

## *Sistemi Operativi – primo modulo*

### *Parte IV - La gestione della memoria*

Augusto Celentano  
Università Ca' Foscari Venezia  
Corso di Laurea in Informatica

© Augusto Celentano, Sistemi Operativi – La gestione della memoria



## *Gestione della memoria centrale (2)*

- **Rilocazione**
  - la posizione di un processo in memoria è nota solo al momento dell'esecuzione e può cambiare nel tempo
  - più processi possono alternarsi in memoria in funzione del loro stato di esecuzione
- **Protezione**
  - un processo non può riferirsi a locazioni di memoria non sue
  - la protezione può essere esercitata solo al momento dell'esecuzione
- **Condivisione**
  - deve essere possibile condividere risorse comuni (es. codice)
  - deve essere consentito a più processi di utilizzare dati comuni in modo controllato

© Augusto Celentano, Sistemi Operativi – La gestione della memoria

3



## *Gestione della memoria centrale (1)*

- La virtualizzazione realizzata dal nucleo di un sistema operativo permette a più processi di comportarsi come se ciascuno avesse una propria unità di elaborazione dedicata, senza la necessità di gestire problemi di competizione e rotazione
- Ogni processo però deve gestire l'uso della memoria centrale in termini di
  - condivisione dello spazio fisico con gli altri processi
  - corrispondenza tra spazio di indirizzamento logico e fisico
- Il gestore di memoria centrale virtualizza lo spazio di indirizzamento di ogni processo indipendentemente dalla sua presenza e posizione nella memoria fisica

© Augusto Celentano, Sistemi Operativi – La gestione della memoria



2

## *Problemi di gestione della memoria a carico del S.O.*

- Le architetture hardware garantiscono i meccanismi di base per la rilocazione, la protezione e la condivisione di memoria
- Il sistema operativo interviene per definire le politiche di gestione
  - allocazione di aree dati private di dimensione variabile per processi e kernel
  - swapping dei processi da e verso la memoria fisica nei sistemi multiprogrammati
  - elaborazione degli eventi di page fault in un sistema a memoria virtuale
  - allocazione di quantitativi adeguati di memoria fisica ai processi
  - gestione dell'utilizzo della memoria fisica in funzione della variabilità di comportamento dei processi

© Augusto Celentano, Sistemi Operativi – La gestione della memoria

4



## Buddy System (1)

- Un sistema di allocazione di quantità variabili di dati all'interno di uno spazio di memoria prefissato
  - buffer temporanei, aree di comunicazione tra processi, aree per i parametri di funzioni di sistema, ...
- Utilizzato (in forma modificata) da Unix SVR4 per l'allocazione della memoria del kernel
- La memoria è allocata a blocchi di dimensione  $2^K$ 
  - $L \leq K \leq U$
  - $2^L$  = blocco di dimensione più piccola allocabile
  - $2^U$  = blocco di dimensione più grande allocabile (l'intera memoria)



## Buddy System (2)

- All'inizio si ha un unico blocco di dimensione  $2^U$
- Quando deve essere allocata un'area di dimensione  $S$ 
  - se  $2^{U-1} < S \leq 2^U$  allora viene allocato l'intero blocco di dimensione  $2^U$
  - altrimenti, il blocco viene diviso in due metà (*buddies*) ciascuna di dimensione  $2^{U-1}$
  - se  $2^{U-2} < S \leq 2^{U-1}$  allora uno dei due blocchi è allocato per intero
  - altrimenti uno dei due blocchi è diviso in due metà ciascuna di dimensione  $2^{U-2}$
  - questo processo viene ripetuto fino ad ottenere il più piccolo blocco di dimensione  $\geq S$  e  $\geq 2^L$
- Due *buddies* vengono riuniti quando diventano entrambi liberi



## Buddy System (3)

1 Mbyte block	1 M				
Request 100 K	A = 128K	128K	256K	512K	
Request 240 K	A = 128K	128K	B = 256K	512K	
Request 64 K	A = 128K	C = 64K	64K	B = 256K	512K
Request 256 K	A = 128K	C = 64K	64K	B = 256K	D = 256K
Release B	A = 128K	C = 64K	64K	256K	D = 256K
Release A	128K	C = 64K	64K	256K	D = 256K
Request 75 K	E = 128K	C = 64K	64K	256K	D = 256K
Release C	E = 128K	128K	256K	D = 256K	256K
Release E	512K		D = 256K	256K	
Release D	1M				

(Stallings, 2011)



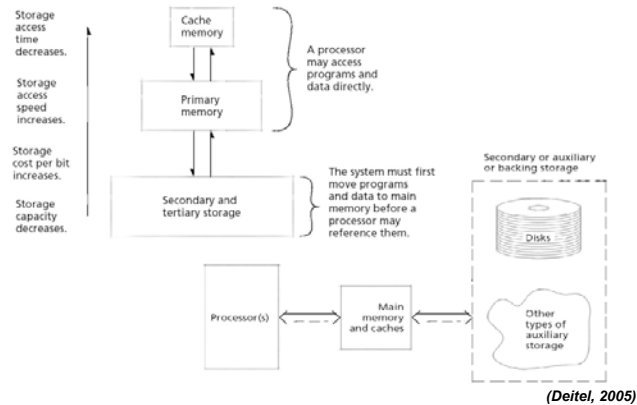
## Buddy System (4)

- L'implementazione si basa su una collezione di liste di aree libere
  - la lista  $i$ -esima raccoglie le aree di dimensione  $2^i$
  - quando si libera una coppia vicina di aree (*buddies*) nella lista  $i$ -esima le aree sono rimosse e memorizzate come un'area unica nella lista  $i+1$ -esima
- Per una richiesta di allocazione di dimensione  $k$ ,
  - $2^{i-1} < k \leq 2^i$
  - si esamina la lista  $i$ -esima
  - se è vuota si esamina la lista  $i+1$ -esima
  - ... e così via fino alla lista di aree di dimensione massima
  - l'area individuata viene divisa secondo l'algoritmo descritto e le liste vengono aggiornate



## Gerarchie di memoria (1)

- Le memorie fisiche sono organizzate in una gerarchia dove variano in modo inverso capacità e velocità di accesso



