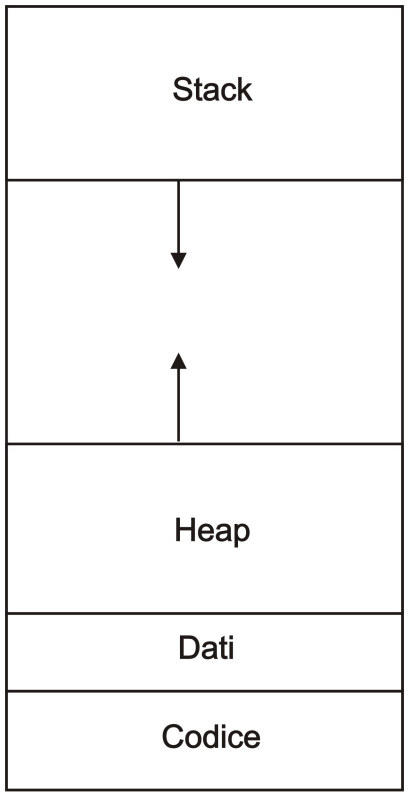
Memoria contigua

Gestione della memoria



Memoria centrale

- Il meccanismo: I processi durante la loro esecuzione richiedono varie quantitá di memoria al sistema operativo.

Esempio di programma C

```
#include <stdio.h>
#define BUFSIZE 30
#define FILEIO 1
void ucase(char *s)
{
   int n,i;

   n=strlen(s); printf("%d\n",n);
   for(i=0;i<n;i++) s[i] -= 32;
}
void main()
{
    int       i,j;
    char      scanStr[BUFSIZE], c;
    size_t    readSize;
    puts("Scrivi una stringa: ");
    gets(scanStr);   // legge una stringa
    ucase(scanStr);  // trasforma in maiuscolo
    puts(scanStr);   // stampa la stringa
    j=0;
    for(i=strlen(scanStr)-1;i>=0;i--)
    {
          c=scanStr[i];
          printf("%c",c);
    }
}
```

Traduzione in assembler

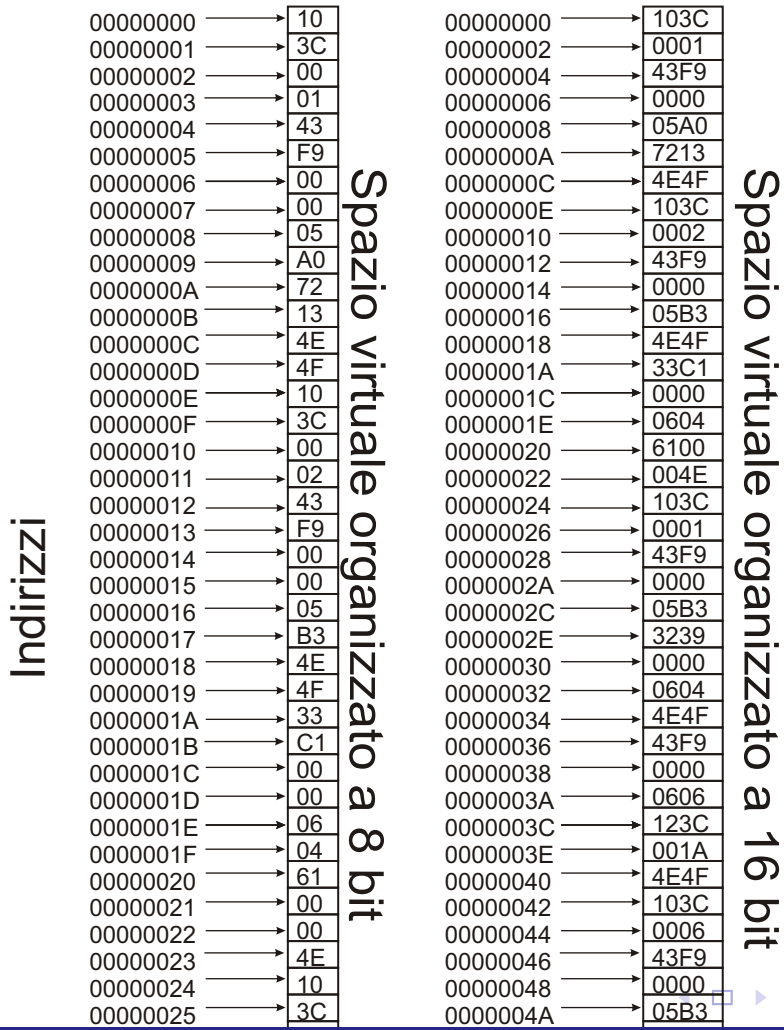
```
start
      move.b  #1,d0
      lea     m0,a1                    - - - -  - - ->
      moveq   #lenm0,d1                |
      trap    #15    ;printf           |       nloop trap    #15      ; CR
      move.b  #2,d0                    |             dbra    d2,nloop
      lea     buff,a1                  |             stop    #$2000
      trap    #15          ;scanf      |       ucase                  ; procedura per maiuscolo
      move    d1,blen                  |             move    blen,d1
      bsr     ucase        ;in maiuscolo|            lea     buff,a1
      move.b  #1,d0                    |             bra     eloop
      lea     buff,a1                  |       loop  move.b  (a1)+,d2
      move    blen,d1                  |             cmp.b   #$61,d2
      trap    #15          ;stampa str |             blt     eloop
      lea     m1,a1                    |             cmp.b   #$7A,d2
      move.b  #lenm1,d1                |             bgt     eloop
      trap    #15          ;stampa msg |             sub.b   #$20,d2
      move.b  #6,d0                    |             move.b  d2,-1(a1)
      lea     buff,a1                  |       eloop dbra    d1,loop
      move    blen,d2                  |             rts
      add     d2,a1                    |       m0    dc.b    'Scrivi una stringa: '
      subq    #1,d1                    |       lenm0 equ     *-m0
rloop move.b -(a1),d1                  |       buff  ds.b    80
      trap    #15          ;str rov.   |             ds.w    0
      dbra    d2,rloop                 |       blen  ds.w    1
      moveq   #1,d2                    |       m1    dc.b    ', la stringa rovesciata e': '
      moveq   #0,d1                    |       lenm1 equ     *-m1
      move.b  #0,d0  - - - - - - - - > |             end     start
```

