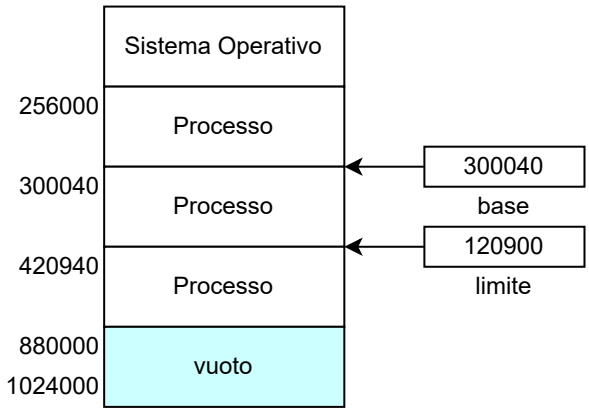
5-Gestione memoria

La gestione della memoria (RAM) è diversa da quella del processore centrale (CPU). La CPU può accedere solamente alla RAM che è abbastanza grande da consentire la multiprogrammazione e ha velocità comparabile a quella del processore.

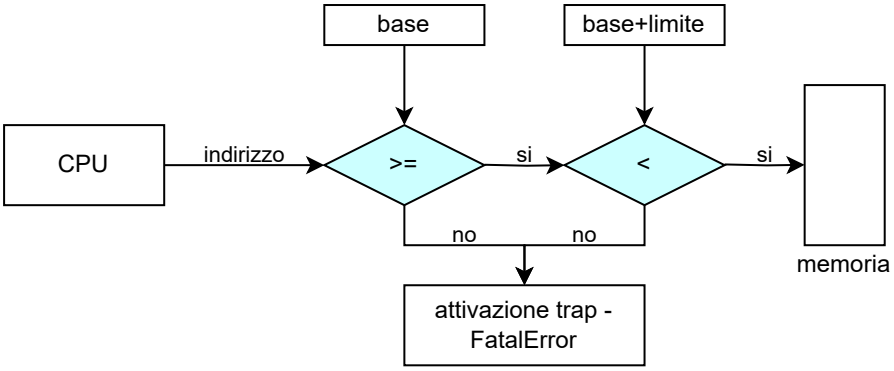
Garantire che ogni processo occupi uno spazio in memoria distinto

Bisogna determinare l'intervallo di indirizzi di memoria validi che il processo può accedere e limitarlo solo ad esso. Questa protezione è fornita da da due registri:

- base: contiene il più piccolo indirizzo di memoria
- limit: specifica la dimensione dell'intervallo.



Un programma eseguito dall'utente che prova ad accedere alla memoria del sistema operativo di un altro utente attiva una trap al SO, restituendo un FatalError. I registri base e limit possono essere modificati solo dal SO tramite un istruzione speciale.



Portare la memoria da disco a RAM

Nei sistemi moderni, un processo utente può risiedere in qualsiasi parte della memoria fisica, quindi non è necessario che l'indirizzo di inizio del processo utente sia 00000. Durante il processo di esecuzione, gli indirizzi vengono rappresentati in diverse fasi:

- Compilazione: Se si conosce già durante la compilazione dove il processo risiederà in memoria, allora è possibile generare un **codice assoluto**. Ad esempio, se si sa che un processo utente risiederà in una certa posizione, il codice del compilatore generato inizierà da lì e si estenderà da lì in su. Tuttavia, se in seguito la posizione di partenza cambia, sarà necessario ricompilare il codice.

- Linking/Caricamento: Se non si conosce durante la compilazione dove il processo risiederà in memoria, allora il compilatore deve generare un **codice rilocabile**. In questo caso, il legame finale viene posticipato al tempo di caricamento. Se la posizione di partenza cambia, sarà sufficiente ricaricare il codice utente per incorporare questo valore modificato.
- Esecuzione: Se il processo può essere spostato durante la sua esecuzione da un segmento di memoria a un altro, allora il legame deve essere posticipato fino al momento dell'esecuzione.

