Gestione della memoria

Obiettivo

- **Allocazione della memoria ai processi**: riservo aree di RAM per le istruzioni, i dati e le variabili dei vari processi
- **Protezione**: ogni processo può accedere solo alle aree che gli sono state riservate in precedenza dal S.O.

Inoltre

- **Trasparenza**: L'operazione di allocazione deve essere invisibile all'utente
  - il programmatore scrive il programma senza tenere conto di dove dati, istruzioni e variabili saranno collocati in memoria
- **Condivisione:** il gestore della memoria deve anche garantire la condivisione delle risorse comuni (codice, librerie ecc.)

Sistemi **mono**programmati

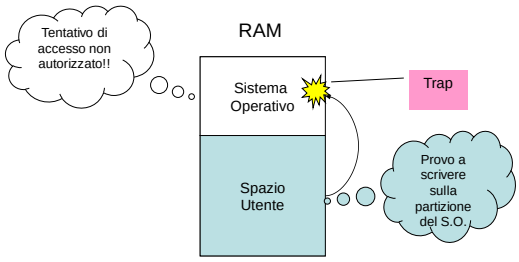
- Esempio: DOS
- **Un solo processo in RAM** (*insieme al sistema operativo*)
- Ai programmi utente viene riservata una parte della memoria(spazio utente) separata da quella del Sistema Operativo

Sistemi **mono**programmati

- **Due partizioni**: S.O. e Spazio Utente
- Se il processo utente tenta di accedere alla partizione del S.O. viene generata un'eccezione(*trap)* ovvero un particolare tipo di *interrupt* (segnale hardware che viene generato quando si verificano eventi che devono essere gestiti dal sistema operativo)

Sistemi **mono**programmati

Sistemi **mono**programmati



Sistemi **mono**programmati

- Vantaggi: *semplicità*
  - E' necessario *un solo registro di confine* per delimitare il confine tra le due partizioni e garantire la protezione del SO dal processo utente
  - Il processo è caricato *integralmente in un blocco contiguo* di memoria

Sistemi **mono**programmati

- Svantaggi: *inefficiente e rigido*
  - Se il processo è fermo in attesa di un'operazione di I/O, la CPU (e la memoria) è *inutilizzata*
  - Una parte della memoria utente sarà *inutilizzata* (perché lo spazio richiesto dal processo è minore di quello disponibile)
  - Se il processo non "entra" nella partizione utente, *non può essere eseguito*

Sistemi **multi**programmati

- **Più processi in RAM**
- *Vantaggi : più efficienti*
  - evito di lasciare inutilizzata la CPU
  - riduco lo spazio sprecato in memoria
- *Svantaggi : più complessi da gestire*
  - Il S.O. deve prendere molte più decisioni: quanto spazio allocare ai vari processi, dove collocarli, cosa fare se lo spazio è insufficiente per ospitare un nuovo processo,etc…

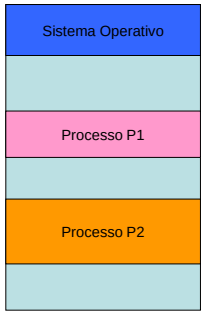
Sistemi **multi**programmati

- Varie strategie:
  - **Allocazione contigua**:
    - Partizioni fisse
    - Partizioni dinamiche(variabili)
  - **Allocazione NON contigua:**
    - Paginazione
    - Segmentazione
    - Memoria virtuale
- Nei moderni sistemi operativi vengono utilizzate insieme:
  - ***Paginazione + Segmentazione + Memoria Virtuale***

Allocazione

- **Contigua**: tutto il processo viene caricato in un blocco di memoria unico
- **Non Contigua**: il processo viene suddiviso in parti e i vari pezzi sono collocati in regioni diverse della memoria

Allocazione contigua



Protezione: garantita tramite una coppia di registri per ogni partizione

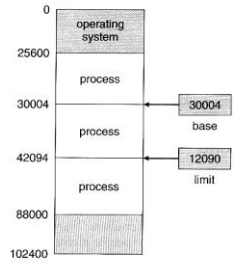


Figure 8.1 A base and a limit register define a logical address space.