Traduzione in codice macchina

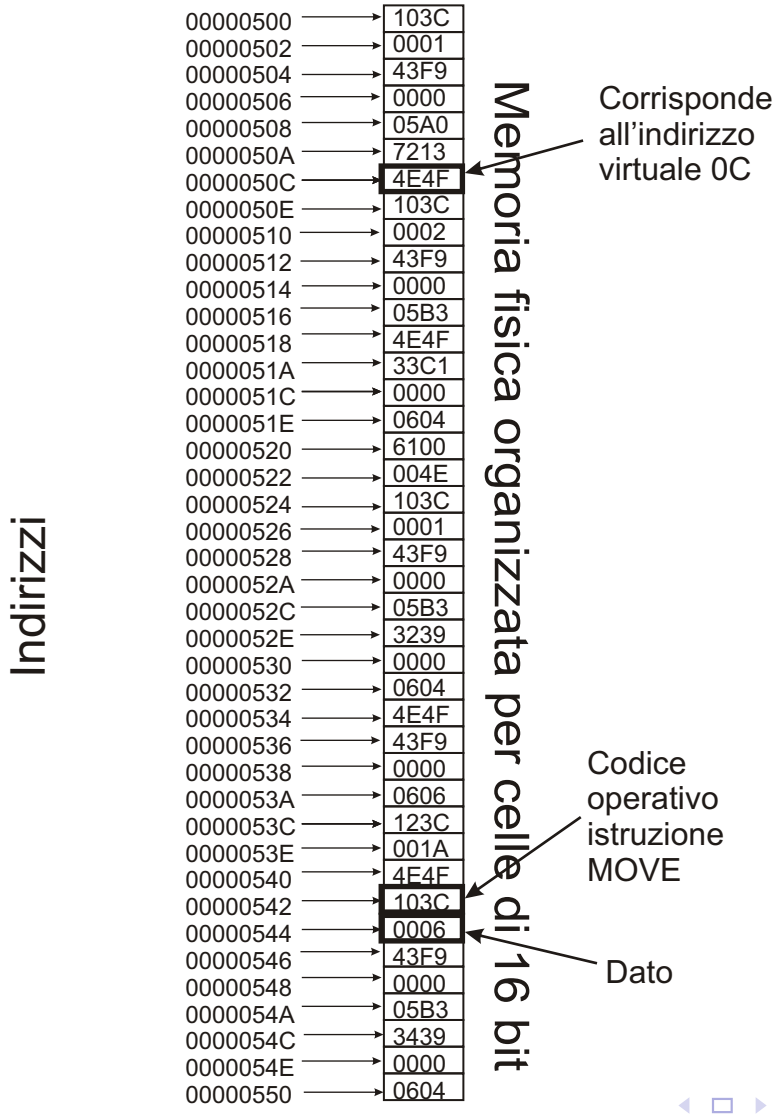
```
00000000                START:
00000000 103C0001               MOVE.B #1,D0    -  - - - - > - - - - - - - - - - - - -
00000004 43F9000000A0           LEA    M0,A1    |     00000070                UCASE:
0000000A 7213                   MOVEQ  #LENM0,D1|00000070 323900000104        MOVE    BLEN,D1
0000000C 4E4F                   TRAP   #15      |00000076 43F9000000B3        LEA     BUFF,A1
0000000E 103C0002               MOVE.B #2,D0    |0000007C 6000001C            BRA     ELOOP
00000012 43F9000000B3           LEA    BUFF,A1  |00000080 1419          LOOP: MOVE.B  (A1)+,D2
00000018 4E4F                   TRAP   #15      |00000082 0C020061            CMP.B   #$61,D2
0000001A 33C100000104           MOVE   D1,BLEN  |00000086 6D000012            BLT     ELOOP
00000020 6100004E               BSR    UCASE    |0000008A 0C02007A            CMP.B   #$7A,D2
00000024 103C0001               MOVE.B #1,D0    |0000008E 6E00000A            BGT     ELOOP
00000028 43F9000000B3           LEA    BUFF,A1  |00000092 04020020            SUB.B   #$20,D2
0000002E 323900000104           MOVE   BLEN,D1  |00000096 1342FFFF            MOVE.B  D2,-1(A1)
00000034 4E4F                   TRAP   #15      |0000009A 51C9FFE4      ELOOP:DBRA    D1,LOOP
00000036 43F900000106           LEA    M1,A1    |0000009E 4E75                RTS
0000003C 123C001A               MOVE.B #LENM1,D1|
00000040 4E4F                   TRAP   #15      |000000A0 576861742069 M0:    DC.B    'Scrivi una stringa: '
00000042 103C0006               MOVE.B #6,D0    |         7320796F7572
00000046 43F9000000B3           LEA    BUFF,A1  |         206E616D653A
0000004C 343900000104           MOVE   BLEN,D2  |         20
