**Memoria**

La gestione della memoria si occupa di allocare la memoria fisica ai processi che ne fanno richiesta. Nessun processo può diventare attivo prima di avere ottenuto un certo quantitativo di memoria. L’utilizzo globale delle risorse e tutte le altre prestazioni di un calcolatore vengono influenzate dalle prestazioni del modulo di gestione della memoria, sia in funzione della sua efficienza nell’allocare memoria, sia per l’influenza che può avere sullo scheduler. Un buon gestore di memoria in un ambiente mutliprogrammato deve garantire la protezione dei dati, impedendo ai processi attivi di sconfinare nello spazio di indirizzamento di altri processi, e permettere allo stesso tempo la loro condivisione affinché i processi cooperanti possano accedere ad aree comuni di memoria. I moduli di gestione della memoria si differenziano in base al tipo di allocazione della stessa, che può essere: **contigua** o **non** **contigua**.

**Allocazione contigua**

La memoria viene allocata in modo tale che ciascun oggetto occupa un insieme di locazioni i cui indirizzi sono strettamente consecutivi. Esistono vari tipi di allocazione contigua:

* Monoallocazione
* Partizionamento statico
* Partizionamento dinamico
* Segmentazione

La **MONOALLOCAZIONE** è lo schema più semplice per la gestione della memoria che viene suddivisa in due aree contigue:

* La prima allocata permanentemente ad una porzione residente del sistema operativo
* La seconda assegnata ai processi in esecuzione, processi utente o porzioni non residenti del sistema operativo, per il tempo necessario al completamento

Il sistema operativo compie diverse funzioni:

* Tiene traccia solo della prima e dell’ultima locazione disponibile per i processi transienti
* Viene posto ad uno dei due estremi della memoria
* Ingloba i vettori di interrupt
* Altri suoi moduli temporanei vengono posti all’altra estremità della memoria

Alla richiesta di esecuzione di un programma**, il sistema operativo** si **assicura** **che** **le** **dimensioni** del processo siano compatibili con la memoria disponibile, **conferisce** **il** **controllo** al processo utente fino al suo completamento o ad eventuali condizioni di errore, **al** **termine** del processo la memoria viene liberata e può essere assegnata ad un altro processo in attesa.

Raramente un monitor monoprocesso supporta meccanismi di protezione tra processi utente in quanto in ogni istante vi può essere al massimo un solo processo residente in memoria. Gli eventuali meccanismi si riferiscono alla protezione del codice del sistema operativo da eventuali sconfinamenti del processo transiente in esecuzione. La protezione del sistema operativo dai processi utente è spesso effettuata mediante supporti hardware come:

* Registri barriera
* Diritti di accesso mediante bit di protezione
* Sistema operativo in memoria a solo lettura

**I registri barriera** sono usati per tracciare un confine tra aree dei processi di sistema e dei processi transienti. Nel registro viene memorizzata la prima locazione disponibile al processo e il sistema operativo effettua i controlli di sconfinamento. Questo metodo richiede la capacità di distinguere l’esecuzione del sistema operativo da quella dei processi utente definendo gli stati utente e supervisore.

Un altro metodo per progettare la memoria consiste nel memorizzare **i diritti di accesso**, mediante bit di protezione, nella memoria stessa. Ad ogni parola di memoria viene associato un bit di protezione che viene posto a 1 nelle zone che contengono il sistema operativo e a 0 nelle restanti. I processi utente accedono solo a parole con bit 0, mentre il sistema operativo ha accesso illimitato. Un modo semplice e diffuso per proteggere il codice del sistema operativo dai programmi utente consiste nel porre **il sistema operativo in memoria a sola lettura**. Questo sistema non è solitamente utilizzato per la sua scarsa flessibilità e impossibilità di correggere o aggiornare il codice del sistema operativo.

**La condivisione** del codice e dei dati in memoria non ha molto senso negli ambienti monoprocesso e raramente viene supportata dai relativi monitor. I programmi utente potrebbero passarsi dei dati, mediante accordi interni, ponendoli in locazioni di memoria che non vengono sovrascritte durante l’esecuzione di processi partecipanti, o mediante file.

Quindi in monoallocazione la mancanza di supporti per la multiprogrammazione **produce un abbassamento dell’efficienza** sia della memoria che della CPU, la memoria può risultare **sovradimensionata** per la maggioranza dei processi, processi **di dimensione più grande** della memoria non possono essere eseguiti, oppure richiedono particolari suddivisioni del codice, i programmi tendono ad essere ottimizzati rispetto alla dimensione comportando spesso sacrifici di funzionalità e velocità. Alcuni pregi della monoallocazione sono: **basso costo di progettazione** del modulo della gestione della memoria, **contenuti e supporti hardware specifici** per la gestione della memoria.