Protezione: registro base e limite

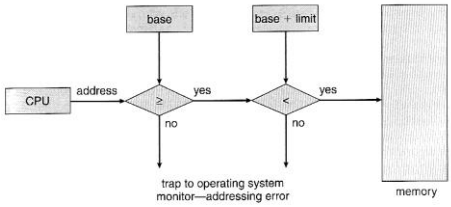
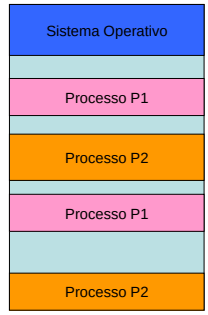


Figure 8.2 Hardware address protection with base and limit registers.

Allocazione ***non*** contigua



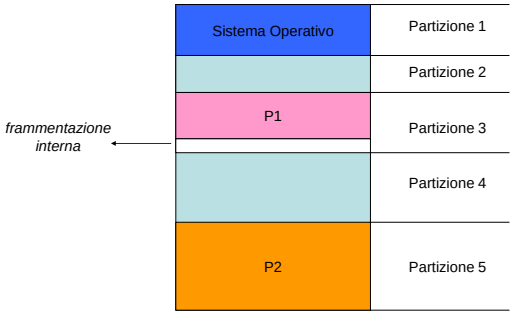
Partizione

- Una partizione è ***un'area di memoria contigua*** che può essere **riservata ad un processo** (utente o di sistema)
- La partizione può essere utilizzata da un **solo processo**, anche qualora vi sia spazio inutilizzato(= frammentazione interna)

Partizioni fisse

- **La dimensione delle partizioni e la loro collocazione è fissa** e non dipende dai processi che vi sono caricati
- La dimensione delle partizioni è **indipendente dalla memoria effettivamente occupata dai vari processi**
- La RAM insomma è suddivisa in una serie di aree(le partizioni appunto) che **possono essere libere o occupate**

Partizioni Fisse



Partizioni fisse

- *Vantaggi*: *semplicità*
- *Svantaggi*: *frammentazione interna*
  - **il processo occupa solo una parte della partizione** (ovvero vi è spazio inutilizzato all'interno della partizione) = ***frammentazione interna***
  - **Il numero di processi è <= numero partizioni** anche se lo spazio inutilizzato è sufficiente per memorizzare altri processi
  - **La dimensione del processo è limitata a quella della partizione più grande**

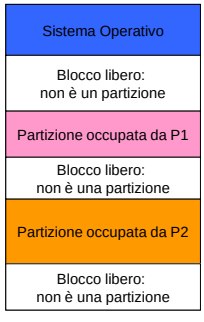
Partizioni dinamiche(variabili)

- **Partizione creata e dimensionata per un processo**
  - *dimensione partizione = dimensione del processo* che deve contenere
  - *numero di partizioni = numero di processi* caricati in memoria(e pertanto varia nel corso del tempo man mano che nuovi processi vengono caricati e successivamente rimossi dalla RAM)

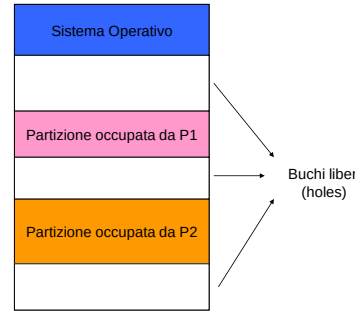
Partizioni dinamiche(variabili)

- Considero **la memoria utente come un unico spazio**
- Quando arriva un processo scelgo un area di memoria libera sufficientemente grande da contenerlo e quella diventa una nuova partizione
- Quando il processo termina l'area di memoria ridiventa libera e forma con le aree libere adiacenti un unico blocco(nel senso che la partizione cessa di esistere insieme al processo)

Partizioni dinamiche



Partizioni dinamiche

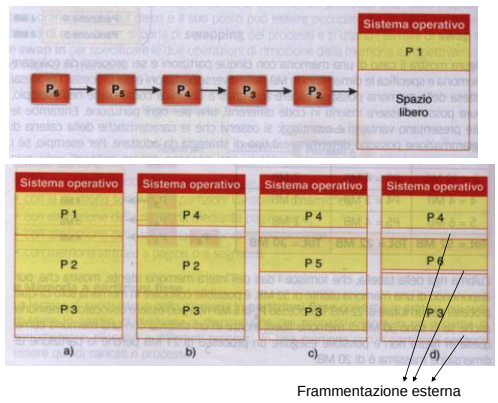


Frammentazione esterna

- Se i buchi(holes) sono troppo piccoli per contenere un nuovo processo si dice che ho frammentazione esterna

Partizioni dinamiche(variabili)