00000052 D2C2                   ADD    D2,A1    |         00000013     LENM0:EQU     *-M0
00000054 5341                   SUBQ   #1,D1    |000000B3 00000050     BUFF: DS.B    80
00000056 1221           RLOOP:MOVE.B -(A1),D1   |00000104 00000000           DS.W    0
00000058 4E4F                   TRAP   #15      |00000104 00000002     BLEN: DS.W    1
0000005A 51CAFFFA               DBRA   D2,RLOOP |00000106 2C20796F7572 M1:    DC.B    ', la stringa rovesciata
0000005E 7401                   MOVEQ  #1,D2    |         206E616D6520
00000060 7200                   MOVEQ  #0,D1    |         6261636B7761
00000062 103C0000               MOVE.B #0,D0    |         726473206973
00000066 4E4F           NLOOP:TRAP   #15        |         3A20
```

Situazione in memoria

Esempio di indirizzi virtuali: situazione in memoria, spazio di indirizzamento



Esempio di allocazione degli indirizzi virtuali dall'indirizzo $500: situazione in memoria



Caratteristiche fondamentali

Allocazione statica o dinamica

Tempo di gestione La gestione della memoria deve avvenire in run time e si divide in tempo di allocazione e tempo di rilascio.

Strutture dati Possono essere liste concatenate, tabelle o bitmap.

Frammentazione É relativa alla memoria utilizzata; definita come

$$f = \frac{\textit{memoria inutilizzata}}{\textit{memoria totale}} \cdot 100$$

Indirizzi virtuali e fisici Virtuali = spazio di indirizzamento del processo. Fisici = legati alla memoria fisica.

Caricamento (program linking) statico e dinamico Statico: librerie caricate assieme al codice; dinamico: librerie caricate in run time (Stub)

Rilocabilitá Esecuzione del processo indipendentemente dalla zona di memoria fisica dove viene caricato.

