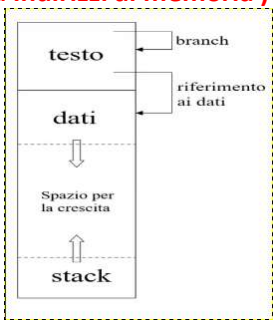
Memoria 1 Partizioni + MMU + swapping

martedì 11 novembre 2025 15:22

Vediamo ora come gestire la memoria (ovvero come gestire l'address space) : quindi si parte dal binding , andando a vedere gli schemi per farlo (partizionamento dinamico e statico) , poi vedendo la tecnica negli attuali so (paginazione/segmentazione) , poi parlando di memoria virtuale e condivisa ; infine vedremo come gestirla in entrambi i sistemi operativi. Andiamo ora a vedere il **binding (Associazione di istruzione e dati ad indirizzi di memoria)** ma prima in generale :



Nel caso più semplice : dato un thread che esegue istruzioni nello stack , si ha la possibilità che vi si arriva ad istruzioni nell RAM (magari esempio con il branch) ; analogamente per l'accesso ai dati .
Questo binding può essere di tre tipi
(condizionamento dell'assegnamento della memoria di lavoro ai processi) :

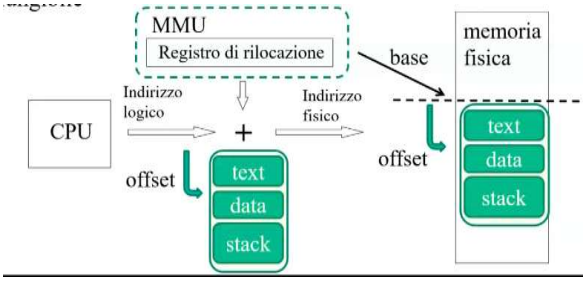
1. **A tempo di compilazione**
   a. La posizione dell'address space del processo è fissa e nota a tempo di compilazione(specifico punto della RAM)
   b. Il codice è assoluto ed ogni riferimento viene risolto tramite l'indirizzo in memoria
   c. Per modificare i riferimenti occorre ricompilare
2. **A tempo di caricamento**
   a. La posizione della memori a dell'applicazione è fissa (definita quando la lanciamo)
   b. Quindi il codice è rilocabile (il riferimento viene risolto tramite offest dall'inizio dell'address space)
   c. La base per l'offset è determinata a tempo di lancio dell'applicazione
3. **A tempo di esecuzione**
   a. La posizione in memoria del processo può variare durante esecuzione
   b. Il codice è rilocabile dinamicamente cioè il riferimento viene risolto per determinare l'indirizzo solo se quel riferimento viene richiesto
4. **Indirizzo logico**
   a. Riferimento nel testo nel nostro programma per localizzare istruzione e/o dati (sempre nell'address space)
5. **Indirizzo fisico**
   a. Posizione reale in memoria di istruzioni e/o dati
   b. Indirizzo fisico nella RAM

Nei primi due tipi di binding l'indirizzo fisico e logico coincidono . Mentre per l'ultimo possono non coincidere , ed il mapping a run-time degli indirizzi fisici su indirizzi logici viene fatto dalla **mmu (memory management unit)** . Facendo un riepilogo :

Binding a tempo di compilazione/caricamento
- ✓adatto a contesti di esecuzione seriale
- ✓adatto a sistemi batch monoprogrammati
- ✓in entrambi i casi è nota la partizione di memoria fisica riservata per il codice dell'applicazione
- ✓la compilazione e/o il caricamento genera un eseguibile "consapevole" di risiedere in quella data regione di memoria fisica

