
Parte IV

Gestione della Memoria

Gestione della memoria

- Una delle due risorse indispensabili all'elaborazione
- Necessità di allocare in memoria lo spazio di lavoro di più processi
- Anche in un contesto uniprogrammato la memoria è condivisa tra processo corrente e sistema operativo
- In un contesto di multiprogrammazione i processi sono molti, e la commutazione deve essere veloce
- Più processi in memoria significa meno probabilità che la CPU sia inattiva: migliore *throughput*

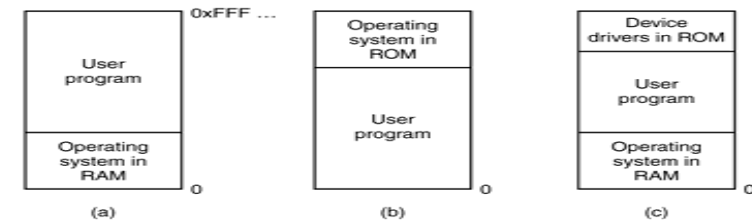
Una risorsa scarsa

- La memoria è sempre stata una risorsa scarsa:
 - inizio anni '80: 128 KB
 - inizio anni '00: 256 MB
- Le dimensioni del software crescono con la disponibilità di memoria (legge di Nathan)
- Frasi celebri:

Chi mai avrà bisogno di più di 640 K di memoria centrale

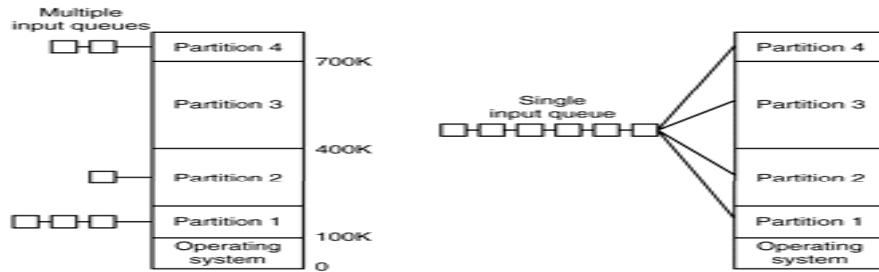
Bill Gates, ~ 1985

Monoprogrammazione



- La memoria divisa tra: parte residente del SO e un'unica applicazione
- Svantaggi:
 - Nessuna protezione
 - Bassa utilizzazione della CPU
- Usato solo nei primi calcolatori e nei Personal Computer della prima generazione

Partizioni Fisse



- Tipico dei SO batch (OS 360)
- Partizioni di diverse dimensioni decise dall'operatore
- Code uniche o separate
- La dimensione della memoria limita il numero di partizioni, e quindi il livello di multiprogrammazione

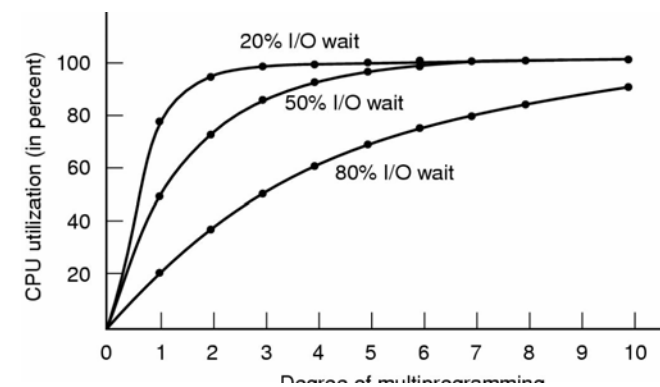
Supporto hardware

- Rilocazione
 - Le diverse partizioni hanno diversi indirizzi iniziali
 - Necessità di traslare gli indirizzi nel codice
 - Rilocazione dinamica effettuata dal caricatore
 - Uso di *indirizzamento autorelativo* e *registri base*
- Protezione
 - I processi non devono accedere al di fuori della loro partizione
 - Uso di registri limite per il controllo dinamico degli indirizzi

