derevolmente l'overhead di CPU, in quanto quest'ultima dovrebbe effettuare spesso spostamenti di dati in memoria.

Una seconda alternativa consiste nell'effettuare la rilocazione solo nei casi in cui non risulti disponibile un'area libera sufficientemente grande per un nuovo lavoro. In questo caso il compattamento avverrebbe più raramente, ma la gestione delle tabelle sarebbe più complessa.

In ogni caso l'allocazione a partizioni dinamiche ha il vantaggio di rendere minima la frammentazione e di aumentare il grado di multiprogrammazione del sistema; contemporaneamente, però, presenta alcuni svantaggi, quali l'aumento di costo dell'hardware, il consumo di tempo per la compattazione e l'obbligo di far risiedere tutto lo spazio degli indirizzi di un programma nella memoria durante l'esecuzione.

PAGINAZIONE

La paginazione fornisce un'altra soluzione al problema della frammentazione e rappresenta una delle tecniche maggiormente utilizzate sugli attuali sistemi di elaborazione, in quanto permette di utilizzare memoria non contigua per contenere lo spazio degli indirizzi.

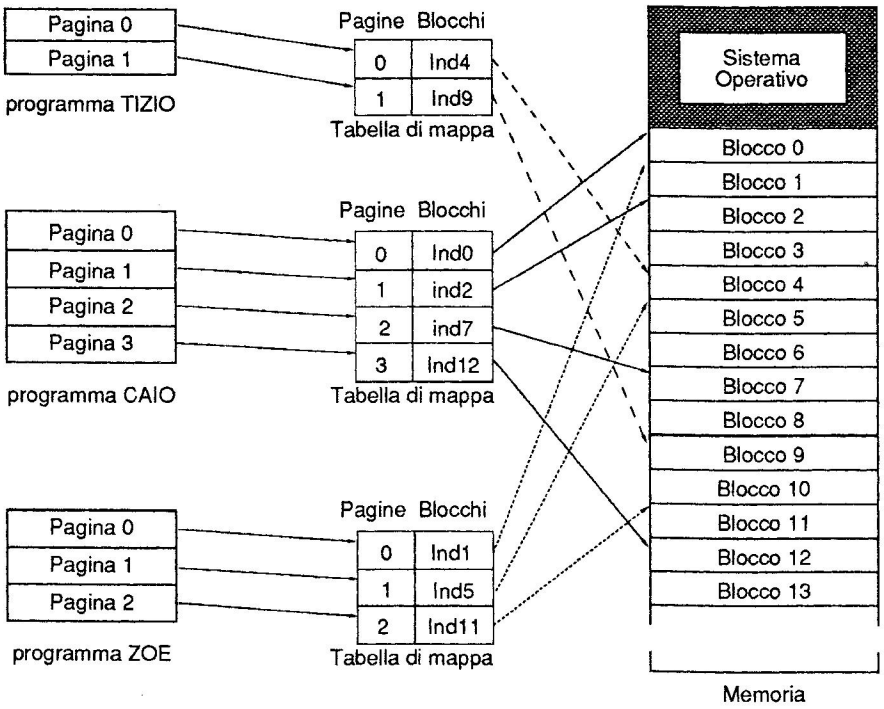
In pratica, lo spazio degli indirizzi di ogni programma viene suddiviso in parti uguali (*pagine*) e l'allocazione avviene ponendo in CORRISPONDENZA BIUNIVOCA le pagine che compongono il programma con i *blocchi* di memoria della medesima dimensione (*frames*), destinati a contenerle.

In questo modo, mentre le pagine del programma sono viste dall'utente logicamente contigue, i blocchi di memoria possono anche non esserlo e lo spazio degli indirizzi del programma si presenterà sparso nella memoria fisica: sarà compito dell'hardware preposto alla traduzione degli indirizzi trasformare gli indirizzi logici in quelli fisici sotto il completo controllo del Sistema Operativo.