Il **PARTIZIONAMENTO STATICO** è un metodo per supportare la multiprogrammazione e consiste nel dividere la memoria in diverse partizioni, ciascuna delle quali può essere allocata in diverse partizioni, ciascuna delle quali può essere allocata a un processo diverso. Il partizionamento statico implica che la suddivisione della memoria venga fatta fuori linea e che da quel momento le partizioni rimangono di dimensioni fisse. Il numero e la dimensione di ciascuna partizione vengono determinati durante la generazione del sistema considerando:

* **Capacità della memoria fisica disponibile**
* **Grado desiderato di multiprogrammazione**
* **Dimensione dei processi più frequentemente eseguiti**

Il sistema operativo deve tener traccia delle partizioni definite. Lo stato corrente delle partizioni ed i loro attributi vengono raccolti in una struttura dati chiamata tabella di descrizioni delle partizioni (TDP). L’unico campo variabile della TDP è lo stato della partizione che indica se la partizione è allocata o no. Gli altri campi contengono valori definiti al momento del partizionamento. Quando un processo non residente deve essere creato o attivato, il sistema operativo tenta di allocare una partizione di memoria libera di dimensioni sufficienti per contenerlo, consultando la TDP. Se il risultato è positivo, il campo stato della partizione viene impostato ad allocata e l’immagine del processo viene caricata nella corrispondente partizione. Quando il processo termina è necessario effettuare uno swapping della memoria. Questa informazione verrà utilizzata per aggiornare la partizione associata al valore libera. I metodi più comuni per la ricerca di una partizione libera sono:

* **First fit**: consiste nell’allocare la prima partizione libera sufficientemente larga per contenere il processo
* **Best fit:** consiste nell’allocare la più piccola delle partizioni libere che contenga il processo

**Il metodo first fit** è il più veloce in esecuzione poiché si arresta, nella ricerca, alla prima partizione libera disponibile, mentre **il metodo best fit** deve controllare, prima di arrestarsi, tutte le partizioni libere disponibili per individuarne la migliore. Il metodo best fit tuttavia è più accurato in esecuzione e quindi può raggiungere un miglior utilizzo della memoria, rispetto al metodo first fit, riducendo al minimo la differenza di dimensione tra il processo e la partizione allocata. Quando però il numero delle partizioni è molto grande nessuno dei due metodi si rivela particolarmente adatto.

Le cause di mancata disponibilità di una partizione sono:

1. **Non vi sono partizioni di memoria** sufficientemente grandi per contenere il processo entrante
2. **Tutte** le partizioni sono **allocate**
3. **Alcune** partizioni sono **libere** ma nessuna di esse è sufficientemente grande

Il primo caso è dovuto a un errore di scelta della dimensione delle partizioni o a un processo eccezionalmente più grande della norma. Si può ridefinire la dimensione delle partizioni o dividere il programma in origine in più sottoprogrammi.

Il secondo caso si può risolvere mediante l’attesa di una partizione libera o mediante lo swapping di un processo. Le stesse soluzioni si possono adottare per il caso 3.

Nel caso 3 l’assegnazione di partizione può continuare:

* Vengono allocate partizioni a processi più piccoli
* L’utilizzo della memoria viene mantenuto più elevato
* Si può violare l’ordinamento di attivazione dei processi dallo scheduler modificando le prestazioni del sistema

Il funzionamento del **gestore della memoria** e dello **scheduler** sono strettamente correlati:

* Lo scheduler determina quali processi sono pronti e quindi residenti nella memoria in attesa della CPU
* Il gestore della memoria può decidere di sfrattare un processo in memoria per liberare una partizione

**Lo swapping** è l’operazione di rimozione dalla memoria di un processo sospeso ed il suo successivo caricamento. Nel partizionamento statico questo aumento il livello di occupazione centrale e conseguentemente il livello di utilizzo della CPU. Quando lo scheduler decide l’introduzione in memoria di un nuovo processo il gestore dello swapping ne sceglie uno da rimuovere in base a:

* **Dimensione della partizione richiesta**
* **Priorità del processo**
* **Tipo di evento atteso dal processo**
* **Tempo trascorso in memoria**

Un processo residente in memoria già parzialmente eseguito è costituito da:

* Codice eseguibile
* Dati finora elaborati
* Stack
* Registri di stato
* File aperti
* Descrittore del processo

