29 Maggio 2023

ARCHITETTURE DI CALCOLO LEZIONE 21

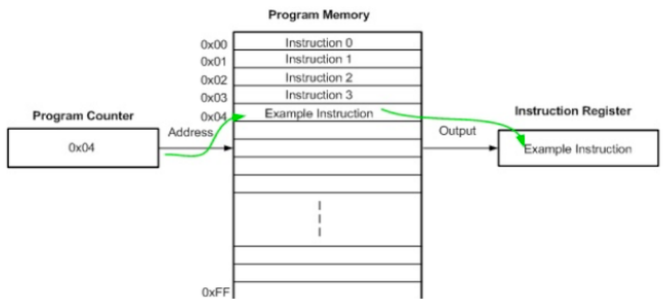
Gestione della memoria centrale

Memoria centrale

La memoria centrale è fondamentale nel computer in quanto contiene dati ed istruzioni prelevati dalla memoria di massa in attesa che siano elaborati dalla CPU.
Così come esistono tecniche di scheduling della CPU, vi sono anche modalità di scheduling della memoria al fine di contenere contemporaneamente più processi possibile, per aumentare il livello di multiprogrammazione. La scelta di un metodo di gestione della memoria dipende dall'architettura del sistema di calcolo: ogni algoritmo dipende dallo specifico supporto hardware.
Una parola (word) di memoria è il minimo gruppo di bit che è possibile leggere o scrivere in memoria. La lunghezza di una parola (ad es. 16, 32 o 64 bit) varia a seconda del tipo di elaboratore.
La memoria centrale può essere rappresentato come un grande vettore di byte, ciascuno con il proprio indirizzo (n.b., organizzata in word).
La memoria centrale è uno spazio per conservare dati facilmente accessibili dalla CPU e dai dispositivi di I/O.
Il sistema operativo che si occupa della gestione della memoria principale è responsabile di:

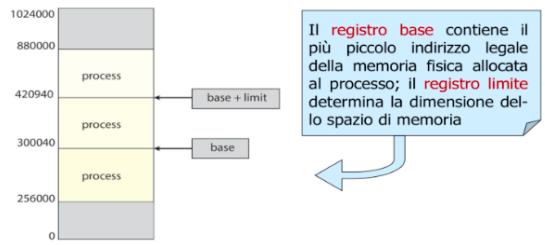
- Tenere traccia di quali parti della memoria sono correntemente usate e da chi.
- Decidere quali processi caricare quando la memoria si rende disponibile.
- Allocare e deallocare lo spazio di memoria.

Quando inizia un programma, il program counter indicizza la prossima istruzione da eseguire. Le istruzioni possono essere, per esempio, operazioni di caricamento di un dato oppure l'inserimento di un dato in una cella di memoria. Quello che il sistema operativo deve fare è supportare i programmi nell'esecuzione di queste istruzioni, accedendo a quello che c'è nella memoria.



Dal punto di vista della memoria, tutte le celle sono uguali; chi programma sa che alcune celle contengono i dati ed altre le istruzioni, ma la RAM questa informazione non la comprende. Quello che la RAM sa è soltanto se chiediamo il contenuto della cella in lettura o in scrittura. Ogni processo deve avere uno spazio di memoria separato.

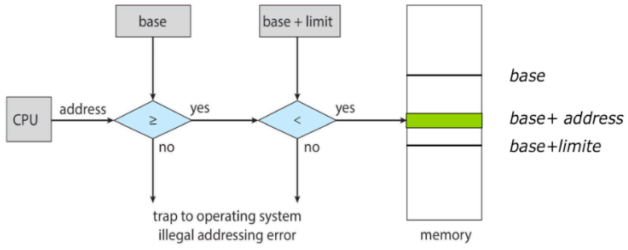
Una soluzione è l'uso di un registro base ed un registro limite.