In pratica si dovrà gestire la trasformazione tra le pagine dello spazio degli indirizzi e i blocchi di memoria, utilizzando opportuni supporti di memorizzazione (*tabelle di mappa di pagina*) che, partendo dall'*indirizzo base* di ogni pagina nella memoria fisica, consentano di gestire la trasformazione dell'indirizzo di pagina in indirizzo di blocco.

La figura illustra come è stata assegnata la memoria ai singoli programmi, reperendola di volta in volta tra le aree disponibili anche non contigue.

> Per esempio, il programma CAIO, lungo 14K, viene suddiviso automaticamente in 4 pagine di 4K, di cui l'ultima non completamente occupata.
> Al momento del caricamento in memoria, vengono reperiti 4 blocchi liberi in memoria (1, 5, 8, 11) e viene creata la tabella di associazione tra pagine e blocchi.



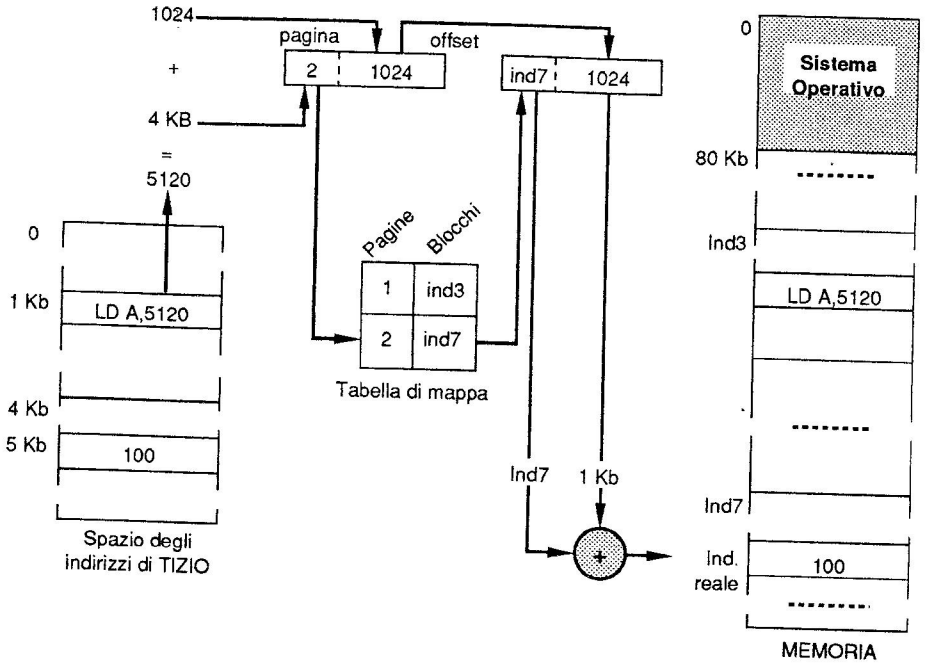
Come si può vedere, quindi, la tecnica di paginazione risolve quasi completamente il problema della frammentazione senza che avvenga alcun movimento fisico dei blocchi in memoria.
Infatti, se nella situazione configurata dovesse essere allocato un nuovo lavoro di 16K, sarebbe possibile associargli i blocchi liberi 2, 9, 10 e 13, senza dover per questo compattare i blocchi già in uso.

Naturalmente la dimensione delle pagine e dei blocchi deve essere fatta tenendo conto dei lavori da effettuare e della struttura hardware del sistema, per evitare, se la misura di una pagina è troppo grande, di ritrovarsi con gli stessi problemi delle partizioni fisse, oppure, se la misura è troppo ridotta, di dover gestire una grossa tabella di mappa.

