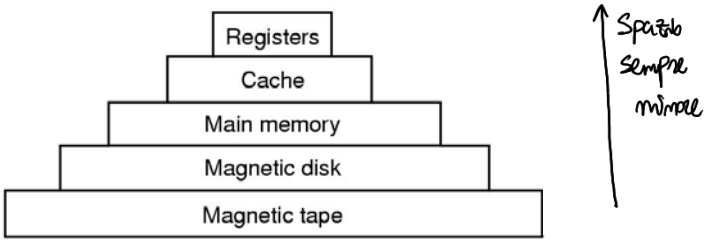
Gestione della Memoria

- Idealmente la memoria dovrebbe essere
  - grande
  - veloce
  - non volatile
- Gerarchia di memorie
  - Disco: capiente, lento, non volatile ed economico
  - Memoria principale: volatile, mediamente grande, veloce e costosa
  - Cache: volatile, veloce, piccola e costosa
- La gerarchia di memorie e' gestita dal "memory manager" (*gestore della memoria*)

1

Tipica Gerarchia di Memoria



La filosofia della gerarchia della memoria consiste nel portare verso il vertice della piramide le informazioni più utilizzate dalla porzione di programma in esecuzione (che possono cambiare nel tempo) così che nella maggior parte dei casi il tempo di accesso coincide con quello delle memorie più rapide.

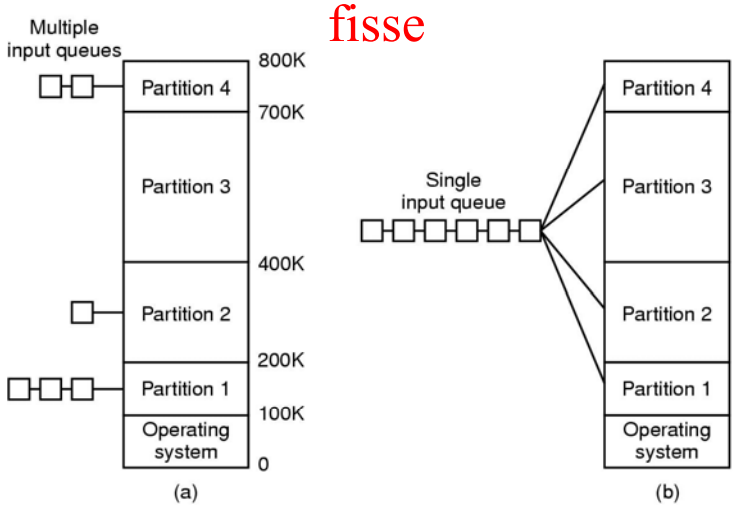
2

Modelli di gestione della memoria

- Modelli che non spostano i processi dalla RAM una volta iniziata l'esecuzione (no rilocazione):
  - **monoprogrammazione**
  - **multiprogrammazione a partizioni fisse**
- Modelli che spostano un processo in esecuzione da RAM a disco (rilocazione):
  - immagine del processo interamente in memoria:
    - **Swapping** (quello visto da noi)
  - immagine del processo parzialmente in memoria:
    - **Paginazione**
    - **Segmentazione con paginazione**

3

Ambiente multiprogrammato con partizioni fisse



- Partizioni fisse
  - Con code dei job distinte per ogni partizione
  - Con unica coda dei job

4

Se conosco indirizzo base, ricavare indirizzo assoluto a partire dal relativo è facile.

Rilocazione e Protezione

Come abbiamo visto, per poter allocare liberamente i processi in memoria (RAM) occorre che gli indirizzi nel codice non siano assoluti: ciò si può realizzare *separando* gli *indirizzi logici* dagli *indirizzi fisici*. Il *binding* (associazione) tra indirizzi logici e indirizzi fisici può avvenire in momenti diversi: durante la compilazione, durante il caricamento oppure a run-time. L'ultimo approccio è il più flessibile (permette di spostare un processo da una parte all'altra della RAM anche durante l'esecuzione).

La MMU (Memory Management Unit) si occupa di operare la traduzione al momento dell'accesso in memoria.

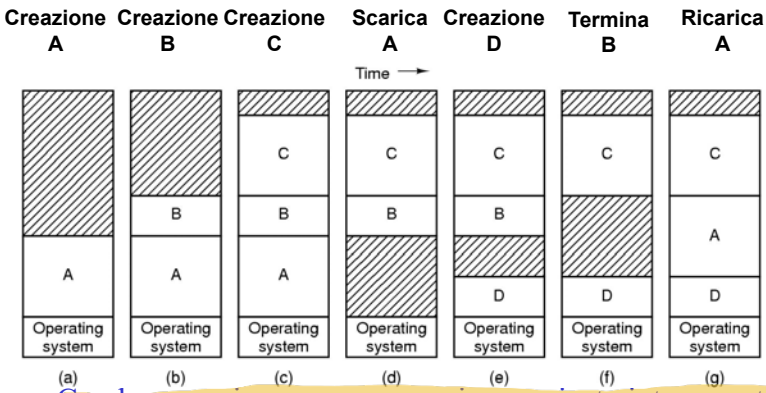
Questa tecnica permette anche di confinare il raggio di azione dei processi alla porzione di memoria ad essi assegnata. Per esempio se si usano un registro base e un registro limite (modificabili solo in modo kernel) e la MMU svolge la funzione:

if (ind_logico>limite) *then* trap(Segmentation fault)

else ind_fisico = ind_logico + base

5

Swapping



- Con lo swapping un processo viene caricato interamente in memoria in indirizzi consecutivi
- Non ci sono partizioni fisse
- L'allocazione della memoria cambia quando:
  - I processi vengono caricati in memoria
  - I processi rilasciano la memoria