Multiprogrammazione

- Modello dell'effetto del numero di partizioni sull'efficienza dell'elaborazione:
 - Utilizzazione della CPU: U_{CPU} , frazione di tempo in cui la CPU è utilizzata
 - Grado di multiprogrammazione: n numero di processi residenti in memoria fra cui la CPU si alterna
 - Frazione di I/O wait: p frazione del tempo di elaborazione che un processo spende in I/O wait
- La CPU è inutilizzata se tutti i processi sono in I/O wait

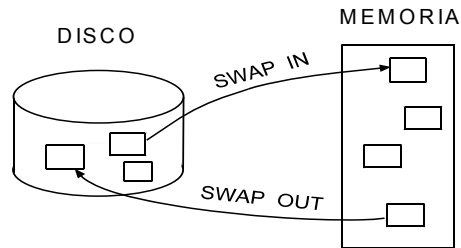
Multiprogrammazione



$$U_{CPU} = 1 - p^n$$

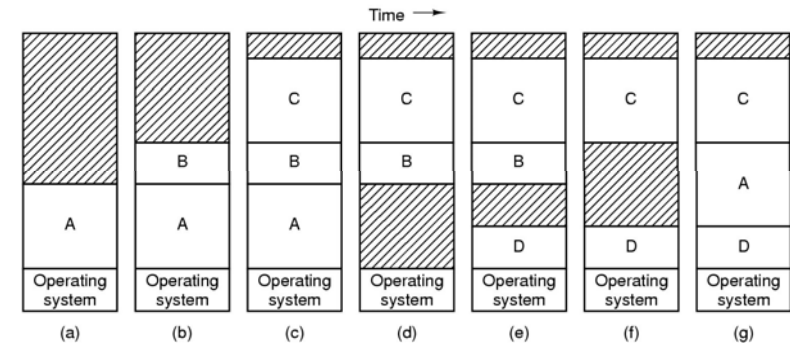
- Se i processi sono I/O bound occorre aumentare il livello di multiprogrammazione a parità di utilizzazione

Partizioni Variabili (Swapping)



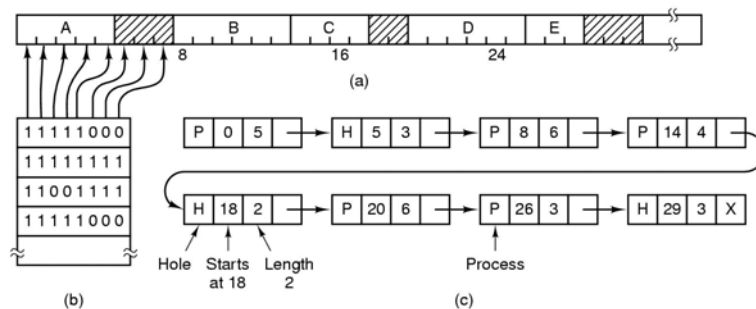
- Introdotto nei primi sistemi interattivi (*time sharing*)
- Alternanza dei processi tra memoria e disco
- *Frammentazione* della memoria
- *Ricompattazione*: costosa (supportata da hardware specifico)
- Problema: crescita dinamica delle partizioni

Frammentazione della memoria



- Ciascuna partizione necessita di blocchi contigui
- La memoria tende a frazionarsi generando frammenti inutilizzabili

Gestione delle partizioni



- (a) Situazione della memoria: 5 partizioni e tre buchi
 (b) Gestione con mappa di bit: 1 occupato 0 libero
 (c) Gestione con liste: P partizione, H hole (buco)

Memoria virtuale: indirizzamento

- I programmi fanno riferimento ad uno spazio di indirizzamento svincolato dalle dimensioni della memoria
- Spazio di indirizzamento virtuale:
 - Cui fanno riferimento i programmi
 - La dimensione dipende dalla piattaforma hardware
Es. indirizzi a 32 bit → spazio virtuale 4 Gbyte
- Spazio di indirizzamento fisico:
 - È quello della memoria centrale
 - La dimensione dipende da quella della memoria