Anche questo tipo di organizzazione richiede un meccanismo hardware, simile a quello usato per la rilocazione, per trasformare ogni indirizzo effettivo contenuto nel programma nel corrispondente indirizzo fisico: si può per esempio pensare di associare ad ogni pagina un apposito registro ad alta velocità (aumento dei costi), oppure riservare un'opportuna area di memoria per queste operazioni (diminuzione della velocità).

Ogni indirizzo generato dalla CPU è suddiviso in due parti: un NUMERO DI PAGINA e un OFFSET DI PAGINA. Il numero di pagina serve come indice per la tabella delle pagine e l'offset permette di individuare l'indirizzo fisico come distanza dal primo indirizzo della pagina individuata.

Per spiegare il meccanismo di trasformazione facciamo riferimento ad un esempio: supponiamo che nel programma TIZIO sia contenuta l'istruzione LD A, 5120 che fa riferimento ad una posizione contenuta nella seconda pagina dello spazio degli indirizzi.



L'indirizzo effettivo 5120 viene diviso per la dimensione della pagina (4*1024 = 4096) in modo da ottenere separatamente il numero della pagina e la distanza in byte della locazione dall'inizio della pagina stessa:

5120 : 4096 = 1 Resto 1024 = 1Kb

Attraverso il numero di pagina si individua, nella tabella di mappa, il corrispondente indirizzo della memoria fisica della pagina e, tramite l'offset si ricava l'indirizzo fisico della locazione in cui è contenuto il dato.

Ogni sistema operativo possiede propri metodi per memorizzare le tabelle di pagina, anche se, in generale, viene comunque allocata una tabella delle pagine per

ogni processo e un puntatore alla tabella delle pagine che, insieme al valore del registro istruzioni, viene memorizzato nel descrittore del processo.

Inoltre, per poter effettuare l'effettiva allocazione, il Sistema Operativo dovrà sempre avere a disposizione, oltre alle tabelle di mappa di pagina, una *tabella dei blocchi* (*tabella dei frame*) in cui, per ogni blocco fisico in memoria, viene memorizzato il suo stato (*libero* o *occupato*) e da quale pagina è eventualmente occupato.

Per concludere, possiamo notare che la tecnica di paginazione ha il vantaggio di eliminare la frammentazione senza richiedere molto tempo di CPU per lo spostamento fisico di blocchi di dati, permettendo così di aumentare il grado di multiprogrammazione e il rendimento generale del sistema.

Persistono però alcuni svantaggi, come l'elevato costo dell'hardware necessario alla trasformazione degli indirizzi, l'occupazione di memoria per le tabelle di mappa e dei blocchi e il consumo di tempo di CPU necessario per mantenere aggiornate le tabelle stesse.

Proposta di lavoro 2

Supponiamo di voler analizzare un sistema la cui memoria viene gestita con la tecnica della paginazione.

Costruire un programma di simulazione che, in base ad una lista di processi in ingresso, permetta di visualizzare, istante per istante, lo stato della memoria e l'assegnazione dei singoli blocchi alle diverse pagine dei processi.

Indicazioni per la soluzione

In questo caso potete provare a costruire il programma in modo che sia l'utente a decidere, istante per istante, se far giungere o no un nuovo processo e ad indicarne, in caso affermativo, i dati caratteristici.
Di conseguenza, il programma sarà formato sostanzialmente da un ciclo in cui per ogni processo in arrivo viene calcolato il numero di pagine necessarie per la sua allocazione in memoria, viene generata la tabella di mappa di pagina e viene verificata l'esistenza di un numero sufficiente di blocchi di memoria per effettuare l'allocazione.
In caso affermativo, deve essere completata la tabella di mappa di pagina e aggiornata quella di occupazione dei blocchi di memoria, mentre in caso negativo il processo dovrà essere posto in una coda di attesa.

Naturalmente, quando un processo ha raggiunto lo stato di terminazione, il programma di simulazione dovrà provvedere alla deallocazione dei blocchi di memoria aggiornando la tabella dei blocchi e rilasciando l'area di memoria occupata dalle tabelle di mappa di pagina associate al processo.

