Sistemi Operativi A
Parte IV - La gestione della memoria

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

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 posizione nella memoria fisica

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 consentito che più processi utilizzino dati comuni in modo controllato
  - deve essere possibile condividere risorse comuni (es. codice)

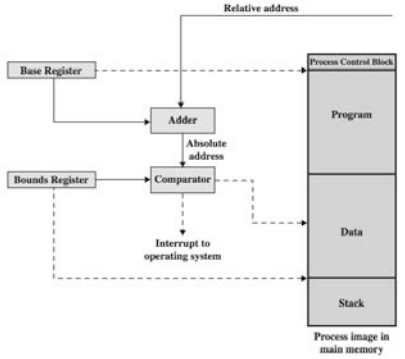
Rilocazione (1)

- Ogni programma viene tradotto in linguaggio macchina come se la sua posizione al momento dell'esecuzione coincidesse con l'inizio della memoria
  - gli indirizzi partono dal valore convenzionale 0, e sono chiamati indirizzi rilocabili
  - le istruzioni che fanno riferimento a questi indirizzi sono contrassegnate come istruzioni rilocabili
- Quando il programma va in esecuzione, gli indirizzi rilocabili sono trasformati in indirizzi assoluti che si riferiscono alle posizioni della memoria fisica assegnate al programma

Rilocazione (2)

- La rilocazione dinamica prevede che ogni riferimento alla memoria da parte di un programma in esecuzione sia espresso in forma indipendente dalla posizione effettiva
  - un programma autorilocante esprime ogni accesso a memoria in modo relativo alla propria posizione
  - un programma che utilizza un registro base esprime tutti gli indirizzi come offset rispetto ad esso
- Con l'utilizzo di un registro base più programmi possono risiedere in memoria in modo indipendente
  - quando il processo che esegue un programma va in esecuzione, si carica nel registro base l'indirizzo della prima cella occupata dal programma

Rilocazione e protezione del codice



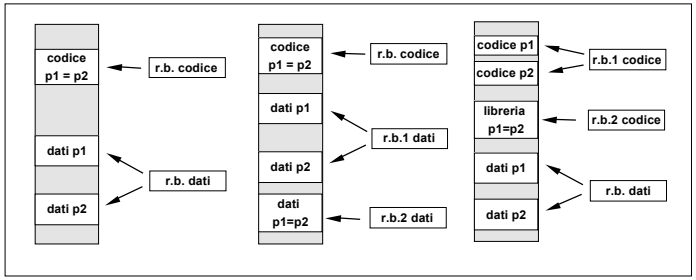
(Stallings, 2005)

Segmentazione e condivisione del codice (1)

- Utilizzando più registri base si possono dividere i programmi in segmenti il cui indirizzamento è indipendente
- E' necessario riconoscere quale registro base deve essere utilizzato per convertire un indirizzo rilocabile in indirizzo assoluto
  - in base al tipo di operazione: riferimento a un operando (accesso a dato, operazione aritmetica, logica o di movimento dati) o a una istruzione (accesso a codice, istruzione di salto)
  - in base al valore dell'indirizzo logico: ogni registro base "copre" un certo intervallo di indirizzi

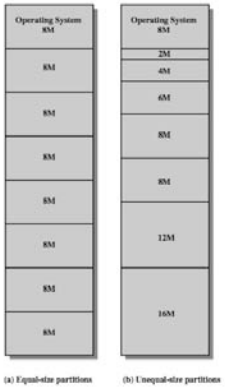
Segmentazione e condivisione del codice (2)

- Con più registri base si possono ottenere
  - la condivisione del codice (es. fork)
  - la condivisione di aree dati (es. memoria condivisa)
  - la condivisione di funzioni (es. librerie)



Partizionamento costante (1)

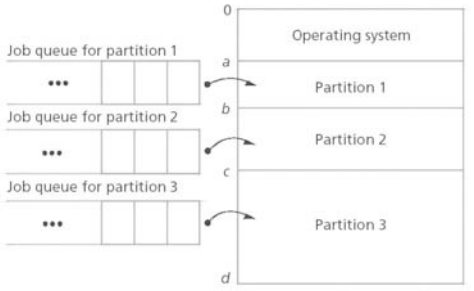
- La memoria centrale è divisa in un insieme di regioni disgiunte chiamate *partizioni*
  - i processi (job) sono allocati in una partizione specifica (semplice) o in una partizione adeguata (efficiente)
  - rimane spazio non occupato all'interno delle partizioni (*frammentazione interna*)
  - se le partizioni sono occupate il S.O. può liberarne una temporaneamente (*swapping*)
  - programmi troppo grandi devono essere segmentati all'origine (*overlay*)



(Stallings, 2005)

Partizionamento costante (2)

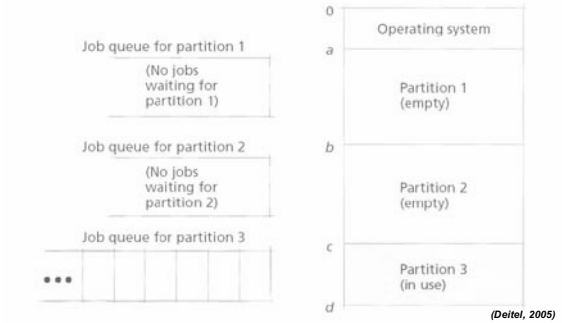
- Ogni partizione può avere ha un propria coda di processi
  - semplice da implementare
  - i processi sono divisi in classi disgiunte



(Deitel, 2005)

Partizionamento costante (3)