6

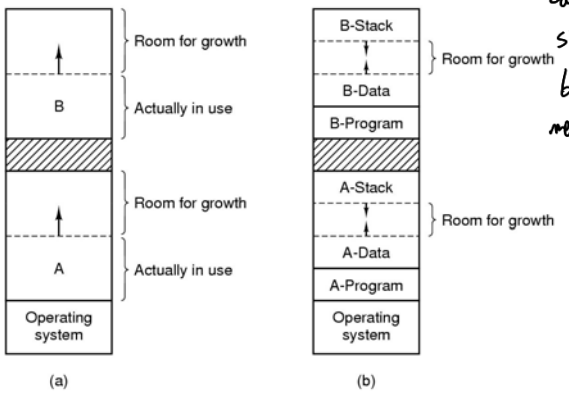
Spazio indirizzamento può avere dimensioni diverse. Per un nuovo processo devo trovare un buco sufficientemente grande da contenere indirizzi consecutivi.
Prima o poi avremo una memoria frammentata con pezzi liberi di memoria ma consecutivi. Devo effettuare riposizionamento in memoria per fare spazio.

Swapping (2)

- Rilocazione e protezione possono usare ancora i registri *base* e *limite*
- Problema: la frammentazione della memoria
  - molte aree piccole
  - compattazione
- Problema: stabilire quanto spazio allocare per ogni processo
  - area dati, stack
- Problema: come tenere traccia della memoria libera

7

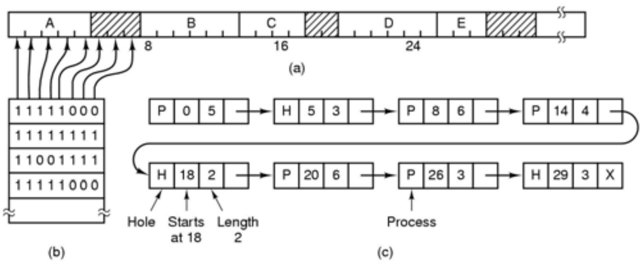
Swapping (3)



Se vi sono strutture dati che possono crescere (come la heap dove allocare dinamicamente strutture dati o lo stack) conviene allocare fin dall'inizio più spazio di quanto richiesto inizialmente. Se si eccede anche quello, o c'è un buco contiguo, oppure devo spostare in uno spazio più grande, o ancora fare swap out.

8

Gestione della Memoria con Bit Map e con lista linkata



Porzione di memoria con 5 processi e 3 aree libere (le suddivisioni indicano l'unità di allocazione)

La lista linkata può essere mantenuta solo per lo spazio libero. Mantenendo la lista degli spazi liberi separata si velocizza la ricerca di uno spazio libero, inoltre gli elementi della lista possono essere memorizzati direttamente nello spazio di memoria libero corrispondente (riducendo l'overhead dovuto alle strutture dati occupate dal sistema operativo).

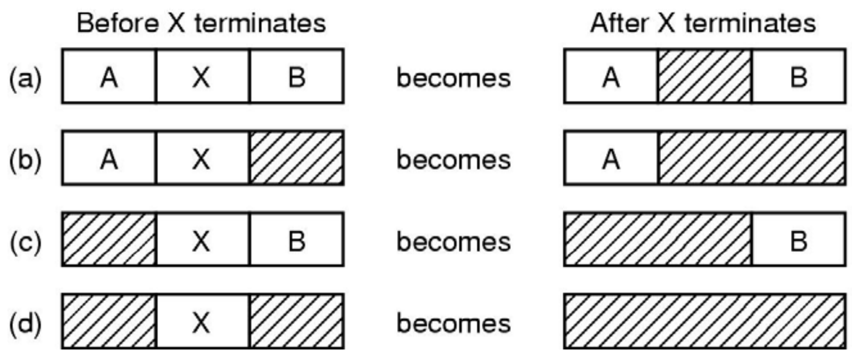
9

BitMap: mappa di bit, tanti quanti sono i pezzi in cui ho diviso la memoria centrale, 1 se occupato, 0 se libero. Possiamo capire subito la dimensione di spazi liberi.
Oppure lista concatenata che hanno record per indicare che P = processo a partire da indirizzo "0" per "5" slot ([P|0|5]→). Ogni tanto, S.O. fa deframmentazione.*

* S.O. cerca un buco con certe dimensioni e inserisce processo così.

Aggiornamento dello spazio libero

L'aggiornamento dello spazio libero è immediato quando si utilizza la mappa di bit (ponendo a 0 i bit delle unità liberate si attua automaticamente l'unificazione di spazi liberi contigui).



Quando termina un processo, se vi sono spazi liberi contigui allo spazio liberato, occorre unificarli (doppia concatenazione).

10

Algoritmi per l'allocazione della memoria

Allocazione di un blocco di memoria di dimensione x:

- First Fit
  - Individua la prima porzione di memoria libera di dimensione $\geq x$ (minimizza i tempi di ricerca)
- Best Fit
  - Individua la più piccola porzione di memoria libera di dimensione $\geq x$ (Consente di evitare di occupare spazi troppo grandi che potrebbero essere necessari per allocare processi più grandi. Ha lo svantaggio di creare dei buchi troppo piccoli per poter essere riutilizzati.) E' reso più veloce se la lista dei buchi è ordinata per dimensione.
- Worst Fit
  - Individua la porzione di memoria libera tale che la differenza fra la dimensione del processo (x) e lo spazio libero sia massima. (Serve per lasciare spazi abbastanza grandi da riuscire ancora ad utilizzarli)

11