Alla chiusura di ogni ciclo di simulazione verrà posto il modulo di visualizzazione dei dati richiesti, con l'indicazione dell'ora.

LA PAGINAZIONE DINAMICA

In tutti gli schemi fin qui visti, un lavoro non può essere eseguito finché non si carica in memoria l'intero spazio degli indirizzi, anche se molto probabilmente alcune parti del programma non vengono quasi mai eseguite (per esempio le routine che vengono attivate solo in presenza di particolari errori).
Ciò vuol dire che, per poter gestire in multiprogrammazione lavori complessivamente molto grandi, occorre possedere una notevole quantità di memoria, anche se poi in realtà buona parte di essa rimane spesso inutilizzata.

La soluzione più idonea sarebbe quindi rappresentata da una gestione della memoria che produca l'illusione di poter avere a disposizione un'area estremamente vasta (*memoria virtuale*), pur utilizzando in realtà solo una memoria fisica di dimensioni limitate.

Una tecnica spesso usata per virtualizzare la memoria di un sistema è la *paginazione dinamica* (*demand paging*), che si basa sulla possibilità di mandare in esecuzione un lavoro, anche se in memoria viene caricata di volta in volta da disco solo la parte interessata a quella fase dell'esecuzione.

La tecnica di paginazione dinamica presenta i seguenti vantaggi:

- un programma non è più vincolato dalla quantità di memoria fisica disponibile e ciò consente agli utenti di scrivere programmi che utilizzano uno spazio degli indirizzi (indirizzi virtuali) con un'ampiezza che può anche superare i limiti fisici della memoria reale;
- può essere sensibilmente aumentato il rendimento del sistema (*throughput*) senza aumentare il tempo di risposta, poiché, dal momento che ogni processo utilizza meno memoria fisica, possono essere fatti avanzare contemporaneamente più programmi (*aumento del grado di multiprogrammazione*);
- sono necessarie meno operazioni di I/O per caricare o salvare un programma per cui l'esecuzione risulta più rapida.

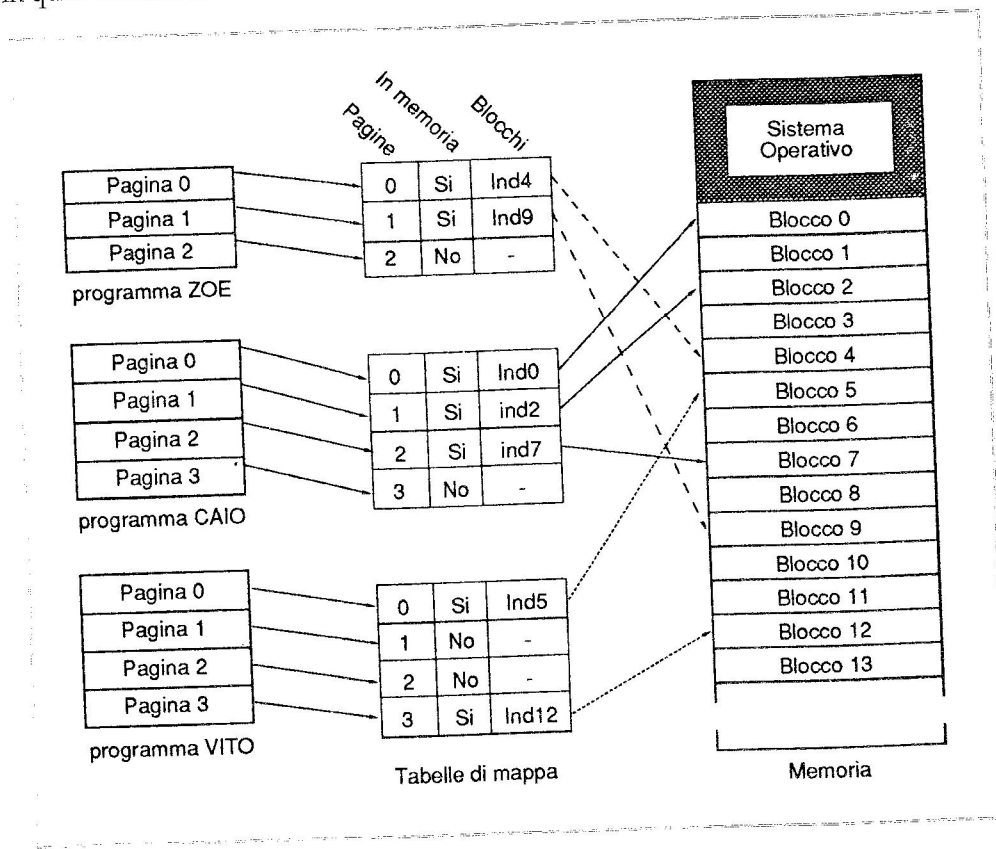
Tutto ciò è reso possibile dal fatto che :