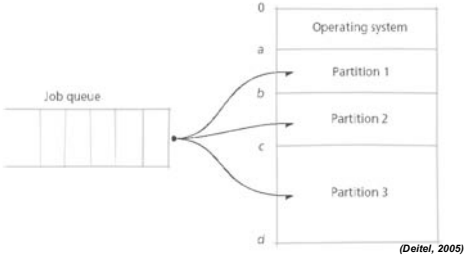
- La soluzione è semplice ma inefficiente
  - possono esserci partizioni vuote e partizioni con lunghe code di processi in attesa



(Deitel, 2005)

Partizionamento costante (4)

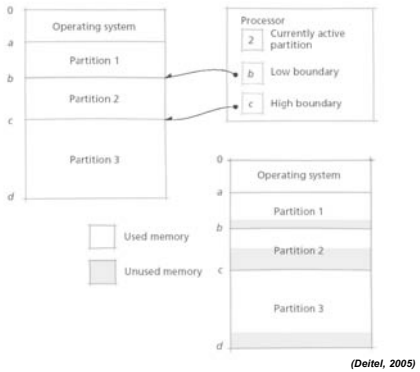
- Una gestione dinamica consente di allocare un processo ad una partizione libera, purché appropriata
  - la coda può bloccarsi se per il primo processo non c'è una partizione adeguata (partizioni di dimensioni diverse)
  - la gestione della coda dei processi è più complessa



(Deitel, 2005)

Partizionamento costante (5)

- E' semplice gestire la protezione ma si crea frammentazione

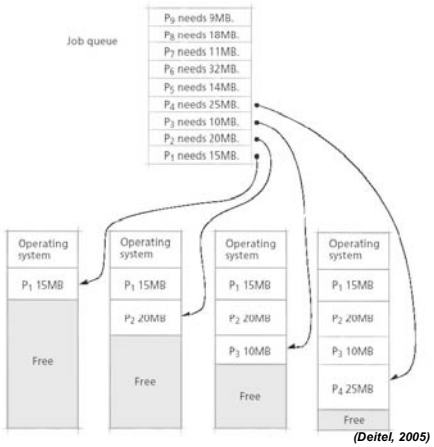


(Deitel, 2005)

Partizionamento dinamico (1)

- La memoria centrale è una partizione unica di cui vengono allocati parti variabili in numero e dimensione in funzione delle esigenze
  - ogni processo (o segmento) è allocato in un'area sufficientemente ampia per contenerlo e occupa esattamente (…) la memoria che serve
  - la decisione su quale area scegliere tra quelle adeguate dipende da algoritmi di allocazione
  - gli spazi liberi tra le aree occupate generano *frammentazione esterna*
  - periodicamente la memoria deve essere ricompattata (*shuffling*)

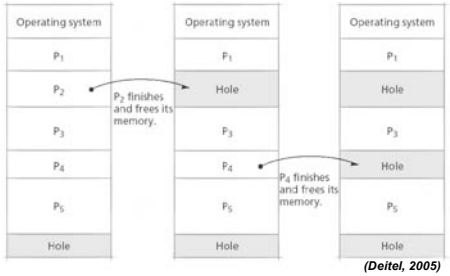
Partizionamento dinamico (2)



(Deitel, 2005)

Partizionamento dinamico (3)

- La dinamica dei processi lascia aree di memoria libere secondo una geometria impredicibile
  - la quantità di memoria cmplessivamente libera può non essere utilizzabile perché frammentata



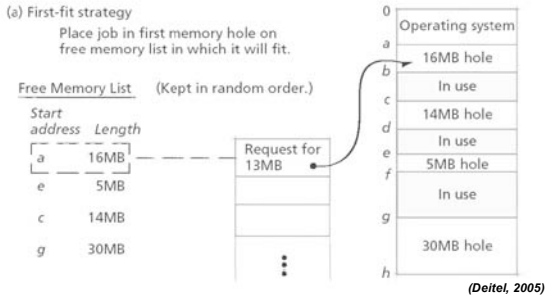
(Deitel, 2005)

Strategie di allocazione (1)

- La allocazione di un nuovo processo o segmento comporta due problemi
  - la scelta dell'area da occupare
  - la possibilità che non vi siano aree contigue adeguate anche in presenza di uno spazio libero totale sufficiente
- La scelta dell'area da occupare può avvenire secondo quattro strategie
  - *first fit*: si sceglie la prima area sufficientemente ampia in una scansione lineare delle aree libere (è veloce)
  - *best fit*: si sceglie l'area di dimensioni più prossime a quelle necessarie (lascia poco spazio difficilmente usabile)
  - *worst fit*: si sceglie l'area più ampia (lascia spazio usabile)
  - *next fit*: variante dell'algoritmo *first-fit*, inizia la scansione dall'ultima area allocata

Strategie di allocazione (2)

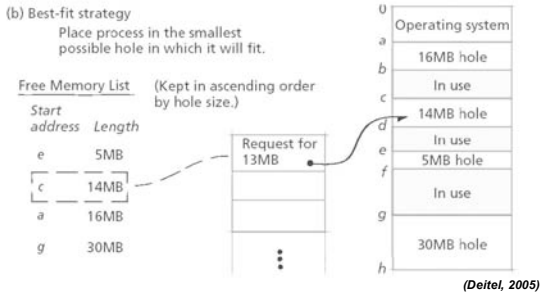
- *First fit*: si sceglie la prima area sufficientemente ampia in una scansione lineare delle aree libere
  - è veloce, ma crea frammentazione in modo casuale



(Deitel, 2005)

Strategie di allocazione(3)