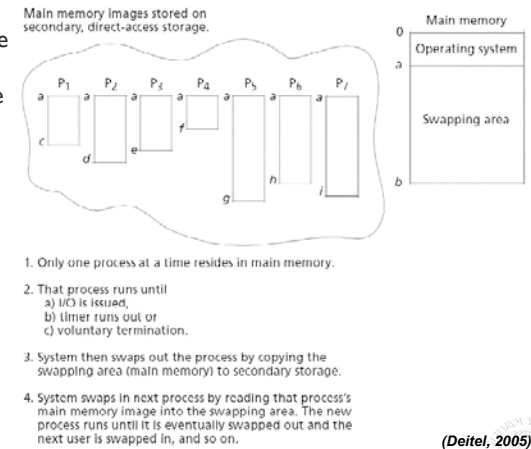
© Augusto Celentano, Sistemi Operativi – La gestione della memoria

9

## Gerarchie di memoria (2)

### Swapping

- un solo processo alla volta è residente in memoria, gli altri risiedono su disco e sono caricati per l'esecuzione



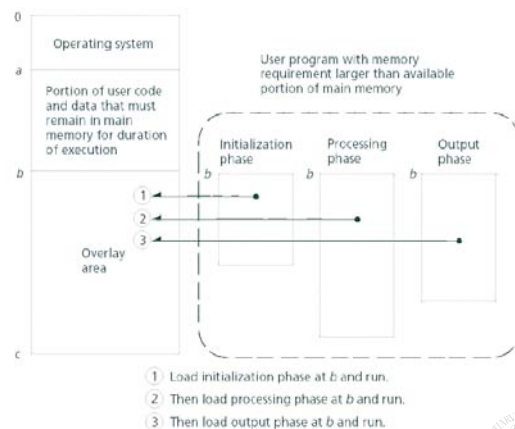
© Augusto Celentano, Sistemi Operativi – La gestione della memoria

10

## Gerarchie di memoria (3)

### Overlay

- parti diverse della memoria di un processo che non sono utilizzate insieme sono residenti su disco e "sovrapposte" nello stesso spazio di indirizzi quando vengono caricate in memoria centrale



© Augusto Celentano, Sistemi Operativi – La gestione della memoria

11

## Memoria virtuale

- La presenza di regole e dispositivi di corrispondenza tra spazi di indirizzamento logico e fisico porta al concetto di *memoria virtuale*

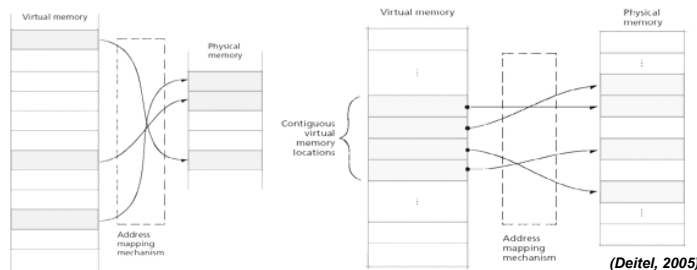
- un processo vede una memoria "logica" indipendente dalle reali caratteristiche "fisiche"
- la memoria logica è organizzata in funzione delle esigenze del processo / linguaggio / ambiente di programmazione
- la memoria fisica è gestita in funzione di obiettivi di efficienza complessiva del sistema
- la corrispondenza tra gli indirizzi logici e gli indirizzi fisici è attuata da meccanismi hw/sw durante l'esecuzione

© Augusto Celentano, Sistemi Operativi – La gestione della memoria

12

## Memoria virtuale (2)

- La contiguità dello spazio di indirizzi è virtuale e realizzata attraverso funzioni e/o tabelle di corrispondenza
  - il processo può vedere una memoria contigua e compatta (entro certi limiti) anche se l'allocazione fisica è sparsa
  - scompaiono o diventano meno importanti i problemi di frammentazione derivanti dalla dinamica dei processi



© Augusto Celentano, Sistemi Operativi – La gestione della memoria

13

## Memoria virtuale (3)

- I metodi di gestione della memoria centrale evolvono nel tempo in funzione delle caratteristiche hardware dei sistemi di indirizzamento e delle esigenze di multiprogrammazione

Real	Real		Virtual		
Single-user dedicated systems	Real memory multiprogramming systems		Virtual memory multiprogramming systems		
	Fixed-partition multi-programming	Variable-partition multi-programming	Pure paging	Pure segmentation	Combined paging and segmentation
	Absolute	Re-locatable			

(Deitel, 2005)

© Augusto Celentano, Sistemi Operativi – La gestione della memoria

14

## Paginazione vs. segmentazione

- La paginazione è un meccanismo di gestione efficiente della memoria
  - è indipendente dalla struttura logica del programma / processo
  - è trasparente al programmatore e all'ambiente di sviluppo
- La segmentazione è (può essere) visibile e controllabile al livello del codice sorgente
  - permette di strutturare i programmi per l'esecuzione: dati / codice, dati locali / dati condivisi, codice locale / librerie
  - i segmenti possono essere soggetti a limitazioni di dimensione (es. 64 KB Intel 80x86, 8 KB PDP-11)
  - ai segmenti possono essere associati attributi di protezione e condivisione

© Augusto Celentano, Sistemi Operativi – La gestione della memoria

15

## Gestione dello spazio di memoria (I)

- In una gestione a pagine la dimensione delle pagine influenza la complessità costruttiva e l'efficacia
  - grandi dimensioni → spreco di spazio
  - piccole dimensioni → tabella delle pagine molto grande
- In una gestione a segmenti non vi è spreco di spazio, ma la memoria fisica può diventare frammentata e non consentire lo sfruttamento totale

© Augusto Celentano, Sistemi Operativi – La gestione della memoria

16

## Gestione dello spazio di memoria (2)

- La richiesta di memoria fisica di un programma è limitata (statisticamente) ad una percentuale dello spazio totale teorico occupato
  - elaborazioni in alternativa non richiedono mai memoria contemporaneamente
  - inizializzazioni e elaborazioni non ripetitive non richiedono memoria dopo la loro esecuzione
  - la maggior parte del tempo di elaborazione è trascorso in elaborazioni concentrate in piccole parti del codice
- E' possibile adottare una politica di contenimento della memoria fisica basata sul riuso delle aree occupate



## Principio di località

- E' un principio osservato sperimentalmente che ha basi teoriche nelle caratteristiche cicliche degli algoritmi e nelle strutture dati aggregate
  - i riferimenti alla memoria tendono a restare confinati in aree localizzate
  - nel tempo la localizzazione cambia ma il fenomeno persiste



(Hartfield, 1972)