Memoria virtuale: gestione

- La corrispondenza tra indirizzi fisici e virtuali è gestita in modo trasparente dal Sistema Operativo
- Una copia dello spazio virtuale di ciascun processo è allocato su memoria secondaria
- Solo una porzione dello spazio virtuale è allocata a ciascun istante in memoria fisica
- L'elaborazione di ciascun processo può procedere finché le locazioni virtuali accedute sono nella porzione presente in memoria fisica
- Il trasferimento delle porzioni necessarie è *dinamico* e gestito in modo trasparente dal SO

Traduzione e trasferimento

Problemi per un'implementazione efficiente:

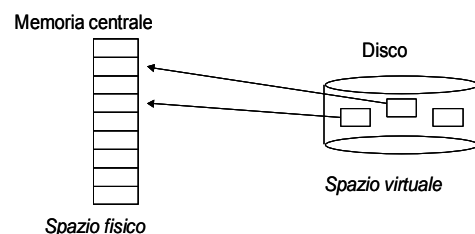
A) Traduzione degli indirizzi:

- La traduzione degli indirizzi da *virtuali* a *fisici* dipende da dove le pagine sono allocate
- La traduzione deve essere effettuata dall'HW come parte del ciclo di esecuzione delle istruzioni

B) Trasferimento delle pagine

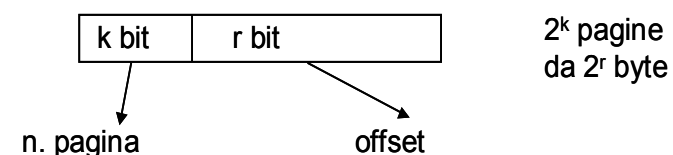
- Scelta delle pagine da mantenere in memoria centrale
- Efficienza del trasferimento e del rimpiazzamento delle pagine

Paginazione



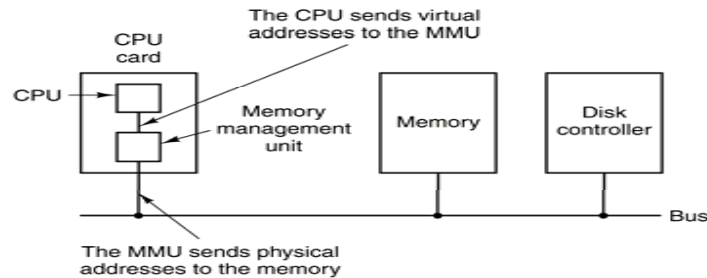
- Sia lo spazio virtuale che quello fisico sono divisi in blocchi di dimensione fissa detti *pagine* (tra 256 byte e 4k):
 - Page virtuali nello spazio virtuale
 - Page frames nello spazio fisico
- Tutti i trasferimenti tra spazio virtuale e fisico avvengono a livello di pagine

Struttura degli indirizzi



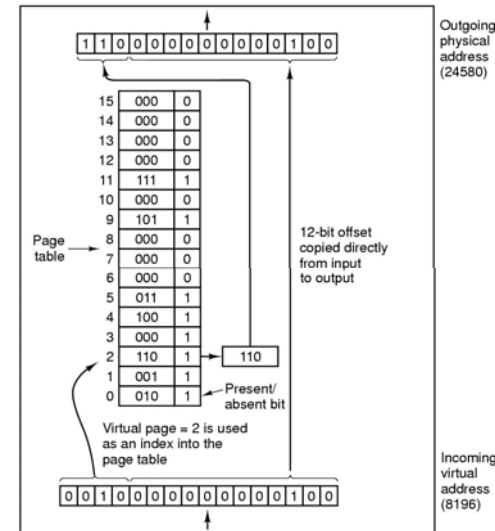
- La struttura dello spazio virtuale si riflette nella struttura degli indirizzi: *indirizzi a due livelli*
- Con pagine di 2^r byte, gli r bit meno significativi costituiscono l'indirizzo nella pagina (*offset*)
- I k bit più significativi costituiscono il *numero della pagina*
- Anche gli indirizzi fisici hanno la stessa struttura
- Il mapping *mantiene l'offset*: l'indirizzo nella pagina non cambia

