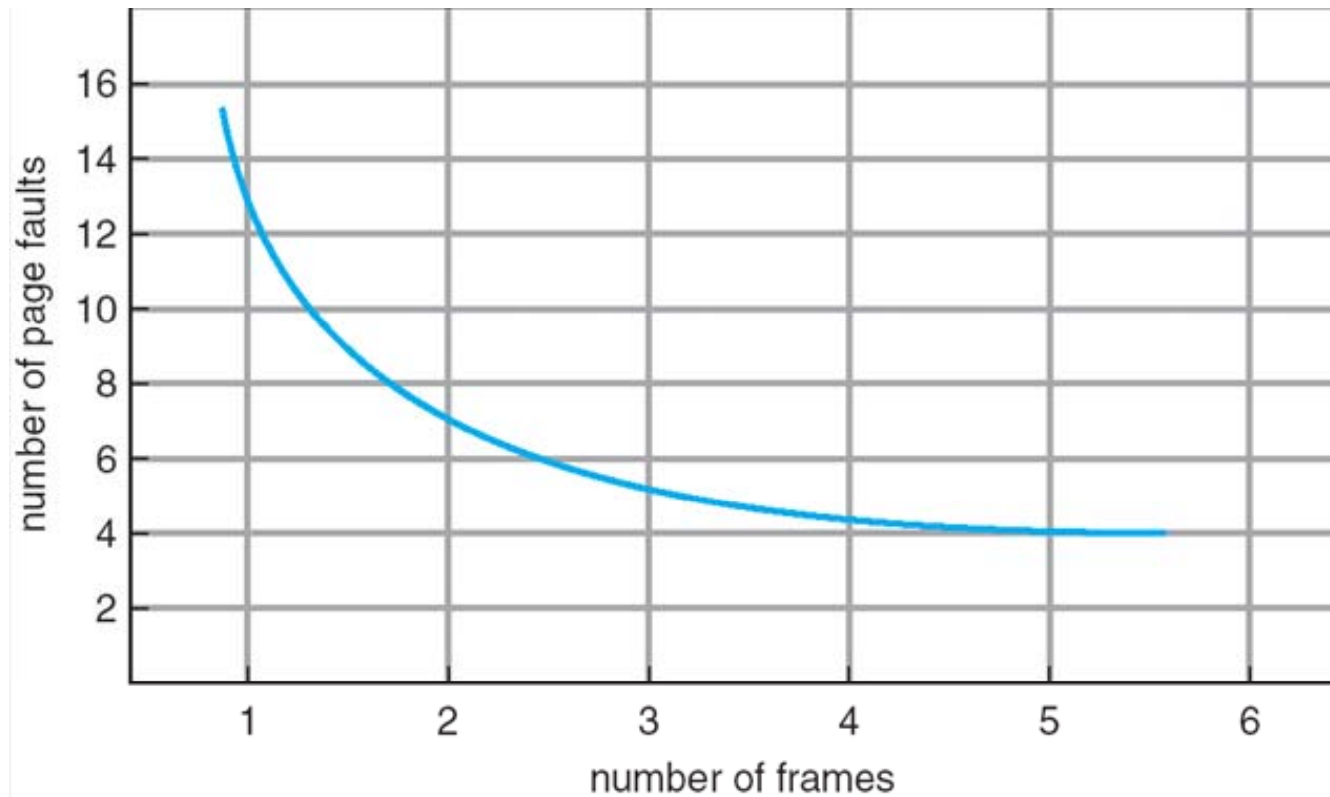
- Questa successione indica che sono stati generati indirizzi di memoria che fanno riferimento alla pagina 10, poi alla pagina 7, alla 4, e così via

Sequenze dei riferimenti - 3



- Consideriamo la seguente successione dei riferimenti:
10, 7, 4, 5, 6, 1, 10, 4
- Il numero di page fault generati dipende dal numero di frame disponibili
- Se esiste un solo frame allora questa successione provoca 8 page fault
- Consideriamo la seguente successione dei riferimenti:
10, 7, 7, 7, 4, 5, 5, 5, 5, 6, 1, 10, 10, 10, 10, 4
- Quanti page fault genera con un solo frame disponibile?