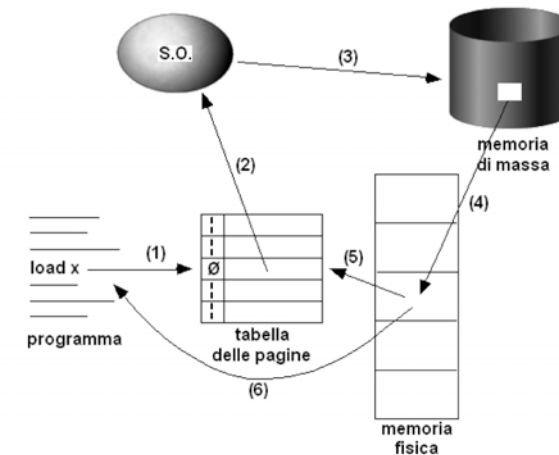


## Paginazione su richiesta (on demand) (1)

- La gestione della memoria a pagine maschera al processo l'effettiva posizione in memoria
- A patto di accettare un rallentamento dell'esecuzione, può anche mascherare la presenza continua in memoria di dati e istruzioni
  - l'istruzione utilizza un dato alla locazione logica  $x$
  - l'hardware di paginazione mappa l'indirizzo logico  $x$  nell'indirizzo fisico  $y$
  - l'indirizzo fisico  $y$  potrebbe non essere valido perché la pagina che lo contiene non è presente in memoria (*page fault*)
  - la pagina viene portata in memoria e il processo continua l'esecuzione
  - un processo per iniziare l'esecuzione ha bisogno che sia presente nella memoria fisica solo la pagina contenente la sua prima istruzione



## Paginazione su richiesta (on demand) (2)



## Trashing

- Per contenere più processi solo una parte dello spazio di indirizzamento di ognuno è conservato in memoria centrale
- Quando la memoria si riempie il sistema deve portare alcune aree (pagine, processi) fuori memoria per fare spazio ad altre (*swap out* - *swap in*)
- La scelta delle aree da scaricare è critica
  - un'area appena scaricata non dovrebbe essere richiesta subito dopo
  - se la scelta non è buona si verifica il fenomeno detto *trashing*: il processore passa più tempo a caricare/scaricare la memoria che ad eseguire istruzioni utili



## Dimensionamento delle pagine (2)

- Dimensioni comprese tra 1KB e 4KB sono frequenti
- Alcuni processori supportano più dimensioni
  - pagine grandi per i segmenti di codice (istruzioni)
  - pagine piccole per i dati dei thread (stack)

Computer	Page Size
Atlas	512 48-bit words
Honeywell-Multics	1024 36-bit word
IBM 370/XA and 370/ESA	4 Kbytes
VAX family	512 bytes
IBM AS/400	512 bytes
DEC Alpha	8 Kbytes
MIPS	4 kbytes to 16 Mbytes
UltraSPARC	8 Kbytes to 4 Mbytes
Pentium	4 Kbytes or 4 Mbytes
PowerPc	4 Kbytes
Itanium	4 Kbytes to 256 Mbytes

(Stallings, 2005)



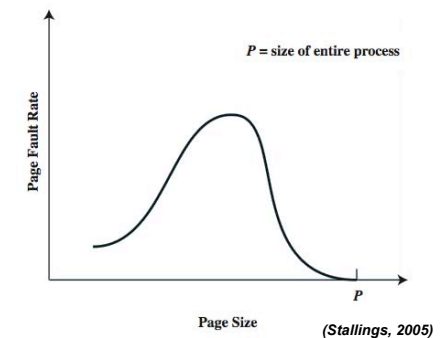
## Dimensionamento delle pagine (1)

- La dimensione delle pagine è definita dall'architettura della macchina ( $2^n$ )
- La scelta della dimensione specifica è un problema serio
  - pagine grandi sono preferibili: meno pagine per processo, la page table richiede meno memoria virtuale
  - pagine piccole sono preferibili: meno frammentazione interna
  - pagine grandi sono preferibili: trasferimento da/a disco più efficiente
  - pagine grandi sono preferibili: meno pagine in memoria, maggiore hit ratio nella cache della TLB
- Quindi?



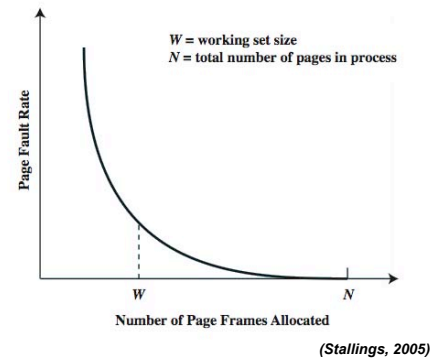
## Dimensionamento delle pagine (3)

- Una dimensione piccola porta pochi page fault perché le pagine sono molte e ogni pagina contiene solo le informazioni che servono (principio di località)
- Dimensioni maggiori possono far aumentare i page fault perché le pagine sono di meno e ogni pagina contiene anche indirizzi non utilizzati
- Se la dimensione della pagina si avvicina a quella del processo i page fault diminuiscono



## Dimensionamento delle pagine (4)

- Il numero di page fault dipende anche dal numero di frame allocati ad ogni processo
- Diventa 0 se il numero di frame è sufficiente per contenere l'intero processo
- Deve essere dimensionato in modo da ottimizzare il rapporto tra page fault e frame allocati



## Allocazione parziale della memoria

- Lo spazio di memoria fisica allocato a un processo può essere minore dello spazio massimo necessario
  - un processo può avere uno spazio di indirizzamento logico maggiore della memoria fisica disponibile
  - principio di località
  - elaborazioni alternative
  - la memoria può essere allocata a mano a mano che serve
- Più processi in memoria aumentano la probabilità che vi sia sempre un processo pronto
  - si deve adottare una strategia di sostituzione delle pagine (scelta della pagina vittima) ispirata a criteri di efficienza



## Strategie di allocazione della memoria

- Decidono quando portare in memoria fisica pagine dello spazio logico di indirizzamento
  - paginazione *on demand*: carica una pagina solo quando è richiesta
  - causa molti page fault all'inizio dell'esecuzione, che poi decrescono
  - *prepaging* (o *precaching*): anticipa il caricamento in memoria rispetto al momento dell'uso
  - il principio di località suggerisce che pagine contigue siano utilizzate insieme
  - non sempre è efficace



