**Parte II - Capitolo 8 - Memoria Centrale**

La ***memoria*** consiste in un ampio vettore di parole o byte, ciascuno con il proprio indirizzo. La CPU preleva le istruzioni dalla memoria sulla base del contenuto del contatore di programma; tali istruzioni possono determinare ulteriori letture o scritture in specifici indirizzi di memoria. La memoria vede soltanto un flusso d'indirizzi di memoria e non sa come sono generati oppure a cosa servano. Di conseguenza, è possibile ignorare come un programma genera un indirizzo di memoria, e prestare attenzione solo alla sequenza degli indirizzi di memoria generati dal programma in esecuzione.

La memoria centrale e i registri incorporati nel processore sono le sole aree di memorizzazione a cui la CPU può accedere direttamente. Qualsiasi istruzione in esecuzione e tutti i dati utilizzati dalle istruzioni, devono risiedere in questi dispositivi. I registri incorporati sono accessibili, in genere, in un ciclo di orologio di sistema. Ciò non vale per la memoria centrale, cui si accede attraverso una transazione sul bus della memoria. Per evitare le situazioni di *stallo* (*stall*), si interpone una memoria veloce tra la CPU e la memoria centrale, un buffer di memoria detto ***cache*** che è in grado di conciliare le diverse velocità. Inoltre è necessario proteggere il sistema operativo dall'accesso dei processi utenti e salvaguardare i processi utenti l'uno dall'altro. Bisogna quindi assicurarsi che ogni processo abbia uno spazio in memoria separato. Occorre quindi determinare l'intervallo di indirizzi a cui un processo può accedere legalmente, e garantire che possa accedere soltanto a questi indirizzi. Si può implementare il meccanismo di protezione tramite due rigistri: il ***registro base***, che contiene il più piccolo indirizzo legale della memoria fisica, e il ***registro limite***, che determina la dimensione dell'intervallo ammesso. Qualsiasi tentativo da parte di un programma eseguito in modalità utente di accedere alle aree di memoria riservate al sistema operativo o a una qualsiasi area di memoria riservata ad altri utenti comporta l'invio di un segnale d'eccezione che restituisce il controllo al sistema operativo, che a sua volta, interpreta l'evento come un errore fatale. Lo schema utilizzato impedisce a qualsiasi programma utente di alterare il codice o le strutture dati. Solo il sistema operativo può caricare i registri base e limite grazie a una speciale istruzione privilegiata.

In genere un programma risiede in un disco in forma di file binario eseguibile. Per esser eeseguito, il programma deve essere caricato in memoria e inserito all'interno di un processo. L'insieme dei processi presenti nei dischi e che attendono di essere trasferiti in memoria per essere eseguiti forma la ***coda d'ingresso*** (*input queue*). Generalmente gli indirizzi del programma sorgente sono simbolici (es.contatore). Un compilatore di solito *associa* (*bind*) questi indirizzi simbolici a indirizzi rilocabili. L'editor dei collegamenti (*linkage editor*) o il caricatore (*loader*), fa corrispondere a sua volta questi indirizzi rilocabili a indirizzi assoluti. Ogni associazione rappresenta una corrispondenza da uno spazio d'indirizzi a un altro. L'associazione di istruzioni e dati a indirizzi di memoria si compie, quindi, in tre fasi: compilazione, caricamento ed esecuzione, come descritto precedentemente.

Un indirizzo generato dalla CPU di solito si indica come ***indirizzo logico***, mentre un indirizzo visto dall'unità di memoria di solito si indica come ***indirizzo fisico***. I metodi di associazione degli indirizzi nelle fasi di compilazione e di caricamento producono indirizzi logici e fisici identici. Con i metodi di associazione nella fase di esecuzione ciò non avviene. In questo caso ci si riferisce agli indirizzi logici col termine di ***indirizzi virtuali***. L'insieme di tutti gli indirizzi logici generati da un programma è lo ***spazio degli indirizzi logici***; l'insieme di tutti gli indirizzi fisici corrispondenti a tali indirizzi logici è lo ***spazio degli indirizzi fisici***. L'associazione nella fase di esecuzione dagli indirizzi virtuali agli indirizzi fisici è svolta da un dispositivo detto ***unità di gestione della memoria*** (***MMU***, ***mermory-management unit***): quando un processo utente genera un indirizzo, prima dell'invio all'unità di memoria, si somma a tale indirizzo il valore contenuto nel *registro di rilocazione*. In questo modo un programma utente non considera mai gli indirizzi fisici reali, ma viene indirizzato verso quelli logici.