L'indirizzo base e l'indirizzo limite servono per mettere in atto il meccanismo di protezione, cioè il meccanismo che evita che un processo generi un indirizzo non valido, accedendo alla memoria di un altro processo.
Quindi, sostanzialmente ogni volta che un programma genera un indirizzo tramite la CPU, questo indirizzo viene controllato rispetto due parametri, per cui l'indirizzo deve essere:

- indirizzo ≥ base
- indirizzo < base+limite

Solo il SO può caricare i registri base e limite tramite istruzioni privilegiate; in quanto tali, nessun processo tranne il kernel può modificare i registri base e limite.
In modalità utente (modalità esecuzione normale del programma) ogni volta che si genera un indirizzo minore della base o maggiore/uguale della somma tra base e limite, viene generata un'eccezione.
Nello schema vediamo che:



•La CPU genera un indirizzo,
•l'indirizzo viene confrontato prima con l'indirizzo base,
•se è maggiore o uguale all'indirizzo base si va a confrontare con l'indirizzo base più limite,
•se è minore di tale somma, infine, si accede alla cella di memoria richiesta.
•Invece, se l'indirizzo non è maggiore o uguale dell'indirizzo base oppure è maggiore dell'indirizzo base più limite, si genera una trap, cioè una eccezione bloccante che fa finire il programma con un messaggio di errore.

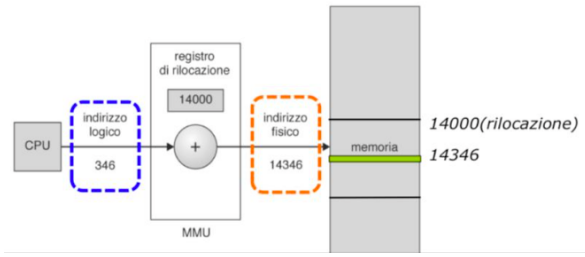
In genere un programma risiede in un disco sotto forma di un file binario eseguibile. Per poter eseguire un programma deve trovarsi (almeno parzialmente) in memoria centrale. I programmi che risiedono sul disco devono essere trasferiti in memoria centrale tramite la coda di input. La coda di input è l'insieme dei processi residenti su disco che attendono di essere trasferiti ed eseguiti. Un programma non viene semplicemente scritto ed eseguito in un passaggio, ma segue una serie di passaggi che portano, a seconda del numero di passaggi che si eseguano, ad un diverso modo di generare gli indirizzi. Questo modo di trasformare gli indirizzi generati dal programma in indirizzi fisici prende il nome di Address binding, cioè collegamento degli indirizzi scritti nel programma.

- Indirizzo logico: generato dalla CPU (o indirizzo virtuale)
- Indirizzo fisico: legato alla cella di memoria.

L'associazione di istruzioni e dati alla memoria (address binding) serve per eseguire i programmi e può avvenire in momenti diversi:

I. **Compilazione**: se la locazione di memoria è conosciuta a priori possono essere generati indirizzi assoluti. Ma in questo caso, la ricompilazione è necessaria quando la locazione di partenza del processo cambia (ad esempio, programmi per MS-DOS nel formato identificato dall'estensione .COM associano gli indirizzi fisici al tempo di compilazione). Esempio: indirizzo logico 0 – cella con indirizzo fisico 0
II. **Caricamento**: se la locazione di memoria non è conosciuta a priori si genera un codice rilocabile (gli indirizzi variano al variare dell'indirizzo iniziale). Il vantaggio del binding in fase di caricamento è che permette di caricare i programmi in un punto diverso della RAM ogni volta senza dover riscrivere il programma; questo perché ad ogni nuovo indirizzo logico crea un indirizzo logico facendo semplicemente una somma fra l'indirizzo logico e l'indirizzo di base. Esempio: indirizzo logico 0 – cella con indirizzo fisico (0 +indirizzo base)
III. **Esecuzione**: se il processo può essere spostato durante l'esecuzione, l'associazione viene ritardata al momento dell'esecuzione. È necessario un hardware specializzato (es: registri base e limite). La maggior parte dei sistemi operativi general purpose impiega questo metodo.