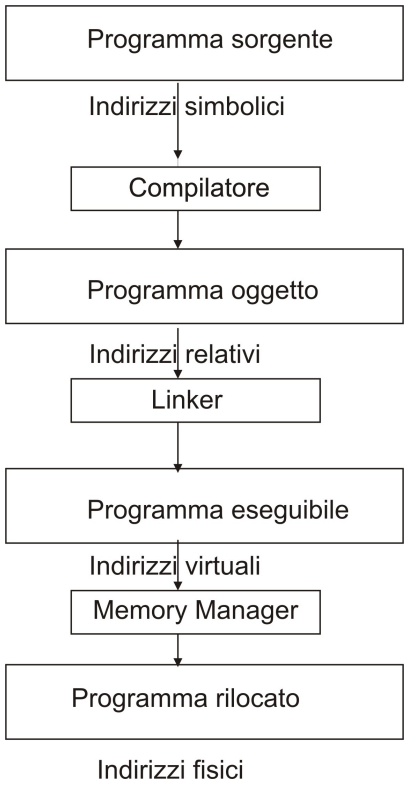
Livelli di memoria Legati al tempo di accesso effettivo T_a e al traffico sui bus.

$$T_a = T_{cache} \cdot h + (T_{cache} + T_m)(1 - h)$$

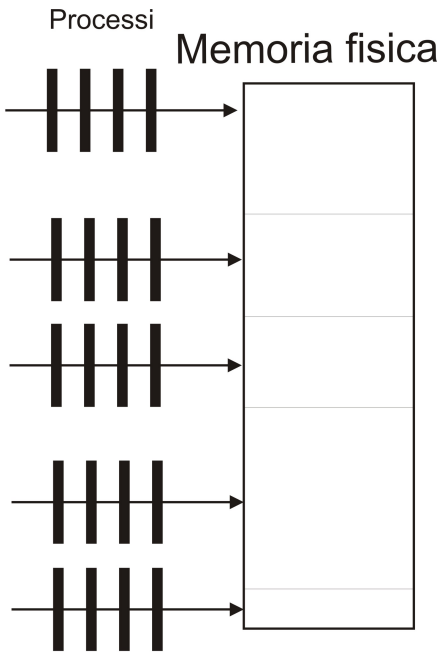
Localitá La probabilitá di richiedere una informazione in memoria é piú alta se le informazioni sono state richieste recentemente. Localitá spaziale e localitá temporale.

Indirizzi

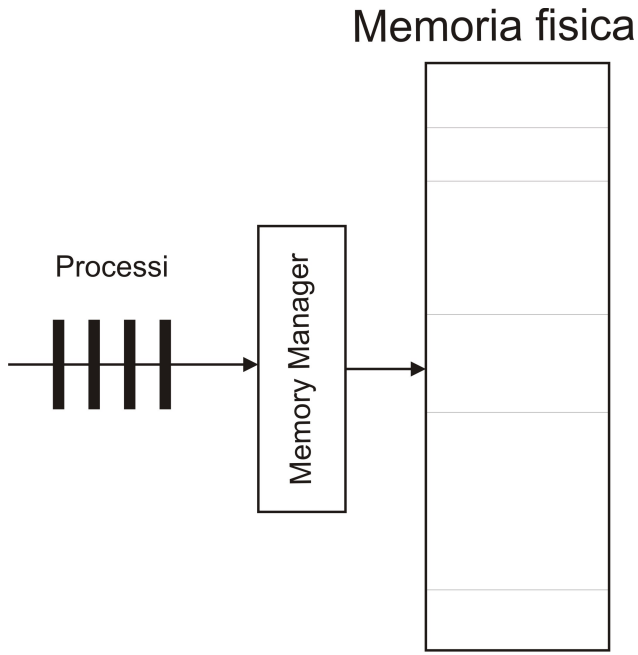
Gli indirizzi coinvolti durante l'esecuzione di un programma sono rappresentati nella seguente figura.