Al momento dello swapping questi oggetti vengono registrati in un file su hard disk detto file di swapping. Tutti i processi che hanno subito uno swapping possono essere mantenuti in un file unico per tutto il sistema o in file separati, ciascuno associato ad un singolo processo. Un file di swapping di sistema viene solitamente creato al momento dell’inizializzazione del sistema e collocato su una periferica veloce di memorizzazione secondaria. L’indirizzo e la dimensione di tale file sono solitamente statici in modo da beneficiare di un indirizzamento diretto su disco. Un parametro critico per la realizzazione di un file di swapping è la dimensione: se troppo grande spreca spazio su dischi veloci, se troppo piccolo può rendere indispensabile l’operazione di swapping. L’alternativa è disporre di un file di swapping per ciascun processo presente nel sistema che po' essere creato dinamicamente al momento della creazione del processo o staticamente al momento della preparazione del programma. Uno swapping efficiente richiede che i processi siano **riallocabili dinamicamente**:

* Il processo può iniziare l’esecuzione in qualunque area di memoria
* Può essere spostato in un’altra area di memoria
* Il calcolo degli indirizzi viene effettuato dinamicamente ad esempio con l’uso di un registro base

**Processi non rilocabili dinamicamente** sono legati alle partizioni in cui iniziano l’esecuzione e rendono inefficiente lo swapping. In linea di principio lo swapping va utilizzato solo in casi di necessità poiché la CPU spreca un tempo elevato nelle operazioni di trasferimento.

**L’integrità di un sistema mutliprogrammato** dipende anche dalla sua capacità di garantire l’isolamento tra spazi di indirizzi separati. Non solo si deve proteggere il sistema operativo dai processi utente ma bisogna anche proteggere i processi utente tra di loro perché nessuno possa invadere lo spazio riservato all’altro. Un supporto di protezione efficace non può prescindere dall’esistenza di un supporto hardware. Nei sistemi che utilizzano per la rilocazione i registri base, vengono di solito usati dei registri limite per la protezione. Il compito base di un registro limite consiste nell’individuare i tentativi di accesso otre i limiti dello spazio di indirizzamento assegnato al programma in esecuzione dal sistema operativo. Il registro limite contiene solitamente il valore dell’indirizzo virtuale più alto dal programma. Ad ogni accesso alla memoria la CPU controlla che il programma non richieda indirizzi oltre il limite. Le violazioni vengono rilevate e riferite al sistema operativo che le gestisce. L’indirizzo più basso del processo è contenuto nel registro base.

Un buon sistema di gestione della memoria deve occuparsi anche della condivisione controllata di dati e codice tra processi cooperanti. I sistemi a partizioni fisse non sono adatti alla condivisione poiché basano i propri meccanismi sull’isolamento degli spazi di indirizzamento. Alcuni metodi di condivisione sono:

1. **Affidare gli oggetti condivisi al sistema operativo** che così è in grado di accedere a tutte le risorse. Gli svantaggi sono che l’area del sistema operativo dovrebbe poter variare dinamicamente e la routine dei processi utente devono poter essere linkate al sistema operativo dinamicamente.
2. **Ogni processo possiede una copia identica** dell’oggetto condiviso, che usa e diffonde gli aggiornamenti. Lo svantaggio di questa tecnica consiste nel fatto che se il sistema supporta lo swapping, uno o più processi potrebbero non essere in memoria e quindi non essere pronti a ricevere gli aggiornamenti
3. **Collocare i dati i una partizione comune dedicata**. In questo caso però, il sistema operativo considera come violazione i tentativi dei processi partecipanti alla condivisione di accedere a zone di memoria esterne alle rispettive partizioni. Se il sistema usa registri base e limite sono necessari accorgimenti per indirizzare partizioni che potrebbero non essere contigue.

Nel **PARTIZIONAMENTO DINAMICO** il sistema delle partizioni è analogo a quello del partizionamento statico. Il gestore della memoria crea e assegna partizioni di dimensione variabile dinamicamente in base alle richieste dei processi. Quando un processo termina o subisce uno swapping, il gestore della memoria restituisce lo spazio liberato all’insieme di aree di memoria libere. Il gestore di memoria può creare e allocare partizioni finché non viene esaurita la memoria fisica o finché non viene raggiunto il massimo grado di multiprogrammazione. Alla richiesta di una partizione il gestore della memoria (GM) ricerca una zona di memoria libera contigua di dimensione sufficiente. Se si trova un’area adatta il sistema operativo vi ricava una partizione in modo da soddisfare esattamente le necessità del processo. L’eventuale parte rimanente di memoria libera viene restituita all’insieme della memoria libera e posta a disposizione del modulo di allocazione. La partizione viene creata registrando la sua base, la sua dimensione e il suo stato nella TDP o in una tabella equivalente. Alcune di queste informazioni vengono anche registrate nel descrittore di processo. Dopo aver caricato l’immagine del processo nella partizione creata, il processo in oggetto viene sottoposto al controllo di un modulo del sistema operativo che si occupa delle operazioni successive. Quando un processo termina o subisce uno swapping, il sistema operativo restituisce la partizione all’area libera e cancella la riga corrispondente nella TDP. L’insieme delle aree libere varia dinamicamente, gli spazi interni all’area libera vengono utilizzati per mantenere una lista a puntatori. Gli algoritmi più comuni per selezionare un’aria di memoria all’interno della quale creare una partizione sono:

* **First-fit e la sua variante next-fit**
* **Best-fit**
* **Worst-fit**

Il metodo **first-fit** termina la ricerca quando viene individuato il primo blocco di memoria sufficientemente grande da contenere il processo.

Il metodo **next-fit** il puntatore alla lista delle aree libere, dopo avere effettuato un’allocazione, viene salvato cosicché la ricerca successiva continua da dove si era fermata la precedente.

Il metodo **best-fit** alloca il blocco di memoria libero più piccolo che può contenere il processo.

Il metodo **worst-fit** alloca il blocco di memoria libero più grande le cui dimensioni superino quelle della partizione richiesta.

Lo schema di partizionamento dinamico **genera buchi di memoria** inutilizzabili. Per questo motivo può anche diventare impossibile l’allocazione di memoria richiesta da un processo, pur contenendo la memoria globalmente un’area di dimensione sufficiente. Questo fenomeno si chiama **frammentazione esterna.** Quando la frammentazione raggiunge un grado elevato è necessario effettuare la compattazione della memoria che consiste nello spostare i processi residenti in memoria in modo da creare partizioni libere più grandi. Questa operazione generalmente richiede un grande costo di CPU. La compattazione può essere effettuata:

1. **Continuamente:**

* La memoria viene compattata ogni volta che un processo libera un’area
* La frammentazione è ridotta al minimo
* Questa operazione ha un grande costo di CPU

1. **Su necessità:**

* La compattazione viene effettuata quando non si riesce ad allocare memoria ad un processo richiedente
* Un controllo preliminare verifica se il totale dell’area liberale è sufficiente a contenere il processo

Inoltre la compattazione può essere effettuata in diversi modi:

1. **Spostamento selettivo incrementale:**

* Ricerca di una strategia ottima di movimento per compattare al meglio la memoria
* Consente di risparmiare tempo negli spostamenti
* Poco utilizzata perché richiede un sovraccarico notevole

1. **Spostamento globale:**
   * Tute le partizioni vengono rilocate ad uno dei due estremi della memoria
   * Il tempo di trasferimento delle partizioni non è ottimizzabile
   * Non richiede strategie particolari

Le tecniche di protezione e condivisione sono analoghe a quelle del partizionamento statico, considerato che il supporto hardware è simile. Nel partizionamento dinamico è possibile una particolare forma di condivisione:

* A due partizioni contigue è consentito sovrapporsi, mettendo un’area in comune
* Questa forma di condivisione è molto restrittiva, in quanto, può avvenire solo tra due processi

La **SEGMENTAZIONE** è uno schema di gestione della memoria che si basa sulla divisione dello spazio di indirizzamento dei processi in entità logiche che possono essere poste in aree non contigue di memoria. La segmentazione fornisce strumenti per la rilocazione dinamica, la protezione e la condivisione. La frammentazione esterna può essere ridotta se la dimensione delle aree richieste è più piccola. Un programma eseguibile può essere suddiviso, ad esempio, in codice, dati e stack; ciascuno di questi può essere posto in un segmento diverso di dimensioni differenti.

La segmentazione condivide alcune proprietà:

* **Dagli schemi di allocazione contigua** relativamente ad un singolo segmento, poiché i dati di ogni singola entità logica devono essere posti in un’area contigua di memoria
* **Dagli schemi di allocazione non contigua** relativamente all’intero spazio di indirizzamento del processo, poiché blocchi logici diversi possono essere messi in segmenti non contigui

La raccolta degli oggetti in segmenti viene predisposta dal programmatore, ciascun segmento inizia all’indirizzo virtuale zero, ogni segmento ha un nome che viene poi tradotto in un indirizzo in fase di caricamento in memoria, un singolo dato all’interno del segmento viene identificato dallo spiazzamento relativo all’indirizzo del segmento cui appartiene.