Per migliorare l'utilizzo della memoria si può ricorrere al ***caricamento dinamico*** (*dynamic loading*), mediante il quale si carica una procedura solo quando viene richiamata; tutte le procedure si tengono in memoria secondaria in un formato di caricamento rilocabile. Si carica il programma principale in memoria e quando, durante l'esecuzione, una procedura deve richiamarne un'altra, si controlla innanzitutto che sia stata caricata, altrimenti si richiama il caricatore di collegamento rilocabile per caricare in memoria la procedura richiesta e aggiornare le tabelle degli indirizzi del programma in modo che registrino questo cambiamento. Il controllo passa poi alla procedura appena caricata. Il vantaggio del caricamento dinamico consiste nel fatto che una procedura che non si adopera non viene caricata e risulta maggiormente utile quando servono grandi quantità di codice per gestire casi non frequenti.

Sono utili anche le ***librerie collegate dinamicamente***. Alcuni sistemi operativi consentono solo il collegamento statico, in cui le librerie e il ilnguaggio sono trattate come qualsiasi altro modulo oggetto e combinate dal caricatore nell'immagine binaria del programma. Il concetto di collegamento dinamico è analogo a quello di caricamento dinamico. Invece di differire il caricamento di una procedura fino al momento dell'esecuzione, si differisce il collegamento. Questa caratteristica si può estendere anche agli aggiornamenti delle librerie come per esempio alla correzione degli errori. A differenza del caricamento dinamico, il collegamento dinamico richiede generalmente l'assistenza del sistema operativo.

Per essere eseguito, un processo deve trovarsi in memoria centrale, ma si può trasferire temporaneamente in ***memoria ausiliaria*** (*backing store*) da cui si riporta in memoria centrale al momento di riprenderne l'esecuzione. Questo procedimento è detto ***avvicendamento dei processi in memoria*** (***avvicendamento*** o ***scambio*** o ***swapping***).

L'avvicendamento dei processi richiede una memoria ausiliaria che deve essere abbastanza grande da contenere le copie di tutte le immagini di memoria di tutti i processi utenti, e deve permettere l'accesso diretto a queste immagini. Il sistema mantiene una ***coda di processi pronti*** (*ready queue*) formata da tutti i processi pronti per l'esecuzione, le cui immagini di memoria si trovano in memoria ausiliaria o in memoria.

In un sistama che usa l'avvicendamento, il tempo di cambio di contesto (*context switch time*) è abbastanza elevato. La maggior parte del tempo d'avvicendamento è data dal tempo di trasferimento che è direttamente proporzionale alla quantità di memoria interessata. Inoltre l'avvicendamento è soggetto ad un altro vincolo: per scaricare un processo dalla memoria è necessario essere certi che questo sia completamente inattivo. Questo problema si può risolvere in due modi: o non si scarica dalla memoria un processo con operazioni pendenti, oppure si eseguono le operazioni solo in aree di memoria del sistema operativo. Attualmente l'avvicendamento semplice si usa in pochi sistemi in quanto richiede un elevato tempo di gestione e consente un tempo di esecuzione troppo breve per essere considerato una soluzione ragionevole al problema di gestione della memoria.

