Giuseppe Pelagatti

Programmazione e Struttura del sistema operativo Linux

Appunti del corso di
Architettura dei Calcolatori e Sistemi Operativi (AXO)

Parte M: La gestione della Memoria

cap. M1 – Memoria virtuale e paginazione

M.1 Memoria Virtuale e Paginazione

1. Indirizzi

Indirizzi, dimensioni, spazio di indirizzamento
Il modello usuale di una memoria è lineare; in tale modello la memoria è costituita da una sequenza di parole o celle numerate da 0 fino al valore massimo. Il numero che identifica ogni cella è detto **indirizzo**.

La dimensione di ogni cella indirizzabile dipende dal tipo di calcolatore; nel seguito supporremo che ogni cella sia costituita da 8 bit, cioè da un **byte**.

E' opportuno distinguere la **dimensione** (effettiva) di una memoria dal suo **spazio di indirizzamento**: lo spazio di indirizzamento è il numero massimo di indirizzi possibili della memoria ed è determinato dalla lunghezza dell'indirizzo, cioè dal numero di bit che costituiscono l'indirizzo; se N è il numero di bit che costituiscono l'indirizzo di una memoria, allora il suo spazio di indirizzamento è 2^N .

La dimensione della memoria è il numero di byte che la costituiscono effettivamente; ovviamente, dato che tutti i byte devono essere indirizzabili *la dimensione della memoria è sempre minore o uguale al suo spazio di indirizzamento*. Ad esempio, quando si installa nuova memoria su un calcolatore si aumentano le dimensioni della memoria, restando entro i limiti dello spazio di indirizzamento.

Le dimensioni della memoria sono generalmente espresse in Kb (Kilobyte), Mb (Megabyte), Gb (Gigabyte), Tb (Terabyte); queste unità corrispondono rispettivamente a 2^{10}, 2^{20}, 2^{30} , 2^{40} byte. Dato che quasi sempre la misura è espressa come numero di byte, la parola byte viene spesso omessa.

Attenzione: talvolta, per semplicità di rappresentazione degli esempi, quando non vi è dubbio che un numero deve rappresentare un indirizzo o una sua parte, si userà la notazione esadecimale senza precederla con 0x ; quindi si scriverà "indirizzo 3472" invece di "indirizzo 0x3472".

Richiamiamo alcuni concetti relativi agli indirizzi dal punto di vista del processo di compilazione:

- le variabili e le funzioni definite nel codice di un programma ricevono un indirizzo che corrisponde alla prima cella di memoria allocata per contenerle; tale indirizzo è rappresentato nella **Tabella Globale dei Simboli,** detta anche **Mappa del Programma (**o **Mappa del Sistema** riferendosi al Sistema operativo);
- le variabili allocate dinamicamente sullo heap o sulla pila invece sono poste in celle di memoria che non hanno un indirizzo associato staticamente e il loro indirizzo verrà determinato al momento dell'allocazione effettiva; al massimo si può sapere a priori quali sono gli indirizzi che delimitano l'area nella quale potranno essere allocate – chiameremo queste celle di memoria **anonime**

2. Memoria fisica e memoria virtuale

Un programma eseguibile è costituito dalle istruzioni che devono essere caricate in memoria per essere eseguite dal processore. Quando il processore esegue il programma, esso legge continuamente gli indirizzi degli operandi e delle istruzioni che sono incorporati nel programma, quindi esso utilizza come indirizzi gli indirizzi contenuti nel programma eseguibile. Tali indirizzi sono detti **indirizzi virtuali** e il modello della memoria incorporato nel programma eseguibile è detto **memoria virtuale**. Ad esempio, in figura 1 è mostrato il processore che legge l'istruzione contenuta all'indirizzo virtuale 2 e, eseguendola, opera sulla cella di indirizzo virtuale 7.