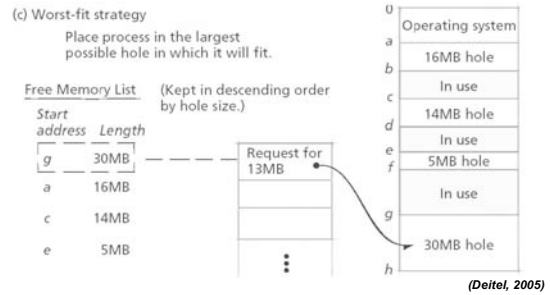
- *Best fit*: si sceglie l'area di dimensioni più prossime a quelle necessarie
  - crea frammentazione perché le aree rimaste libere sono difficilmente usabili



(Deitel, 2005)

Strategie di allocazione (4)

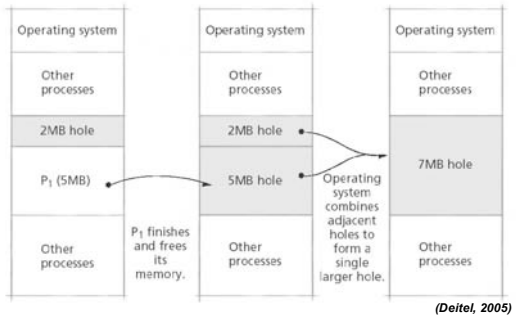
- Worst fit: si sceglie l'area più ampia
  - lascia spazio libero più facilmente riusabile



(Deitel, 2005)

Ricomposizione della memoria libera (1)

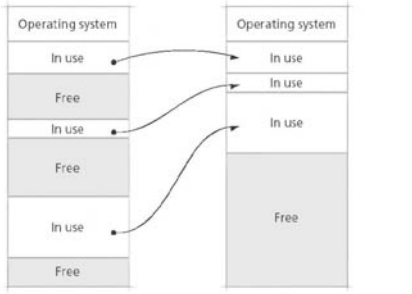
- Aree contigue liberate dall'evolvere dei processi vengono collassate



(Deitel, 2005)

Ricomposizione della memoria libera (2)

- Periodicamente il S.O. può ricompattare la memoria per creare uno spazio libero contiguo più grande (deframmentazione)



(Deitel, 2005)

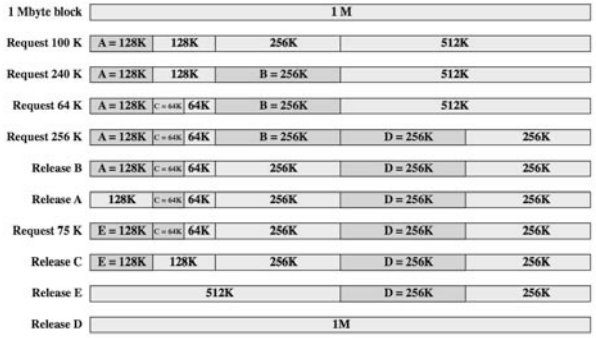
Buddy System (1)

- Un compromesso tra gli schemi di allocazione per partizionamento fisso e dinamico
- Utilizzato (in forma modificata) da Unix SVR4 per l'allocazione della memoria del kernel
- La memoria è allocata a blocchi di dimensione 2^K
  - $L <= K <= U$
  - 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 $2^{U-1} < S <= 2^U$ 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 $2^{U-2} < S <= 2^{U-1}$ allora uno dei due blocchi è allocato per intero
  - altrimenti uno dei due blocchi è diviso in due metà 2^{U-2}
  - questo processo viene ripetuto fino ad ottenere il più piccolo blocco di dimensione >= S
- Due *buddies* vengono riuniti quando diventano entrambi liberi

Buddy System (3)



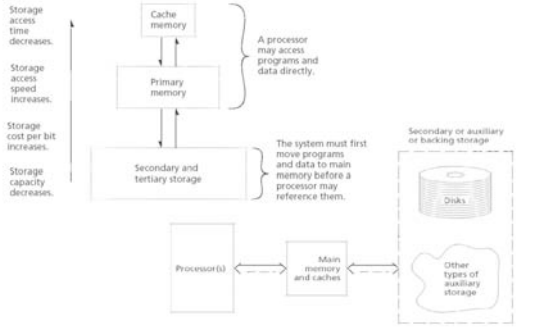
(Stallings, 2005)

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, $2^{i-1} < k <= 2^i$
  - si esamina la lista i-esima
  - se è vuota si esamina la lista i+1-esima
  - … e così via fino alla lista j-esima
  - l'area individuata viene divisa secondo l'algoritmo descritto e le liste vengono aggiornate

Gerarchie di memoria (1)

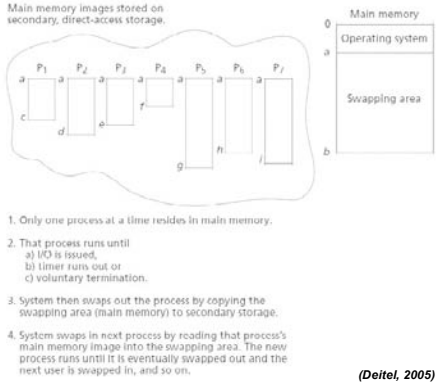
- Le memorie fisiche sono organizzate in una gerarchia dove variano in modo inverso capacità e velocità di accesso



(Deitel, 2005)

Gerarchie di memoria (2)

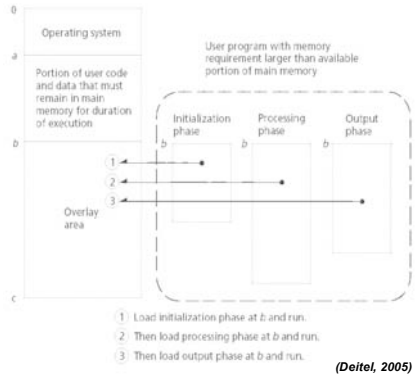
- Swapping
  - un solo processo alla volta è residente in memoria, gli altri risiedono su disco e sono caricati per l'esecuzione