La memoria centrale deve contenere sia il sistema operativo sia i vari processi utenti, perciò è necessario assegnare le diverse parti della memoria centrale nel modo più efficiente. La memoria centrale di solito è divisa in due partizioni: una per il sistema operativo residente e una per i processi utenti. In generale si vuole che i processi utenti risiedano contemporaneamente in memoria centrale, quindi è necessario considerare come assegnare la memoria disponibile ai processi presenti nella coda d'ingresso che attendono di essere caricati in memoria. Con l'***allocazione contigua della memoria***, ciascun processo è contenuto in una singola sezione contigua di memoria. Prima di trattare l'allocazione, si devono comprendere al meglio le questioni relative alla rilocazione e alla protezione della memoria. Con i registri di rilocazione e limite, ogni indirizzo logico deve essere minore del contenuto del registro limite; la MMU fa corrispondere dinamicamente l'indirizzo fisico all'indirizzo logico sommando a quest'ultimo il valore contenuto nel registro di rilocazione. Quando lo scheduler della CPU seleziona un processo per l'esecuzione, il dispatcher, durante l'esecuzione del cambio di contesto, carica il registro di rilocazione e il registro limite con i valori corretti. Poichè si confronta ogni indirizzo generato dalla CPU con i valori contenuti in questi registri, si possono proteggere il sistema operativo, i programmi e i dati di altri utenti da tentativi di modifiche da parte del processo in esecuzione. Passiamo ora all'allocazione della memoria. Uno dei metodi più semplici consiste nel suddividere la memoria in ***partizioni*** di dimensione fissa che deve contenere esattamente un processo. Con il ***metodo delle partizioni multiple*** quando una partizione è libera può essere occupata da un processo; terminato il processo, la partizione diventa nuovamente disponibile per un altro processo. Questo metodo non è più in uso. Nello schema a partizione fissa il sistema operativo conserva una tabella in cui sono indicate le partizioni della memoria disponibili e quelle occupate. Inizialmente tutta la memoria è a disposizione dei processi utenti, quindi appare come un grande blocco di memoria disponibile, un ***buco*** (*hole*). Quando entrano nel sistema, i processi vengono inseriti in una coda d'ingresso. Per determinare quali processi assegnare alla memoria, il sistema operativo tiene conto dei requisiti di memoria di ciascun proceso e delle quantità di spazio di memoria disponibile. Quando a un processo si assegna dello spazio, il processo stesso viene caricato in memoria e può quindi competere per il controllo della CPU. Al termine, rilascia la memoria che gli era stata assegnata. In generale, è sempre presente un insieme di buchi di diverse dimensioni sparsi per la memoria. Quando si presenta un processo che necessita di memoria, il sistema cerca nel gruppo un buco di dimensioni sufficienti per contenerlo; se il buco è troppo grande, questo viene diviso in due parti. I criteri più usati per scegliere un buco libero tra quelli disponibili sono: ***first-fit*** (si assegna il primo buco abbastanza grande), ***best-fit*** (si assegna il più piccolo buco in grado di contenere il processo), ***worst-fit*** (si assegna il buco più grande). Attraverso le simulazioni si è dimostrato che il first-fit e il best-fit sono migliori rispetto al worst-fit in termini di risparmio di tempo e di energia utilizzata. Entrambi i criteri soffrono di ***frammentazione esterna***: quando si caricano e si rimuovono i processi dalla memoria, si frammenta lo spazio libero della memoria in tante piccole parti. Si ha la frammentazione esterna se lo spazio di memoria totale è sufficiente per soddisfare una richiesta, ma non è contiguo, cioè la memoria è frammentata in tanti piccoli buchi. Questo problema può essere molto grave e, nel caso peggiore, può verificarsi un blocco di memoria libera tra ogni coppia di processi. Una soluzione al problema è data dalla ***compattazione*** (***garbage collection***). Lo scopo è quello di riordinare il contenuto della memoria per riunire la memoria libera in un unico grosso blocco. La compattazione non è sempre possibile (solo se la rilocazione è dinamica e si compie nella fase di esecuzione). Il più semplice algoritmo di compattazione consiste nello spostare tutti i processi verso un'estremità della memoria, mentre tutti i buchi vengono spostati nell'altra direzione formando un grosso buco di memoria. Questo metodo può essere molto oneroso. Un'altra possibile soluzione al problema della frammentazione esterna è data dal consentire la non contiguità dello spazio degli indirizzi logici di un processo, permettendo così di assegnare la memoria fisica ai processi dovunque sia possibile.

