Gestione della memoria

> **Legge di Parkinson**
>
> I programmi si espandono in modo da riempire tutta la memoria che hanno a disposizione per contenerli.

Situazione migliore -> Memoria infinitamente grande, non volatile e a basso costo.

Siccome questa cosa è attualmente impossibile abbiamo creato il concetto di **gerarchia della memoria.**

Il compito del Sistema Operativo è quello di astrarre la gerarchia in un modello comodo e facilmente gestibile.

> **Gestore della memoria**
>
> Parte del Sistema Operativo che ha il compito di gestire la memoria:
>
> - Tiene traccia di quali parti della memoria sono in uso.
> - Allloca memoria ai processi.

Esistono diversi schemi di gestione della memoria.

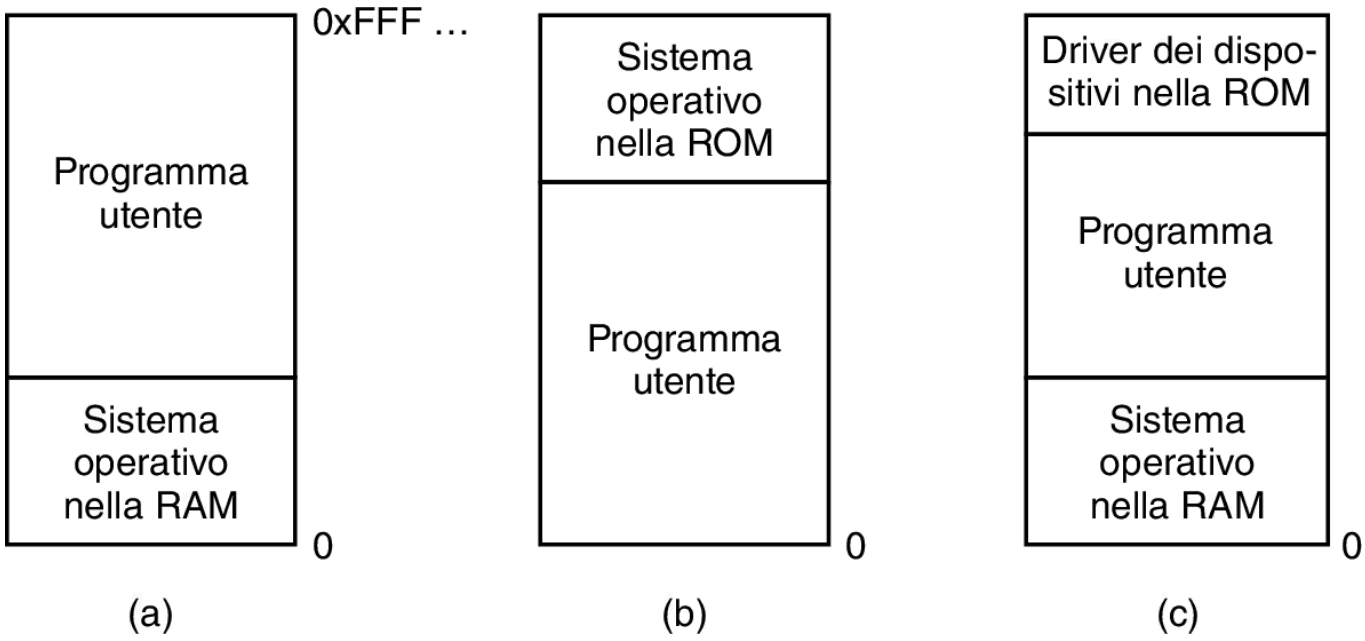
Nessuna astrazione di memoria

La più semplice astrazione della memoria è l'assenza dell'astrazione.

> **Periodo Storico**
>
> Primi Computer mainframe (1960), primi minicomputer (1970), primi personla computer (1980).

Quando si eseguiva un istruzione venivano spostati fisicamento i bit all'interno della memoria. Era impossibile avere due programmi eseguiti contemporaneamente, inoltre non ci sta nessuna sicurezza sulla cancellazione dei dati.

Anche in questo modello ci sono diverse possibilità riguardanti lo storage del SO:



- SO in fondo alla memoria nella **RAM**.
- SO in cima alla memoria nella **ROM**.
- Driver in cima alla memoria (nella **ROM**) e il resto del SO nella **RAM** in fondo.