(Deitel, 2005)

Gerarchie di memoria (3)

- Overlay
  - parti diverse della memoria di un processo che non sono utilizzate insieme sono residenti su disco e "sovrapposte" sullo stesso spazio di indirizzi



(Deitel, 2005)

Memoria virtuale (1)

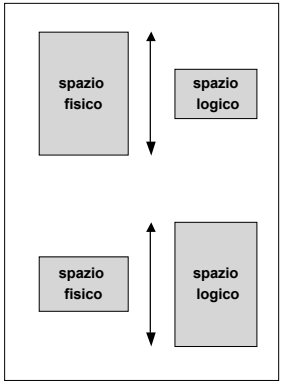
- La presenza di regole e dispositivi di corrispondenza tra spazi di indirizzamento logico e fisico porta al concetto di *memoria virtuale*
  - un processo vede una memoria "logica" indipendente dalle reali caratteristiche "fisiche"
  - la memoria logica è organizzata in funzione delle esigenze del processo / 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

Memoria virtuale (2)

- Siano *v* la dimensione dello spazio di indirizzamento virtuale di un processo, e *f* quella dello spazio di indirizzamento fisico della macchina
  - *v* dipende dalla dimensione dei registri di indirizzamento e PC
  - *f* dipende dal numero di linee del bus indirizzi
- Si hanno tre relazioni tra le due quantità
  - $v < f$: un programma non può indirizzare tutta la memoria fisica
  - $v > f$: un programma può indirizzare uno spazio logico più ampio di quello fisico
  - $v = f$: può essere visto come caso particolare del caso v > f, dipende dalla memoria effettivamente installata

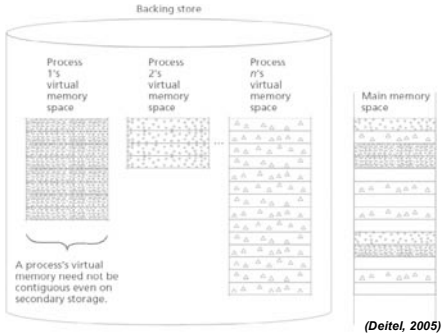
Memoria virtuale (3)

- Se $v < f$
  - più programmi possono risiedere contemporaneamente in memoria
  - ciascuno vede al massimo una "finestra" di dimensioni pari allo spazio di indirizzamento logico
- *Se* $v > f$
  - occorre un meccanismo hw/sw che, di volta in volta, porti nella memoria fisica una parte della memoria logica, "parcheggiando" il resto su memoria di massa
  - la memoria fisica è una "finestra" sulla memoria virtuale



Memoria virtuale (4)

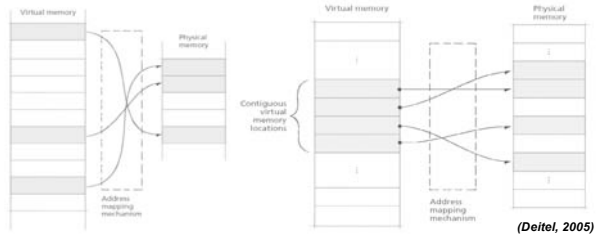
- Parte dello spazio di indirizzi può essere presente nella memoria centrale solo quando è necessario accedervi
  - ricorda i metodi di allocazione basati su swapping e su overlay, ma la gestione è a cura dello hardware di indirizzamento



(Deitel, 2005)

Memoria virtuale (5)

- 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



(Deitel, 2005)

Memoria virtuale (6)

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

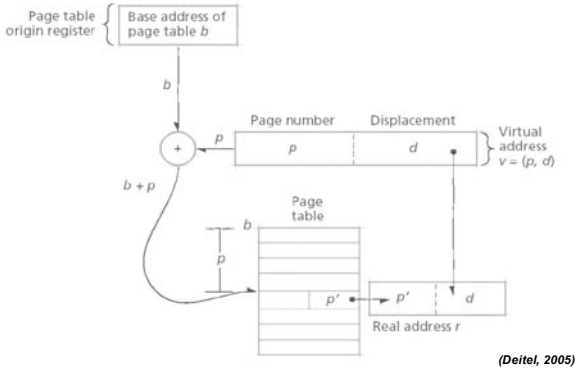
Real	Real		Virtual		
Single-user dedicated systems	Real memory multiprogramming systems		Virtual memory multiprogramming systems		
	Fixed-partition multi-programming	Variable-partition multi-programming	Pure paging	Pure segmentation	Combined paging and segmentation
	Absolute	Re-locatable			

(Deitel, 2005)

Memoria virtuale a pagine (1)

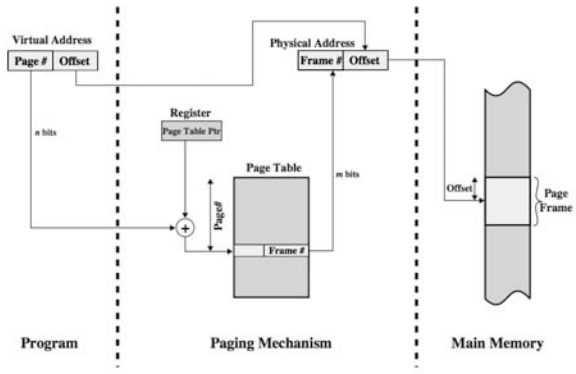
- Sono definite le seguenti quantità
  - t il numero di linee del bus indirizzi, 2^t il numero di celle della memoria fisica teorica, $t = f + s$
  - i il numero di bit del registro di indirizzamento, 2^i la dimensione dello spazio di indirizzamento logico, $i = p + s$
- La memoria fisica teorica complessiva è divisa in 2^f pagine fisiche (*frame*) di 2^s celle ciascuna
- La memoria logica indirizzabile da un programma è divisa in 2^p pagine logiche di 2^s indirizzi ciascuna
- L'associazione tra pagine logiche e pagine fisiche è descritta dalla *tabella della pagine* (*page table*)