Numero di page fault rispetto al numero di frame



Algoritmi di sostituzione delle pagine



- Sostituzione delle pagine secondo l'ordine di arrivo (FIFO)
- Sostituzione ottimale delle pagine
- Sostituzione delle pagine usate meno recentemente (LRU)
- Algoritmo della seconda chance
- Algoritmo della seconda chance migliorato

Sostituzione delle pagine secondo l'ordine di arrivo

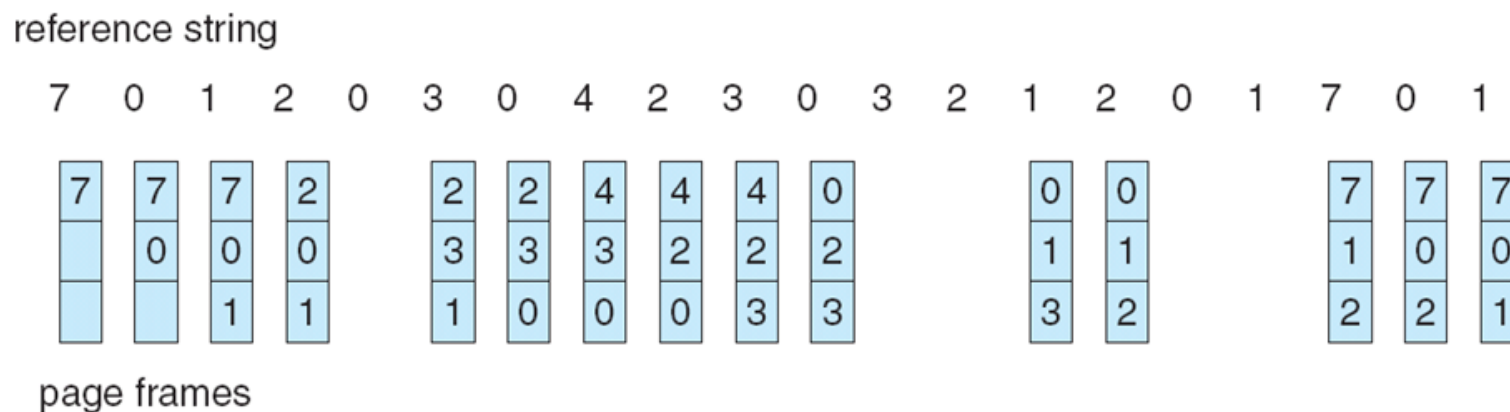


- **Nell'algoritmo di sostituzione delle pagine secondo l'ordine di arrivo** la pagina vittima è selezionata tra quelle presente da più tempo in memoria
- Per tenere traccia del tempo di arrivo è sufficiente una coda FIFO
- L'idea è semplice...
 - Se la pagina contiene del codice di inizializzazione del processo, utilizzato solo in fase di inizializzazione allora quella pagina può essere rimossa
 - Ma se la pagina contiene una variabile inizializzata all'inizio che sarà utilizzata nel corso dell'esecuzione allora non conviene scaricarla

Esempio di Algoritmo FIFO



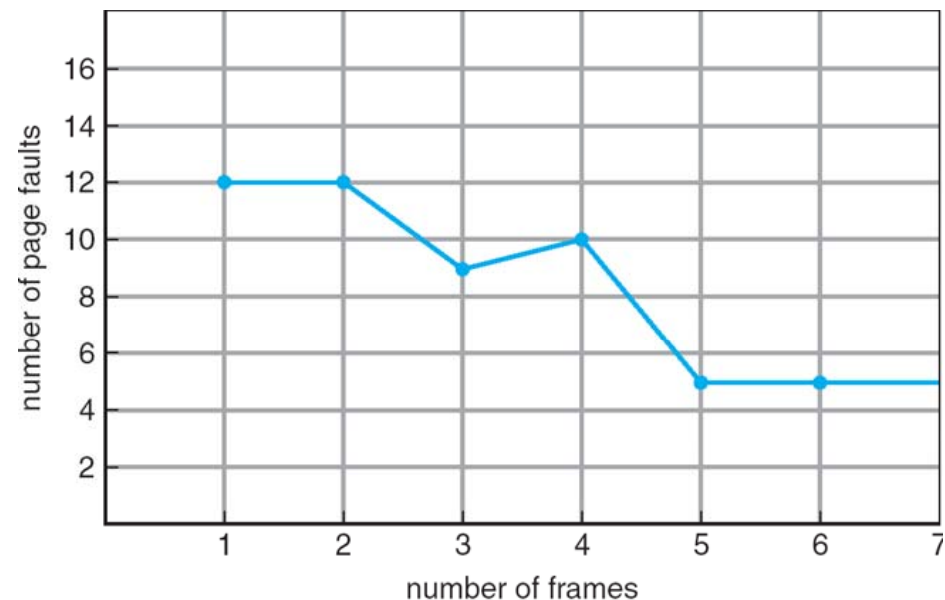
- Supponiamo di avere la seguente sequenza dei riferimenti
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1
e 3 frame a disposizione
- Quanti page fault produce l'algoritmo FIFO?
 - 15