① i programmi spesso contengono delle routine per la gestione degli errori eseguite solamente in casi rarissimi, per cui è inutile mantenerle sempre in memoria centrale;

② i programmi sono spesso organizzati "a menù" e l'esecuzione di un'opzione, in generale, non richiede che i blocchi di programma relativi alle altre scelte siano presenti in memoria;

- nei programmi sono spesso presenti parti indipendenti tra loro, la cui esecuzione non richiede la presenza in memoria del resto del programma (per esempio, la routine di input non richiede per l'esecuzione la presenza della routine di stampa);
- anche se il programma gestisce una grande tabella di dati, per la maggior parte delle operazioni ne usa solo un sottoinsieme (spesso ad un array viene riservata più memoria di quanta ne necessiti effettivamente) ed è quindi sufficiente mantenere in memoria la parte più usata, richiamando il resto solo all'occorrenza.

La paginazione dinamica utilizza le stesse tecniche della paginazione, basate sulla suddivisione del programma in pagine e della memoria in blocchi (*page frame*) della stessa dimensione, solo che in ogni istante non tutte le pagine hanno associato un reale blocco di memoria.
Anche in questo caso è necessaria una tabella di mappa di pagina che ponga in relazione pagine e blocchi di memoria e che consenta al sistema operativo, tramite un bit detto *bit di validità*, di sapere quali pagine sono realmente presenti in memoria e in quali blocchi sono memorizzate.



Resta ancora aperto il problema di come agire quando un lavoro deve accedere ad un indirizzo contenuto in una pagina non presente in memoria.

Quando, durante la trasformazione dell'indirizzo effettivo in un indirizzo fisico, si accede alla tabella di mappa per determinare il blocco corrispondente, poiché l'indicatore di stato segnala che la pagina non è allocata in memoria, viene generata un'interruzione (*interruzione per mancanza di pagina* - *page fault*) e viene richiesto al Sistema Operativo di caricare la pagina dal disco in memoria.

Se vi sono blocchi liberi, questa operazione consisterà solamente nell'aggiornamento della tabella di mappa e nel caricamento effettivo della pagina nel blocco prescelto, mentre quando non vi sono blocchi disponibili, occorrerà scegliere tra le pagine presenti in memoria quale trasferire sul disco, per creare spazio libero in memoria centrale.