Memoria virtuale a pagine (2)



(Deitel, 2005)

Memoria virtuale a pagine (3)

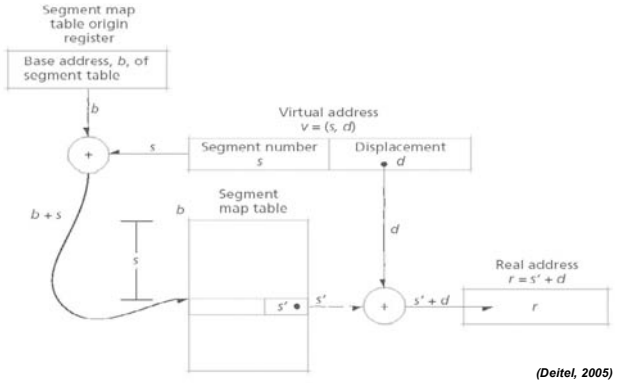


(Stallings, 2005)

Memoria virtuale a segmenti (1)

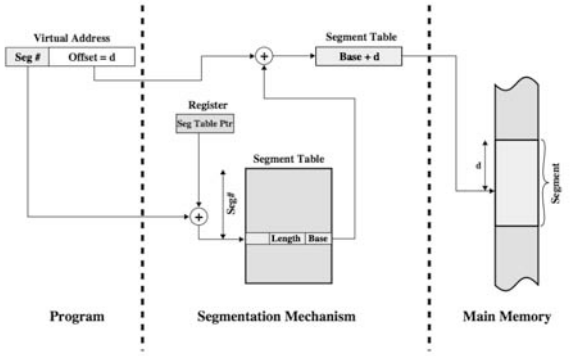
- Sono definite le seguenti quantità
  - t il numero di linee del bus indirizzi, 2^t il numero di celle della memoria fisica massima teorica
  - i il numero di bit del registro di indirizzamento, 2^i la dimensione dello spazio di indirizzamento logico, i = p + s
- Un programma è diviso in blocchi logici (al massimo 2^p) corrispondenti a parti di codice o di dati, detti *segmenti*
  - i segmenti possono essere di dimensioni diverse, ognuno può contenere al massimo 2^s indirizzi
  - i segmenti possono essere allocati in memoria indipendentemente, purché ogni segmento sia compatto
  - i segmenti di un programma sono complessivamente individuati da un insieme di *registri di segmentazione* (*segment table*)

Memoria virtuale a segmenti (2)



(Deitel, 2005)

Memoria virtuale a segmenti (3)



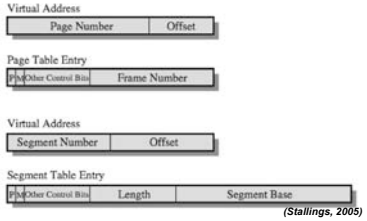
(Stallings, 2005)

Paginazione vs. segmentazione (1)

- 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

Paginazione vs. segmentazione (2)

- Paginazione e segmentazione utilizzano strutture informative simili
  - ogni elemento della *page table* ha un bit di presenza e un bit di modifica
  - anche ogni elemento della *segment table* ha un bit di presenza e un bit di modifica
  - altri bit possono descrivere protezioni e condivisioni



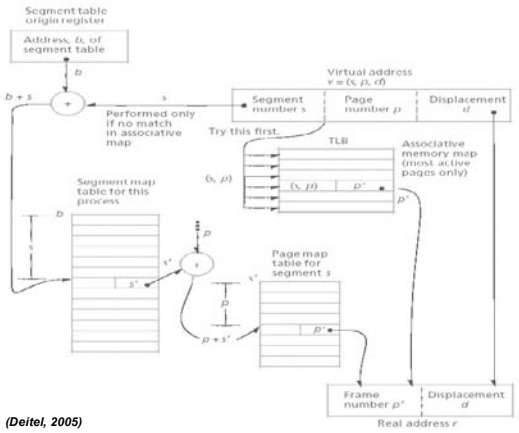
(Stallings, 2005)

- La traduzione da indirizzo logico a indirizzo fisico è simile ma nel caso della segmentazione si esegue un'operazione di somma

Paginazione vs. segmentazione (3)

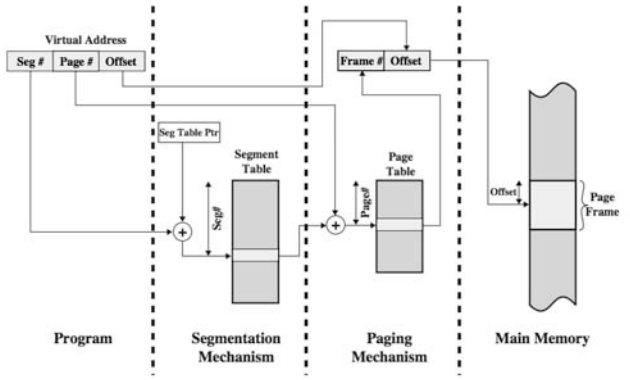
- Nella *segment table* ogni elemento descrive l'indirizzo di partenza e la lunghezza del segmento
  - un segmento può aumentare o diminuire di dimensione (es. allocazione dinamica)
  - la validità di un indirizzo si misura sulla lunghezza del segmento
- Segmenti di dimensioni variabile causano frammentazione esterna e sono più critici nei confronti delle operazioni di swap in/swap out
- Protezione e condivisione a livello dei segmenti sono visibili al programmatore