Traduzione degli indirizzi



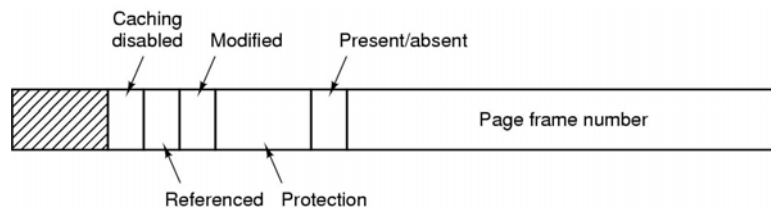
- La traduzione degli indirizzi avviene a livello HW
- La corrispondenza tra pagine virtuali e pagine fisiche (page frames) viene mantenuta nella *Tavola delle Pagine*
- C'è una tavola per ogni processo e viene conservata in memoria fisica

Traduzione degli indirizzi (2)



Spazio Virtuale: 16 pagine da 4K
Spazio Fisico: 8 pagine da 4K
N.B. Il mapping mantiene l'offset

Tavola delle pagine



- Una tavola delle pagine per ogni processo
- Un *descrittore* nella tavola per ciascuna pagina virtuale
 - Presente/assente in memoria fisica
 - Numero del page frame
 - Informazioni sull'uso: usata, modificata
 - Informazioni sulla protezione

Tavola delle pagine: dimensioni

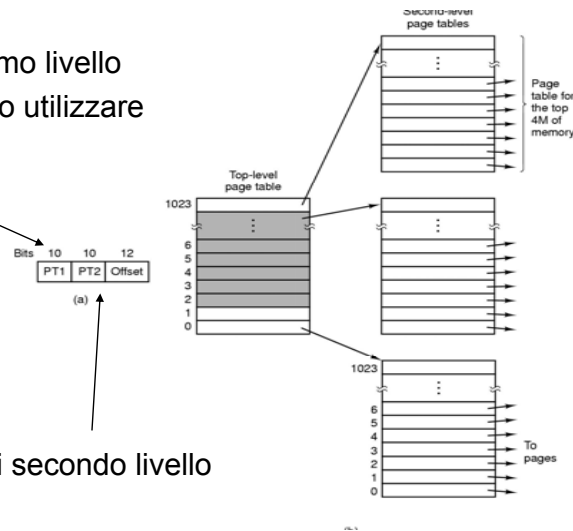
- La tavola delle pagine deve essere in memoria centrale
- Ciascun descrittore occupa più byte
- La tavola rischia diventare troppo grande

ES

- Indirizzi virtuali a 32 bit \Rightarrow spazio virtuale 2^{32} byte = 4 Gbyte
- Pagine da 2^{12} byte = 4 Kbyte
- Numero pagine virtuali $\Rightarrow 2^{32} / 2^{12} = 2^{20}$
- Dimensione dei descrittori $2^2 = 4$ byte
- Dimensione della tavola delle pagine $2^{20} \cdot 2^2$ byte = 4 Mbyte

Tavole delle pagine a più livelli

PT1 indica nella tavola di primo livello quale tavola di secondo livello utilizzare



PT2 indica nella tavola di secondo livello il descrittore di pagina

Accessi a memoria

- Per tradurre un indirizzo occorre consultare la tavola delle pagine
- La traduzione genera nuovi accessi a memoria
- Gli accessi a memoria degradano le prestazioni
- Grazie alla località è possibile utilizzare memorie associative
- Il TLB (Translation Lookaside Buffer) mantiene le ultime corrispondenze tra pagine virtuali e fisiche
- La tavola delle pagine è acceduta solo se l'accesso nel TLB fallisce, cioè raramente

Translation Lookaside Buffer

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

- Il bit valid ha la stessa funzione che nelle cache
- Il TLB contiene anche informazioni di uso e protezione

Page fault

- Quando la pagina virtuale indirizzata non è presente in memoria centrale l'HW genera un trap
- Si dice che il processo è andato in *Page Fault*
 - Il processo viene interrotto e messo in stato *blocked*
 - Il S.O. cerca e carica la pagina
 - Il processo è pronto a ripartire e diventa *ready*
- Nell'ambito della multiprogrammazione i page fault sono semplicemente una quota di I/O addizionale
- È come se il processo fosse un po' più I/O bound
- Basta compensare con il livello di multiprogrammazione