Il primo modello è raro incontrarlo ed era usato su mainframe e minicomputer.

Il secondo è usato su computer palmari e sistemi embedded.

Il terzo è stato usato sui primi personal computer dove parte della ROM era chiamata **BIOS**.

> ⚠ **Problema dei modelli (a) e (c)**
>
> Un difetto nel programma utente può cancellare il SO con conseguenze disastrose.

In questi modelli si può eseguire un processo alla volta. Il SO copia dalla memoria non volatile alla memoria il programma richiesto e lo esegue, poi attende per il comando successivo. Quando riceve il nuovo comando lo carica in memoria sovrascrivendo il primo.

Per avere un grado di parallelismo sarebbe utile usare dei thread, ma è ==comunque un uso limitato ==perchè gli utenti spesso vogliono eseguire contemporaneamente programmi non correlati, cosa che non è permessa dall'astrazione dei thread.

Esecuzione di più programmi senza astrazione della memoria

Il SO deve salvare l'intero contenuto della memoria in un file su memoria non volatile, quindi prelevare ed eseguire il programma succesivo. Finchè in memoria ci sta un solo programma

non ci sono conflitti (**Swapping**).

E' possibile eseguire più programmi senza scambio con l'uso di hardware speciale.

> **Primi modelli 360 IBM**
>
> Suddividevano la memoria in blocchi di 2Kb, ognuno gli veniva assegnata una chiave di protezione di 4 bit, salvati nei registri.
> La **PSW(Program Status Word)** conteneva una chiave a 4 bit.
> L'hardware del 360 **bloccava** ogni tentativo da parte dei processi in esecuzione di accedere alla memoria con un codice protettivo diverso da quello della chiave del PSW.
> Siccome solo il SO può cambiare le chiavi, i processi utente non potevano interferire con nessun'altro processo.

> **Problema con questo modello**
>
> I Programmi utilizzano indirizzi assoluti di memoria fisica, portando a conflitti durante l'esecuzione. La mancanza di astrazione dell'indirizzo può causare il crash dei programmi.

Un'astrazione della memoria: gli spazi di indirizzi

I problemi di non avere alcun tipo di astrazione sono i seguenti:

- Esporre la memoria fisica ai processi -> Possibile eliminazione del SO e interferenze con altri processi.
- Niente multiprogrammazione.

Per mettere che i processi non interferiscano l'uno con l'altro dobbiamo risolvere due problemi:

- **Protezione**
- **Rilocazione**

Per quanto riguarda il primo, abbiamo già una soluzione introdotta da IBM, ma non risolve il secondo problema.

La soluzione migliore è inventare una nuova astrazione per la memoria: **lo spazio degli indirizzi**

> **Astrazioni**

Processo --> CPU astratta per eseguire i programmi
Spazio degli indirizzi --> Memoria astratta utilizzabile dai programmi

Uno spazio degli indirizzi è l'insieme degli indirizzi che un processo può usare oer indirizzare la memoria, ogni processo ha il suo spazio personale, indipendente da quello degli altri.

Come facciamo a dare ad ogni processo il SUO spazio di indirizzi?, tramite **registri base e limite**

ⓘ Rilocazione dinamica

Mappa lo spazio degli indirizzi di ogni processo su una parte della memoria fisica.
Funziona tramite due registri speciali nella CPU: **registro base e limite**.
Quindi carichiamo i programmi normalmente sulla memoria, ma al momento dell'esecuzione inseriamo nel registro base l'indirizzo fisico in cui comincia il programma e nel registro limite la lunghezza del programma.
Ogni volta che un processo fa riferimento alla memoria, prima di inviare l'indirizzo sul bus di memoria, l'hardware della CPU aggiunge il valore di base all'indirizzo generato dal processo e controlla se l'indirizzo offerto sia uguale o maggiore a quello limite, se si viene generato un errore.