Memoria contigua



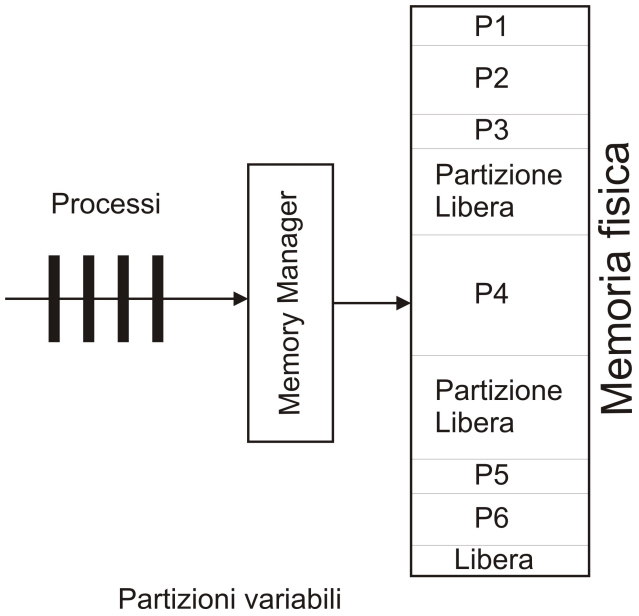
Partizioni fisse, code distinte



Partizioni fisse, coda unica

Problema: frammentazione interna

Memoria contigua

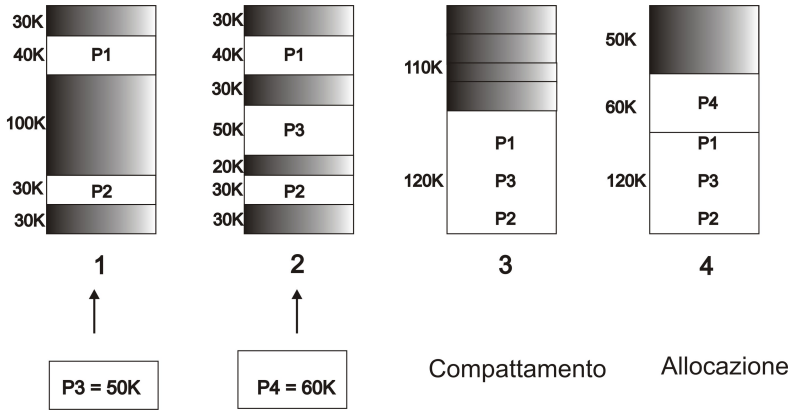


Partizioni variabili

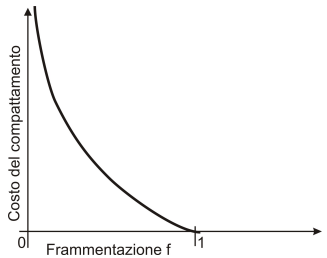
Problema: frammentazione esterma

Compattamento della memoria

Scopo del compattamento é di risolvere il problema della frammentazione



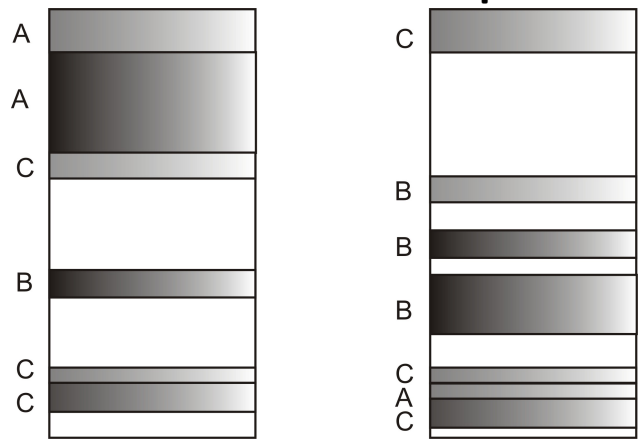
- Il costo del compattamento é duplice:
  - Quanto tempo blocco gli altri processi? é il tempo di compattamento
  - Compattamento contrastato dal riempimento: bisogna minimizzare il rapporto $\frac{TempoCompattamento}{TempoRiempimento}$. Forma $K\frac{1-f}{f}$ dove f é la frammentazione:



- In conclusione il compattamento conviene ad alte frammentazioni, quando ho molta memoria inutilizzata.