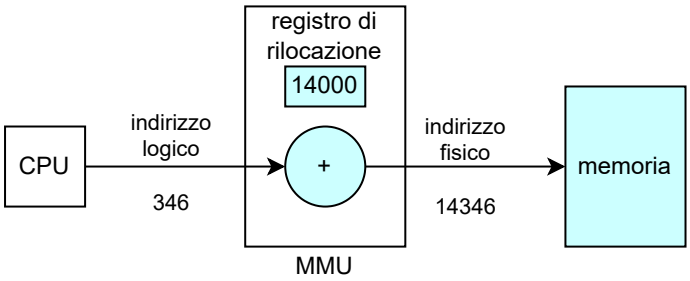
Indirizzo logico e indirizzo fisico

Un indirizzo generato dalla CPU è l'indirizzo logico, mentre quello caricato nel memory address register (MAR) è l'indirizzo fisico.

1. **Indirizzo Simbolico**: Gli indirizzi simbolici sono utilizzati nel codice sorgente del programma e rappresentano simboli come nomi di variabili, funzioni o etichette. Questi indirizzi non sono direttamente associati a posizioni specifiche in memoria, ma sono piuttosto nomi convenzionali o simbolici utilizzati dal programmatore per riferirsi a parti specifiche del codice o dei dati. Gli indirizzi simbolici nel codice sorgente vengono tradotti in indirizzi logici durante la compilazione del programma.
2. **Indirizzo Logico (o Virtuale)**: Gli indirizzi logici sono quelli utilizzati dal programma durante l'esecuzione. Sono generati dal sistema operativo e rappresentano gli indirizzi a cui il programma accede, ma non corrispondono direttamente a posizioni fisiche nella memoria del computer. Gli indirizzi logici sono utilizzati per creare una mappatura astratta e conveniente dello spazio di memoria del programma. Durante l'esecuzione del programma, gli indirizzi logici possono essere mappati o tradotti in indirizzi fisici dal sistema operativo e dall'hardware (MMU).
3. **Indirizzo Fisico**: Gli indirizzi fisici rappresentano le posizioni reali della memoria del computer. Sono indirizzi diretti utilizzati dal processore e dalla memoria per accedere fisicamente ai dati e alle istruzioni. Gli indirizzi fisici sono associati direttamente alla posizione effettiva di celle di memoria nel hardware del computer. L'associazione tra indirizzi logici e indirizzi fisici può essere dinamica e può variare durante l'esecuzione del programma, soprattutto nei sistemi di memoria virtuale dove la gestione della memoria viene ottimizzata per fornire uno spazio degli indirizzi logici più grande di quello fisicamente disponibile.

Memory Management Unit (MMU)

La MMU è un dispositivo hardware che converte gli indirizzi virtuali in indirizzi fisici. Il valore del registro di rilocazione è generato da un processo dell'utente quando viene inviato in memoria. Il programma dell'utente usa indirizzi logici, non vede mai gli indirizzi fisici.



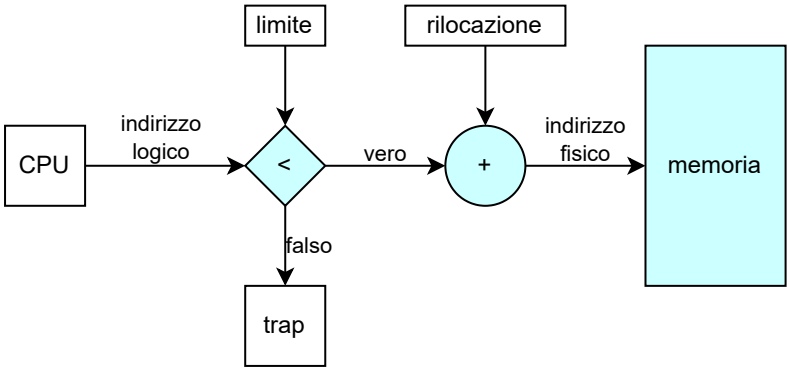
Swapping

Un processo deve trovarsi in memoria per essere eseguito, ma può essere temporaneamente spostato dalla memoria a uno spazio di archiviazione secondario (detto backing store) e successivamente riportato in memoria per continuare l'esecuzione. Lo swapping rende possibile superare lo spazio fisico totale della memoria disponibile nel sistema, permettendo così un maggior grado di multiprogrammazione.