Il concetto di spazio di indirizzi logici/virtuali che è legato allo spazio degli indirizzi fisici è molto importante nella gestione della memoria centrale.
Gli indirizzi logici e gli indirizzi fisici sono uguali nel caso dell'address binding durante la compilazione e nel caricamento.
Nell'address binding durante l'esecuzione gli indirizzi logici sono detti virtuali e differiscono dagli indirizzi fisici.
Se in fase di compilazione dico che l'indirizzo logico corrisponde all'indirizzo fisico la condizione viene mantenuta; in tutti gli altri casi i due tipi di indirizzi sono legato tramite una somma rispetto ad una base.
Per poter gestire la traduzione tra indirizzi logici e indirizzi fisici nel computer esiste un componente che si chiama Memory-Management Unit (MMU).
La MMU realizza l'associazione tra indirizzi seguendo lo schema di gestione della memoria centrale. Importante ricordare che il programma lavora con gli indirizzi logici ma ovviamente la RAM lavora con gli indirizzi fisici, quindi la MMU è indispensabile per poter accedere ai dati.



Il binding in fase di caricamento prende il nome di rilocazione dinamica, una tecnica che permette di spostare indirizzi logici da un indirizzo di base ad un altro nel tempo secondo necessità.
Questa rilocazione dinamica si fa nel caso più semplice con un esempio di operazione di somma.

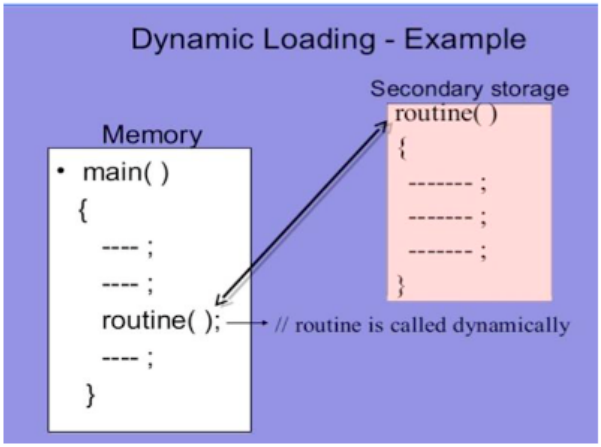
Se la CPU genera un indirizzo logico, per esempio 346, e l'indirizzo di base attuale del processo è 14000 semplicemente si fa la somma (346+14000) e si ottiene 14346, quindi si chiede alla RAM di accedere la cella 14346.
In questo caso esistono due diversi tipi di indirizzi:
- gli indirizzi logici (nell'intervallo da 0 a max)
- gli indirizzi fisici (nell'intervallo da r + 0 a r + max per un valore di base r).

Il programma utente genera solo indirizzi logici e pensa che il processo sia eseguito nelle posizioni da 0 a max. Tuttavia, questi indirizzi logici devono essere mappati in indirizzi fisici prima d'essere usati. Il concetto di spazio d'indirizzi logici mappato su uno spazio d'indirizzi fisici separato è fondamentale per una corretta gestione della memoria.

Problema: Nei S.O. più vecchi, l'intero programma e i dati di un processo dovevano essere presenti in RAM perché il processo potesse essere eseguito. La dimensione di un processo era quindi limitata alle dimensioni della memoria fisica. Per migliorare l'utilizzo della memoria fu introdotto il cosiddetto caricamento dinamico (dynamic loading). Tramite il caricamento dinamico, le routine (procedure, funzioni, metodi) vengono caricate quando sono chiamate (se non sono già in memoria). Per il caricamento dinamico si intende caricare in memoria il codice delle funzioni solo quando li evochiamo; in altri termini, se una funziona particolare non viene mai usata durante una sessione non viene mai caricata nella RAM, per cui non occuperà la memoria "a vuoto". La prima volta che chiamiamo una funziona, la funzione viene caricata nella RAM e vi rimane cosicché le volte successive sia già pronta all'uso.