Paginazione a domanda

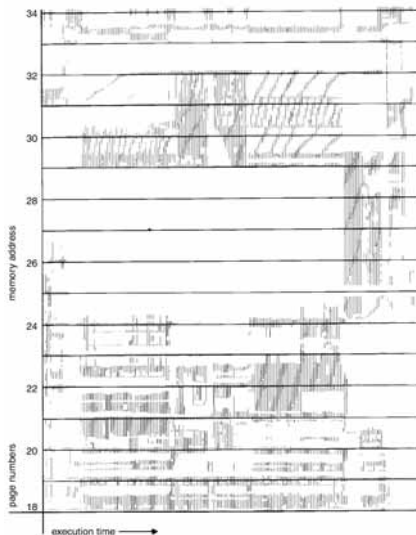
Due politiche fondamentali, spesso usate in modo ibrido

- Demand Paging
 - Si caricano le pagine a richiesta
 - Transitorio ogni volta che un processo viene portato fuori dalla memoria
- Swapping
 - Tutte le pagine che processo usa vengono spostate tra memoria centrale e secondaria in blocchi contigui
 - Quando il processo riparte trova subito tutte le pagine che gli servono

Località dei riferimenti

- Località Spaziale
 - | forte probabilità di fare riferimento a indirizzi contigui in tempi vicini
- Località Temporale
 - | forte probabilità di fare riferimento allo stesso indirizzo in tempi vicini
- Tutti i programmi presentano, seppure in misura diversa, una forte località dei riferimenti
- La località consente di mantenere la frequenza di page fault (PFR: Page Fault Rate) ragionevolmente bassa

Mapa di località



- Comportamento tipico
- Ascissa: tempo
- Ordinata: indirizzi acceduti
- In ciascun intervallo di tempo i riferimenti si concentrano su poche pagine
- Notare l'evoluzione del working set

Rimpiazzamento delle pagine

- Ogni processo ha un numero fisso di page frames a disposizione
- Problema: scelta della pagina da eliminare quando si verifica un *page fault* (quella con minore probabilità di essere usata)
- Strategie base di rimpiazzamento
 - Random: la prima che capita (semplice da implementare)
 - FIFO: quella che è entrata per prima (indipendentemente dai riferimenti)
 - LRU: (*Least Recently Used*) quella che non viene usata da più tempo

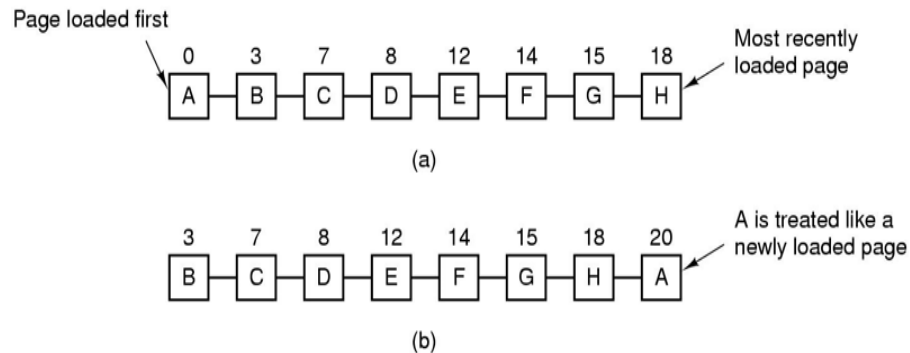
Algoritmo ottimo

- Convieni sempre buttare fuori la pagina che verrà usata *nel futuro più remoto*
- L'algoritmo ottimo è quello che agisce conoscendo l'intera stringa dei riferimenti (che non si conosce!)
- È un caso ideale e costituisce un punto di riferimento
- Gli algoritmi reali si basano sulla conoscenza del passato
- L'ipotesi che si fa è che il comportamento nel *futuro prossimo* sarà simile a quello nel *passato prossimo*