Combinazione di segmentazione e paginazione (1)



(Deitel, 2005)

Combinazione di segmentazione e paginazione (2)



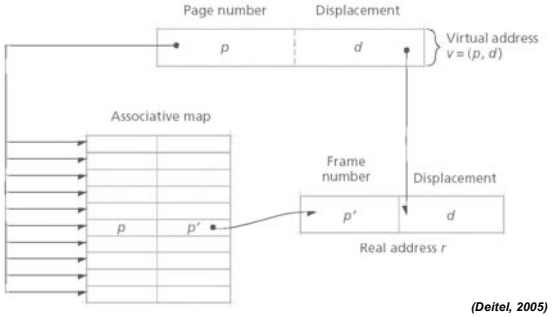
(Stallings, 2005)

Struttura della page table

- Ogni processo ha una propria *page table* il cui uso è trasparente al processo
  - insieme di registri veloci: dimensioni ridotte, una sola tabella per il processo attivo, impostata durante il *context switch*
  - area di memoria riservata: dimensioni ampie, una copia differente per ogni processo, puntatore alla tabella corrente impostato durante il *context switch*
  - problema: tempi di accesso raddoppiati
  - area di memoria più mappa associativa (*translation lookaside buffer*): tempi di accesso statisticamente ragionevoli (*hit ratio*)
  - la mappa associativa è azzerata ad ogni *context switch*

Mappa associativa delle pagine (1)

- E' un'area *cache* che contiene gli elementi della *page table* che sono stati usati più recentemente

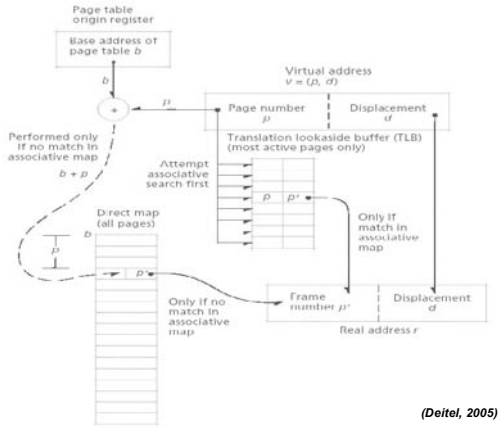


(Deitel, 2005)

Mappa associativa delle pagine (2)

- Dato un indirizzo logico si esamina la mappa
  - se l'elemento della *page table* cercato è presente (*hit*) di recupera il numero di pagina fisica (*frame*) e si costruisce l'indirizzo fisico
  - se l'elemento è assente (*miss*) si accede alla *page table* del processo
  - se il bit che segnala la pagina valida è1 la pagina fisica è presente in memoria, si compone l'indirizzo fisico e si accede
  - se il bit è 0 si segnala la mancanza di pagina (*page fault*) e la pagina viene portata in memoria centrale
  - la mappa associativa viene aggiornata con il nuovo elemento

Mappa associativa delle pagine (3)



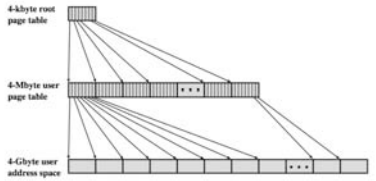
(Deitel, 2005)

Gestione della page table

- Le moderne architetture supportano spazi di indirizzamento molto grandi
  - da 32 a 64 bit di indirizzamento logico
  - una dimensione di pagine di 4KByte richiede una *page table* potenziale di 2^{20} elementi su un'architettura a 32 bit
- La *page table* non può risiedere completamente in memoria centrale
  - le *page table* dei processi sono allocate in memoria virtuale e soggette a paginazione
  - durante l'esecuzione una parte della *page table* deve essere in memoria centrale → deve comprendere il riferimento alla pagina in esecuzione

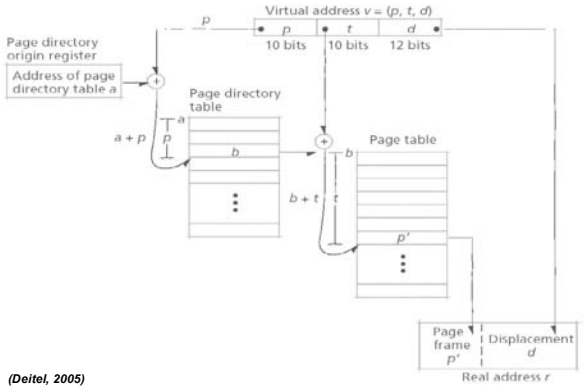
Page table multilivello (1)

- Page table grandi richiedono più pagine per l'allocazione
- Le page tale possono essere organizzate in gerarchie
  - es. Intel: 386, Pentium hanno *page table* a due livelli (p1, p2)
  - p1 indicizza il primo livello i cui elementi puntano a pagine che contengono elementi indicizzati dal secondo livello (p2)
  - il primo livello è in memoria (*directory*), il secondo è soggetto a swap



(Stallings, 2005)

Page table multilivello (2)

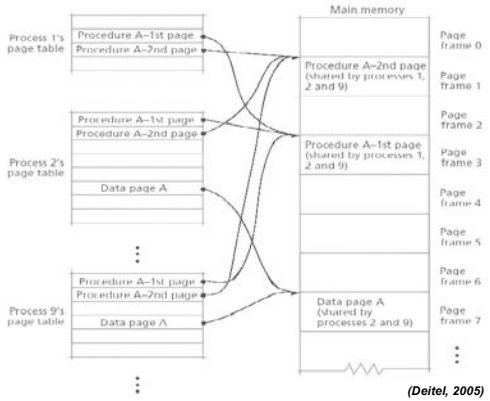


(Deitel, 2005)