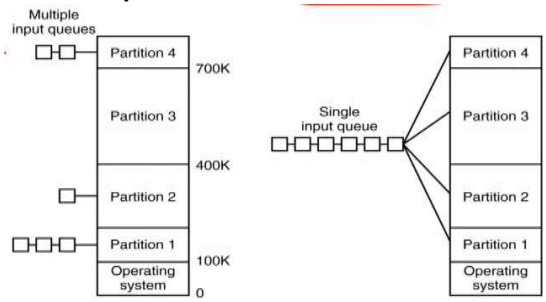
Binding a tempo di esecuzione
- ✓adatto a sistemi batch multiprogrammati
- ✓adatto a sistemi time-sharing
- ✓ogni processo potrà essere caricato e/o spostato dinamicamente in zone di memoria fisica differenti in modo trasparente alla struttura del codice

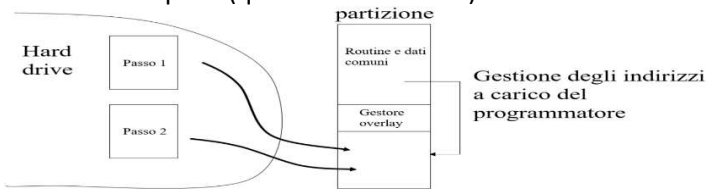
Andiamo ora a vedere la struttura basica **MMU** :



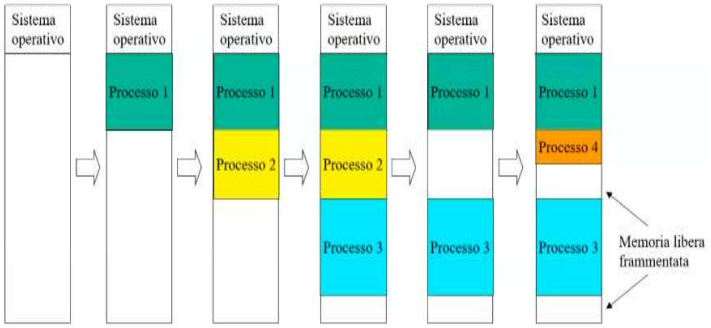
Il registro di rilocazione rappresenta indirizzo al quale incomincia applicazione (indirizzo fisico in RAM). Una domanda sorge: come posso determinare la taglia dell'address space ? Dipende dall'architettura della cpu dove gira : in generale si può avere spazio di indirizzamento di (2^x)-1; ma attenzione se spiazzamento cade all'esterno dell'address space. Quindi per risolvere questo problema, all'interno della MMU è stato aggiunto altro registro (**registro limite**) , il quale indica il limite massimo all'interno della memoria fisica dove un'applicazione è collocata.

Questo controllo è sempre fatto a run-time, il quale controllo viene effettuato attraverso hardware . Vediamo ora lo scenario nel quale dobbiamo gestire più applicazioni attive :

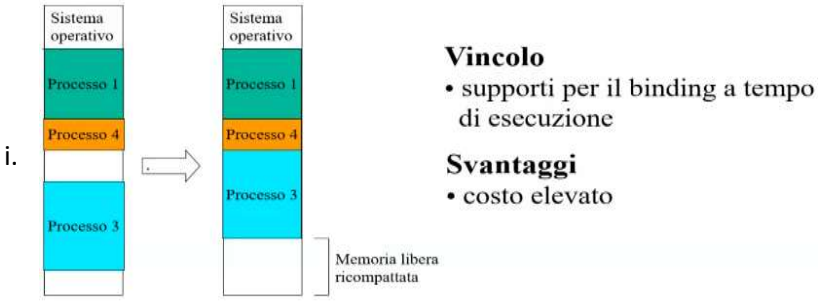
1. **Partizionamento statico a partizioni multiple**
    a. Sistemi multi programmati
    b. Quindi lo spazio della ram è suddiviso in partizione fisse di taglia uguale o diversa , la quale ognuna ospita un singolo processo
    c. Ogni processo occupa una partizione intera della ram
    d. Le partizioni statiche possono portare a **frammentazione interna**
    e. Il numero massimo di partizioni diminuisce se non c'è swap in/out
    f. Come si risolve questo problema della taglia delle partizioni? Si fa un mix tra partizioni di taglia piccola e taglia grande
    g. **Allocazione dei processi**
        i. 