- *Vantaggi*: *utilizzo meglio la memoria*
  - Frammentazione interna = 0
  - Posso eseguire processi di dimensione maggiore se ho un blocco vuoto abbastanza grande da contenerlo(non vi sono vincoli rigidi relativi alla dimensione delle partizioni)
- *Svantaggi: frammentazione esterna*



Frammentazione esterna

- Con il passare del tempo creando ed eliminando partizioni si finisce per avere una serie di zone vuote tra le partizioni troppo piccole per poter contenere da sole nuovi processi
- Poiché queste aree vuote sono "esterne" alle varie partizioni si parla di frammentazione **esterna**
- Infatti si tratta di memoria inutilizzabile

Compattazione di buchi adiacenti

- Aggregazione degli spazi liberi adiacenti in un unico blocco libero
- Eseguita *periodicamente*
- Prende il nome di *coalescenza*

Compattazione partizioni

- Posso "compattare" le varie partizioni **ottenendo un unico spazio vuoto alla fine** che posso utilizzare per nuovi processi
- Questo riduce di molto la frammentazione esterna
- Purtroppo richiede tempo quindi è eseguita raramente per non rallentare troppo il sistema

Algoritmi di allocazione

- Nel caso di partizioni variabili…
- Come scelgo l'area di memoria libera in cui allocare il processo?
- Varie strategie:
  - First Fit
  - Best Fit
  - Next Fit
  - Worst Fit

First Fit = Prima Adatta

- First Fit = Prima Adatta
- **Scelgo la prima zona libera abbastanza grande da contenere il processo**
- Questo algoritmo tende a collocare tutti i processi all'inizio della memoria
- Lascia una grossa area libera alla fine della memoria

Best Fit = Migliore Adatta

- Best Fit = Migliore Adatta
- **Scelgo la zona libera più piccola in grado di contenere il processo**
- Cerco di ridurre lo spazio sprecato, occupando tutta la memoria disponibile
- Non frammento aree molto grosse se non quando è strettamente necessario
- *Però…* frammentazione esterna spazio rimanente troppo piccolo per essere utilizzato

Next Fit = Prossima Adatta

- Next Fit = Prossima Adatta
- *Variante di First Fit*
- **Scelgo la prima area in grado di contenere il processo non dall'inizio della memoria, ma a partire dall'ultima area allocata**
- Mira a distribuire i processi su tutta la memoria
- *Però*…frammenta il blocco alla fine, quindi lascia poco spazio per i grandi processi

Worst Fit= Peggiore Adatta

- Worst Fit = Peggiore Adatta
- **Sceglie la zona di dimensione maggiore in grado di contenere il processo**
- Si basa sul presupposto che rimanga sufficiente spazio libero per allocare altri processi e non vada sprecato
- *Però…* tende a non lasciare zone libere grandi, quindi potrebbe provocare problemi in caso di programmi di grosse dimensioni, in quanto vado sempre a occupare la zona libera più grande, anche se si liberano altre zone più piccole;

Partizioni *Variabili*

- **In generale**
  - First Fit migliore
  - Next Fit
  - Best Fit
  - Worst Fit peggiore
- **Tuttavia dipende dal tipo di processi eseguiti**

Riassumendo…

in ordine di efficienza:

- *first fit* = migliore sia per velocità che per frammentazione
- *next fit* = frammenta il blocco alla fine, quindi lascia poco spazio per i grandi processi
- *best fit* = lascia poco spazio difficilmente utilizzabile
- *worst fit* = non lascia grandi zone libere, quindi per inserire processi grandi bisogna ricompattare i blocchi

Partizioni *Fisse*

- Anche nel caso di partizioni fisse posso utilizzare una delle strategie viste
- In questo caso però la migliore risulta essere la Best Fit
- Spreca meno spazio e permette di far eseguire + processi

In finale

- Partizioni fisse : Best Fit
- Partizioni variabili : First Fit

Paginazione : allocazione non contigua

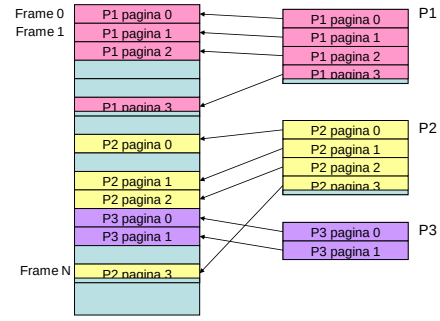
- Suddivido il processo in vari **blocchi di dimensione fissa** detti **pagine**: ogni pagina è identificata da un numero
- Ogni pagina può essere collocata in un punto diverso della memoria
- La **memoria fisica** infatti è **suddivisa** in blocchi di dimensione uguale a quella delle pagine detti **frames**: anch'essi numerati
- **Associo pagine e frames** attraverso una **tabella delle pagine**
- Il sistema operativo mantiene una **"lista" dei frames liberi**