Migliore uso dello spazio di memoria

1. Le routine mai usate non vengono mai caricate in memoria centrale. Utile quando molta parte del codice è usata raramente.
2. Nessun speciale supporto del S.O. In alcuni casi esistono librerie per il caricamento dinamico.



Questo schema è un esempio di un programma in C in cui la routine viene presa dalla memoria secondaria e il suo codice (i dati) viene copiato in RAM. Spesso, quando si scrive un programma facciamo uso di librerie esterne, soprattutto per i programmi di traffico o programmi che fanno uso di dispositivi esterni. Ci sono librerie che possono essere collegate al programma principale; il collegamento può essere fatto in due modi:

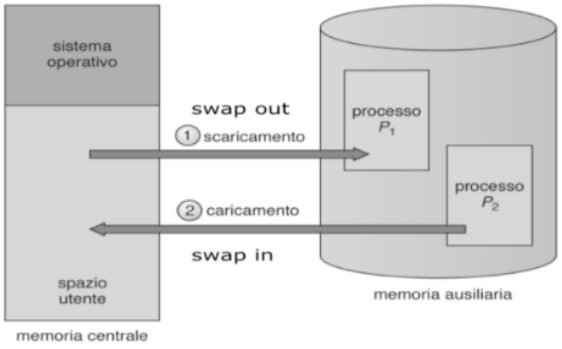
- **Link statico**: librerie di sistema e codice del programma combinati dal caricatore nell'immagine binaria.
- **Link dinamico**: il link viene rinviato fino al momento dell'esecuzione. Per esempio, in windows è possibile identificare le cosiddette librerie DLL, librerie di sistema che permette di caricare dinamicamente i programmi quando serve.

L'approccio basato sulle librerie dinamiche è molto efficiente dal punto di vista dell'occupazione della memoria. Da un punto di vista pratico, quando in un programma vogliamo far uso di una libreria dinamica, inseriamo nel codice, quello che prende il nome di stub. Gli stub (piccole porzioni di codice) vengono impiegate per localizzare la routine

appropriata nella libreria residente in memoria. Lo stub rimpiazza sé stesso con l'indirizzo della routine e la esegue.
Un altro vantaggio delle librerie dinamiche è quello di poter fare uso delle librerie condivise. Si immagini di avere due o più programmi nel SO che devono fare uso dello stesso codice di librerie; si potrebbe caricare la DLL una volta sola e poi, se un secondo programma vuole fare l'uso della stessa DLL, si può evitare di caricare nella memoria un'altra copia, usando come riferimento la copia precedentemente caricata.

Swapping

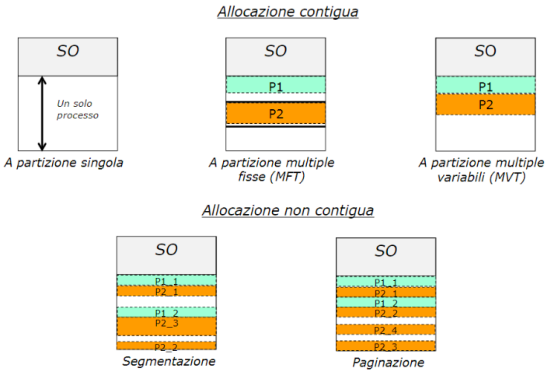
Un processo può essere temporaneamente riportato (swapped out) su disco (backing store) e quindi riportato in memoria (swapped in) al momento di riprendere l'esecuzione. Chi si occupa di fare lo swapped in o lo scaricamento di un processo?