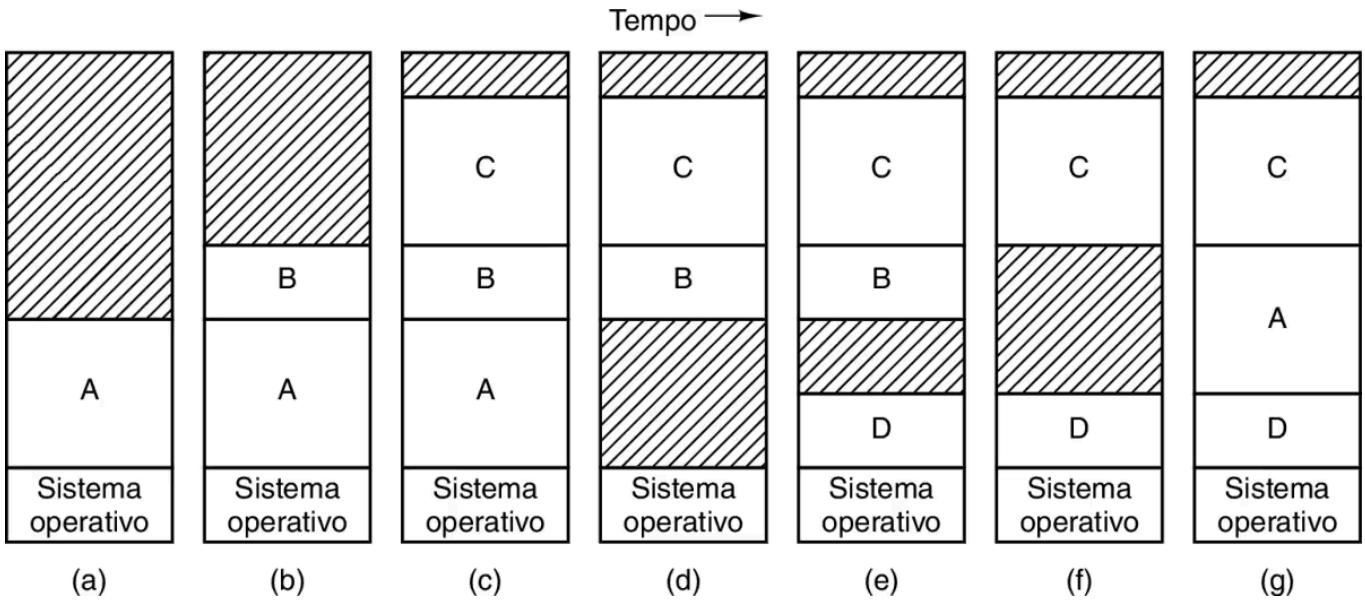
Uno svantaggio della rilocazione con registri base e limite è la necessità di eseguire una somma e un confronto a ogni riferimento alla memoria.

Swapping

La quantità totale di RAM necessaria per i processi per mantenere tutti i processi è spesso più grande della memoria fisica.
Per gestire il sovraccarico di memoria sono stati sviluppati due approcci generali:

- **Swapping**
- **Memoria virtuale**

Lo swapping consiste prelevare un processo nella sua totalità, eseguirlo per un certo periodo di tempo, per poi porlo di nuovo nella memoria non volatile.



> **Memory Compaction**
>
> Tecnica che compatta tutti gli spazi vuoti in un unico grande spazio vuoto.
> Chiede molto tempo di CPU.

Se sappiamo che i dati di dei processi possono crescere è opportuno allocare un pò di memoria extra ogni volta che un processo viene scambiato o spostato.

Gestione della memoria libera

Quando la memoria è assegnata dinamicamente il SO deve trovare un modo per gestirla, ci sono in particolare due metodi:

- **Bitmap**
- **Liste**

Gestione della memoria con bitmap:

La memoria, con una bitmap, viene divisa in unità di allocazione alle quali è associato un bit della bitmap:

- **0** --> L'unità è libera.
- **1** --> L'unità è piena.