Anomalia di Belady



- L'algoritmo FIFO nonostante sia semplice soffre della cosiddetta **Anomalia di Belady**
- Aumentando il numero di frame, e usando l'algoritmo FIFO, il numero di page fault potrebbe addirittura aumentare
- Intuitivamente ci si aspetta che avendo più frame il numero di page fault diminuisca



Sostituzione ottimale delle pagine



- **Nell'algoritmo di sostituzione ottimale delle pagine (OPT o MIN)** la pagina vittima è quella che non si userà per il periodo di tempo più lungo
 - Abbiamo già incontrato un'idea del genere?
- **Le caratteristiche di questo algoritmo sono:**
 - Assenza dell'anomalia di Belady
 - Numero di page fault minimo (a parità di frame disponibili)
- **In effetti OPT è utilizzato solo come termine di paragone per misurare le prestazioni di altri algoritmi**

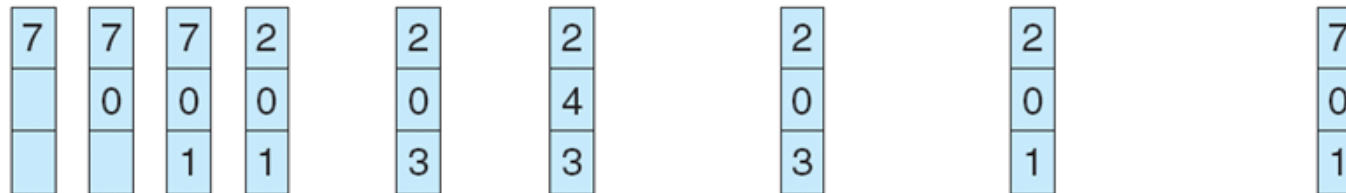
Esempio di Algoritmo OPT



- Supponiamo di avere la seguente sequenza dei riferimenti
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1
e 3 frame a disposizione
- Quanti page fault produce questo l'algoritmo OPT?
 - 9

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



page frames

Sostituzione delle pagine usate meno recentemente



- Poiché non possiamo utilizzare OPT **l'algoritmo di sostituzione delle pagine usate meno recentemente** (*Least Recently Used* - LRU) cerca di approssimarlo
- La pagina vittima è selezionata tra quelle non usate da più tempo
- L'idea è di prevedere il futuro guardando al passato che ci è noto
- Le prestazioni di LRU sono migliori di FIFO in quanto si avvicina più a OPT ma l'implementazione resta critica per l'efficienza

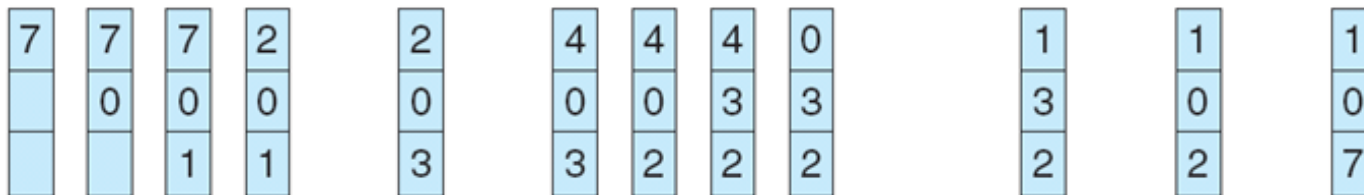
Esempio di Algoritmo LRU



- Supponiamo di avere la seguente sequenza dei riferimenti
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1
e 3 frame a disposizione
- Quanti page fault produce questo l'algoritmo LRU?
 - 12