Ovviamente, quando lo scheduler della CPU decide di eseguire un processo, richiama il dispatcher, che controlla se il processo si trova in memoria centrale. Se non si trova in memoria o non c'è spazio libero, il dispatcher scarica un processo dalla memoria e vi carica il processo richiesto dallo scheduler della CPU, quindi ricarica normalmente i registri e trasferisce il controllo al processo selezionato. In un tale sistema d'avvicendamento, il tempo di cambio di contesto (context-switch time) è piuttosto elevato.
La maggior parte del tempo di swap è il tempo di trasferimento e il tempo totale è proporzionale alla dimensione dell'area di memoria sottoposta a swap.
In un tale sistema d'avvicendamento, il tempo di cambio di contesto (context switch time) è piuttosto elevato. Per avere un'idea della sua durata si pensi a un processo utente di 100 MB e una memoria ausiliaria costituita da un normale hard disk con velocità di trasferimento di 50 MB al secondo.
Il trasferimento effettivo del processo di 100 MB da e in memoria richiede:
100 MB / 50 MB al secondo = 2 secondi
Il tempo di avvicendamento è di 2000 millisecondi.
Dal momento che dobbiamo effettuare lo spostamento in entrambe le direzioni (scaricare e poi ricaricare il processo), il tempo totale di avvicendamento è di circa 4.000 millisecondi!

Allocazione in memoria

Così come esistono diversi algoritmi per lo scheduling della CPU che permettono di ottenere le migliori prestazioni possibili, lo stesso vale per l'allocazione in memoria dei processi; in particolare esistono 5 metodologie che si dividono in due categorie:



•Allocazione contigua: il processo viene allocato in memoria in una sequenza contigua di celle;
•Allocazione non contigua: il processo viene allocato in memoria in una sequenza non contigua di celle.

Allocazione contigua a partizione singola

In memoria vi è il SO ed un unico processo (viene lasciato un po' di spazio tra processo ed il SO, perché quest'ultimo potrebbe "crescere" allocando nuovi dati e funzioni), portando ad una mancanza di multiprogrammazione (come in MS-DOS). È l'algoritmo più semplice perché comporta la necessità di un solo indirizzo base ed un solo indirizzo limite.
Allocazione contigua a partizioni multiple fisse (MTF – Multiprogramming with a Fixed number of Tasks)
Le partizioni, ovvero le sezioni in cui è divisa la memoria, hanno dimensione fissa; all'interno di ogni partizione può essere allocato un solo processo.
Questo algoritmo è stato introdotto negli anni '60 e fu implementato per la prima volta nel IBM OS/360.
Le partizioni multiple fisse hanno il vantaggio di permettere al processo di crescere, sebbene ciò può avvenire fino al limite della partizione, ma lo svantaggio di creare il fenomeno della frammentazione interna (porzione inutilizzata di memoria interna alla partizione).

Allocazione contigua a partizioni multiple variabili

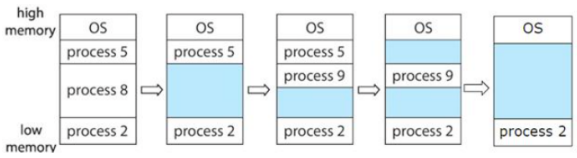
(MVT - Multiprogramming with a Variable number of Tasks)

È un approccio in cui ogni partizione è delle esatte dimensioni del processo da allocare. È utilizzato in ambienti batch o time-sharing con gestione della memoria tramite segmentazione semplice.

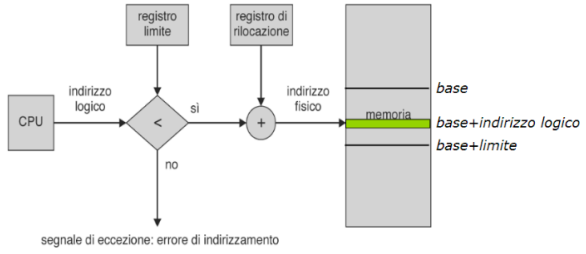
Vantaggi (vs MFT)
- Si elimina la frammentazione interna: ogni partizione è dell'esatta dimensione del processo allocato;
- Il grado di multiprogrammazione è variabile;
- La dimensione massima dei processi è limitata dalla disponibilità di spazio fisico.