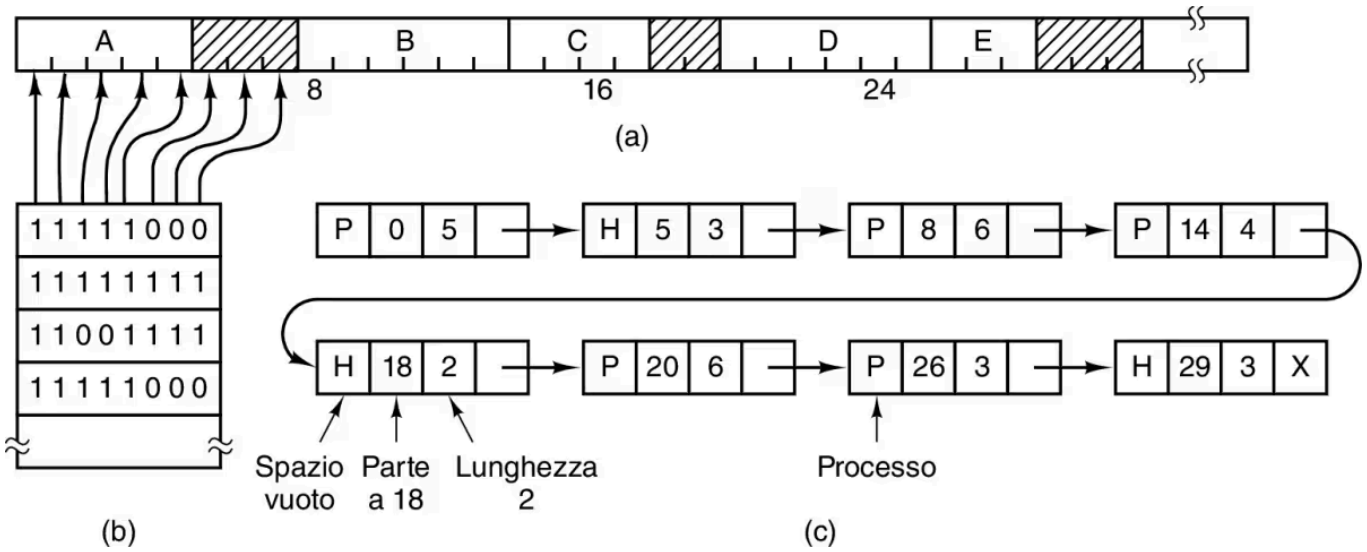
La dimensione dell'unità d'allocazione è importante, se scegliessimo una quantità troppo piccola la bitmap risulterebbe poco utile, perché andremmo a mappare l'intera memoria fisica, mentre se scegliessimo una quantità troppo grande ma ci potrebbe essere uno spreco grande di memoria.

La dimensione della bitmap dipende solo dalla dimensione della memoria e dalla dimensione dell'unità di allocazione.

> ⚠ **Problema**
>
> Quando dobbiamo allocare un processo nella bitmpa dobbiamo cercare una serie di k bit 0 consecutivi, il che è un operazione lenta.

Gestione della memoria con liste concatenate:

Manteniamo una lista concatenata di segmenti di memoria allocati e liberi, ogni segmento è uno spazio vuoto fra due processi o contiene un processo.



Gestiamo la memoria tramite una lista come in C, viene creata una struttura per ogni segmento della memoria, nella lista abbiamo:

- **Processo/buco**
- **L'inizio dell'indirizzo di memoria**
- **Lunghezza processo**
- **Puntatore al nodo successivo**

Esistono diversi algoritmi per allocare memoria nel momento in cui nasce un nuovo processo.

- **First fit:** Appena trovo uno spazio libero nella lista inserisco il nuovo processo, è molto veloce e la situazione peggiore si verifica se lo spazio libero si trova alla fine.
- **Next fit**: Variante del First fit, funziona uguale al First fit, ma tiene traccia della posizione dove ha tenuto l'ultimo spazio libero, il problema è che non si tiene conto di possibili posti di memoria che si liberano, quindi si hanno perfomance peggiori.

- **Best fit**: Scorre la lista è cerca la segmentazione (Quella più piccola che può contentere il processo) migliore per il processo, problema deve scorrere tutta la lista, ciò crea tanti piccoli spazi vuoti fra dei processi, ci sta uno spreco di memoria, first fit ha buchi più grandi.
- **Worst fit**: Logica inversa del Best fit non è buon algoritmo, quindi si creano buchi di memoria piu grandi da riutilizzare, fa un po schifo.
- **Quick fit**: Ci sono liste separate di dimensioni diverse, il processo è inserito nella lista della stessa dimensione, ha lo stesso tipo di problematiche degli altri processi, quello della ricerca, ma ciò è un problema della struttura dati lista che ti obbliga a vedere tutta la lista per la ricerca.

> ⓘ **Vantaggio della lista ordinata per indirizzi**
>
> Quando un processo termina o viene scambiato su disco, l'aggiornamento della lista è molto semplice.