La regola del 50%

Il numero di partizioni libere é la metá delle partizioni occupate. Esempio:



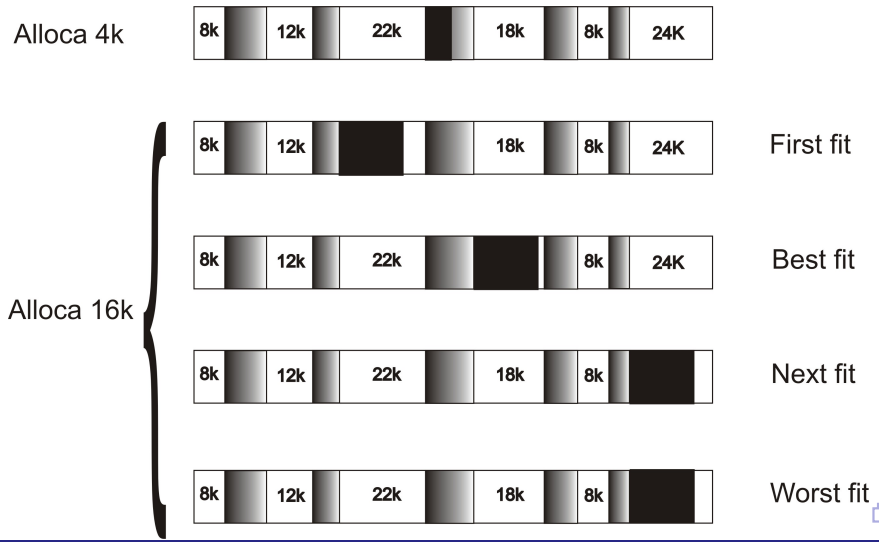
Una considerazione.

- Chiamando k il rapporto tra la dimensione media delle partizioni libere e quella delle partizioni occupate e D la dimensione media delle partizione occupate, cioé dei processi, la frammentazioneé $f = \frac{mem.libera}{mem.totale} = \frac{k}{k+2}$.
- Quindi: la frammentazione é una funzione crescente con k. Per ridurre la frammentazione, la **dimensione delle partizioni libere deve essere piccola rispetto a quella delle partizioni occupate**.
- La soluzione normalmente usata alla frammentazione sta nella fusione delle aree rilasciate

Costo del compattamento.
Il tempo del compattamento é $t_c = T_{rw}(1-f)M$ dove M é la dimensione totale di memoria, f la frammentazione e T_{rw} é il tempo di lettura e scrittra di 1 byte.
Regola del 50%.
N = numero partizioni occupate = $n_A + n_B + n_C$ dove n_A, n_B e n_C sono il numero di partizioni occupate di tipo A, B e C. Il tipo A aumenta il numero di partizioni libere quando viene disallocato, il tipo B diminuisce e il tipo C resta uguale.
Naturalmente a regime deve essere $n_A = n_B$, per cui N=numero partizioni occupate=$2n_B + n_C$.
Il numero di partizioni libere é M = numero partizioni libere = $\frac{n_c+2n_B}{2} + \xi$ dove ξ vale 0, 1 o 2 e serve per sistemare i conti. Se il numero delle partizioni é grande, allora M = $\frac{n_c+2n_B}{2}$.
La frammentazione é: $f = \frac{\text{memoria libera}}{\text{memoria totale}} = \frac{M*D_L}{M*D_L+N*D_O} = \frac{1}{1+\frac{ND_O}{MD_L}} = \frac{1}{1+2/k}$ dove D_O e D_L sono le dimensioni medie delle partizioni occupate e libere e $k = \frac{D_L}{D_O}$.

Algoritmi di allocazione

- First-Fit: il processo viene allocato nella prima partizione libera che puó contenerlo. Vantaggio: basso tempo di allocazione
- Best-Fit: il processo viene allocato nella partizione libera che lascia meno spazio inutilizzato. Minimo rapporto tra dimensione delle partizioni occupate e partizioni libere. Svantaggio: tempo di allocazione alto (bisogna esaminare tutte le partizioni libere).
- Next-Fit: il processo viene allocato nella prossima partizione libera. Vantaggio: tempo di allocazione
- Worst-Fit: il processo viene allocato nella partizione che lascia piú spazio libero. Due svantaggi: massimo rapporto partizioni occupate/partizioni libere e tempo alto