Algoritmo NRU

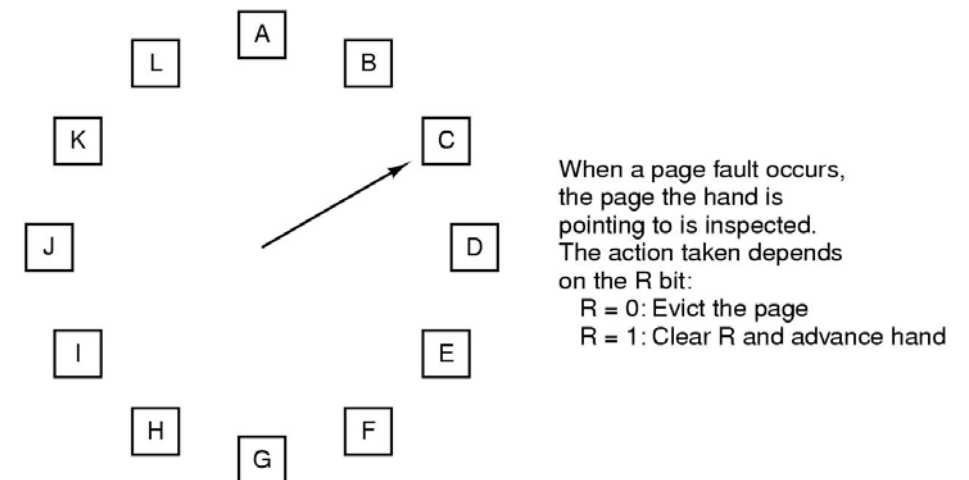
- NRU (Not Recently Used)
- Ogni pagina ha un *Reference bit* e un *Modified bit*
- Le pagine vengono classificate
 1. Non referenziate e non modificate
 2. Non referenziate e modificate
 3. Referenziate e non modificate
 4. Referenziate e modificate
- La pagina da buttare viene selezionata random dalla classe non vuota più bassa

Algoritmo 'second chance'



- Lista gestita FIFO: in base all'ordine di entrata
- Se la pagina in testa ha $R=1$ viene messa in coda con $R=0$
- Si passa alla successiva finché non se ne trova una con $R=0$

Algoritmo dell'orologio



Working set

- $W(k,t)$, detto *working set* all'istante t , è l'insieme delle pagine riferite negli ultimi k accessi (k è la *finestra*)
- La cardinalità di $W(k,t)$ varia con t , perché il processo ha bisogno di più o meno page frames
- L'assunzione è che nei prossimi k riferimenti il processo accederà solo a pagine dell'insieme $W(k,t)$
- Al processo viene garantito un numero di page frames variabile e pari a $W(k,t)$
- Questo consente di mantenere in memoria le pagine del suo working set $W(k,t)$
- Una scelta adeguata di k consente di limitare il PFR

Anomalia di Belady del FIFO

All pages frames initially empty

	0	1	2	3	0	1	4	0	1	2	3	4	
Youngest page		0	1	2	3	0	1	4	4	4	2	3	3
			0	1	2	3	0	1	1	1	4	2	2
Oldest page				0	1	2	3	0	0	0	1	4	4
				P	P	P	P	P	P		P	P	

9 Page faults

FIFO con 3 frames

(a)

	0	1	2	3	0	1	4	0	1	2	3	4	
Youngest page		0	1	2	3	3	3	4	0	1	2	3	4
			0	1	2	2	2	3	4	0	1	2	3
Oldest page				0	1	1	1	2	3	4	0	1	2
					0	0	0	1	2	3	4	0	1
					P	P	P	P	P	P	P	P	P

10 Page faults

FIFO con 4 frames

(b)

Comportamento dello LRU

Reference string 0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 1 3 4 1

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	1	3	4	1
		0	2	1	3	5	4	6	3	7	4	7	7	3	3	5	3	3	3	1	7	1	3	4
			0	2	1	3	5	4	6	3	3	4	4	7	7	7	5	5	5	3	3	7	1	3
				0	2	1	3	5	4	6	6	6	6	4	4	4	4	4	4	4	4	4	5	5
					0	2	2	1	1	1	1	1	1	1	1	6	6	6	6	6	6	6	6	6
						0	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
							0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Page faults P P P P P P P P P P P P P