        ii. Quindi vi sono code (sia nel caso di una coda per ogni partizione , sia nel caso di unica coda per tutte le partizioni , la quale riduce la frammentazione esterna)
    h. **Overlay**
        i. Tecnica utilizzata per risolvere il seguente problema: dato un address space di taglia T_1 maggiore della taglia della ram T ; quindi carico in ram solo una parte dell'address space (quello che mi serve)
        ii. 

        iii. Questa situazione la gestisce il programmatore
        iv. Questo concetto ha un collegamento con le DLL (librerie caricate dinamicamente)
2. **Partizionamento dinamico**

a. 

b. Il numero delle partizione e la loro lunghezza è variabile
c. La memoria allocata è pari a quella dell'address space
d. L'MMU evita i seguenti bug : accesso ad area di memoria fuori da quella allocata per il processo stesso : il sistema invoca un a trap
e. Anche qui vi è <u>frammentazione esterna</u> : la memoria è libera ma non contigua . Quindi per evitare questo problema si usa la **ricompattazione** : sposto le zone libere per averne una zona libera di dimensione ampia

i. 

f. Sorge una domanda : date due o più aree di memoria libere quale si sceglie? Vediamolo attraverso degli algoritmi

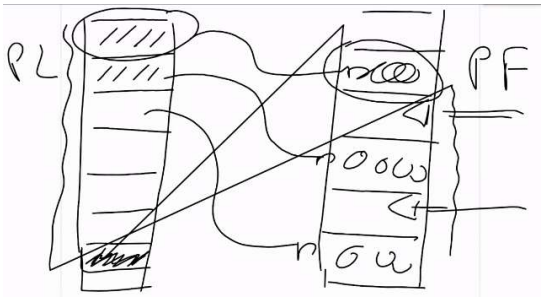
g. **First fit**
- Il processo viene allocato nel <u>primo "buco" disponibile</u> sufficientemente grande
- La ricerca può iniziare sia dall'inizio dell'insieme dei buchi liberi che dal punto in cui era terminata la ricerca precedente

Best fit
- Il processo viene allocato nel buco <u>più piccolo che può accoglierlo</u>
- La <u>ricerca</u> deve essere effettuata su tutto l'insieme dei buchi disponibili, a meno che questo non sia ordinato in base alla taglia dei buchi
- Questa strategia tende a produrre frammenti di memoria di dimensioni minori, lasciando non frammentate porzioni di memoria più grandi

Worst fit
- Il processo viene allocato nel buco <u>più grande in grado di accoglierlo</u>
- Anche in questo caso la <u>ricerca</u> deve essere effettuata su tutto l'insieme dei buchi disponibili, a meno che questo non sia ordinato in base alla taglia
- Questa strategia tende a produrre frammenti relativamente grandi, utili quindi ad accogliere una quantità di nuovi processi di taglia ragionevole

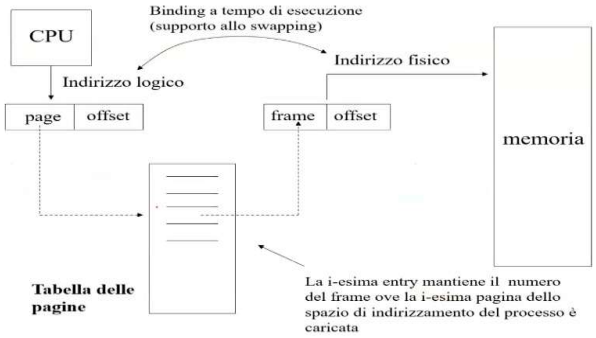
h. <u>Il primo è il basico; il secondo ha che nonostante l'applicazione viene inserita nella memoria che calza, vi è sempre una porzione della stessa partizione utilizzata; nel terzo si ha il duale : la partizione rimanente è grande</u>