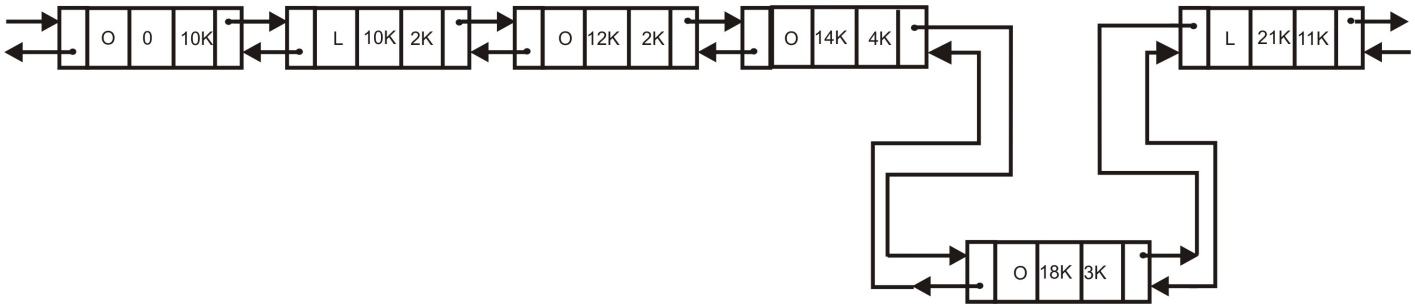


Allocazione a liste concatenate: due possibilitá

* lista delle partizioni libere/occupate
* lista delle sole partizioni libere Allocazione di un blocco di memoria con le liste

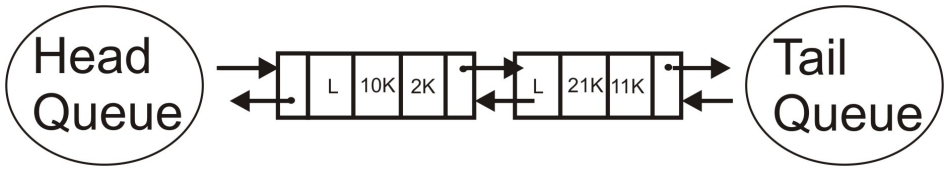
libere/occupate: ricerca della partizione libera, occupazione, restituzione della zona libera

* richiesta di 3K

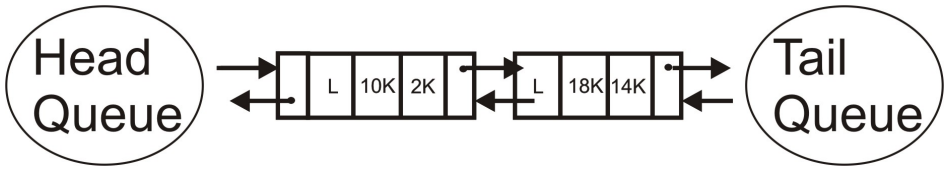


Implementazione

Allocazione di un blocco di memoria con le **sole liste libere**: ricerca della partizione libera, occupazione, modifica della dimensione della partizione libera



Dopo l'allocazione di un processo di 3K:

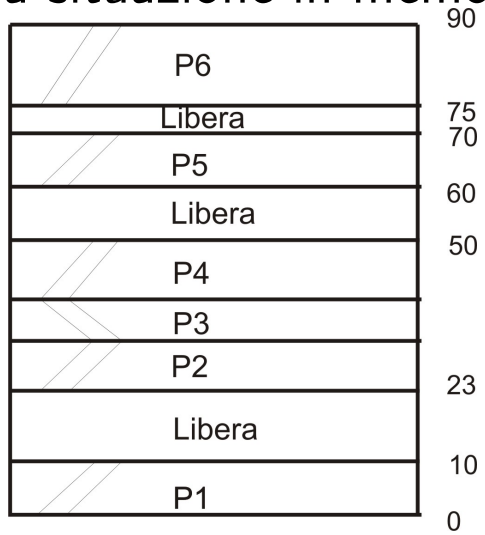


Gestione delle liste:

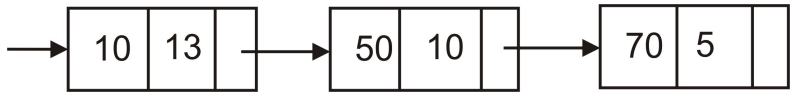
- liste ordinate secondo l'indirizzo iniziale della partizione libera
- liste ordinate secondo la dimensione crescente/decrescente delle partizioni libere

Esempio

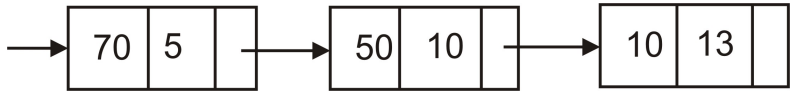
Una situazione in memoria



Corrispondente lista partizioni libere ordinate secondo indirizzo



Corrispondente lista partizioni libere ordinate secondo dimensione (FirstFit=BestFit)