L’indirizzo nella segmentazione è di tipo bidimensionale, infatti la designazione univoca di un dato o di un’istruzione richiede il nome del segmento e lo spiazzamento all’interno del segmento. La memoria fisica mantiene l’indirizzo lineare per cui è necessario un meccanismo per la traduzione degli indirizzi bidimensionali di segmenti virtuali in indirizzi fisici. **Il caricamento** di un processo segmentato avviene nel seguente modo:

* Il S.O. cerca di allocare n partizioni per adatte per gli n segmenti in cui è diviso il processo
* Il S.O. crea un descrittore di segmento registrandovi l’indirizzo fisico in cui è stato posto il segmento e la sua dimensione
* L’insieme dei descrittori di segmento di un processo viene raccolto nella tabella dei descrittori di segmento

**La traduzione** dell’indirizzo avviene nel seguente modo:

* L’indirizzo bidimensionale virtuale I (segment, displacement) è tradotto in modo equivalente al meccanismo del registro base
* Il segment viene utilizzato per ritrovare l’indirizzo fisico del segmento nella TDS
* Il displacement viene sommato all’indirizzo fisico del segmento per ottenere l’indirizzo fisico richiesto
* La dimensione del segmento, riportata nella TDS, viene utilizzata anche per controllare eventuali violazioni

Le dimensioni della TDS sono legate alle dimensioni dello spazio virtuale di indirizzamento di un processo. Essa viene tenuta in memoria, in uno dei segmenti associati al processo. Due registri hardware, RBTDS e RLTDS, contengono l’indirizzo di inizio e fine della TDS associata a processo. Ciò consente anche il controllo d’accesso a segmenti non assegnati al processo. Quando un processo subisce uno swapping, al ritorno in memoria va aggiornata la sua TDS. Generalmente si preferisce rigenerarla totalmente utilizzando la mappa statica fornita dal linker. L’eventuale compattazione richiede l’aggiornamento nella TDS delle righe relative ai segmenti spostati. La traduzione di ciascun indirizzo bidimensionale virtuale richiede due accessi in memoria:

* Uno alla TDS per tradurre l’indirizzo del segmento virtuale a fisico
* L’altro per accedere fisicamente alla locazione richiesta

La segmentazione raddoppia il tempo per l’accesso ad una locazione di memoria.

La forma più naturale di protezione per un sistema segmentato è l’utilizzo di registri base e limite. La protezione tra spazi di indirizzamento di processi differenti è garantita dalla collocazione dei segmenti in aree disgiunte di memoria. Una caratteristica offerta dai sistemi segmentati consiste nel fornire un livello di protezione all’interno dello spazio di indirizzamento di un singolo processo. È possibile definire diritti e modalità di accesso per ogni tipo di segmento che possono essere registrati in un campo della TDS.

La segmentazione offre una condivisione semplice e flessibile. Gli oggetti condivisi sono collocati in segmenti dedicati e separati. Il segmento condiviso può essere mappato, attraverso la TDS, nello spazio di indirizzamento virtuale di tutti i processi autorizzati ad accedervi. Lo spiazzamento interno di un dato risulta identico per tutti i processi autorizzati ad accedervi. Lo spiazzamento interno di un dato risulta identico per tutti i processi che lo condividono.

Quando si applica **L’ALLOCAZIONE** **NON** **CONTIGUA** la memoria viene allocata in modo tale che un unico oggetto logico viene posto in aree separate e non adiacenti. Durante l’esecuzione di un processo viene eseguita la traduzione degli indirizzi per ristabilire la corrispondenza tra lo spazio virtuale di indirizzamento (contiguo) e gli indirizzi fisici. Vi sono due metodi attraverso i quali si implementa l’allocazione non contigua: la paginazione e la memoria virtuale.

Quando si effettua la **PAGINAZIONE** la memoria fisica viene concettualmente suddivisa in pagine di dimensione fissa chiamate pagine fisiche. Anche lo spazio di indirizzamento dei processi viene diviso in pagine virtuali della stessa dimensione delle pagine fisiche. Il meccanismo di traduzione fa corrispondere indirizzamenti all’interno di pagine virtuali ad indirizzamenti in pagine fisiche. Ogni pagina è mappata separatamente, per cui non è necessaria la contiguità delle pagine fisiche. Ogni indirizzo all’interno di un processo è diviso in:

* Bit più significativi che rappresenta il numero della pagina virtuale
* Bit meno significativi che rappresentano lo spiazzamento reale

Il sistema operativo tiene traccia della corrispondenza tra pagine virtuali e pagine fisiche mediante **la tabella delle pagine (TDP)**. La TDP viene costruita al momento del caricamento del processo, **s** della tabella contiene l’indirizzo di partenza della pagina fisica in cui è allocata la corrispondente pagina virtuale, la tabella è costituita da un numero di righe pari al numero di pagine occupate.