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



page frames

Caratteristiche di LRU



- L'algoritmo LRU non soffre dell'anomalia di Belady
- Si dimostra che gli **algoritmi a pila** non soffrono dell'anomalia di Belady
- In un algoritmo a pila:
 - l'insieme delle pagine tenute in memoria principale dall'algoritmo per n frame disponibili è sempre un sottoinsieme dell'insieme delle pagine tenute in memoria se sono disponibili $n+1$ frame*
- Nel caso di LRU se l'insieme delle pagine in memoria rappresentano le n pagine utilizzate più recentemente aumentato il numero di frame queste continuano ad essere quelle più recentemente utilizzate e quindi sono ancora in memoria

Implementazione di LRU



- L'algorithmo LRU può essere implementato in due modi:
 - Mediante uno stack
 - Mediante un contatore

Implementazione di LRU mediante uno stack



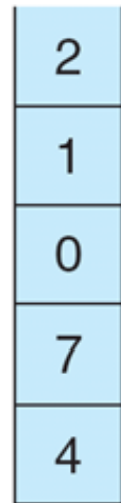
- Utilizziamo uno stack contenente i numeri di pagina acceduti
 - La dimensione dello stack è uguale al numero di frame del processo
- Ad ogni accesso di pagina il suo numero viene spostato in cima allo stack
- L'elemento il fondo allo stack è la pagine utilizzata meno recentemente ed è quindi la pagina vittima
- Gestire uno stack del genere è una operazione costosa in quanto va aggiornato ad ogni accesso in memoria

Uso di uno stack per LRU



reference string

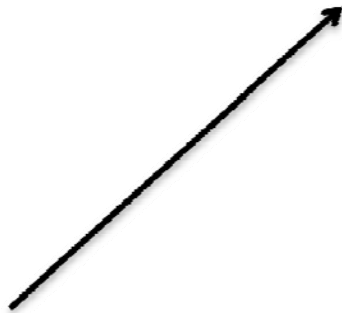
4 7 0 7 1 0 1 2 1 2 7 1 2



stack
before
a



stack
after
b



Implementazione di LRU mediante contatore



- Per ogni pagina abbiamo un campo contatore
- Ad ogni accesso di pagina accade che:
 - Una variabile globale contatore viene incrementata
 - Il valore di questo contatore viene copiato nel campo contatore della pagina
- La pagina con il valore del campo contatore più piccolo è la pagina vittima
 - Alternativamente si potrebbe utilizzare anche il clock della CPU

Bit di riferimento



- Le due implementazioni necessitano di un supporto hardware per essere implementato poiché sia lo stack che il contatore devono essere aggiornati ad ogni riferimento in memoria
 - Questa operazione implementata via software introdurrebbe troppo overhead
- Oltretutto il supporto hardware per LRU è troppo costoso e molti processori di conseguenza non lo supportano
- Una variante supportata dai processori è il cosiddetto **bit di riferimento** associato ad ogni pagina della tabella delle pagine di un processo

Approssimazione di LRU



- In fase di partenza i bit di riferimento di un processo nella sua tabella delle pagine sono impostati tutti a 0 dal Sistema Operativo
- Quando si fa riferimento ad una pagina sia in lettura che in scrittura l'architettura imposta ad 1 il bit di riferimento della pagina
- Questo bit ci permette di determinare quali pagine sono state utilizzate ma non l'ordine con cui sono state referenziate