Problematiche
- Scelta dell'area da allocare;
- Frammentazione esterna.

Nell'esempio si vede graficamente la collocazione di diversi processi in memoria.



Inizialmente vi sono i processi 5, 8, 2; successivamente il processo 8 termina e viene allocato inmemoria il processo 9. Terminato anche il processo 5, sebbene in totale vi è una quantità di memoria sufficiente per contenere un nuovo processo delle stesse dimensioni del processo 8, non vi è uno slot unico capace di farlo, a meno che si verifichi una deframmentazione della memoria (operazione che ha un costo elevato in termini di tempo).
Questo approccio porta a un considerevole "spreco" di memoria a causa del fenomeno della frammentazione esterna.
In caso di allocazione a partizioni multiple variabili, ad ogni processo viene associata un'area di memoria distinta.
I registri di rilocazione e limite vengono utilizzati per proteggere reciprocamente i processi utente e per prevenirne eventuali accessi a codice e dati di sistema; il registro di rilocazione contiene il valore del più piccolo indirizzo fisico dell'area di memoria allocata ad un processo mentre il registro limite definisce l'intervallo di variabilità degli indirizzi logici (ciascun indirizzo logico deve essere minore del valore contenuto nel registro)
La MMU mappa dinamicamente gli indirizzi logici negli indirizzi fisici.
Lo schema con registro di rilocazione consente al sistema operativo di cambiare dinamicamente le proprie dimensioni (codice transiente). Infatti, se un servizio (ad esempio un driver) non è comunemente usato, è inutile
tenerne in memoria codice e dati, poiché lo spazio occupato si potrebbe usare per altri scopi.



Il meccanismo di protezione della memoria si articola in due fasi:
•viene verificato che l'indirizzo logico è minore del registro limite;
•l'indirizzo logico viene sommato al registro di rilocazione (ovvero il registro di base).

Questo nuovo ordine nelle operazioni di verifica è dovuto al fatto che sappiamo che l'indirizzo di base è un numero non negativo.
Il problema principale nell'allocare dinamicamente la memoria è riempire nel modo più conveniente possibile gli slot disponibili della RAM. Questo può avvenire seguendo tre metodologie diverse:

- **First-fit**: il processo viene allocato nel primo slot grande abbastanza→garantisce migliore prestazioni in termini di velocità;
- **Best-fit**: il processo viene allocato nello slot più piccolo capace di contenerlo (è necessario scandire tutta la lista degli slot liberi)→garantisce un miglior impiego della memoria;
- **Worst-fit**: il processo viene allocato nello slot più grande tra quelli disponibili (è necessario scandire tutta la lista degli slot liberi)→si basa su un ragionamento opposto rispetto al best- fit; invece di puntare a creare il più piccolo sfrido (spazio residuo di uno slot, dopo l'allocazione di un processo) possibile, ne genera uno più grande possibile,

cosicché lo sfrido creatosi sia grande abbastanza da poter effettivamente contenere un altro processo.

Numerose indagini statistiche hanno rivelato che la metodologia di allocazione migliore è best-fit.

Nell'esempio seguente si confrontano le prestazioni degli algoritmi best-fit e first-fit.

	Dimensioni	Indirizzo
N_1	100K	1K
N_2	500K	610K
N_3	200K	1200K
N_4	300K	1520K
N_5	600K	2000K

Si supponga che la lista delle partizioni libere di memoria (di cui conosciamo l'indirizzo di partenza in memoria e la dimensione), mantenuta in ordine di indirizzo, consista di cinque elementi N1, N2, N3, N4, N5, le cui dimensioni e gli indirizzi sono riportati nella tabella.