La traduzione dell’indirizzo virtuale avviene nel seguente modo:

1. Un meccanismo hardware divide l’indirizzo virtuale in:

* 12 bit più significativi che rappresentano il numero della pagina
* 8 bit meno significativi che rappresentano il displacement interno alla pagina

1. Il numero della pagina viene usato all’interno della TDP per ottenere il corrispondente indirizzo della pagina fisica
2. L’indirizzo ottenuto, concatenato con il displacement, fornisce l’indirizzo fisico cercato

Il sistema operativo tiene traccia dello stato di tutte le pagine fisiche mediante **la tabella di memoria (TDM):**

* In ogni riga della TDM è indicato lo stato della relativa pagina
* Il numero di righe della tabella è uguale al numero di pagine fisiche

Quando un processo termina, subisce uno swapping:

* Restituisce tutte le pagine fisiche ad esso assegnate
* La sua TDP viene eliminata

Quando il processo richiedente ha uno spazio di indirizzamento non multiplo della dimensione delle pagine fisiche si crea il fenomeno della frammentazione interna della pagina. Questo è comunque un fenomeno molto ridotto e legato alla dimensione della pagina fisica.

Particolare attenzione va posta nella scelta della tecnica di memorizzazione delle aree libere. La consultazione della TDM statica per trovare n pagine fisiche libere richiede una ricerca su un numero di righe in media x=n/q. q è la probabilità che una pagina sia libera e si lega alla percentuale di memoria libera u: q=u/100 0<=q<=1

In sostanza x è legata ad 1/q e quindi aumenta con l’aumentare della memoria utilizzata. Per questo motivo si preferisce sostituire la TDM con una lista a puntatori contenente i numeri delle pagine fisiche:

* N pagine da allocare si individuano nei primi n nodi della lista
* In fase di disallocazione vengono inseriti in testa alla lista gli n nodi relativi alle pagine liberate

**I meccanismi hardware** per la paginazione vengono utilizzati per due scopi:

1. Risparmiare la memoria necessaria per le TDP
2. Velocizzare la traduzione da indirizzi virtuali a fisici

Nel primo caso è ragionevole costruire tabelle con un numero di righe uguale alle pagine effettivamente utilizzate. Si rende necessario quindi l’uso di due registri hardware:

* Registro limite della tabella TDP che costituisce il numero di pagina virtuale più alto utilizzato dal processo
* Registro base della tabella TDP che contiene l’indirizzo di partenza della TDP

I valori dei registri sono di norma memorizzati nel descrittore di processo. Questo meccanismo consente anche il controllo di accesso a pagine non assegnate al processo.

Nel secondo caso la traduzione di ciascun indirizzo virtuale richiede due accessi in memoria:

* Uno alla TDP per prelevare il numero di pagina fisica
* L’altro per accedere fisicamente alla locazione richiesta

In sostanza la paginazione raddoppia il tempo per l’accesso ad una locazione di memoria. Un metodo di velocizzazione è l’uso di una cache di traduzione in cui vengono mantenuti gli indirizzi delle pagine usate più frequentemente.

Spazi di indirizzamento distinti vengono di norma allocati in zone disgiunte della memoria fisica, gli accessi alla memoria sono ristretti al proprio spazio di indirizzamento, il registro limite della TDP blocca i tentativi di accedere a pagine virtuali oltre il limite assegnato al processo, i valori dei registri sono modificati esclusivamente da istruzioni privilegiate del sistema operativo, l’aggiunta di alcuni bit alle righe delle TDP consente di definire e controllare il tipo di accesso alla pagina, nella paginazione non ha senso la protezione interna allo spazio di indirizzamento del processo, poiché non prevede tipizzazione di dati.

La condivisione è immediata con lo schema della paginazione, infatti:

* Una copia unica di una pagina fisica può essere mappata in molti spazi di indirizzamento
* I diversi processi possono avere un tipo di accesso diverso alla pagina
* La condivisione viene riconosciuta ed effettuata da programmi di sistema, poiché la paginazione è trasparente all’utente
* Il codice condiviso deve naturalmente essere rientrante o eseguito in mutua esclusione.

La **MEMORIA** **VIRTUALE** è uno schema di gestione della memoria in cui soltanto una parte dello spazio di indirizzamento virtuale di un processo residente viene effettivamente caricata in memoria. Con questo tipo di gestione si ha che:

* La somma di tutti gli spazi di indirizzamento virtuali dei processi attivi può superare la capacità della memoria fisica
* La dimensione ammissibile per lo spazio di indirizzamento virtuale di un singolo processo può superare la capacità della memoria fisica disponibile in un sistema

Gli schemi precedenti della gestione della memoria tendevano ad una occupazione della memoria vicina la 100%, mentre lo schema con memoria virtuale sembra fornire un grado di occupazione superiore al 100%. Questo si ottiene:

* Mantenendo in memoria secondaria l’immagine dell’intero spazio virtuale di indirizzamento del processo
* Trasferendo in memoria fisica soltanto alcune parti al momento in cui sono necessarie
* Il sistema operativo sceglie tempi e modalità di trasferimento in memoria fisica tenendo conto di fattori quali: richieste dei processi attivi, priorità dei processi e disponibilità globali del sistema.