3. **Swapping**
   a. Tecnica che permette di caricare/scaricare in ram address space <u>solo nel momento che vi è completa inattività</u>
4. **Paginazione**
   a. Ulteriore schema di gestione della memoria (collocazione address space logico all'interno dell'area di lavoro)
   b. Permette di risolvere il problema delle partizioni (statiche / dinamiche) : allocazione nella ram partendo dalla base ed arrivando ad un limite . Quindi **permette di carica in una data zone della ram una data zona dell'address space** (levo la contiguità) . Quindi non ci basta la basica MMU .
   c. **Ogni zona viene chiamata pagina logica** mentre nella ram si hanno **pagine fisiche (frames)**
   d. Non c'è ne frammentazione esterna ne' interna in quanto la taglia della pagina fisica è uguale a quella logica

e. 

f. Attenzione se pagina logica è minore di pagina logica : frammentazione esterna possibile

g. Quindi la MMU diventa la seguente :

i. 

ii. La tabella delle pagine (tipicamente in ram e puntata da un registro nel processore) viene mandata in setup quando viene fatto il setup dell'address space : viene consultata ogni volta dal processore attraverso il sotto sistema di controllo

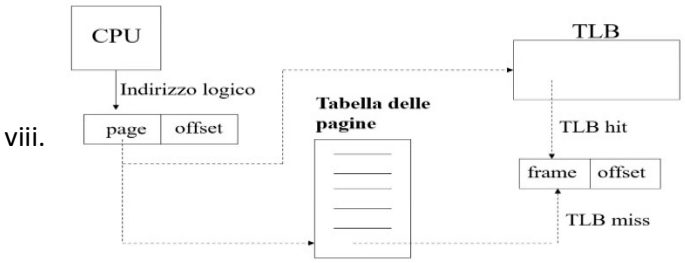
iii. Vediamo come ottimizzare : <u>usiamo una cache all'interno del processore</u> **TLB (translation lookaside buffer)** :

iv. Mantiene associazioni tra numero di pagina e frame

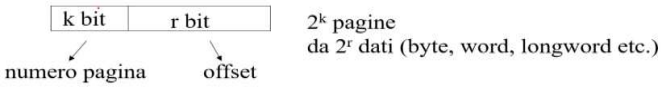
v. Permette di bypassare la tabella delle pagine

vi. Se hit ok altrimenti se miss vado a consultare la tabella dei processi e si carica nella TLB

vii. <u>Se più processi più tabelle dei processi</u>

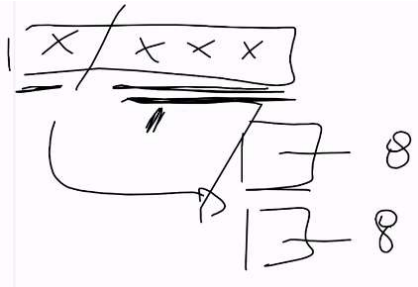
viii. 

ix. Vediamo ora come è organizzato un indirizzo

1. 

2. Si ha quindi <u>protezione della memoria</u> (non si esce da offset della pagina fisica)

3. Vediamo un esempio :

4. 

5. Dove si hanno due pagine per il primo bit (0/1) ed 8 possibili entry per via dei 3 bit dell'offset (8=2^3)

6. Nelle macchine attuli la dimensione di una pagina e **4kb (2^12)**

5. Riassumendo (paginazione e dipendenza da hw)

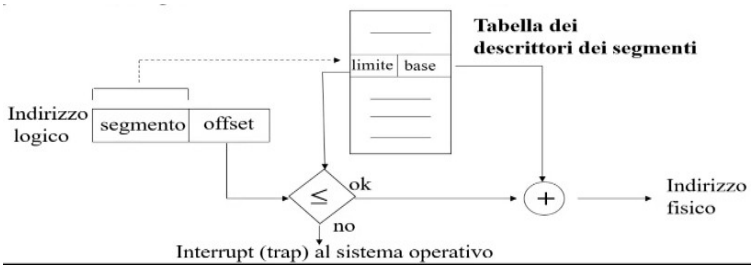
a.
- Setup/gestione della tabella delle pagine via software (di livello kernel) non da luogo ad indipendenza dalla specifica architettura hardware
- La risoluzione di un miss sul TLB viene infatti effettuata dal microcodice della specifica CPU
- Questo effettua uno o piu' accessi alla memoria di lavoro per consultare la tabella delle pagine del processo corrente
- La struttura della tabella delle pagine (e delle relative informazioni) dipende quindi dalla logica del microcodice