Rilascio

- Quando termina un processo, viene inserita una partizione libera
- Dove inserirla? dipende dall'ordinamento
- Unione(fusione) delle zone libere:
  - Trova il punto di inserimento
  - Controlla se il blocco rilasciato puó essere unito con la partizione libera precedente
  - Controlla se il blocco rilasciato puó essere unito con la partizione libera seguente

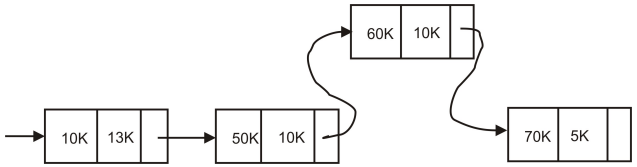
Tre scenari:

- Unione della partizione rilasciata con la partizione libera precedente
- Unione della partizione rilasciata con la partizione libera seguente
- Impossibilitá di fusione e inserimento di una nuova partizione libera

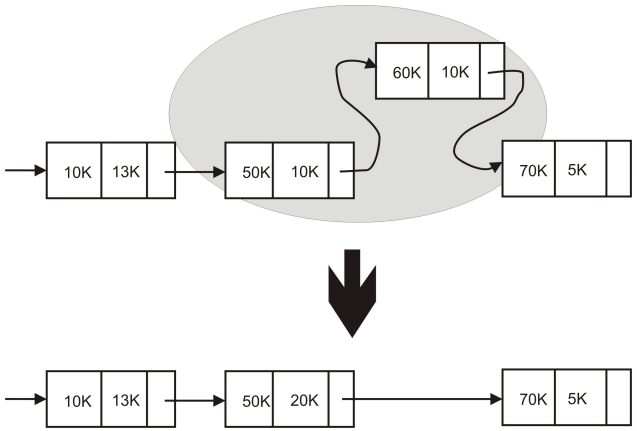
Esempio

Fine P5 $\rightarrow$ creo elemento con Stat=60K e Size=10K

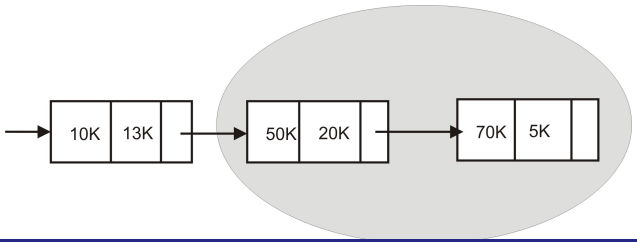
Inserimento nella lista



fusione con la precedente



fusione con la seguente



Considerazioni

Considerazioni finali:

- Il tempo di gestione memoria é **overhead**
  - Tempo di allocazione: minore con next fit
  - Tempo di allocazione: maggiore con first fit/worst fit
- Dove allocare le liste? nelle partizioni libere!
- Se la lista é ordinata per dimensione crescente: first fit equivale a best fit
- Se decrescente, first fit equivale a worst fit
- L'unione dei frammenti liberi richiede che la lista sia ordinata per indirizzo

Bitmap

- La memoria é divisa in blocchi di dimensione fissa
- Bitmap: matrice di bit, dove ogni bit rappresenta lo stato di un blocco di memoria
- Esempio: memoria di 10 MB, blocchi di 1 bit $\Rightarrow$ bitmap di 80Mbit cioé 10 MB
- memoria di 10 MB, blocchi di 1 byte $\Rightarrow$ bitmap di 10Mbit cioé 1.250 MB
- memoria di 10 MB. blocchi di 2 byte $\Rightarrow$ bitmap di 5Mbit cioé 625 Kbyte
- allocazione: vedo se esistono sufficienti bit a zero
- rilascio: azzeramento dei bit
- necessitá di una supporto hardware

Buddy systems

Memoria divisa in blocchi con dimensioni potenza di 2:

size	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768
nr.blk	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

- Una richiesta di memoria vá nel blocco PIÚ PICCOLO capace di contenerla.
- I blocchi sono raggruppati in coppie di compagni (buddies): 1MB $\rightarrow$ 2 buddies da 512K; ogni buddy da 512 genera 2 buddies da 256K etc.
- Una allocazione é un compagno, un rilascio consente subito di unire i compagni liberi.