Dal punto di vista del programmatore:

* I dettagli della gestione sono trasparenti
* Non è necessario l’adeguamento di un programma ad una memoria limitata
* Un programma può essere eseguito su sistemi con disponibilità di memoria fisica diversa, senza alcun adattamento

Dal punto di vista del S.O:

* Un processo può essere caricato in uno spazio di dimensione arbitrarie
* Si riduce la frammentazione esterna
* Non è necessario modificare l’ordine di esecuzione di un processo
* Il quantitativo di memoria assegnato ad un processo può variare durante la sua durata
* Il sistema operativo può dinamicamente scegliere di velocizzare l’esecuzione di un processo allocando un quantitativo maggiore di memoria o, in alternativa, aumentare il grado di multiprogrammazione

L’importanza dell’uso dello schema di memoria virtuale si basa sulle seguenti osservazioni:

* Durante una specifica esecuzione alcune parti del programma non vengono indirizzate
* Le diverse condizioni interne o esterne causano diversi tracciati ad ogni esecuzione di un processo
* Alcune parti di un programma vengono eseguite molto raramente
* Alcuni programmi contengono tracciati complementari

Il funzionamento della memoria virtuale è analogo alla paginazione ed alla segmentazione e quindi con uso di tabelle di traduzione. La differenza sostanziale è che alcune pagine possono essere mancanti, per cui il processo di traduzione è più articolato.

**Lo schema di memoria** **virtuale** fa uso della tabella di traduzione delle pagine con aggiunta di 1 bit di presenza che viene posto a 1 se la pagina è in memoria reale mentre viene posto a 0 se si tratta di una pagina mancante. Il caricamento di un processo avviene nel seguente modo: il sistema operativo crea la TDP, ponendo ad 1 i bit di presenza delle pagine portate in memoria. Se viene richiesta una pagina mancante, il processo viene sospeso in attesa del suo caricamento, altri processi vengono posti in esecuzione nel frattempo.

Lo schema di memoria virtuale fa uso anche della tabella di descrizione del file (TDF). La TDF è equivalente alla TDP, associa al numero di pagini virtuali l’indirizzo di residenza di queste sulla memoria secondaria. In generale i sistemi supportano l’interruzione di un processo tra due istruzioni, ma non supportano l’interrompibilità della singola istruzione. Lo schema della memoria virtuale è implementabile solo su sistemi che supportano l’interrompibilità della singola istruzione. L’assenza della memoria della pagina successiva causa una interruzione dell’istruzione, l’hardware deve essere in grado di gestire i cambiamenti avvenuti durante l’esecuzione della stessa. Le possibili soluzioni a questo problema sono:

* Gli effetti parziali di un’istruzione vengono cancellati e l’istruzione viene ricominciata dopo aver risolto il page fault: dispende strettamente dal tipo di istruzioni, non è facile generalizzare, alcuni cambiamenti possono essere irreversibili, può richiedere molto lavoro
* L’istruzione viene fatta partire dal punto esatto in cui era stata interrotta, questo tipo di soluzione richiede forme sofisticate di memorizzazione dello stato CPU
* Prima di iniziare l’esecuzione si controllano gli accessi in memoria che l’istruzione eseguirà, però può non essere possibile giustificare il controllo.

Le soluzioni adottate sono spesso miste e rientrano nelle scelte architetturali.

La gestione della memoria virtuale segue gli schemi analoghi alla paginazione o alla segmentazione, con l’aggiunta della TDF. Ai fini dell’efficienza della memoria virtuale, l’allocazione di un sottoinsieme delle pagine virtuali su pagine fisiche richiede l’applicazione di specifiche strategie. Tali strategie si possono classificare in:

* **Strategie di allocazione**: quanta memoria reale allocare a ciascun processo
* **Strategia di ricerca:** quali oggetti trasferire e quando dalla memoria secondaria alla memoria centrale
* **Strategie di sostituzione**: in mancanza di spazio, quali oggetti rimuovere per assegnare spazio alle nuove richieste
* **Strategie di posizionamento:** dove collocare un oggetto che entra in memoria.

