Partizionamento della memoria

Index

Partizionamento

Uno dei primi metodi per la gestione della memoria è il **partizionamento**, ma è comunque utile per capire la memoria virtuale (la memoria virtuale è l'evoluzione moderna delle tecniche di partizionamento)

Tipi di partizionamento

- Partizionamento fisso
- Partizionamento dinamico
- Paginazione semplice
- Segmentazione semplice
- Paginazione con memoria virtuale
- Segmentazione con memoria virtuale

Partizionamento fisso uniforme

Con il partizionamento fisso, il SO, all'avvio, segmenta la memoria in partizioni di ugual lunghezza. Dunque in ognuna partizione posso mettere un processo che occupa al più la dimensione della partizione. Il sistema operativo può decidere di swappare un processo per toglierlo da un segmento (es. suspended)

Problemi

Quando un processo era troppo grande per la partizione, il programmatore doveva usare la tecnica dell'*overlays* (gestire esplicitamente lo swap) per fare in modo di non occupare più memoria di quella disponibile
Un altro problema sta nel fatto che la memoria viene utilizzata in modo inefficiente, infatti anche se un programma occupava meno memoria della dimensione della partizione, comunque gli veniva affidata una partizione intera (**frammentazione interna**)

Partizionamento fisso variabile

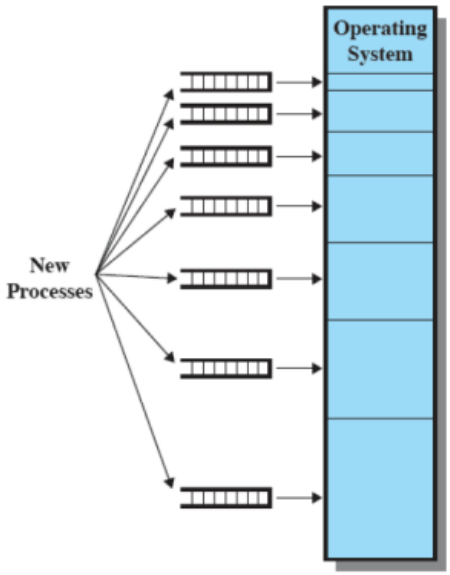
Nel partizionamento fisso variabile, così come nel partizionamento fisso, le partizioni vengono create all'inizializzazione del sistema operativo, ma a differenza di prima qui le partizioni non hanno tutte la stessa dimensione, mitigando quindi i problemi del

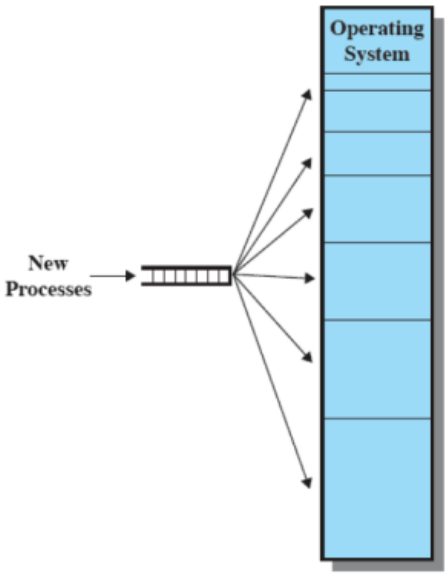
partizionamento precedente (senza però risolverli)

Algoritmo di posizionamento

Per capire in quale partizione posizionare un determinato processo è necessario un algoritmo di posizionamento. Un processo infatti, in questo caso, viene posizionato nella partizione più piccola che può contenerlo, minimizzando la quantità di spazio sprecato

Ciò ci pone però di fronte a due scelte: o utilizzo una coda per ogni partizione oppure utilizzo una singola coda per tutti i processi e solo alla fine decido in quale partizione posizionarlo





Problemi irrisolti

Seppur abbia mitigato i problemi del partizionamento precedente, lascia dei problemi irrisolti.
C’è un numero massimo di processi in memoria principale (corrispondente al numero di partizioni deciso inizialmente). Se ci sono molti processi piccoli, la memoria verrà usata in modo inefficiente (seppur questo problema è risolvibile utilizzando una sola coda per i processi e mettendo il processo in una partizione più grande nel caso in cui quella in cui dovrebbe andare è già occupata)