Quando il sistema ha esaurito la memoria fisica dedicata ai processi (RAM), può utilizzare lo swapping per liberare memoria spostando momentaneamente alcuni processi su uno spazio di archiviazione secondario, come un disco rigido. In questo modo, la memoria fisica viene liberata per l'esecuzione di nuovi processi. Quando un processo spostato è richiamato, viene riportato in memoria principale dallo spazio di archiviazione secondario.

Protezione Memoria

Per evitare che un processo acceda ad un indirizzo di memoria non suo si utilizza sia il relocation register che contiene l'indirizzo più piccolo che il limit register che contiene l'intervallo degli indirizzi logici.



Inizialmente tutta la memoria disponibile ai processi dell'utente è vuota: è costituita da un unico **hole**. Il SO gestisce l'allocazione della memoria, decidendo quali processi ricevono spazio di memoria e quali devono attendere in coda. Quando un processo termina, rilascia la memoria utilizzata, che può essere riutilizzata per caricare un nuovo processo dalla coda di input.
La memoria viene allocata ai processi finché lo spazio di memoria richiesto da un processo non è disponibile (non esistono hole abbastanza grandi). Il SO può allora aspettare che si liberi un blocco sufficientemente grandi o può scorrere la coda degli input per controllare se c'è un processo con requisiti di memoria più piccoli. Soluzioni per gestire questo tipo di richieste (dynamic storage allocation) sono:

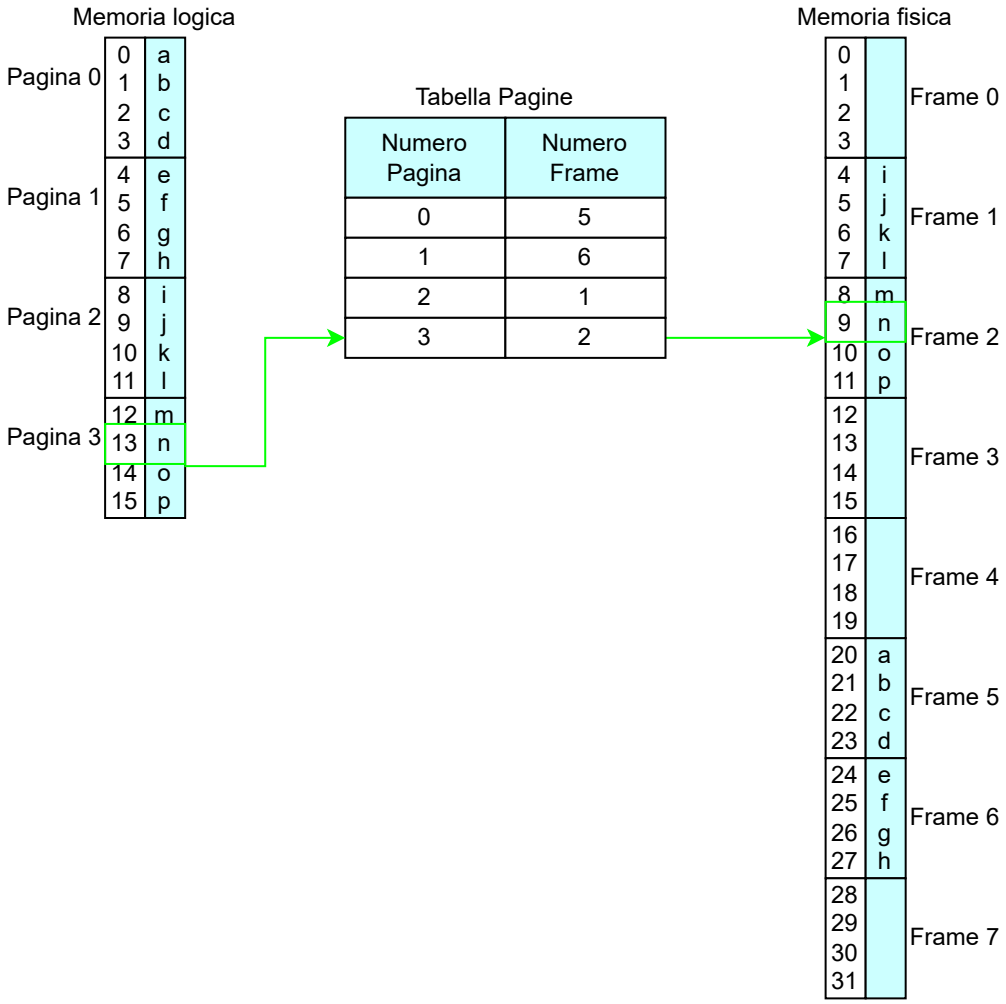
- First fit: Si alloca il primo buco abbastanza grande
- Best fit: si alloca il più piccolo buco sufficientemente grande
- Worst fit: si alloca il buco più grande. Questa strategia produce buchi più grandi, che sono preferiti a quelli piccoli.