Le strategie di allocazione e posizionamento sono immediate in quanto gli oggetti vengono trasferiti quando servono, si parla infatti di paginazione o segmentazione a richiesta, non sono facilmente applicabili strategie di previsione e infine il posizionamento delle pagine segue le regole dello schema con cui è realizzata la memoria virtuale, ossia paginazione e segmentazione. Queste due strategie sono particolarmente importanti per l’efficienza della gestione della memoria virtuale e richiedono particolare attenzione. Importante è analizzare il comportamento di un processo, dal punto di vista delle sequenze di accesso della memoria durante la sua esecuzione al fine di minimizzare il tempo di residenza in memoria di un processo e al fine di minimizzare i trasferimenti di pagine.

All’occorrenza di una page fault, se il gestore della memoria non ha pagine disponibili può disporre due operazioni:

* **Sospensione** del processo che ha generato la page fault
* **Sostituzione** di una pagina in memoria

La prima operazione è raramente scelta perché:

* L’eccezione di page fault è dovuta allo schema di gestione e non al processo
* La sospensione del processo influirebbe sullo scheduling e sul tempo di risposta relativi al processo
* In condizioni di memoria piena il tempo che una pagina si liberi può essere lungo

La seconda operazione è la via generalmente scelta perché:

* Richiede preferibilmente la presenza di 1 bit per il controllo di pagina modificata
* Il dirty bit può rientrare tra i fattori presi in considerazione per la pagina da sostituire

Esistono numerosi algoritmi per la soluzione delle pagine tra cui:

* **Algoritmo FIFO (First In First Out)**
* **Algoritmo LRU (Least Recently Used)**
* **Algoritmo di Belady**
* **Algoritmo NRU (Not Recently Used)**

**L’algoritmo FIFO** sostituisce le pagine che da più tempo risiedono in memoria. Il gestore della memoria deve tenere traccia dell’ordine di caricamento delle pagine in memoria, ad esempio con una coda FIFO sei numeri di pagina È di semplice realizzazione, richiede di aggiornare la coda solo ad ogni page fault e non richiede supporti hardware specifici, ma contemporaneamente la questa politica spesso getta via pagine più frequentemente indirizzate, poiché per loro natura permangono più tempo in memoria perché non si tiene traccia dell’ordine di riferimento delle pagine.

**L’algoritmo LRU** sostituisce le pagine residenti che meno recentemente sono state utilizzate. L’LRU considera la frequenza del comportamento di un programma con l’ipotesi che la pagina utilizzata recentemente è quella che ha la minore probabilità di essere referenziata in futuro. Una possibile modalità operativa è:

* Memorizzare l’suo delle pagine attraverso l’uso di una struttura simile ad uno stack
* Ogni volta che la pagina viene referenziata, viene ricercata nello stack e posta in cima ad esso
* La pagina da rimuovere viene presa dalla parte bassa dello stack

In media l’algoritmo LRU in media si comporta meglio dell’algoritmo FIFO, ma presenta dei difetti quali l’aggiornamento dello stack va effettuato ad ogni riferimento alla pagina (praticamente ad ogni accesso in memoria) e la sua organizzazione è tale da richiedere un supporto hardware sofisticato e dedicato alle operazioni relative all’aggiornamento dello stack.

**L’algoritmo di Belady** sostituisce le pagine che saranno referenziate nel più lontano futuro. Non è realizzabile, ma è di importanza teorica.

**L’algoritmo NRU** combina il basso costo dell’algoritmo FIFO con la registrazione delle pagine residenti tipico dell’LRU sostituendo le pagine non recentemente usate. Viene associato 1 bit a ciascuna pagina residente, il bit viene posto a 1 ad ogni riferimento alla pagina, successivamente i bit di tutte le pagine vengono posti a zero ciclicamente, quando si deve scegliere la pagina, l’algoritmo sceglie tra quelle con bit a 0. Facendo uso di una lista circolare riparte ogni volta dall’ultima pagina esaminata.

Gli algoritmi di sostituzione globale scelgono la pagina da rimuovere non solo tra quelle appartenenti al processo ma tra tutte quelli residenti. Questi algoritmi guardano più alle esigenze di sistema che all’andamento individuale di ciascun processo e variare la strategia degli algoritmi di allocazione.

Le strategie di allocazione servono per decidere quanta memoria fisica allocare a ciascun processo attivo. L’allocazione di un numero di pagine ottimale:

* Riduce la frequenza di page fault
* Riduce al minimo indispensabile il tempo di esecuzione
* Aumenta l’uso della CPU

L’allocazione di un numero di pagine molto grande riduce il numero di processi attivi e abbassa il fattore di utilizzo della CPU. Mentre l’allocazione di un numero di pagine troppo ridotto produce un aumento delle page fault, aumenta il tempo di residenza in memoria dei processi e aumenta lo spreco di CPU e dei dispositivi di I/O per il trasferimento delle pagine.