Le funzioni del gestore della memoria

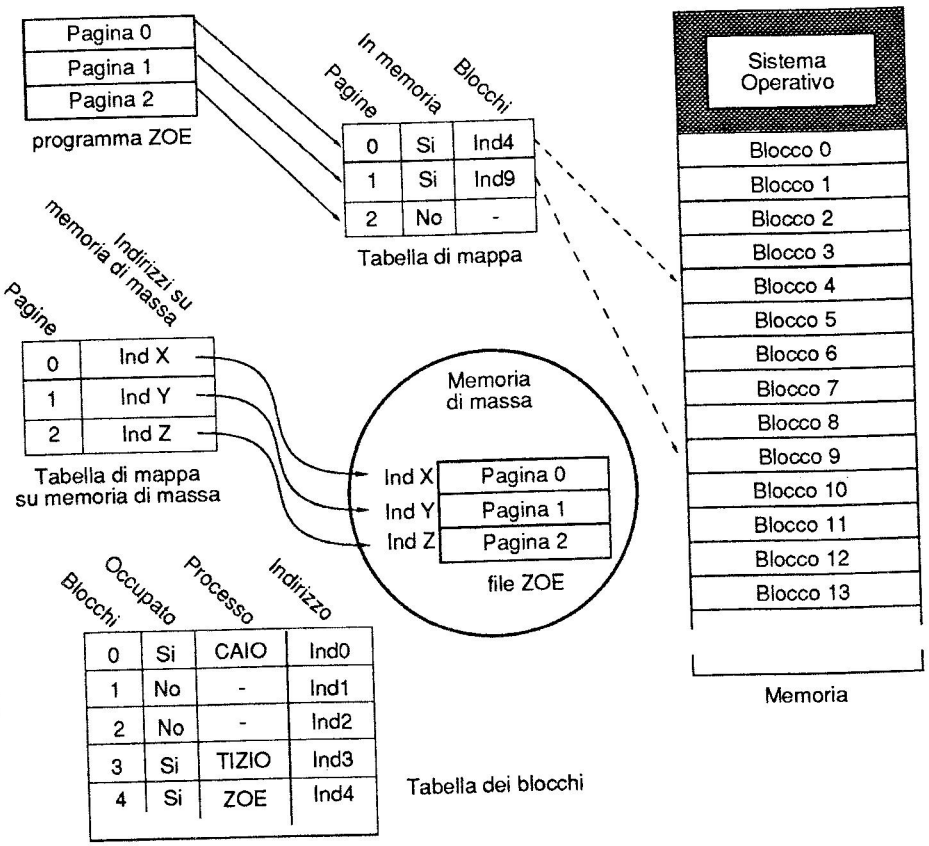
Con questo tipo di organizzazione il gestore della memoria diviene più complesso, ma garantisce una maggiore flessibilità.

In particolare esso esplica quattro funzioni fondamentali:

1. mantenere aggiornato lo stato dei lavori e della memoria attraverso tre ti-pi di tabelle:
   - TABELLA DI MAPPA DI PAGINA PER OGNI SPAZIO DI INDIRIZZI *(o tavole indirizzi)*, che contiene le informazioni necessarie per verificare se ad una pagina è assegnato un blocco (pagina residente in memoria) e quale, oppure se la stessa risiede su memoria di massa,
   - TABELLA DEI BLOCCHI, UNICA PER L'INTERO SISTEMA *(tavola sistema)*, che, per ogni blocco, contiene informazioni sul suo stato (libero o occupato), l'eventuale lavoro a cui è associato, l'eventuale pagina che vi è memorizzata, l'indirizzo di partenza e così via,
   - TABELLA DI MAPPA SU DISCO PER OGNI LAVORO *(tavole lavori)*, che, per ogni pagina, contiene le informazioni necessarie per il suo reperimento in memoria di massa,
2. determinare, in base ad un'opportuna politica di gestione, a quale lavoro assegnare blocchi, quante pagine caricare in memoria e, all'occorrenza, a quale lavoro devono essere tolti dei blocchi di memoria e quindi quali pagine devono essere così trasferite su memoria di massa;
3. trasferire le pagine nei blocchi, aggiornando opportunamente le tabelle;
4. liberare i blocchi di memoria, trasferendo su memoria di massa, quando necessario, le pagine in essi contenute.