NOTA BENE:

➢ dato che mantenere la tabella delle pagine via software significa che essa e' accessibile tramite indirizzi logici, sara' necessario avere il supporto hardware per mantenere ad ogni istante la corrispondente traduzione fisica

➢ questo supporto e' tipicamente esterno al TLB (ad esempio su x86 e' un registro dedicato denominato CR3)

6. **Segmentazione**

a.


b. Lo spazio di indirizzamento viene visto come un insieme si segmenti (possibile diversa taglia) scorrelati tra loro.

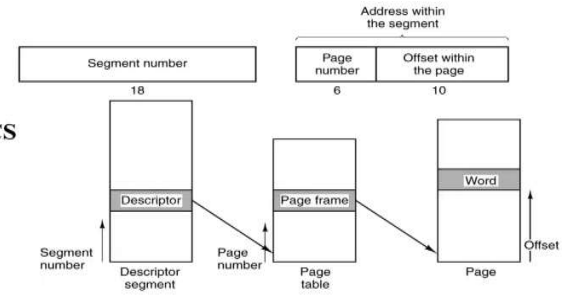
c. Nota : la **tabella dei descrittori è simile a quella delle pagine** , ma le entry sono differenti : limite + base (nella RAM) , con eventuali informazioni di protezione

d. Attenzione alla generazione dell'indirizzo fisico (se ok si calcola altrimenti lancio una trap)

e. Nota : entrambe se prese da sole non bastano!!

7. **Segmentazione e paginazione**

a. **Es. MULTICS (1964 – MIT/Bell-Labs)**



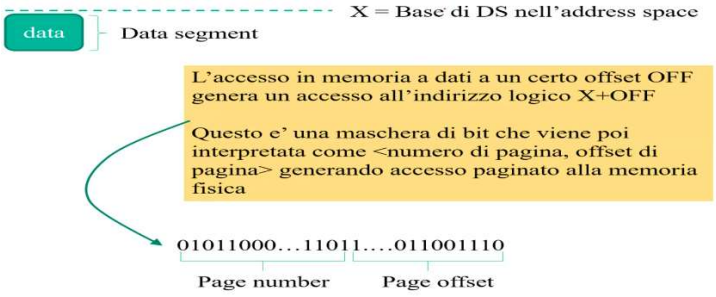
b. Address space è insieme di segmenti i quali sono a loro volta suddivisi in pagine : si risolve frammentazione esterna

c. Vediamolo come funziona sui processori attuali :

i. Nella tabella dei segmenti vi è offset della memoria logica : quindi per avere la posizione dell'istruzione si somma la l'indirizzo base + offset -> **indirizzo lineare**

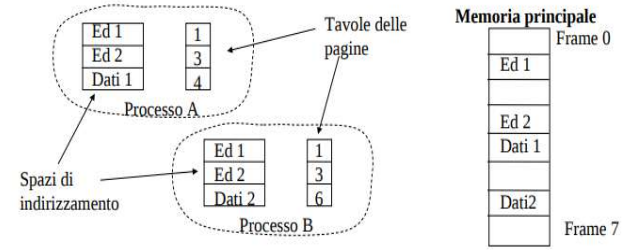
ii. **Se non specificato si intende DS(Data segment) altrimenti si usa il CS(Code segment)**

1. In dettaglio

2.


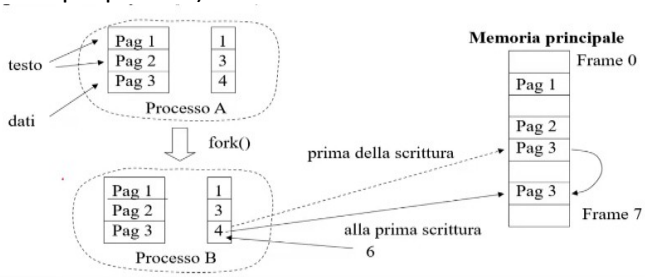
d. Quindi combinando queste due tecniche si ha ulteriore vantaggio : **condivisione della memoria principale** , evitando il problema di avere due processi che puntano alla stessa entry in ram . In dettaglio

i.