Algoritmo della seconda chance - 1



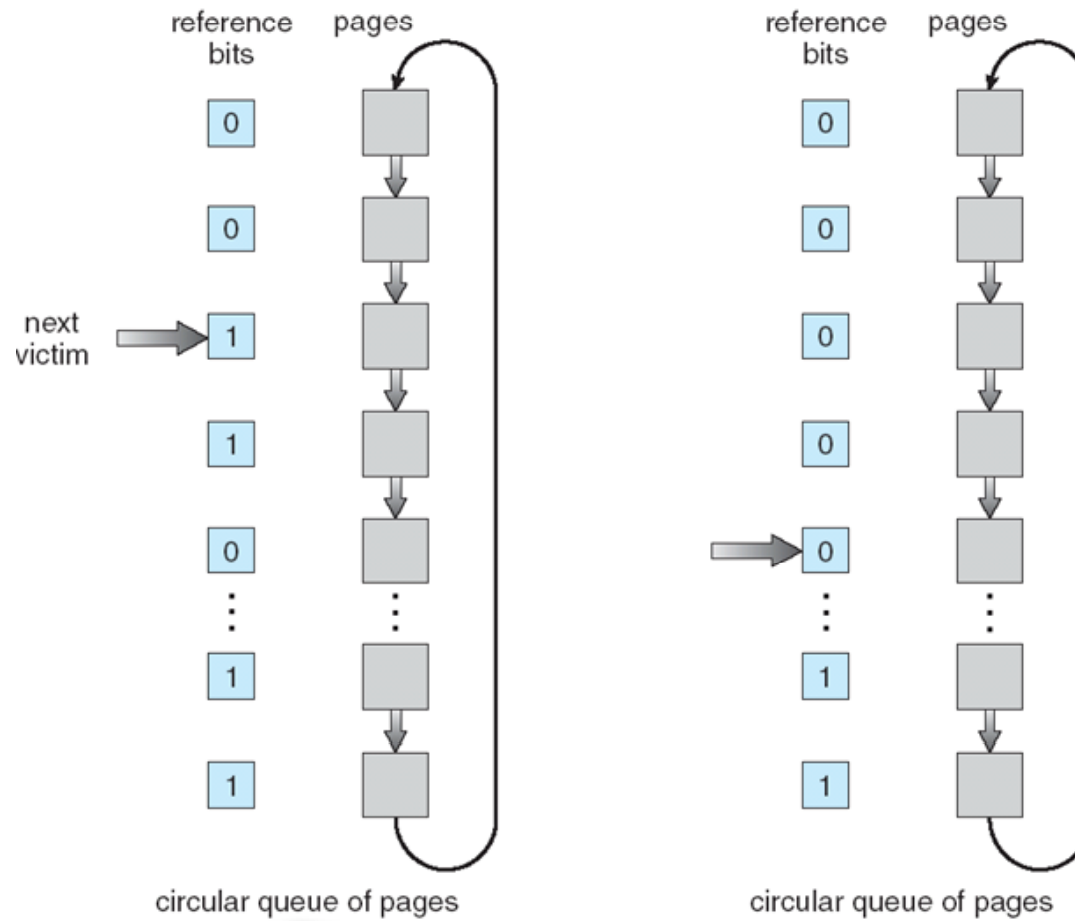
- L'**algoritmo della seconda chance** si basa su una variante dell'algoritmo FIFO
- In caso di page fault consideriamo la pagina in fondo alla coda (la più vecchia)
 - Se il bit di riferimento è (ancora) a 0 allora la selezioniamo come pagina vittima
 - Se il bit di riferimento è a 1 allora lo si azzerà e quindi alla pagina viene data una **seconda chance** e si passa ad analizzare la seconda pagina più vecchia e così via...
- Cosa accade nel caso peggiore?

Algoritmo della seconda chance - 2



- Nel caso peggiore, l'algoritmo trova che tutte le pagine in cui il bit di riferimento è impostato ad 1
- Al termine della coda (cioè dopo che ha analizzato tutte le pagine nella coda FIFO) ritorna alla pagina più vecchia e la seleziona come pagina vittima
- Quindi nel caso peggiore l'algoritmo della seconda chance si riduce all'algoritmo FIFO

Esempio Algoritmo della seconda chance



Algoritmo della seconda chance migliorato



- Se l'HW fornisce sia il bit di riferimento sia il bit dirty, le pagine possono essere raggruppate in 4 classi:
 - (0, 0) né usata né scritta \Rightarrow ottima da rimpiazzare
 - (0, 1) non usata di recente, ma modificata \Rightarrow meno buona
 - (1, 0) usata di recente \Rightarrow non è ancora opportuno rimpiazzarla
 - (1, 1) in uso e modificata \Rightarrow peggiore candidata come vittima
- A partire da questa classificazione si può utilizzare la tecnica della seconda chance, selezionando come vittima la prima pagina che si trova nella classe migliore non vuota
- Unix e MacOS utilizzano un sistema simile