## Strategie di sostituzione delle pagine (I)

- Il problema è simile al problema dello scheduling dell'unità centrale nella gestione dei processi
  - *ottima*: la pagina vittima è quella che non servirà per più tempo (come individuarla?)
  - *LRU*: la pagina vittima è quella che da più tempo non viene utilizzata (least recently used)
  - *FIFO*: la pagina vittima è quella caricata da più tempo
  - *clock*: la pagina vittima è scelta tra quelle non utilizzate recentemente
- Varianti dei casi precedenti privilegiano pagine a sola lettura o non modificate
- Le pagine sostituite possono essere bufferizzate per migliorare l'efficienza



## Strategie di sostituzione delle pagine (2)

- Non è possibile scegliere una pagina vittima in modo completamente libero
  - alcuni frame devono rimanere allocati permanentemente o per periodi determinati (*locked frame*)
  - codice del kernel e strutture dati del sistema operativo
  - buffer di I/O per la durata di un'operazione
- L'insieme di pagine da esaminare per la scelta della pagina vittima può essere
  - locale: nell'ambito dello stesso processo
  - globale: la memoria è un'unica risorsa comune
- La quantità di memoria fisica allocata ad ogni processo può variare nel tempo (*working set*)



## La strategia di sostituzione ottima

- La pagina vittima è quella che non verrà utilizzata per più tempo
  - genera il minor numero di page fault
  - non è prevedibile se non su base statistica
  - è una strategia di riferimento per valutare le altre strategie
- Si introduce il concetto di *page address stream*
  - sequenza di riferimenti alle pagine nel corso dell'esecuzione di un processo

Page address stream	2	3	2	1	5	2	4	5	3	2	5	2																																				
OPT	<table><tr><td>2</td></tr><tr><td></td></tr><tr><td></td></tr></table>	2			<table><tr><td>2</td></tr><tr><td>3</td></tr><tr><td></td></tr></table>	2	3		<table><tr><td>2</td></tr><tr><td>3</td></tr><tr><td></td></tr></table>	2	3		<table><tr><td>2</td></tr><tr><td>3</td></tr><tr><td>1</td></tr></table>	2	3	1	<table><tr><td>2</td></tr><tr><td>3</td></tr><tr><td>5</td></tr></table>	2	3	5	<table><tr><td>2</td></tr><tr><td>3</td></tr><tr><td>5</td></tr></table>	2	3	5	<table><tr><td>4</td></tr><tr><td>3</td></tr><tr><td>5</td></tr></table>	4	3	5	<table><tr><td>4</td></tr><tr><td>3</td></tr><tr><td>5</td></tr></table>	4	3	5	<table><tr><td>4</td></tr><tr><td>3</td></tr><tr><td>5</td></tr></table>	4	3	5	<table><tr><td>2</td></tr><tr><td>3</td></tr><tr><td>5</td></tr></table>	2	3	5	<table><tr><td>2</td></tr><tr><td>3</td></tr><tr><td>5</td></tr></table>	2	3	5	<table><tr><td>2</td></tr><tr><td>3</td></tr><tr><td>5</td></tr></table>	2	3	5
2																																																
2																																																
3																																																
2																																																
3																																																
2																																																
3																																																
1																																																
2																																																
3																																																
5																																																
2																																																
3																																																
5																																																
4																																																
3																																																
5																																																
4																																																
3																																																
5																																																
4																																																
3																																																
5																																																
2																																																
3																																																
5																																																
2																																																
3																																																
5																																																
2																																																
3																																																
5																																																
				F		F				F																																						

(Stallings, 2011)



## La strategia LRU, Least Recently Used (1)

- La pagina vittima è quella che da più tempo non viene utilizzata
  - per il principio di località non dovrebbe servire nell'immediato futuro
  - si avvicina all'efficienza della strategia ottima

Page address stream	2	3	2	1	5	2	4	5	3	2	5	2
OPT	<div><div>2</div><div></div><div></div></div>	<div><div>2</div><div>3</div><div></div></div>	<div><div>2</div><div>3</div><div></div></div>	<div><div>2</div><div>3</div><div>1</div></div>	<div><div>2</div><div>3</div><div>5</div></div>	<div><div>2</div><div>3</div><div>5</div></div>	<div><div>4</div><div>3</div><div>5</div></div>	<div><div>4</div><div>3</div><div>5</div></div>	<div><div>4</div><div>3</div><div>5</div></div>	<div><div>2</div><div>3</div><div>5</div></div>	<div><div>2</div><div>3</div><div>5</div></div>	<div><div>2</div><div>3</div><div>5</div></div>
LRU	<div><div>2</div><div></div><div></div></div>	<div><div>2</div><div>3</div><div></div></div>	<div><div>2</div><div>3</div><div></div></div>	<div><div>2</div><div>3</div><div>1</div></div>	<div><div>2</div><div>3</div><div>5</div></div>	<div><div>2</div><div>3</div><div>5</div></div>	<div><div>2</div><div>3</div><div>5</div></div>	<div><div>2</div><div>3</div><div>5</div></div>	<div><div>3</div><div>5</div><div>4</div></div>	<div><div>3</div><div>5</div><div>4</div></div>	<div><div>3</div><div>5</div><div>2</div></div>	<div><div>3</div><div>5</div><div>2</div></div>

(Stallings, 2011)



## La strategia LRU, Least Recently Used (2)

- E' una strategia che richiede la gestione di informazioni temporali associate ad ogni pagina
  - ogni pagina deve essere marcata con l'istante di tempo in cui si è verificato l'ultimo accesso/modifica (*time stamp*)
  - la pagina vittima è quella con il time stamp minore
  - alternativamente si organizzano le pagine in una struttura a lista aggiornata ad ogni accesso
  - la presenza di un algoritmo di ricerca richiede hardware specializzato o un notevole overhead software
- Si preferiscono algoritmi simili, meno efficaci ma più semplici
  - strategie *Least Frequently Used* e *Not Used Recently*
  - strategia di sostituzione *clock*, o *FIFO second chance*





## La strategia LRU, Varianti LFE e NUR

- **LFE, Least Frequently Used**
  - sostituisce al tempo di accesso un'informazione sulla frequenza di utilizzo
  - sono candidate alla sostituzione le pagine utilizzate meno frequentemente
  - ogni pagina è associata a un contatore che si incrementa ad ogni accesso
  - le pagine usate meno frequentemente potrebbero essere le più recenti
- **NUR, Not Used Recently**
  - approssima LRU campionando il tempo ad intervalli molto ampi
  - ad ogni pagina è associata una coppia di bit: *utilizzata* e *modificata*
  - periodicamente si azzerano i bit di utilizzo
  - ad ogni scrittura si imposta il bit di modifica
  - si seleziona una pagina non utilizzata dall'ultimo reset; se manca, si sceglie una pagina non modificata (per non doverla riscrivere su memoria esterna)
  - è migliorato dalla strategia *clock*