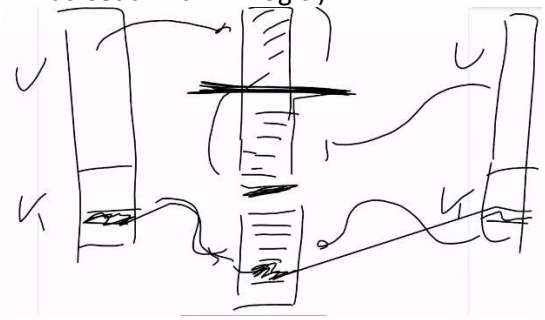
e. Attenzione al bit **copy on write** il quale bit è di protezione , quindi vediamo un esempio (si fanno copie private)

i.



f. La paginazione quindi ci dà ulteriore vantaggio ovvero : <u>la condivisione della zona kernel</u> (determinati set di indirizzi logici)

i.

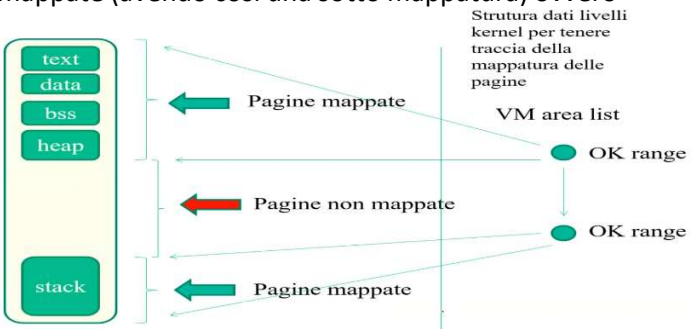


ii. In dettaglio LINUX a 32 bit

iii.

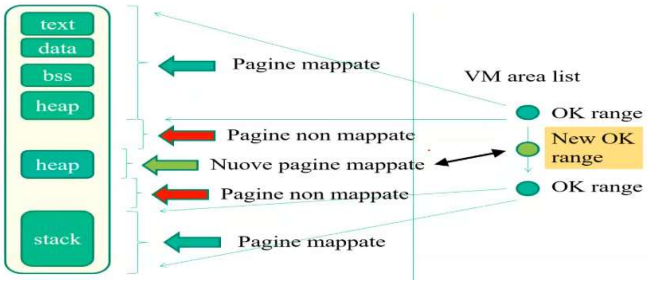
8. **Struttura interna address space**
   a. Chiaramente se indirizzamento lineare a X bit si hanno massimo 2^x locazioni , le quali non vengono usate tutte in un determinato momento
   b. Attenzione però **alla mappatura dell'address space** ovvero ci possono essere zone non mappate (avendo cosi una sotto mappatura) ovvero

c.



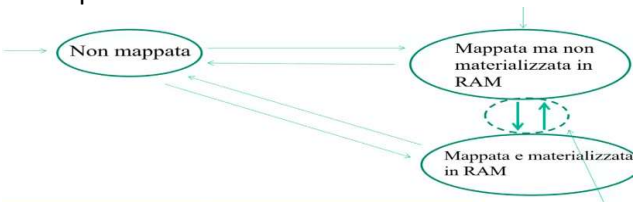
   d. Dove la VM area list rappresenta l'elenco delle zone correntemente mappate
   e. Quindi si potrebbe avere una situazione del genere (dopo aver mappato delle pagine)

i.



f. In generale quindi su un so moderno si ha che

i.



Materializzata in RAM: la corrispettiva entry della tabella delle pagine mantiene valide informazioni di posizionamento in RAM della pagina

Transizioni legata alla **Memoria Virtuale**

ii. Possibile che una pagina mappata sia solo presenti nell'address space e per renderla mappata la dobbiamo materializzare in ram ; oppure pagina presente solo in address space ma non in RAM (swap out)