Allocazione dei frame ai processi



- In un sistema multiprogrammato esistono diverse possibilità con il quale distribuire i frame tra i processi
 - Lo stesso numero per tutti i processi
 - In proporzione alla dimensione del processo
 - In proporzione alla priorità dei processi
- Inoltre bisogna considerare se tenere dei frame liberi per i processi futuri che possono entrare nel sistema

Metodi di allocazione



- Allocazione uniforme
- Allocazione proporzionale
- Allocazione in base alla priorità
- Allocazione globale e locale

Allocazione uniforme



- Nell'**allocazione uniforme** allochiamo lo stesso numero di frame a tutti i processi
- Quindi se abbiamo n frame a disposizione ed p processi ciascun processo avrà a disposizione n/p

Allocazione proporzionale



- Nell'**allocazione proporzionale** teniamo in considerazione la diversa dimensione dei processi
- Ad esempio se i processi P_1 , P_2 e P_3 hanno la seguente dimensione:

$$P_1 = 4, P_2 = 6, P_3 = 12$$

con 11 frame a disposizione l'allocazione sarà:

$$P_1 = 4/22 \times 11 = 2$$

$$P_2 = 6/22 \times 11 = 3,$$

$$P_3 = 12/22 \times 11 = 6$$

Allocazione in base alla priorità



- Nell'**allocazione basata sulla priorità** si considera il fatto che i processi hanno priorità diverse
- Quindi il numero di frame assegnati ad ogni processo dipende dalla propria priorità ed eventualmente dalla propria dimensione
- Il Sistema Operativo fa sì che il numero dei frame allocati ad ogni processo cambi in base al grado di multiprogrammazione

Allocazione globale e locale



- Nel caso in cui dobbiamo scegliere una vittima sono possibili due strategie in cui cercare la pagina da rimuovere
 - **Allocazione globale** – La vittima è selezionata tra tutte le pagine allocate in memoria principale (tranne quelle del Sistema Operativo)
 - ✦ E' possibile che la vittima sia una pagina di un altro processo
 - **Allocazione locale** – La vittima è selezionata tra le pagine del processo che ha generato il page fault
 - ✦ In questo caso il numero di frame per processo rimane costante

Problemi dell'allocazione globale e locale



- **Allocazione globale**
 - Poiché il numero delle pagine di un processo presenti in memoria non dipende dai riferimenti che il processo stesso effettua il suo comportamento è variabile ed è funzione del comportamento di paginazione degli altri processi
- **Allocazione locale**
 - Nel caso in cui un processo ha avuto troppe pagine che non utilizzerà processi che ne hanno maggiormente bisogno potrebbero risentirne
- **In generale l'allocazione globale è preferita in quanto il Sistema Operativo riesce a gestire la multiprogrammazione in maniera più flessibile ottenendo un throughput maggiore**
 - Utilizzato da Windows XP