La ***paginazione*** è un metodo di gestione della memoria che permette che lo spazio degli indirizzi fisici di un processo sia contiguo. Il metodo di base per implementare la paginazione consiste nel suddividere la memoria fisica in blocchi di dimensione costante, detti ***frame*** o ***pagine fisiche***, e nel suddividere la memoria logica in blocchi di pari dimensioni detti ***pagine***. Quando si deve eseguire un processo, si caricano le sue pagine nei frame disponibili, prendendole dalla memoria ausiliaria, divisa in blocchi di dimensione fissa, uguale a quella dei frame della memoria. Ogni indirizzo generato dalla CPU è diviso in due parti: un ***numero di pagina*** e uno ***scostamento di pagina*** (*offset*). Il numero di pagina serve come indice per la ***tabella delle pagine***, contenente l'indirizzo di base in memoria fisica di ogni pagina. Questo indirizzo di base si combina con lo scostamento di pagina per definire l'indirizzo di memoria fisica. La dimensione di una pagina, così come quella di un frame, è definita dall'architettura del calcolatore ed è, in genere, una potenza di 2 compresa tra 512 byte e 16 MB. La scelta di una potenza di 2 come dimensione di una pagina facilita notevolmente la traduzione di un indirizzo logico in corrispondenti numero di pagina e scostamento di pagina. Quando si deve eseguire un processo, si esamina la sua dimensione espressa in pagine. Poichè ogni pagina di un processo necessita di un frame, se il processo richiede n pagine, devono essere disponibili almeno n frame che si assegnano al processo stesso. Si carica la prima pagina del processo in uno dei frame assegnati e s'inserisce il numero di frame nella tabella delle pagine realtiva al processo in questione. La pagina successiva si carica in un altro frame e, anche in questo caso, si inserisce il numero del frame nella tabella delle pagine. Un aspetto importante della paginazione è la netta distinzione tra la memoria vista dall'utente e l'effettiva memoria fisica: il programma utente vede la memoria come un unico spazio contiguo, contenente solo il programma stesso; in realtà il programma utente è sparso in una memoria fisica contenente anche altri programmi. La differenza tra la memoria vista dall'utente e la memoria fisica è colmata dall'architettura di traduzione degli indirizzi, che fa corrispondere gli indirizzi fisici agli indirizzi logici generati dai processi utenti. Queste trasformazioni non sono visibili all'utente e sono controllate dal sistema operativo. Un processo non ha modo di accedere alla memoria se non attraverso quello che è previsto dalla tabella delle pagine. Poichè il sistema operativo gestisce la memoria fisica, deve essere informato dei relativi particolari di allocazione (quali frame sono assegnati, quali sono disponibili, il loro numero totale); queste informazioni sono contenute in una struttura dati detta ***tabella dei frame***, contenente un elemento per ogni frame, indicante se sia libero o meno e, se è assegnato, a quale pagina di quale processo o di quali processi. Il sistema operativo conserva una copia della tabella delle pagine per ciascun processo, così come conserva una copia dei valori contenuti nel contatore di programma e nei registri. Questa copia serve per tradurre gli indirizzi logici in indirizzi fisici ogni volta che il sistema operativo deve svolgere questa funzione. La stessa copia è usata dal dispatcher della CPU per impostare l'architettura di paginazione quando a un processo sta per essere assegnata la CPU (la paginazione fa aumentare la durata dei cambi di contesto). Ogni sistema operativo segue metodi propri per memorizzare le tabelle delle pagine. La maggior parte dei sistemi impiega una tabella delle pagine per ogni processo. L'architettura d'ausilio alla tabella delle pagine si può realizzare in modi diversi. Nel caso più semplice, si usa uno specifico insieme di ***registri*** che devono essere costruiti in modo da operare a una velocità molto elevata. Il dispatcher carica questi registri come gli altri, ma le istruzioni di caricamento e modifica dei registri della tabella delle pagine sono privilegiate e quindi eseguibili solo dal sistema operativo. L'uso dei registri per la tabella delle pagine è efficientese la tabella stessa è ragionevolmente piccola. La maggior parte dei calcolatori usa tabelle molto grandi quindi non si possono impiegare registri veloci. La tabella delle pagine viene mantenuta in memoria principale e un ***registro di base della tabella delle pagine*** (*page-table base register*, *PTBR*) punta alla tabella stessa. Il cambio delle tabelle delle pagine richiede soltanto di modificare questo registro, riducendo considerevolmente il cambio di contesto. Questo metodo comporta, però, un rallentamento del tempo di accesso alla memoria in quanto occorre accedere due volte alla memoria (uno per l'elemento della tabella delle pagine e uno per il byte stesso). La soluzione tipica a questo problema consiste nell'impiego di una speciale, piccola cache di ricerca veloce detta ***TLB*** (***translation look-aside buffer***). Questa è una memoria associativa ad alta velocità in cui ogni suo elemento consiste di due parti: una chiave, o un indicatore, e un valore. Quando si presenta un elemento, la memoria associativa lo confronta con tutte le chiavi e, se trova corrispondenza, riporta il valore correlato. La ricerca è molto rapida, ma le memorie associative sono molto costose. La TLB si usa nel modo seguente insieme con le tabelle delle pagine. La TLB contiene una piccola parte degli elementi della tabella delle pagine; quando la CPU genera un indirizzo logico, si presenta il suo numero di pagina alla TLB: se tale numero è presente, il corrispondente numero del frame è immediatamente disponibile e si usa per accedere alla memoria; se tale numero non è presente, la situazione è nota come ***insuccesso della TLB*** (*TLB miss*) e si deve consultare la tabella delle pagine in memoria. Gli elementi nella TLB possono essere sostituiti attraverso criteri che vanno dall'elemento usato meno recentemente alla scelta casuale. Inoltre, alcune TLB, consentono che certi elementi siano *vincolati* (*wired down*) e che non possano essere rimossi dalla TLB. Alcune TLB memorizzano gli ***identificatori dello spazio d'indirizzi*** (*address-space identifier*, *ASID*) in ciascun elemento della TLB. La mancata corrispondenza dell'ASID viene trattata come una TLB miss. Se la TLB non permette l'uso di ASID, ogni volta che si seleziona una tabella delle pagine si deve *cancellare* (*flush*) la TLB, in modo da assicurare che il successivo processo in esecuzione non faccia uso di errate informazioni di traduzione. La percentuale di volte che un numero di pagina si trova nella TLB è detta ***tasso di successo*** (*hit ratio*). In un ambiente paginato, la protezione della memoria è assicurata dai bit di protezione associati ad ogni frame; normalmente tali bit si trovano nella tabella delle pagine. Un bit può determinare se una pagina si può leggere e scrivere o soltanto leggere. Tutti i riferimenti alla memoria passano attraverso la tabella delle pagine per trovare il numero corretto del frame. Quindi, mentre si calcola l'indirizzo fisico, si possono controllare i bit di protezione per verificare che non si scriva in una pagina di sola lettura. Un tentativo di questo tipo causerebbe l'invio di un segnale d'eccezione al sistema operativo, poichè si avrebbe una violazione della protezione della memoria. Questo metodo si può facilmente estendere per fornire un livello di protezione perfezionato. Si può progettare un'architettura che fornisca la protezione di sola lettura, di sola scrittura o di sola esecuzione. Di solito a ciascun elemento della tabella si associa un ulteriore bit detto ***bit di validità*** impostato a *valido* se la pagina corrispondente è nello spazio d'indirizzi logici del processo, o a *non valido* se la pagina non è nello spazio degli indirizzi logici del processo. Il bit di validità consente quindi di riconoscere gli indirizzi illegali e di notificarne la presenza attraverso un segnale di eccezione; il sistema operativo concede o revoca la possibilità d'accesso a una pagina impostando in modo appropriato tale bit. Un altro vantaggio della paginazione consiste nella possibilità di condividere codice comune. Il ***codice rientrante***, detto anche ***codice puro***, è un codice non automodificante: non cambia durante l'esecuzione. Quindi due o più processi possono eseguire lo stesso codice nello stesso momento. Ciascun processo dispone di una propria copia dei registri e di una memoria dove conserva i dati necessari alla propria esecuzione. I dati per due differenti processi variano per ciascun processo.