Dovendo inserire P1, P2, P3, P4, rispettivamente di 212K, 417K, 112K, 426K (in questa successione), come e dove saranno memorizzati, usando First-Fit e Best-Fit?

Calcolare la frammentazione in entrambi i casi.

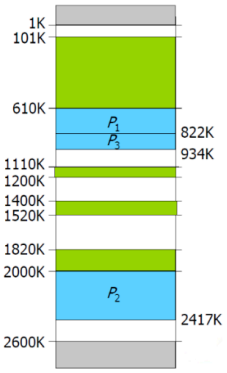
❖ Soluzione

	Dimensioni	Indirizzo
N_1	100K	1K
N_2	500K	610K
N_3	200K	1200K
N_4	300K	1520K
N_5	600K	2000K

FIRST-FIT
P_1, P_2, P_3, P_4,
di 212K, 417K, 112K, 426K

Frammentazione esterna:

100K+176K+200K+300K+183K=959K



La cosa che salta più all'occhio è che non si è potuto allocare in memoria tutti i processi; P4, infatti, non ha trovato uno slot sufficientemente capiente per contenerlo. Da notare anche che la frammentazione esterna ha dato origine ad una quantità di spazio inutilizzato non indifferente (959K), che sarebbe stato in grado di contenere P4, di soli 426K.

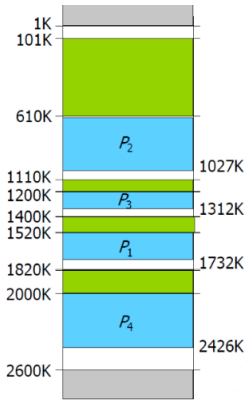
❖ Soluzione

	Dimensioni	Indirizzo
N_1	100K	1K
N_2	500K	610K
N_3	200K	1200K
N_4	300K	1520K
N_5	600K	2000K

BEST-FIT
P_1, P_2, P_3, P_4,
di 212K, 417K, 112K, 426K

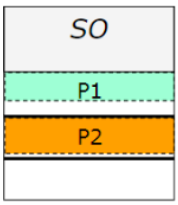
Frammentazione esterna:

100K+83K+88K+88K+174K=533K



A differenza della soluzione first-fit, tutti i processi sono stati allocati in memoria e lo spazio non sfruttato (533K) è molto minore rispetto all'esempio precedente.

Non in tutti i casi, l'algoritmo best-fit da prestazioni migliori rispetto al first-fit.



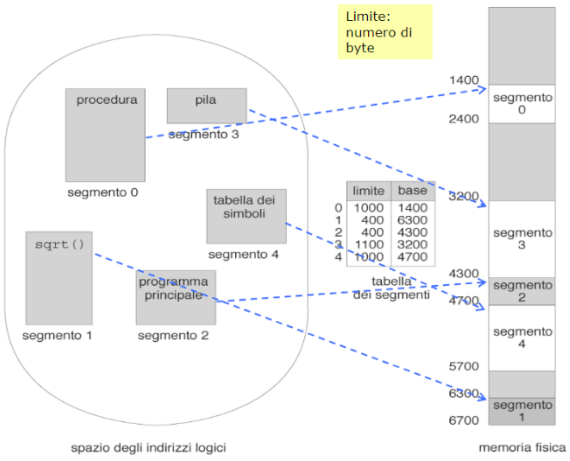
Nell'allocazione dinamica della memoria è importante gestire in modo appropriato il fenomeno della frammentazione; questa si distingue in:

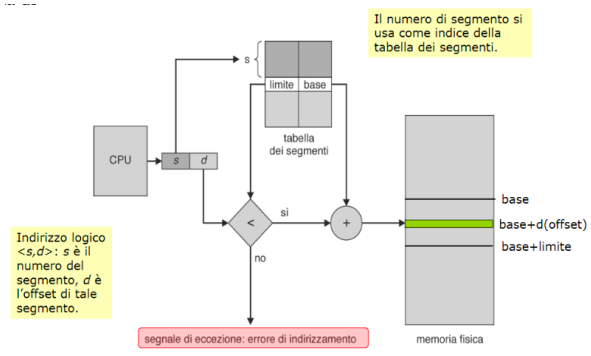
- *frammentazione interna*: la memoria allocata può essere un po' più grande di quella richiesta; la parte in eccesso è interna alla partizione ma non è usata (spazi bianchi in figura compresi tra la linea tratteggiata e la linea continua).