## La strategia FIFO (1)

- Le pagine fisiche sono gestite secondo una struttura a coda
- La pagina vittima è quella che da più tempo risiede in memoria
  - è indipendente dalla quantità e frequenza dei riferimenti
  - non coincide con la strategia LRU
  - per il principio di località la pagina potrebbe essere utilizzata spesso lungo un intervallo di tempo non breve
  - è molto semplice da implementare (buffer circolare)



## La strategia FIFO (2)

- La strategia FIFO non è efficiente
  - non considera l'uso delle pagine ma solo il momento del primo caricamento
  - nell'esempio, non riconosce il maggior utilizzo delle pagine 2 e 5 rispetto alle altre

Page address stream	2	3	2	1	5	2	4	5	3	2	5	2
LRU	2	2	2	2	2	2	2	2	3	3	3	3
		3	3	3	5	5	5	5	5	5	5	5
				1	1	1	1	4	2	2	2	2
					F		F		F	F		
FIFO	2	2	2	2	5	5	5	5	3	3	3	3
		3	3	3	3	2	2	2	2	2	5	5
				1	1	1	4	4	4	4	2	2
					F	F	F		F	F	F	F

(Stallings, 2011)

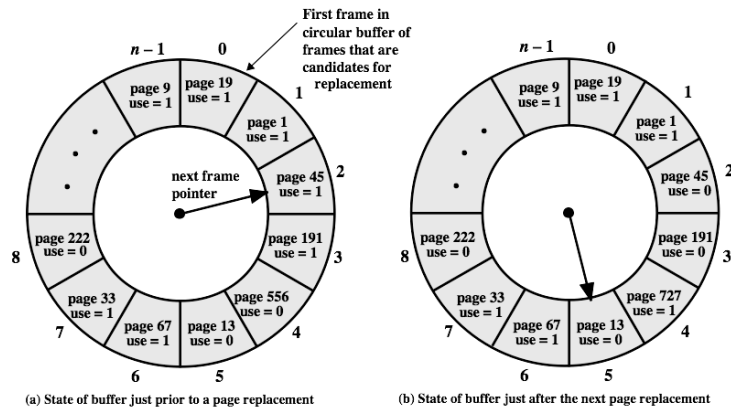


## La strategia dell'orologio (clock) (1)

- Le pagine fisiche sono organizzate in un buffer circolare
- Ogni pagina fisica (*frame*) ha associato un bit di uso che viene impostato a 1 quando
  - una pagina logica viene caricata per la prima volta nel frame
  - la pagina logica caricata nel frame viene utilizzata
- La scelta della pagina vittima avviene scandendo il buffer
  - la pagina vittima è il prossimo frame che ha il bit di uso a 0  
→ non è utilizzato recentemente
  - durante la scansione del buffer i bit di uso dei frame esaminati vengono rimessi a 0
  - in questo modo una pagina usata frequentemente rimetterà il bit di uso a 1 e non verrà selezionata al prossimo giro



## La strategia dell'orologio (clock) (2)



(Stallings, 2011)

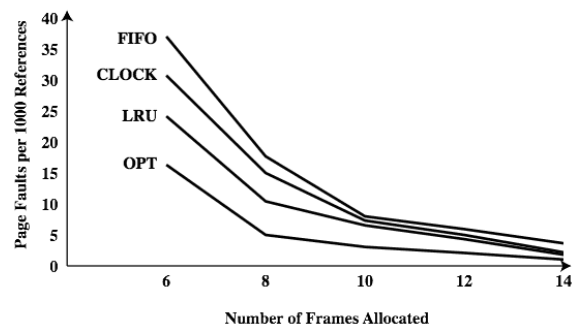
## Un confronto tra le strategie (1)

Page address stream	2	3	2	1	5	2	4	5	3	2	5	2
OPT	2	2	2	2	2	2	4	4	4	2	2	2
LRU	2	2	2	2	2	2	2	2	3	3	3	3
FIFO	2	2	2	2	5	5	5	5	3	3	3	3
CLOCK	2*	2*	2*	2*	5*	5*	5*	5*	3*	3*	3*	3*

Nella strategia clock l'asterisco indica che il bit di uso è 1

(Stallings, 2011)

## Un confronto tra le strategie (2)

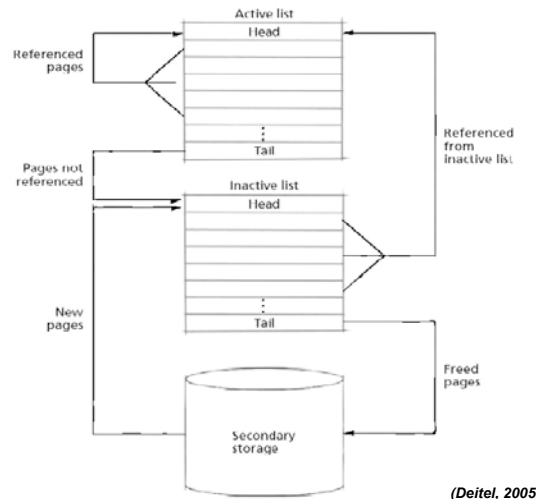


(Stallings, 2011)

## Gestione bufferizzata dei frame sostituiti (1)

- I frame utilizzati per la sostituzione delle pagine sono conservati in una coda in memoria centrale
  - un frame da sostituire viene accodato alla lista e il corrispondente elemento della page table viene azzerato
  - la pagina rimane in memoria e può essere riutilizzata senza ricaricarla da memoria secondaria
  - durante un page fault si esamina la lista per verificare se la pagina è ancora presente
  - se non è presente viene caricata nel frame in cima alla coda (meno utilizzato)
  - per minimizzare i tempi di swap si gestiscono due liste diverse per pagine modificate e pagine non modificate

## Gestione bufferizzata dei frame sostituiti (2)



## Allocazione dei frame a un processo

- L'allocazione di frame a un processo deve bilanciare due esigenze contrastanti
  - pochi frame comportano molti page fault
  - molti frame comportano pochi processi in memoria
- Si possono adottare diverse strategie di allocazione
  - fissa in ambito locale
  - variabile in ambito globale
  - variabile in ambito locale
  - la distinzione fissa-variabile determina se la quantità di memoria allocata ad un processo cambia o no
  - la distinzione locale-globale determina lo spazio di ricerca della pagina vittima