Vediamo ora come possono essere strutturate le tabelle delle pagine. La maggior parte dei sistemi moderni dispone di uno spazio d'indirizzi logici molto grande, quindi anche la tabella delle pagine diventerebbe troppo grande. Una soluzione a questo problema consiste nel suddividere la tabella delle pagine in parti più piccole. Questo risultato si può ottenere in molti modi. Un primo metodo consiste nell'adottare un algoritmo di paginazione a due livelli, in cui la tabella stessa è paginata. L'indirizzo logico quindi è formato da un indice della tabella delle pagine di primo livello (o tabella esterna delle pagine), dallo scostamento all'interno della pagina indicata dalla tabella esterna delle pagine e dallo scostamento di pagina. Poichè la traduzione degli indirizzi si svolge dalla tabella esterna delle pagine verso l'interno, questo metodo è anche noto come ***tabella delle pagine ad associazione diretta*** (*forward-mapped page table*). Un metodo di gesione molto comune consiste nell'impiego di una ***tabella delle pagine di tipo hash***, in cui l'argomento della funzione hash è il numero di pagina virtuale. Per la gestione delle collisioni, ogni elemento della tabella hash contiene una lista concatenata di elementi che la funzione hash fa corrispondere alla stessa locazione. Ciascun elemento è composto da tre campi: il numero della pagina virtuale, l'indirizzo del frame corrispondente alla pagina virtuale e un puntatore al successivo elemento della lista. L'algoritmo opera in questo modo: si applica la funzione hash al numero della pagina virtuale contenuto nell'indirizzo virtuale (identificando un elemento della tabella), si confronta il numero di pagina virtuale con il primo campo del primo elemento della lista concatenata: se i valori coincidono si usa l'indirizzo relativo al frame per generare l'indirizzo fisico desiderato, altrimenti si passa ad esaminare gli elementi successivi. Le tabelle delle pagine di tipo hash sono particolarmente utili per gli spazi d'indirizzi sparsi, in cui i riferimenti alla memoria non sono contigui ma distribuiti per tutto lo spazio d'indirizzi. Un altro metodo utile per contrastare l'incoveniente delle grandi dimensioni di ciascuna tabella delle pagine è legato all'utilizzo della ***tabella delle pagine invertita***. Una tabella delle pagine invertita ha un elemento per ogni pagina reale (o frame), costituito dall'indirizzo virtuale della pagina memorizzata in quella reale locazione di memoria, con informazioni sul processo che possiede tale pagina. Nel sistema dunque esiste una sola tabella delle pagine che ha un solo elemento per ciascuna pagina di memoria fisica. Le tabelle invertite richiedono spesso la memorizzazione di un identificatore nello spazio di indirizzi in ciascun elemento della tabella delle pagine, perchè essa contiene di solito molti spazi d'indirizzi diversi associati alla memoria fisica. L'identificatore garantisce che una data pagina logica relativa a un certo processi sia associata alla pagina fisica corrispondente. Ciascun indirizzo virtuale è una tripla formata da id-processo, numero di pagina, scostamento. Ogni elemento della tabella delle pagine invertita è una coppia id-processo, numero di pagina. Sebbene questo schema riduca la quantità di memoria necessaria per la memorizzazione di ogni tabella delle pagine, aumenta però il tempo di ricerca nella tabella quando si fa riferimento a una pagina.