Posizionamento dinamico

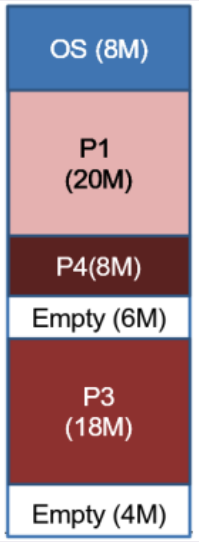
In questo tipo di partizionamento, le partizioni variano sia in misura che in quantità, allocando per ciascun processo esattamente la quantità di memoria che serve

Esempio >

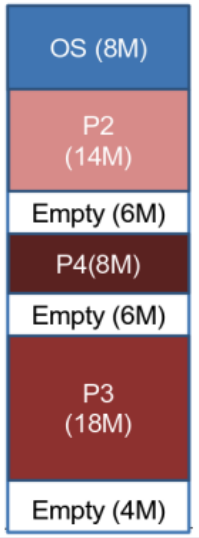
Esempio

Quando il sistema operativo viene inizializzata la memoria senza alcun processo al suo interno

In un certo momento arrivano 3 processi che lasciano 4M di memoria libera



Ad un certo momento arriva un processo P4 che richiede 8M, quindi il sistema operativo decide che per qualche motivo P4 è più importante di P2, che viene swappato per mettere al suo posto P4



Quindi è il momento di P1 di essere swappato su disco per mettere al suo posto P2

Se per caso arriva un processo da 8M questo non potrà essere inserito in quanto

la memoria a disposizione non è contigua, sarebbe quindi necessario rimuovere un altro processo per fargli spazio

Problemi

Qui a differenza dei precedenti, si ha un problema di **frammentazione esterna**, infatti i processi, quando terminano o vengono swappati, lasciano all'interno della memoria dei "frammenti" di memoria libera. Questo è risolvibile con la **compattazione**, con la quale il SO sposta tutti i processi in modo tale che siano contigui (però ha un alto overhead)

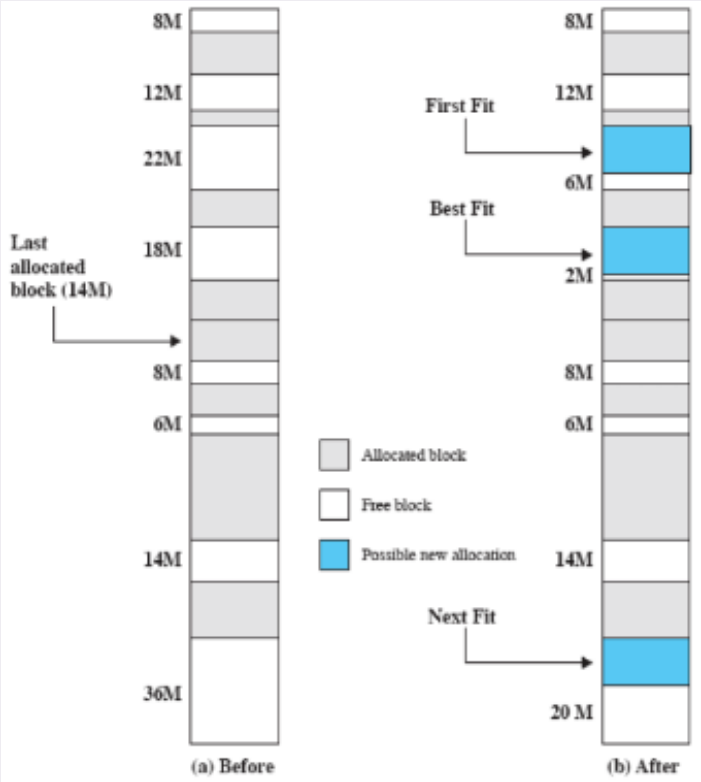
Il SO inoltre deve scegliere a quale blocco libero assegnare un processo. Ciò si può banalmente pensare possa avvenire, come per gli altri partizionamenti, tramite l'algoritmo *best-fit*: sceglie il blocco la cui misura è la più vicina (in eccesso) a quella del processo da posizionare. Ma questo algoritmo risulta essere quello con i risultati peggiori in quanto lascia frammenti molto piccoli che costringono a fare spesso la compattazione