Tabella delle pagine

- Dice in quale frame è stata caricata la pagina cercata
- E' del tipo:

Pagina	Frame
0	0
1	2
2	6

Paginazione



Paginazione

- **Ogni** processo ha la **propria** tabella delle pagine (page table)
- La tabella delle pagine dice in quali frames sono state collocate le pagine caricate in RAM
- Tutte le pagine del processo attivo vengono caricate in memoria, ma non necessariamente in frames adiacenti per cui l'allocazione non è contigua

Osservazione

- Processi caricati in RAM → tabelle delle pagine caricate in RAM
- Quella del processo attivo viene caricata in parte in una cache "speciale"(TLB)
- Alcune pagine del processo attivo saranno caricate nelle cache ordinarie(I,II,III livello)

Paginazione
Caricamento di un nuovo processo

- Sul disco(rigido) ho un processo che occupa 4 pagine
- Scorro la lista dei frame liberi
- Ne scelgo 4 e vi carico le 4 pagine eliminando i rispettivi frame dalla suddetta lista

Paginazione

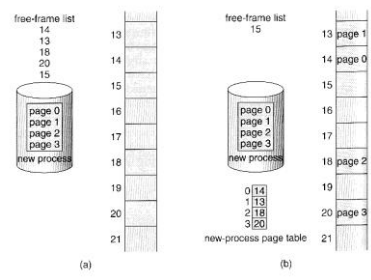
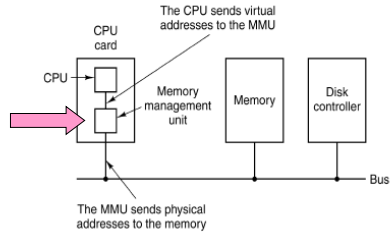


Figure 8.10 Free frames (a) before allocation and (b) after allocation.

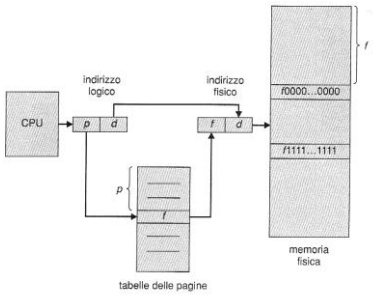
Traduzione degli indirizzi

- La traduzione degli indirizzi avviene a livello hardware
- L'unità preposta alla traduzione è chiamata **MMU** (*Memory Management Unit*) ovvero unità di gestione della memoria

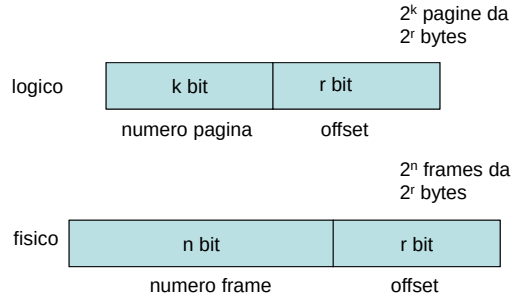
Traduzione degli indirizzi: MMU(Memory Management Unit)



Traduzione indirizzi



Indirizzo logico-fisico



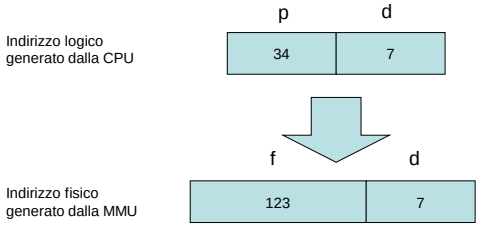
Indirizzo logico - fisico

- P = numero della pagina in cui si trova la locazione di memoria da caricare (per ogni processo parto sempre da 0)
- D = scostamento dall'inizio della pagina (displacement o offset) è la posizione della locazione di memoria rispetto all'inizio della pagina
- F = numero del frame in cui viene caricata la pagina

Indirizzo logico - fisico

- Esempio:
  - il processo è suddiviso in 70 pagine
  - La **locazione di memoria** indirizzata si trova
    - **pagina 34**
    - **ottava parola** di memoria (a partire dall'inizio pagina)
  - L'indirizzo logico sarà: P = 34 D = 7
  - Se la pagina 34 è caricata nel frame 123 l'indirizzo fisico sarà : F = 123 D = 7