Frammentazione

Con il first fit e best fit si incorre in un problema di frammentazione esterna. Mentre i processi vengono caricati nella memoria disponibile, si formano piccoli buchi non contigui di memoria, e nessuno di questi è in grado di avere un processo allocato. Questi buchi sono molto numerosi, e per ogni n processi che occupano memoria, altri $n/2$ blocchi vengono persi per colpa della frammentazione.
Una soluzione a questo problema è la compattazione: si mischiano i contenuti per piazzare tutta la memoria libera in un unico grande blocco, tuttavia questo processo è costoso.

Paginazione

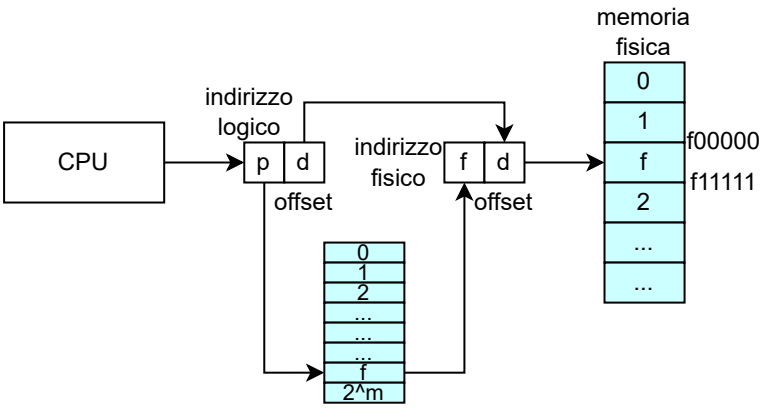
La paginazione permette di gestire la memoria in modo flessibile, consentendo ai processi di avere spazi di indirizzi non contigui. La paginazione risolve problemi come la frammentazione esterna, dove la memoria libera è divisa in blocchi non contigui, riducendo la complessità del sistema e semplificando la gestione della memoria. Inoltre evita la necessità di compattare la memoria per riorganizzare i blocchi di memoria libera, che può essere un'operazione dispendiosa in termini di tempo e risorse del sistema.
Si implementa suddividendo la memoria fisica in blocchi di dimensioni fisse detti **frame**, e suddividendo la memoria logica in blocchi detti **pagine**. Ogni indirizzo generato dalla CPU ha un numero di pagina (p) e un offset (d). Il numero della pagina è usato come indice della **tabella delle pagine**, che contiene gli indirizzi di ogni pagina in memoria fisica che vengono poi combinati con l'offset per determinare l'indirizzo nella memoria fisica.
La dimensione della memoria logica è 2^m, mentre quello di una pagina è 2^n. Esempio con $m = 2$ e $n = 4$.