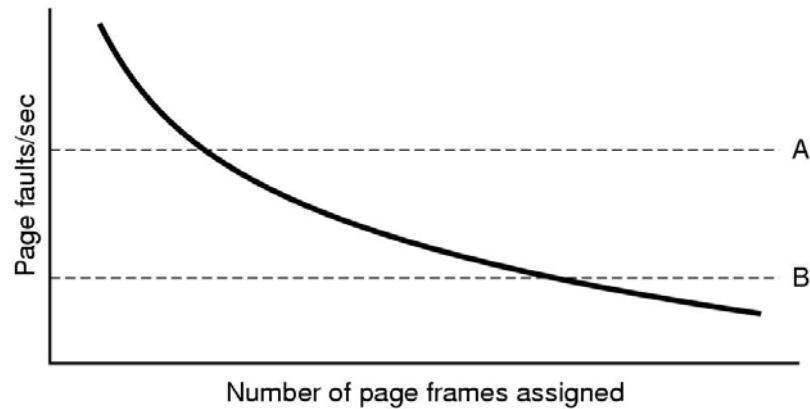
Distance string ∞ ∞ ∞ ∞ ∞ ∞ ∞ 4 ∞ 4 2 3 1 5 1 2 6 1 1 4 2 3 5 3

- Una riga per ogni pagina virtuale (nell'esempio 8)
- Le prime quattro sono in memoria
- $M(m,r)$ sono i primi m elementi dall'alto della colonna r
- È verificato che $M(m,r) \subseteq M(m+1,r)$

Algoritmi a stack

- L'anomalia di Belady mostra come non sempre un aumento del numero di page frame migliora il PFR
- Per LRU è verificato che $M(m,r) \subseteq M(m+1,r)$
 - Con $m+1$ page frame ad ogni passo ci sono almeno tutte le stesse pagine che c'erano con m page frame
 - Il PFR (Page Fault Rate) non può peggiorare: è *monotono crescente* con m
- Algoritmi che godono di questa proprietà vengono detti *Algoritmi a Stack*
- Gli algoritmi a stack consentono il *controllo adattivo* del PFR (Page Fault Rate)

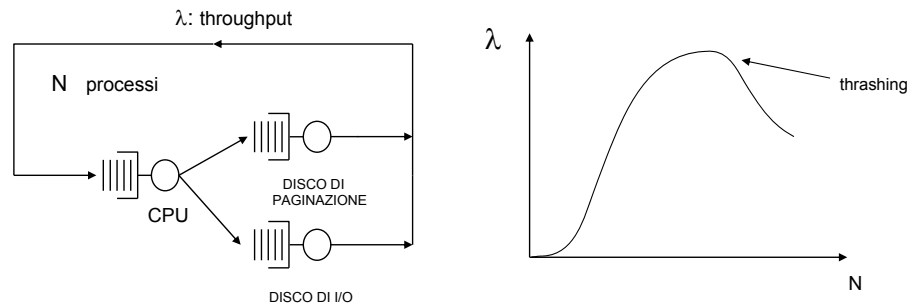
Andamento del PFR



Thrashing

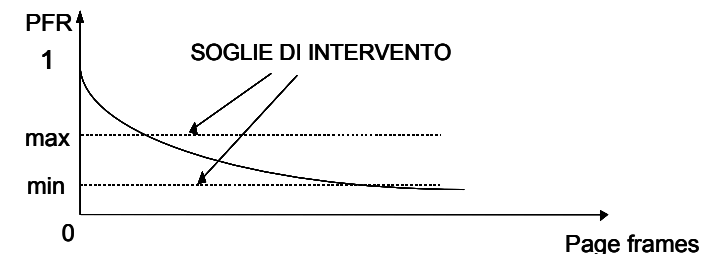
- Il SO cerca aumentare il numero di processi in memoria e quindi l'utilizzazione della CPU e il throughput
- Diminuiscono i page frames assegnati a ciascun processo e quindi aumenta il suo *Page Fault Rate (PFR)*
- Se ciascun processo ha troppi pochi page frames il suo PFR cresce bruscamente e il sistema rallenta
- Il sistema va in *thrashing* (*to trash*: battere con un bastone)
- Corrisponde alla situazione limite in cui disco di paginazione viene saturato