Come alternativa a questo algoritmo vennero proposte due alternative:

- Algoritmo *first-fit* → scorre la memoria dall'inizio, il primo blocco con abbastanza memoria viene subito scelto e per questo motivo è molto veloce. Ha come problema il fatto che tende a riempire solo la prima parte della memoria, seppur è il **migliore** tra quelli proposti
- Agloritmo *next-fit* → come il first-fit, ma anziché partire da capo ogni volta, parte dall'ultima posizione assegnata ad un processo. Sperimentalmente si nota che assegna più spesso il blocco alla fine della memoria che tendenzialmente è il più grande

Algoritmi di posizionamento

Memoria prima e dopo l'allocazione di un blocco da 16M

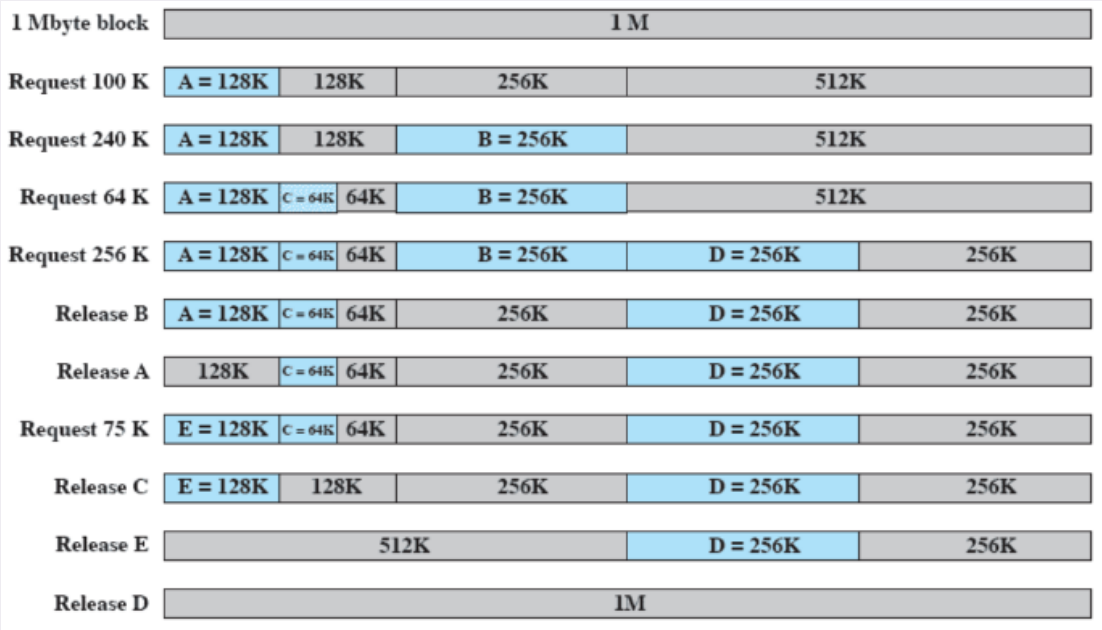


Buddy System (sistema compagno)

E' un compromesso tra il partizionamento fisso e il partizionamento dinamico: è un partizionamento dinamico nel senso he le partizioni si creano man mano che arrivano processi, ma fisso perché non possono essere create tutte le partizioni possibili ma occorre seguire uno schema ben definito

Sia 2^U la dimensione dello user space e s la dimensione di un processo da mettere in RAM. Quello che fa il buddy system è cominciare a dimezzare lo spazio fino a trovare un X t.c. $2^{X-1} < s \leq 2^X$ con $L \leq X \leq U$ e una delle due porzioni è usata per il processo (L serve per dare un lower bound per evitare che si creino partizioni troppo piccole)
Ovviamente, occorre tener presente le partizioni già occupate.
Quando un processo finisce, se il buddy è libero si può fare una fusione; la fusione può essere effettuata solo nel caso in cui è possibile costruire la partizione più grande ovvero 2^{X+1}

Example



Rappresentazione ad albero (quinta riga)