Per trovare la posizione di un processo da memoria fisica f conoscendo la sua posizione in memoria logica l:

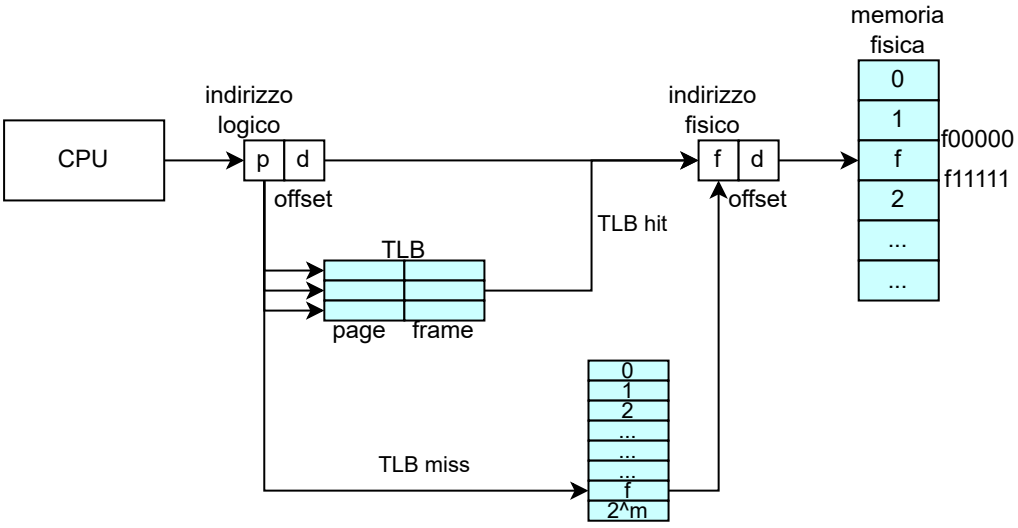
- `dim_pagina=2^m`
- `pagina=l/dim_pagina`
- `offset=l%dim_pagina`
- `frame=page_table[pagina]`
- `f=(frame*m)+offset`
  Questo processo richiede operazioni costose come la divisione e la moltiplicazione, e può essere semplificato per sfruttare la natura binaria del calcolatore.

- base 10 per trovare la posizione in memoria fisica di 6:
  $6/4 = 1r2 \quad pag1 \rightarrow frame6 \quad f = (6 * 4) + 2 = 26$
- base 2 per trovare la posizione in memoria fisica di 6:
  $110/100 = 1r10 \quad pag1 \rightarrow frame6 \quad f = (100 * 100) + 10 = 11010$
  Questo processo in binario si semplifica in:
- `exp= x` del divisore scritto in forma 2^x in base 10
- `offset=prime x cifre di l` (da destra)
- `pagina= resstanti cifre di l`
- `f=page_table[pagina] + offset` dove la somma è un unione dei due numeri.



Implementazione della Tabella di Paginazione

- La tabella di paginazione viene mantenuta nella memoria principale.
- Il registro base della tabella di pagina (PTBR) punta alla tabella di pagina.
- Il registro di lunghezza della tabella di pagina (PRLR) indica la dimensione della tabella di pagina.
  In questo schema, ogni accesso ai dati/istruzioni richiede **due accessi alla memoria**. Uno per la tabella di pagina e uno per i dati/istruzioni.
  Il problema dei due accessi alla memoria può essere risolto utilizzando una cache hardware speciale per la rapida ricerca chiamata memoria associativa o buffer di traduzione (translation look-aside buffer, TLB).
  Quando alla memoria associativa viene presentato un oggetto, questo viene subito comparato con tutte le chiavi contemporaneamente. Se l'oggetto è presente sulla TLB allora il suo valore viene restituito (TLB hit). Se non è presente (TLB miss), bisogna fare una richiesta di accesso alla memoria principale.



Tempo di Accesso Effettivo

- ϵ: tempo per fare ricerca associativa
- x: durata ciclo di memoria
- α: hit ratio (percentuale di volte che un numero è trovato sui registri associativi)