Inoltre, dato che ogni lavoro risiede in parte in memoria centrale ed in parte su memoria di massa ed i trasferimenti dall'una all'altra sono molto frequenti, il gestore della memoria dovrà sempre operare in stretta collaborazione con il gestore delle periferiche e con quello delle informazioni.

Diamo ora uno schema generale del collegamento tra memoria centrale e memoria di massa ottenuto attraverso le tre tabelle fondamentali.



Le politiche di gestione

La paginazione dinamica ha indubbiamente l'effetto di appesantire l'esecuzione dei processi, infatti ogni volta che avviene la richiesta di accesso ad una pagina non presente in memoria si innesca una successione di operazioni la cui esecuzione non è affatto trascurabile per il sistema:

- viene lanciata un'interruzione al Sistema Operativo (*trap*);

viene salvato lo stato del processo P che aveva fatto la richiesta e che viene posto in stato di attesa;

viene riconosciuto dal sistema il tipo di interruzione;

viene effettuato un controllo sulla correttezza del riferimento alla pagina e viene determinata la locazione della pagina su disco;

viene effettuato un trasferimento dati da memoria di massa al blocco libero (ammesso che ve ne sia uno libero) e ciò implica:

- un'attesa nella coda associata al dispositivo di I/O fino a che la richiesta non è stata esaudita,
- un'attesa relativa al tempo di ricerca e/o latenza del dispositivo,
- un tempo di effettivo trasferimento tramite DMA dei dati da memoria di massa a memoria centrale (durante l'attesa, la CPU può essere assegnata ad altro processo P2);

viene corretta la tabella delle pagine e dei blocchi per segnalare al gestore della memoria che la pagina è ora presente in memoria;

viene riportato in stato di pronto il processo P;

si deve attendere che la CPU venga nuovamente assegnata al processo P1;

viene ripresa l'esecuzione del programma associato a P.

Da tutto ciò, appare evidente l'importanza di tenere bassa la frequenza delle richieste di pagina per non rallentare eccessivamente l'esecuzione del processo, cosa che può essere ottenuta effettuando in modo oculato tre tipologie di scelte:

determinare, di volta in volta, quale lavoro servire, in base alle strategie già viste nel capitolo precedente;

determinare quante e quali pagine devono essere caricate in memoria per avviare l'esecuzione.
Nell'effettuare questa scelta, il gestore potrà assegnare ad ogni processo un numero di blocchi pari a quelli disponibili in memoria divisi per il numero dei processi allocati (ALLOCAZIONE UGUALE), oppure assegnare i blocchi in proporzione alla dimensione dei programmi (ALLOCAZIONE PROPORZIONALE), o in rapporto al grado di priorità di un processo (ALLOCAZIONE A PRIORITA') o ancora nessun blocco ed effettuare l'allocazione quando necessario (DEMAND PAGING);

individuare il "miglior" blocco da liberare per poter servire la richiesta di pagina di un lavoro quando non vi sia più memoria disponibile (SOSTITUZIONE DI PAGINA).

Metodi di sostituzione delle pagine

Esistono due tipi di politiche per gestire la sostituzione di pagina: una politica ed una politica
Le politiche globali presuppongono che la scelta della pagina da eliminare possa avvenire tra tutte le pagine residenti in memoria, senza distinzione tra i processi, mentre quelle locali, per determinare la pagina da eliminare, si limitano alle pagine residenti di quel processo.

Ma le politiche globali non sono facilmente applicabili; per esempio un page fault su un processo si ripercuoterebbe anche su altri, e quindi vengono di fatto utilizzate il meno possibile.

Rimane allora da determinare quale pagina estrarre tra quelle associate ad un processo qualora si verifichi un page fault.

L'ideale sarebbe poter scegliere quella pagina che non servirà più, ma ciò presupporrebbe una conoscenza del comportamento "futuro" del programma, che risulta praticamente impossibile; ci si deve perciò accontentare di strategie meno soddisfacenti ma applicabili, che basano la loro previsione sul calcolo probabilistico e sull'uso "passato" delle pagine residenti in memoria.