## Allocazione fissa in ambito locale

- Ad ogni processo è assegnato un numero fisso di frame che dipende dalle esigenze applicative
- Durante la gestione di un page fault i frame esaminati per la sostituzione sono quelli allocati al processo
  - semplice da implementare
  - segue una delle strategie esaminate
- E' difficile determinare una dimensione soddisfacente per il processo in modo statico e costante
  - troppo basso: molti page fault
  - troppo alto: multiprogrammazione limitata



## Allocazione variabile in ambito globale

- I frame liberi sono organizzati in una lista globale
  - in caso di page fault al processo viene assegnato un frame libero indipendentemente dalla sua attuale dimensione fisica
- Il numero di frame allocato a un processo aumenta
  - le richieste di altri processi nel tempo possono ribilanciare la distribuzione di frame
  - se la lista si esaurisce la scelta di quale processo penalizzare può essere arbitraria
  - la gestione bufferizzata dei frame liberi può migliorare l'efficienza
- E' semplice da implementare
- E' utilizzato da Unix SVR4

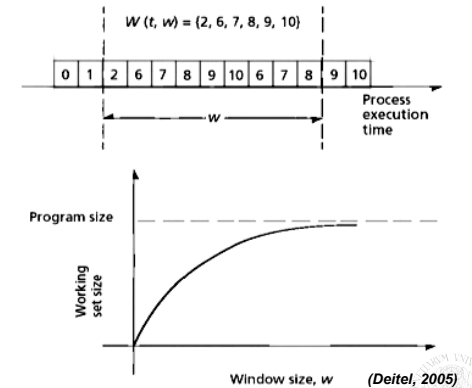


## Allocazione variabile in ambito locale

- Ad ogni processo è assegnato un numero predefinito di frame all'inizio dell'esecuzione
- L'occorrenza di un page fault viene risolta nell'ambito dei frame allocati al processo
  - non si creano squilibri nei confronti di altri processi
- Periodicamente la quantità di memoria fisica allocata a un processo è ridefinita sulla base del comportamento osservato
  - migliora le prestazioni adattandosi alla dinamica reale dei processi
- E' utilizzato da Windows 2000/XP

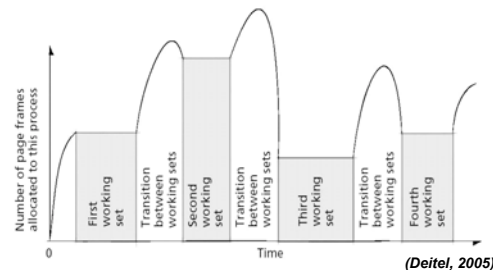
## La strategia Working Set (1)

- E' una strategia di allocazione variabile in ambito locale che si basa sul principio di località
- Il *working set* di un processo al tempo  $t$ ,  $W(t, w)$ , è l'insieme di pagine utilizzate nelle ultime  $w$  unità di tempo
  - $W(t, w)$  descrive la località del programma
  - è una funzione non decrescente di  $w$



## La strategia Working Set (2)

- Il *working set* di un processo ha una variazione prevedibile
  - aumenta all'inizio dell'esecuzione, poi si stabilizza per il principio di località
  - cresce nuovamente quando il processo sposta il proprio indirizzamento verso un altro ambito di località
  - raggiunge un massimo quando contiene pagine che appartengono alle due località
  - quindi decresce nuovamente quando il processo si assesta sul nuovo ambito di località



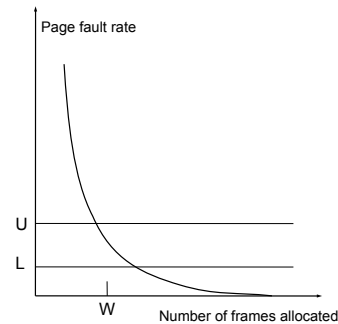
## La strategia Working Set (3)

- La determinazione della dimensione dell'insieme di frame assegnati ad un processo si basa sull'osservazione del comportamento del WS
  - si controlla la variazione di dimensione del WS
  - periodicamente si rimuovono dall'insieme di frame assegnati WS le pagine non utilizzate dal WS
  - quando l'insieme di frame allocati è minore del WS vengono allocati ulteriori frame
  - se non sono disponibili l'intero processo viene sospeso fuori memoria
- L'osservazione della dimensione del *working set* è complessa
  - si osservano i *page fault*

## La strategia Working Set (4)

- Si realizza attraverso una strategia *Page Fault Frequency (PFF)*

- si definiscono un limite superiore  $U$  e un limite inferiore  $L$  per il tasso di page fault
- se il tasso di page fault supera il valore  $U$  si allocano più frame
- se il tasso di page fault scende sotto il valore  $L$  si liberano frame
- in questo modo il numero di frame allocati segue da vicino l'effettivo valore del *working set*



## Problemi attuali nella gestione della memoria

- Pur restando validi in linea generale, i principi che ispirano la gestione della memoria virtuale risentono di alcuni aspetti dell'evoluzione delle tecnologie informatiche

- la dimensione della memoria fisica è aumentata: le tecniche che esaminano sistematicamente tutti i frame fisici sono penalizzati
- le gerarchie di memoria sono cresciute in termini di livelli: il costo di un *cache miss* è molto alto rispetto a uno *hit*
- il principio di località è contrastato da alcune tecniche di programmazione: la programmazione a oggetti e le strutture dati dinamiche tendono a occupare la memoria secondo schemi meno regolari
- la gestione dell'I/O si appoggia a un meccanismo di *caching*: le funzioni del gestore di memoria virtuale sono duplicate, si tende a unificarle attraverso un modulo "allocatore di memoria" più generale

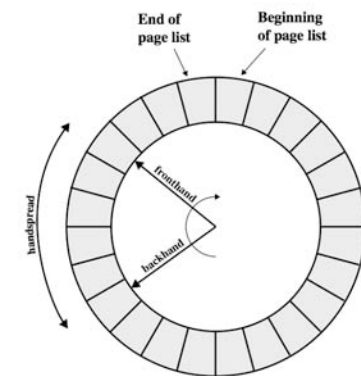
## La gestione della memoria in Unix (1)