Condivisione di memoria (1)

- La condivisione di memoria tra più processi dipende da fattori legati alla logica dei processi e dei dati che elaborano
  - condivisione del codice (es. due processi eseguono lo stesso programma)
  - condivisione di parte del codice (es. librerie condivise)
  - condivisione di parte dei dati (es. comunicazione tra due processi, accesso a dati globali condivisi)
- La condivisione si appoggia naturalmente ad un concetto di segmento
- I segmenti condivisi hanno diritti di accesso che dipendono dal tipo di condivisione
  - es. lettura (dati globali), scrittura (comunicazione), esecuzione (codice)
- La condivisione di memoria paginata è significativa solo nell'ambito di una segmentazione logica

Condivisione di memoria (2)



(Deitel, 2005)

Gestione dello spazio di memoria (1)

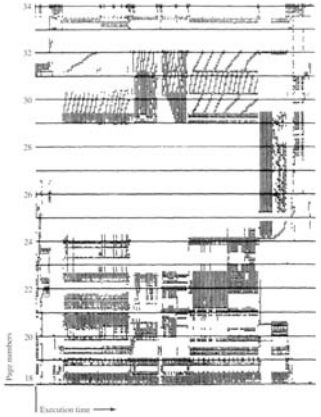
- In una gestione a pagine la dimensione delle pagine influenza la complessità costruttiva e l'efficacia
  - grandi dimensioni → spreco di spazio
  - piccole dimensioni → tabella della pagine molto grande
- In una gestione a segmenti non vi è spreco di spazio, ma la memoria fisica può diventare frammentata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

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

- I riferimenti alla memoria tendono a restare confinati in aree localizzate
- Nel tempo la localizzazione cambia ma il fenomeno persiste

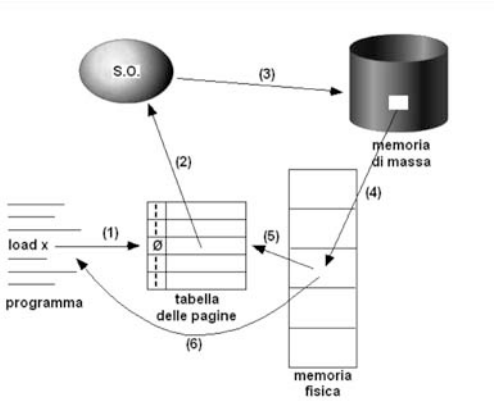


(Hartfield, 1972)

Paginazione su richiesta (1)

- La gestione della memoria a pagine o a segmenti 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 venga caricata solo la pagina contenente la sua prima istruzione

Paginazione su richiesta (2)



Trashing

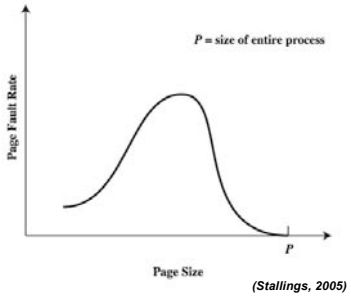
- Per contenere più processi solo una parte dello spazio di indirizzamento di ognuno è conservato in memoria centrale
- Quando la memoria si riempie il sistema deve portare alcune aree fuori memoria per fare spazio ad altre (*swap out* / *swap in*)
- La scelta delle aree da scaricare è critica
  - un'area 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 (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* TLB
- Quindi?

Dimensionamento delle pagine (2)

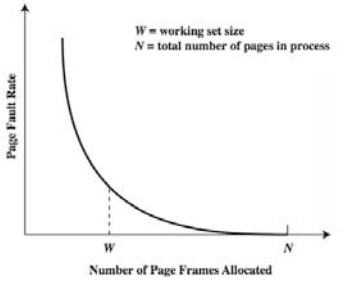
- Una dimensione piccola porta pochi page fault perché le pagine sono molte e ogni pagina contiene solo le informazioni che servono (principio di località)
- Dimensioni maggiori possono far aumentare i page fault perché le pagine sono di meno e ogni pagina contiene anche indirizzi non utilizzati
- Se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



(Stallings, 2005)

Dimensionamento delle pagine (3)

- 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



(Stallings, 2005)

Dimensionamento delle pagine (4)

- 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 kbyes 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)

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 (1)

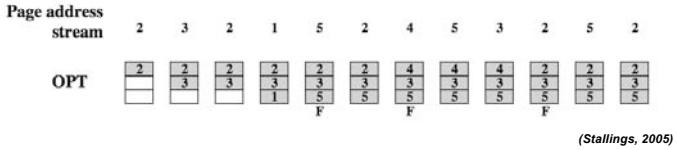
- Il problema è simile al problema dello scheduling dell'unità centrale nella gestione dei processi
  - *ottima*: la pagina vittima è quella che non servirà per più tempo (come individuarla?)
  - *LRU*: la pagina vittima è quella che da più tempo non viene utilizzata (*least recently used*)
  - *FIFO*: 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



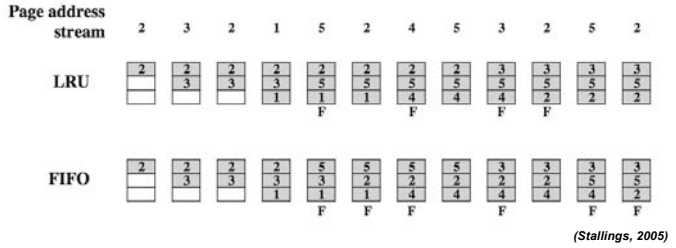
(Stallings, 2005)

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)