- *frammentazione esterna*: esiste uno spazio totale di memoria disponibile per soddisfare una richiesta ma non è contiguo (come P4 in first-fit).
  La gravità del problema della frammentazione esterna per allocazione contigua dipende dalla quantità totale di memoria e dalla dimensione media dei processi; la "regola del 50%" rivela che nell'algoritmo First-fit, per n blocchi assegnati, 0.5n blocchi possono andare "persi" per frammentazione. La compattazione riduce il problema della frammentazione esterna, in quanto la memoria libera viene compattata in un unico blocco, spostando i processi in slot contigui.
  Anche la memoria di massa (tra cui l'hard disk), se allocata in modo contiguo, soffre del problema della frammentazione.

Allocazione non contigua con segmentazione

Nella segmentazione si considera un programma come una come costituito da più segmenti (ovvero moduli del programma, creati dal compilatore, che hanno una funzione precisa) di dimensioni differenti e conservati in punti diversi della memoria.
Per semplicità di implementazione, i segmenti sono numerati, ed un indirizzo logico di una locazione consiste di una coppia di valori "numero-segmento (numero che rappresenta il segmento), offset (la distanza dell'indirizzo generato a partire dalla posizione 0 del segmento)".





Es. Numero-segmento (s)= 3; offset (d)= 1000→l'indirizzo si trova a distanza 1000 dall'inizio del segmento 3. La posizione del segmento 3 è nota al SO ma non al programma.
Il mapping tra l'indirizzo logico e l'indirizzo fisica prevede che l'indirizzo di base del segmento sia trasformato in un indirizzo di rilocazione da sommare all'offset; per cui: indirizzo fisico = offset + indirizzo di base del segmento.
Nel mapping, inoltre, si fa uso della tabella dei segmenti, in cui viene memorizzato, per ogni segmento, la coppia indirizzo di base e limite del segmento. Ciò che avviene è che:
•la CPU genera un indirizzo costituito dalla coppia di valori s-d;
•conoscendo s, si va nella tabella dei segmenti e si verifica l'indirizzo limite e l'indirizzo base relativo ad s;
•si confronta d (offset) con il limite;
•se d < limite, allora d è sommato alla base (ovvero l'indirizzo da cui inizia il segmento in RAM)

Vi sono diversi aspetti che devono essere gestiti in un sistema basato sulla segmentazione, tra cui:

- **Rilocazione**: possibilità di togliere un segmento (quindi copiarlo nel disco) e poi aggiungerlo nuovamente in seguito, in caso vi sia necessità di liberare memoria. La tabella dei segmenti permette di tenere traccia delle diverse posizioni occupate da un segmento.
- **Condivisione**: permette di rendere del codice leggibile da più programmi, evitando di ripeterlo più volte e di occupare inutilmente spazio in memoria. La condizione fondamentale per mettere in pratica la condivisione è che il segmento condiviso sia di sola lettura e, quindi, non sia possibile modificarne i dati (es. funzione “radice quadrata”)
- **Allocazione**: scegliere l’algoritmo migliore per allocare i segmenti in memoria

Nella segmentazione è implementato un sistema di protezione basato su un bit di validazione che, se assume il valore 0, indica l’illegalità di un segmento. Un segmento è illegale se è stato spostato dalla memoria, per cui, se la CPU genera un indirizzo che coinvolge questo segmento, esso va ricaricato in RAM.