- Unix è ampiamente indipendente dalla piattaforma hw
  - versioni diverse hanno meccanismi diversi di gestione della memoria
  - le prime versioni usavano sistemi di partizionamento variabile
  - SVR4 e Solaris presentano meccanismi simili di paginazione
- La gestione memoria si basa su due strategie
  - paginazione per i processi
  - allocazione di memoria per il kernel

## La gestione della memoria in Unix (2)

- La paginazione dello spazio virtuale dei processi si basa su una strategia clock modificata

- due puntatori scandiscono il buffer
- il primo azzerava periodicamente i bit di uso
- il secondo identifica i *frame* da sostituire esaminando il bit di uso



(Stallings, 2005)

## La gestione della memoria in Unix (3)

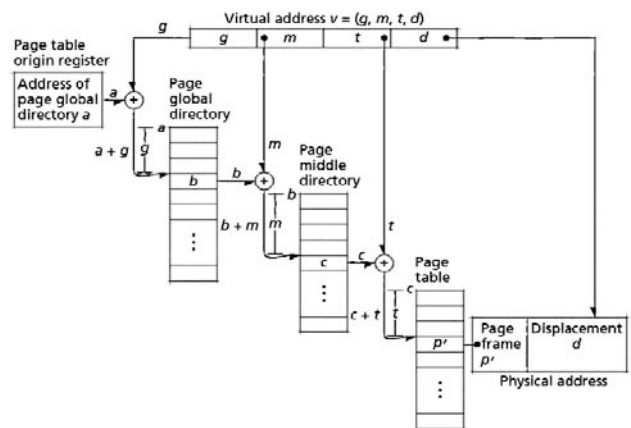
- L'allocazione di memoria per il *kernel* non può basarsi efficientemente sui meccanismi di paginazione
  - l'allocazione è molto dinamica e riguarda numerose strutture dati
  - la maggior parte delle richieste è sensibilmente minore della dimensione di una pagina
- Si utilizza uno schema basato sull'algoritmo di allocazione *buddy system*
  - si osserva che le richieste di memoria sono brevi e uniformi
  - un algoritmo *buddy system* comporterebbe la riunione di aree libere e l'immediata divisione per frequenti richieste omogenee
  - lo schema adottato rimanda nel tempo la riunione delle aree libere

## La gestione della memoria in Linux (1)

- La gestione della memoria dipende dall'architettura della macchina ospite
  - per i sistemi a 64 bit si utilizza una paginazione a quattro livelli
  - la memoria del kernel è residente e occupa la parte iniziale della memoria fisica, la parte più alta della memoria virtuale (> 3GB)
  - la memoria viene allocata secondo un algoritmo buddy system
  - le pagine vittime non sono rimosse immediatamente dalla memoria fisica ma restano "parcheeggiate" in un'area buffer prima della definitiva rimozione

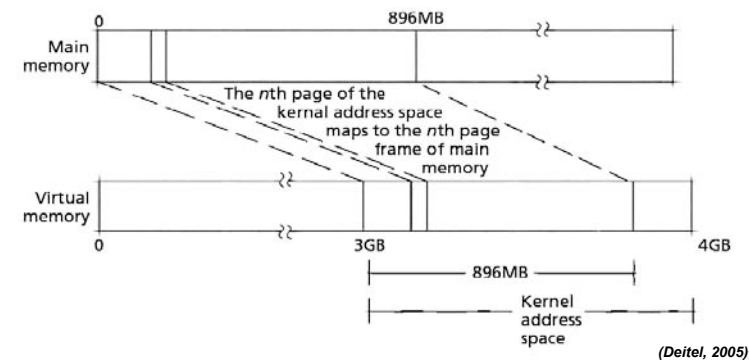
## La gestione della memoria in Linux (2)

- Paginazione a quattro livelli per i sistemi a 64 bit

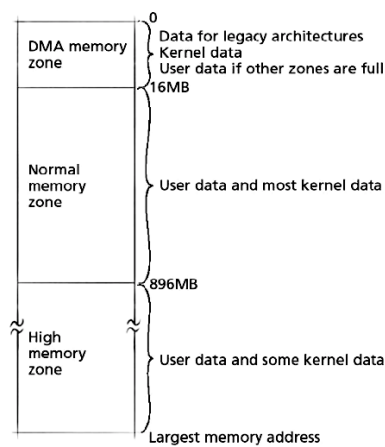


## La gestione della memoria in Linux (3)

- La memoria del kernel è residente e occupa la parte iniziale della memoria fisica, la parte più alta della memoria virtuale (> 3GB)



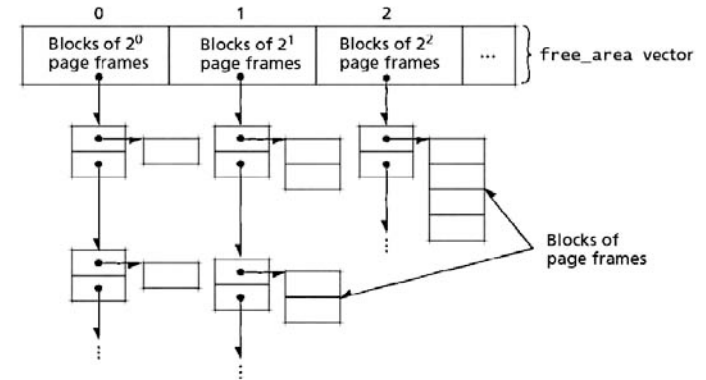
## La gestione della memoria in Linux (4)



(Deitel, 2005)

## La gestione della memoria in Linux (5)

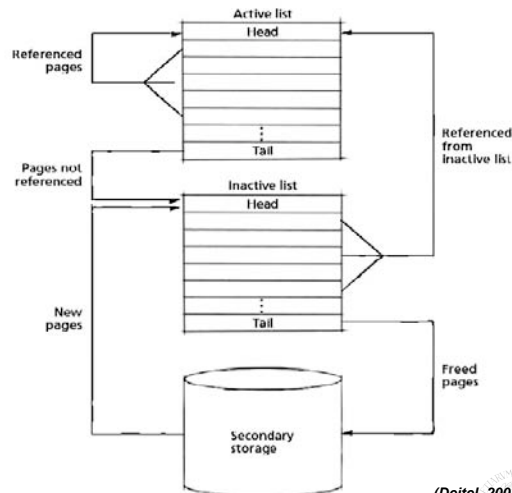
- La memoria del kernel viene allocata secondo un algoritmo buddy system



(Deitel, 2005)