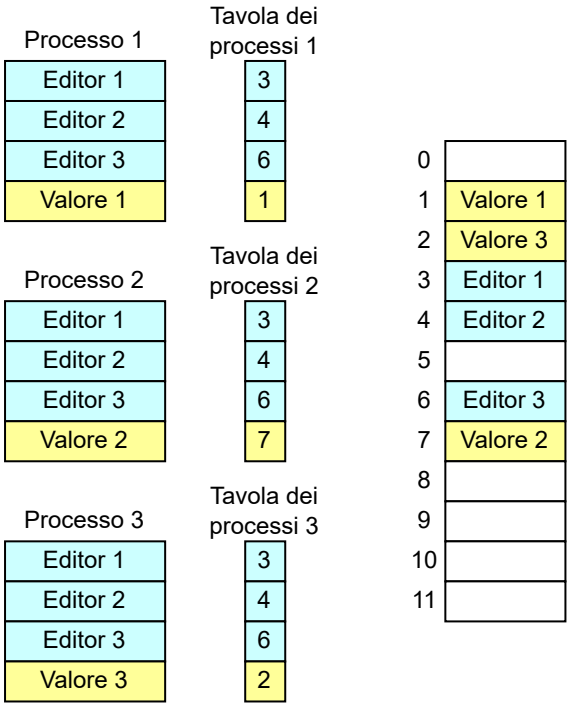
$$EAT = (1+\epsilon)\alpha + (2 \times x + \epsilon)(1-\alpha) = 2 \times x + \epsilon - \alpha$$

Protezione Memoria

La protezione di memoria viene implementata estendendo la tabella della pagina per includere un bit di validità. Se il bit di validità di un elemento è "valido" significa che la pagina associata è nello spazio logico e dunque è una pagina legale. "invalido" indica che la pagina non è nello spazio logico del processo.

Pagine Condivise

Un vantaggio della paginazione è la possibilità di condividere il codice comune. Questo è particolarmente utile in un ambiente con time-sharing. Il codice **re-entrante** può essere condiviso. Del codice re-entrante è progettato in modo tale da essere sicuro per l'uso in ambienti concorrenti. Quindi può essere chiamato da più parti del programma o da più thread simultaneamente senza causare conflitti o corruzioni dei dati. Questo viene realizzato spesso evitando l'uso di dati globali condivisi e assicurandosi che il codice non abbia effetti collaterali imprevisti quando viene interrotto e ripreso in un altro contesto.
Solo una copia del codice re-entrante viene condivisa tra i processi. Tuttavia per funzionare il codice condiviso deve apparire nella stessa posizione in memoria logica per tutti i processi.



Struttura della paginazione

Paginazione gerarchica

Nei SO moderni con 2^{32} o 2^{64} indirizzi logici, la tabella di paginazione diventa eccessivamente grande, dunque si vuole evitare di allocarla tutta nella memoria principale. Si può usare una paginazione a 2 livelli, dove la page table è a sua volta paginata.

Tabelle di paginazione Hashed

Un'altra alternativa sono le tabelle di paginazione hashed, dove l'hash è il numero della pagina virtuale. Ogni elemento della page table è composto da 3 campi:

1. Numero di pagina virtuale
2. Valore del frame della pagina mappata
3. Puntatore all'elemento successivo della linked list.
   L'algoritmo funziona così:
4. Il numero della pagina virtuale nell'indirizzo virtuale è mappato nella hash table.
5. Si controlla se il numero della pagina virtuale è uguale al primo elemento della lista.
6. Se è uguale il frame corrispondente è usato per creare l'indirizzo fisico richiesto.
7. Se è diverso, si scorre tutta la lista per cercare un valore uguale.

Tabelle di paginazione invertite

Le tabelle di paginazione invertite hanno un'entry per ogni frame di memoria (reale). Ogni entry consiste nell'indirizzo virtuale della pagina memorizzata in quella locazione di memoria e informazioni sul processo di cui fa parte la pagina.
Poiché una singola tabella di paginazione invertita può essere condivisa tra più processi, è fondamentale includere un identificatore dello spazio degli indirizzi in ogni voce della tabella per distinguere quali indirizzi virtuali appartengono a quale processo. Questo identificatore dello spazio degli indirizzi consente al sistema operativo di determinare correttamente a quale processo appartiene un dato indirizzo virtuale quando viene richiesta una traduzione da indirizzo virtuale a indirizzo fisico.
Per trovare questo identificatore bisogna accedere alla memoria centrale ogni volta che si cerca sulla tabella di paginazione, quindi per evitare di fare tante richieste quante sono le entry nella tabella, si usa una hash map al posto di una lista.
Le tabelle di paginazione invertita non furono mai ampiamente diffuse.