Il fenomeno del thrashing



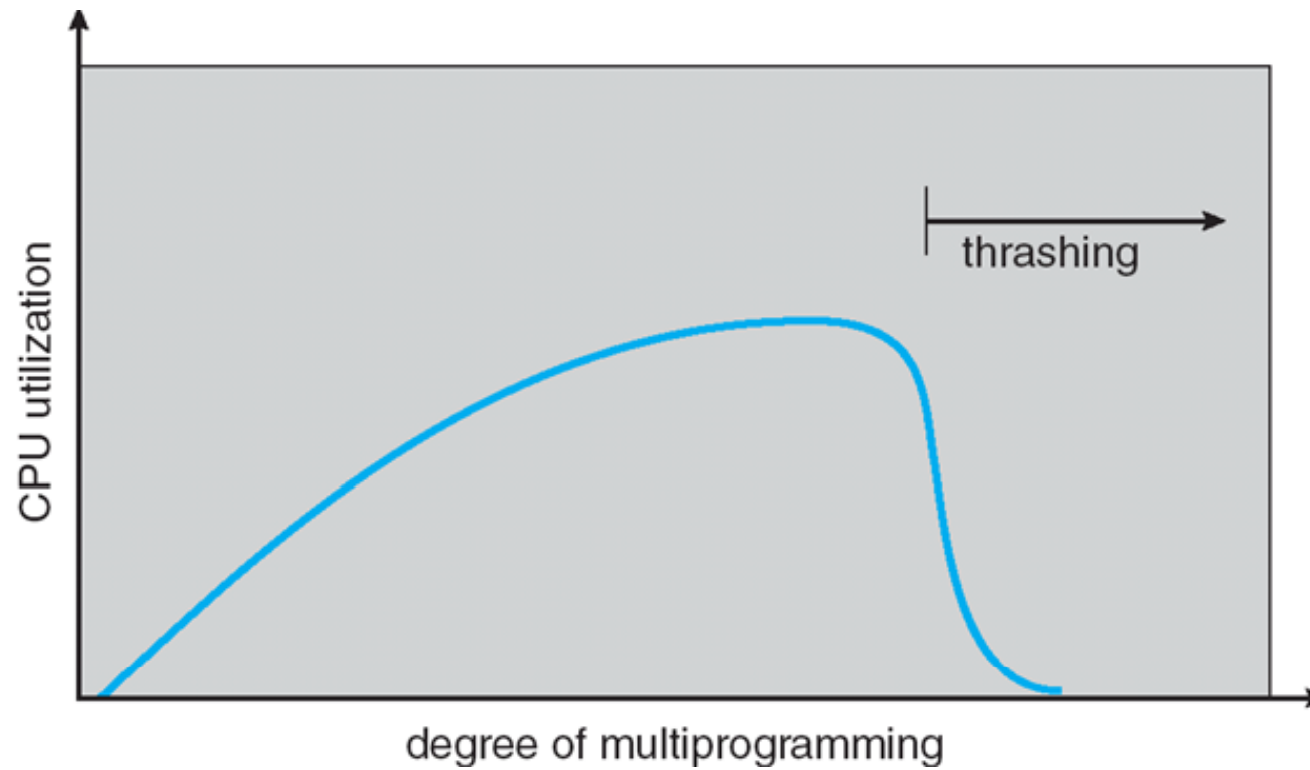
- Supponiamo che in un determinato istante ogni processo ha un numero minimo di frame per essere eseguito
- Inoltre il sistema utilizza una politica di allocazione globale
- Poiché ogni processo ha un'alta probabilità di richiedere un frame inizierà a sottrarlo a qualche altro processo che a sua volta sottrarrà un frame ad un processo e così via...
 - Ogni processo “ruberà” le pagine agli altri processi
- Questo effetto a cascata è chiamato **thrashing**

Cause del thrashing



- Il fenomeno del thrashing si verifica quando si tenta di aumentare troppo il grado di multiprogrammazione per cercare di sfruttare al massimo il tempo di CPU e incrementare il throughput del sistema, ma...
- Esiste un limite oltre il quale i processi passano più tempo a generare page fault che ad essere eseguiti sulla CPU
- In conclusione il throughput crolla

Paginazione degenera



Cause del thrashing



- In generale in un sistema se la CPU è sottoutilizzata si potrebbe ritenere di alzare il grado di multiprogrammazione
 - Ad esempio permettendo a più utenti di connettersi, o di lanciare un maggior numero di processi
- Questa scelta comporta che i nuovi processi iniziano ad “allargarsi” sottraendo pagine ai processi già presenti

Cause del thrashing



- Accade quindi che i processi che in un determinato istante generano page fault vengono tolti dalla code dei processi pronti ed inseriti nella coda di wait
- Con il tempo la coda dei processi pronti si svuota e tutti i processi entrano nella coda di wait in attesa della pagina
- La CPU in questo contesto risulta ancora una volta sottoutilizzata ed altri processi potrebbero essere lanciati...
- Il sistema ad un certo punto si “ingolfa”

Prevenire il thrashing



- Una soluzione potrebbe essere quella di adoperare un algoritmo di allocazione locale in modo che nessuno fregghi le pagine agli altri
- Un processo in thrashing non danneggia gli altri
- Ma se ad un processo sono concessi pochi frame per aumentare il grado di multiprogrammazione allora tutti potrebbero andare lo stesso in thrashing

Osservare la frequenza dei page fault



- In base ad osservazioni sperimentali si può osservare la frequenza dei page fault
- In base alla frequenza osservata:
 - Se è troppo bassa allora è possibile togliere qualche frame ai processi per concederli ad altri processi ed aumentare quindi il grado di multiprogrammazione
 - Se è troppo alta allora diminuiamo il grado di multiprogrammazione ridistribuiamo i frame liberati fra i processi non rimossi

Frequenza delle assenze delle pagine