## La gestione della memoria in Linux (6)

- Le pagine vittime non sono rimosse immediatamente dalla memoria fisica ma restano “parcheeggiate” in un’area buffer prima della definitiva rimozione

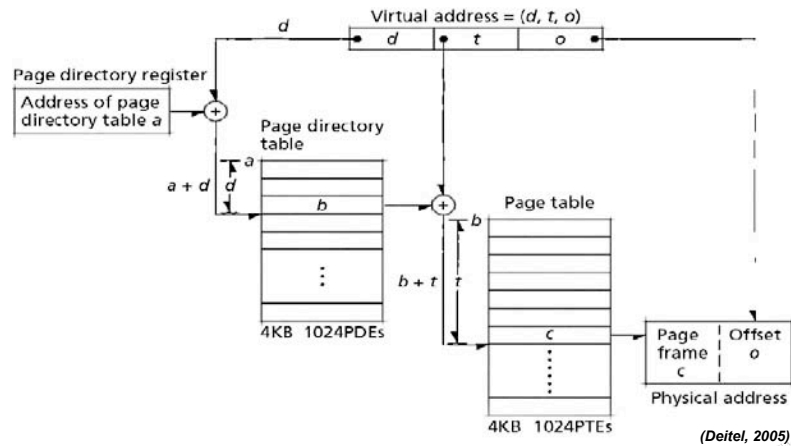


(Deitel, 2005)

## La gestione della memoria in Windows 2000/XP (1)

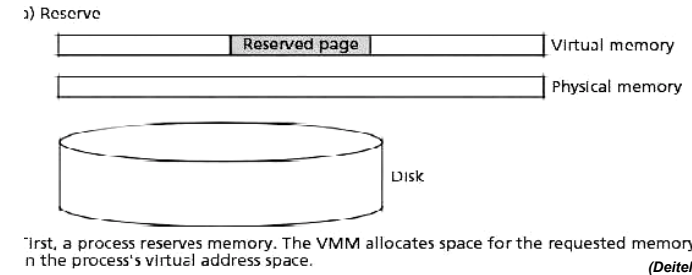
- Windows 2000 utilizza uno schema di paginazione con pagine di 4Kbyte e due livelli di page table
  - un *page directory* contiene 1024 elementi di 4 byte che puntano ciascuno ad una diversa *page table*
  - ogni *page table* ha 1024 elementi di 4 byte che puntano ai frame, per un totale di 4 Mbyte di *page table* per processo
  - il *page directory* è in memoria centrale, le *page table* sono soggette a swap
  - gli indirizzi virtuali sono organizzati su due livelli di paginazione

## La gestione della memoria in Windows 2000/XP (2)



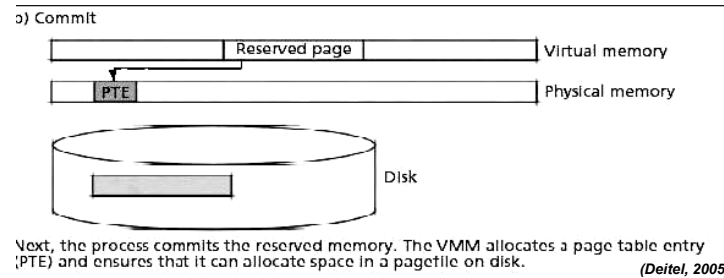
## La gestione della memoria in Windows 2000/XP (3)

- L'allocazione di memoria segue uno schema transazionale (richiesta-commit-esecuzione) per garantire la disponibilità di memoria



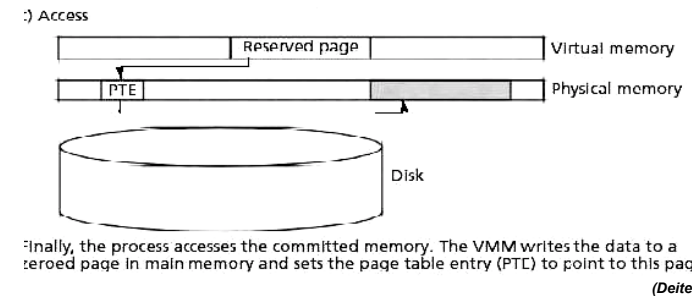
## La gestione della memoria in Windows 2000/XP (4)

- L'allocazione di memoria segue uno schema transazionale (richiesta-commit-esecuzione) per garantire la disponibilità di memoria



## La gestione della memoria in Windows 2000/XP (5)

- L'allocazione di memoria segue uno schema transazionale (richiesta-commit-esecuzione) per garantire la disponibilità di memoria



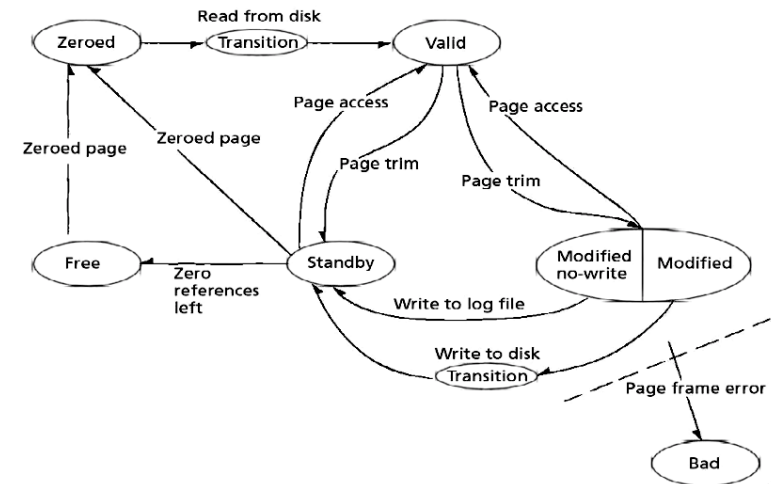


## La gestione della memoria in Windows 2000/XP (6)

- L'allocazione di frame si basa su una strategia variabile in ambito locale (working set)
  - alla creazione di un processo solo un numero limitato di pagine viene assegnato al processo, e un altro insieme di pagine è "prenotato" per usi futuri
  - una pagina può essere in stato committed (effettivamente utilizzata), reserved (prevista ma non ancora utilizzata) oppure not used (non utilizzata dal processo)
- La page table contiene bit di controllo che indicano
  - se la pagina è presente o no → il numero di frame indica la posizione nell'area di swap
  - se la pagina è modificata
  - altre informazioni di protezione



## La gestione della memoria in Windows 2000/XP (7)



(Deitel, 2005)