Vi sono molti modi di gestire questo tipo di problema e tra questi i più significativi si basano sul , per il quale (dopo aver fatto riferimento ad una locazione di memoria successivamente farà accessi a locazioni vicine) (dopo aver fatto riferimento ad una locazione di memoria in tempi ravvicinati farà ancora riferimento a quella locazione di memoria).

Algoritmo FIFO (*First In First Out*)

L'idea che sta alla base dell'algoritmo è quella di rimuovere ogni volta la pagina che risiede in memoria da più tempo.
L'implementazione di un algoritmo di questo tipo è piuttosto semplice e richiede che ad ogni pagina sia associato un contatore in cui registrarne il tempo di permanenza in memoria della pagina oppure che le pagine del processo siano organizzate in una coda, in modo che la prima ad essere stata caricata in memoria sia anche la prima ad uscire.

La tecnica FIFO ha sicuramente il vantaggio di proteggere dalla rimozione una pagina appena caricata e non ancora pienamente utilizzata, ma, per contro, non tiene conto di una considerazione piuttosto elementare: se una pagina è utilizzata molto spesso, facilmente diverrà quella con maggior tempo di permanenza in memoria e a quel punto verrà inesorabilmente rimossa, anche se subito dopo dovrà essere nuovamente richiamata.
Si può inoltre verificare una particolare situazione ( ) per cui, all'aumentare dei blocchi a disposizione del processo, contrariamente a quanto ci si

aspetterebbe, il numero di page fault aumenta e di conseguenza conviene ricorrere ad altre strategie.

Algoritmo LRU (*Least Recently Used*)

Con questa strategia, che si basa sul principio di località temporale, viene sostituita la pagina in memoria centrale che non è stata usata per il periodo di tempo più lungo, supponendo che, se una pagina non è usata da molto tempo, probabilmente non verrà più riutilizzata.
Occorre quindi associare ad ogni pagina l'ora in cui è stata utilizzata l'ultima volta ed effettuare la ricerca all'indietro nel tempo.

La politica LRU viene considerata abbastanza valida, ma può richiedere hardware aggiuntivo per la gestione della sostituzione (ad ogni elemento della tabella delle pagine viene aggiunto un registro che contiene l'ora di utilizzo e alla CPU un contatore o un clock logico).

Un soluzione alternativa, gestibile via software, consiste nell'organizzare in una pila i descrittori delle pagine usate, con l'ultima pagina in cima e la LRU in fondo: prevedendo un puntatore al fondo della pila è facilmente eliminabile la pagina più vecchia.

Algoritmo LFU (*Least Frequently Used*)

Con questa politica, basata sul principio di località spaziale, viene sostituita la pagina usata meno frequentemente.
Risulta quindi necessario associare ad ogni pagina un contatore di accessi e ricercare tra le pagine associate ad un processo quella con il conteggio più basso.

Il difetto di questo algoritmo è rappresentato dai casi in cui una pagina viene utilizzata molto intensamente per un certo periodo e poi non viene più usata. Il numero globale di accessi, in questo caso, infatti, rimarrà alto e la pagina non verrà sostituita anche se inutile in memoria.

Algoritmo NUR (*Not Used Recently*)

Con questa strategia la scelta della pagina da sostituire viene ancora fatta tra quelle poco usate, ma dando la precedenza a quelle che non hanno subito modifiche, in modo da rendere inutile la fase di aggiornamento su disco.

Questo algoritmo utilizza, per effettuare la scelta, due tipi di segnalatori: il primo specifica, per ogni pagina logica, se la stessa è già stata utilizzata oppure no (
), mentre il secondo specifica, per le pagine usate, se durante l'accesso sono state modificate le informazioni in essa contenute (
).