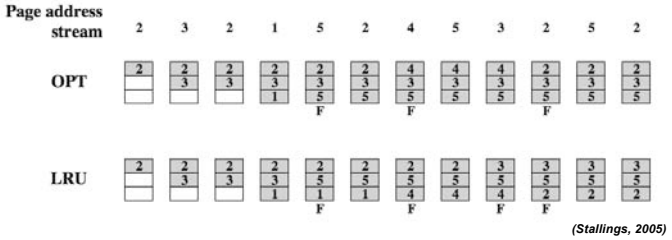
- La strategia FIFO non è efficiente
  - non considera l'uso delle pagine ma solo il momento del primo caricamento
  - nell'esempio, non riconosce il maggior utilizzo delle pagine 2 e 5 rispetto alle altre



(Stallings, 2005)

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



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 hw specializzato o un notevole overhead sw
- 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*

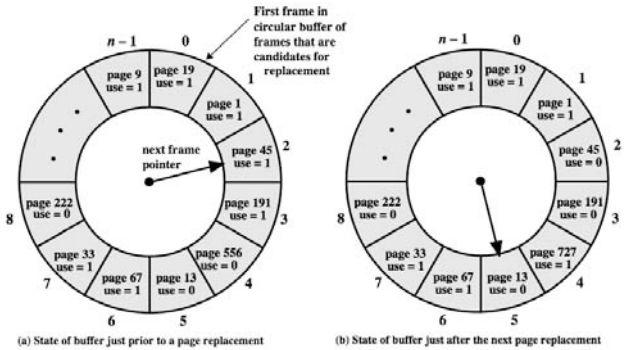
La strategia LRU, Varianti LFE e NUR

- LFE, Least Frequently Used
  - sostituisce all'istante di accesso un'informazione sulla frequenza di utilizzo
  - sono candidate alla sostituzione le pagine utilizzate meno frequentemente
  - ad ogni pagina è associato 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 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)

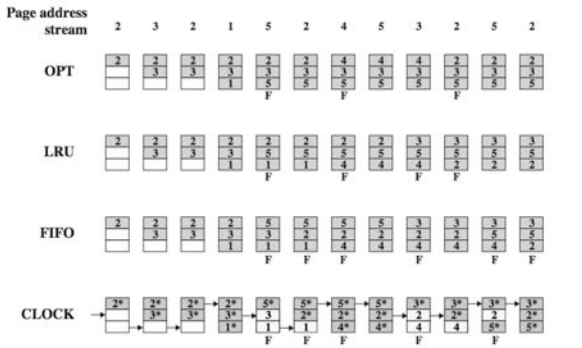


(a) State of buffer just prior to a page replacement

(b) State of buffer just after the next page replacement

(Stallings, 2005)

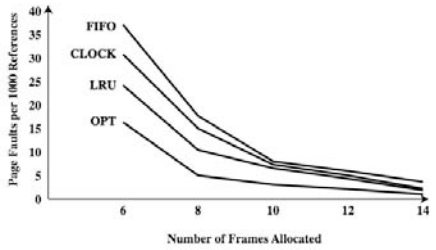
Un confronto tra le strategie (1)



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

(Stallings, 2005)

Un confronto tra le strategie (2)



(Stallings, 2005)

Gestione bufferizzata dei frame sostituiti

- 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

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

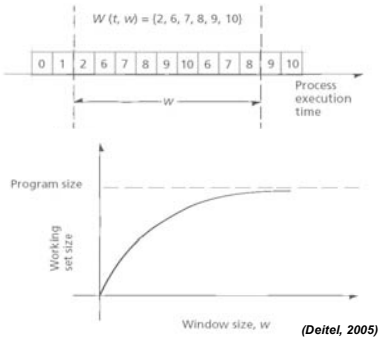
- I frame liberi sono organizzati in una lista globale
  - in caso di *page fault* al processo viene assegnato un *frame* libero indipendentemente dalla sua attuale dimensione fisica
- Il numero di *frame* allocato a un processo aumenta
  - le richieste di altri processi nel tempo possono ribilanciare 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

La strategia Working Set (1)

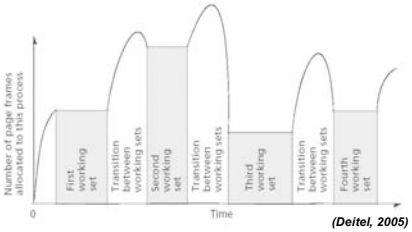
- E' una strategia di allocazione variabile in ambito locale che si basa sul principio di località
- Il *working set* di un processo al tempo *t*, *W*(*t*,*w*), è l'insieme di pagine utilizzate nelle ultime *w* unità di tempo
  - *W*(*t*,*w*) descrive la località del programma
  - è una funzione non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



(Deitel, 2005)

La strategia Working Set (2)

- Il WS 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 appartengo alle due località
  - quindi decresce nuovamente quando il processo si assesta sul nuovo ambito di località



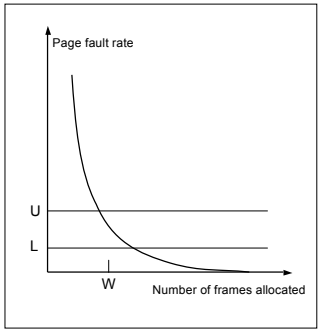
(Deitel, 2005)

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 *working set*
  - 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

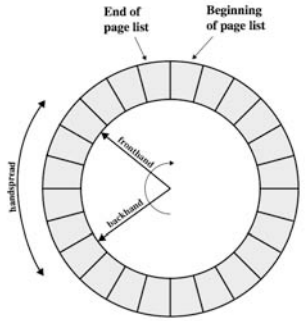


La gestione della memoria in Unix (1)

- 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 azzera 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 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 (2)

- 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