Indirizzo logico - fisico



Tutto ovviamente dovrebbe essere espresso in binario

Traduzione indirizzi

- **indirizzo logico** = indirizzo generato dalla CPU
  - formato da p (**numero di pagina**) e d (**scostamento**)
- **indirizzo fisico** = indirizzo corrispondente nella memoria fisica
  - d resta lo stesso
  - p viene sostituito da f (numero di frame in cui è caricata la pagina)

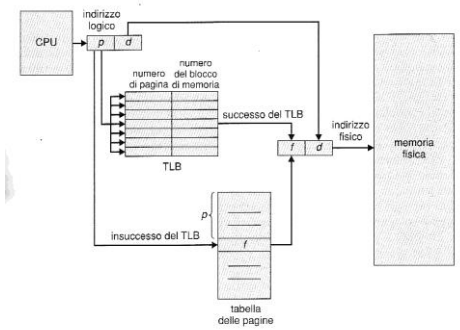
TLB = Translation Lookaside Buffer

- **Cache della page table** per velocizzare l'accesso alle **pagine di uso più frequente del processo attivo (TLB).**
- Le **tabelle delle pagine molto grandi**, infatti verranno per lo più mantenute in memoria fisica, caricando nel TLB(che in quanto cache è molto piccola) solo le corrispondenze di uso più frequente
- Nel TLB è caricata una parte della page table del processo attivo ciò consente di accelerare le ricerche

TLB

- Infatti caricare una parola di memoria in assenza del TLB richiede di effettuare almeno due accessi in RAM
  - Il primo per accedere alla tabella delle pagine
  - Il secondo per accedere alla pagina
- Il TLB permette di evitare il primo accesso

TLB



TLB

- La ricerca della pagina avviene in contemporanea sia nella Cache(TLB) che nella RAM(dove però richiede molto più tempo)
- Se la trovo in Cache posso procedere subito altrimenti dovrò attendere che ve

- Ad ogni pagina si associa (almeno) un bit di protezione:
  - pagina *read-only*
  - pagina *read/write*
- Sovente i bit di protezione sono tre, ciascuno che abilita o non abilita una funzione tra *read-write-execute*.
- Non tutte le combinazioni di valori sono "sensate, ma le seguenti si:

3 bits di protezione per regolamentare l'accesso alla pagina

R	*W*	*E*	*Significato*	*Esempio*
N	N	N	nessun accesso al processo	pagine di sistema
N	N	Y	sola esecuzione	codice condiviso tra processi
Y	N	N	sola lettura	data base protetto
Y	N	Y	lettura o esecuzione	consente copia non modifica
Y	Y	N	lettura o scrittura	area dati - no codice
Y	Y	Y	ogni accesso è consentito	

Dimensione Pagina

- se troppo **piccola** avrò un ***page table* enorme**
- se troppo **grande** avrò **frammentazione interna** (perché l'ultima parte dell'ultima pagina di un processo sarà in parte inutilizzata), inoltre **dovrei caricare in memoria anche parti del programma che magari non vengono utilizzate**

Paginazione

- Gli **attributi(Read,Write,Execute,…)** associati ad ogni pagina possono essere utilizzati per garantire:
  - Protezione
  - Condivisione
- Ad esempio: se una pagina è in sola lettura posso condividerla con processi (cooperanti) senza doverla duplicare in memoria

Paginazione
Esempio di condivisione

- La seguente figura mostra 3 processi P1,P2,P3 che condividono una parte il codice (ed1, ed2, ed3) supposto accessibile in sola lettura.
- La condivisione del codice avviene semplicemente duplicando una
- parte della tabella delle pagine di P1(i primi 3 elementi) nella tabella delle pagine di P2 o P3
- In tal modo ogni riferimento alle pagine di P2 e P3 si traduce in un riferimento alle medesime pagine di P1 evitando di avere 3 copie delle stesse 3 pagine in memoria

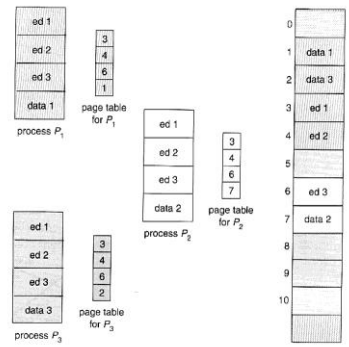


Figure 8.13 Sharing of code in a paging environment.