Modello del thrashing



- Sistema di code chiuso con N processi in multiprogrammazione
- Al crescere di N i processi hanno meno frame, e il PFR sale
- La coda di paginazione satura e quella di CPU svuota
- Il throughput λ invece di salire scende

Controllo del Thrashing



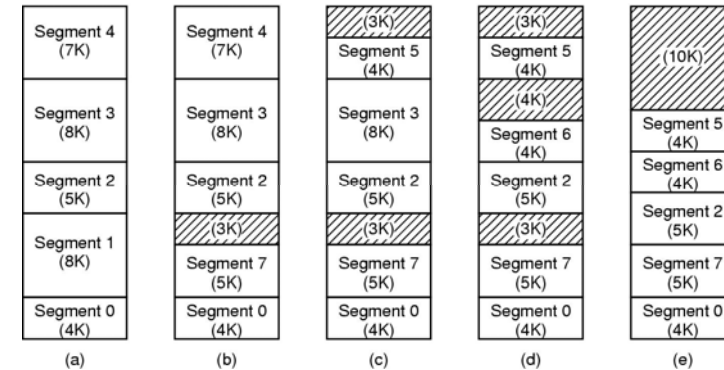
- Il S.O. varia dinamicamente il numero dei processi in memoria ed il numero di page frame loro assegnate
- Il PFR di ciascun processo viene monitorato
- Il numero di page frame assegnate al processo viene variato in modo da mantenere il PFR nella fascia stabilita:

$$\min < PFR < \max$$

Segmentazione

- Ciascun processo vede più spazi di indirizzamento indipendenti: *segmenti*
- Vantaggi della segmentazione
 - Gestione di strutture dinamiche (stack)
 - Migliore località: associazione di segmenti a procedure e strutture di dati
 - Unità logiche: condivisione/protezione
- Svantaggi della segmentazione
 - Più complessa da gestire
 - La dimensione variabile dei segmenti causa la frammentazione della memoria

Frammentazione della memoria



- (a) Situazione iniziale
- (d) Frammentazione eccessiva
- (e) Ricompattazione

Frammentazione esterna ed interna

- Frammentazione esterna
 - Si verifica nei sistemi a segmentazione
 - Spazi inutilizzabili in memoria tra i segmenti
 - Spreco della memoria centrale (anche il 40%)
- Frammentazione interna
 - Si verifica nei sistemi a paginazione
 - Un programma non occupa un numero intero di pagine
 - Spreco medio del 50% sull'ultima pagina
 - Occorre *limitare* la dimensione delle pagina

Dimensione delle pagine

- Compromesso tra:
 - Dimensione della tavola delle pagine
 - Spreco dovuto a frammentazione interna

$$\text{overhead} = \frac{s \cdot e}{p} + \frac{p}{2}$$

s : dimensione media processo (bytes)
 p : dimensione pagina (bytes)
 e : dimensione page table entry (bytes)

The diagram shows the formula with annotations: 'page table space' points to the first term, and 'internal fragmentation' points to the second term.

- Si cerca di minimizzare la quantità di memoria complessiva 'sprecata' per ogni pagina (overhead)
- Il minimo si ha per $p = \sqrt{2se}$

Segmentazione impaginata

- Ciascun segmento è diviso in pagine
- Le pagine allocate in modo non contiguo
- Eliminata la frammentazione esterna
- Indirizzi a tre livelli:
 - *Segmento*
 - *Pagina*
 - *Offset*
- Traduzione degli indirizzi più complessa
 - *Tavola dei segmenti*
 - *Tavola delle pagine per ciascun segmento*
- Adottato in Multics e nella piattaforma Intel (dal 386 in poi)

Memoria virtuale nel Pentium

- 16 K segmenti per ogni processo
- Offset nei segmenti a 32 bit: segmenti da 4 Gbyte
- I registri di segmento 'puntano' ai segmenti correntemente utilizzati dal processo
- Gli indirizzi a 32 due bit nelle istruzioni sono gli offset nei segmenti
- Segmenti impaginati in pagine da 4 K byte
- Molti sistemi operativi usano un unico segmento di 4 Gbyte per la memoria virtuale: paginazione pura