La ***segmentazione*** è uno schema di gestione della memoria che consente di gestire la rappresentazione della memoria fin qui vista dal punto di vista dell'utente. Uno spazio d'indirizzi logici è una raccolta di segmenti, ciascuno dei quali ha un nome e una lunghezza. Gli indirizzi specificano sia il nome sia lo scostamento all'interno del segmento, quindi l'utente fornisce ogni indirizzo come una coppia ordinata di valori: nome di segmento e scostamento. Per semplicità i segmenti sono numerati e ogni riferimento si compie per mezzo di un numero anzichè di un nome. Sebbene l'utente possa fare riferimento agli oggetti di un programma per mezzo di un indirizzo bidimensionale, la memoria fisica è in ogni caso una sequenza di byte unidimensionale. Per questo motivo occorre tradurre gli indirizzi bidimensionali definiti dall'utente in indirizzi fisici unidimensionali. Questa operazione si compie attraverso una ***tabella di segmenti***: ogni suo elemento è una coppia ordinata formata da *base del segmento*, che contiene l'indirizzo fisico iniziale della memoria dove il segmento risiede, e *limite del segmento*, che contiene la lunghezza del segmento.

Esempio del Pentium Intel (pag. 330)

Segmentazione in Pentium

Paginazione in Pentium

Linux su sistemi Pentium