Anche per la memoria virtuale di un programma possiamo parlare di spazio di indirizzamento e di dimensione; lo spazio di indirizzamento è determinato dal numero di indirizzi virtuali disponibili, mentre la **dimensione iniziale** (virtuale) è determinata dal collegatore durante la costruzione del programma eseguibile e scritta nel file che contiene l'eseguibile. La dimensione virtuale del programma è soggetta a variazione durante l'esecuzione del programma stesso, in base alle esigenze di allocazione di memoria per nuovi dati.

Ad esempio, nel x64, la lunghezza degli indirizzi è di 64 bit; lo spazio virtuale disponibile è di 2^{48} byte, quindi solo 48 dei 64 bit di indirizzamento sono utilizzati effettivamente per costruire indirizzi validi.

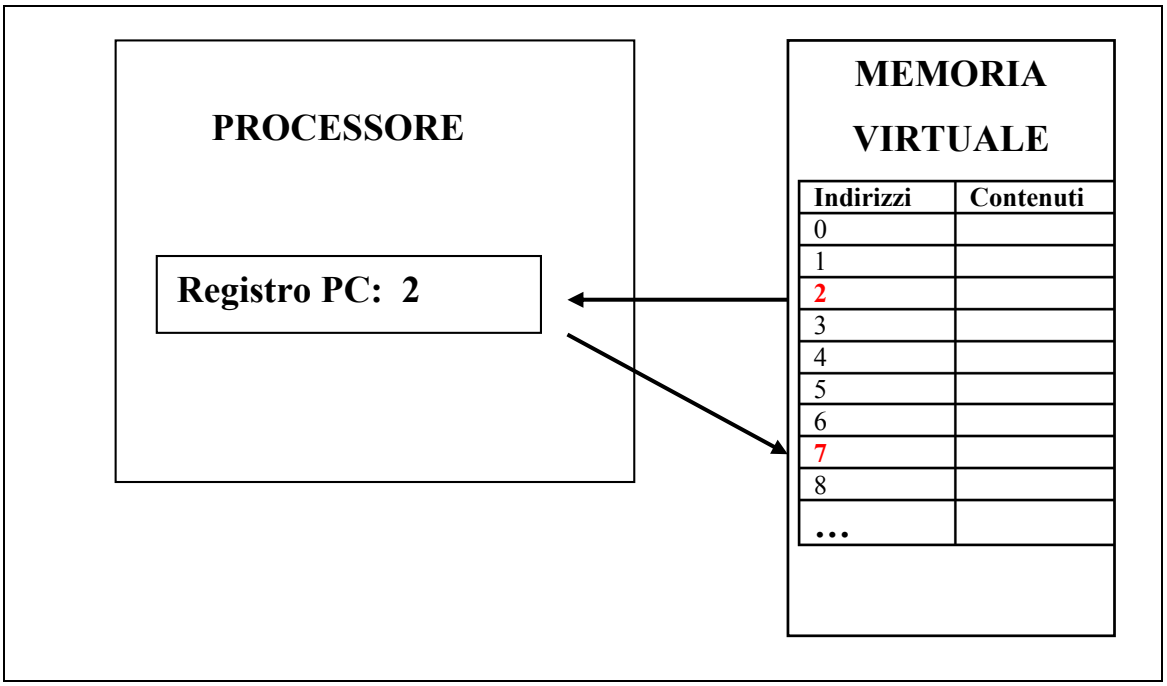

Figura 1 – Modello di esecuzione del programma: il Processore legge un'istruzione all'indirizzo virtuale 2 e modifica un operando all'indirizzo virtuale 7

La memoria effettivamente presente sul calcolatore è chiamata **memoria fisica**, e i suoi indirizzi sono detti **indirizzi fisici**. E' evidente che la corretta esecuzione del programma di figura 1 richiederebbe di caricarlo nella memoria fisica a partire dall'indirizzo 0, in modo da far coincidere la memoria virtuale e la memoria fisica durante l'esecuzione.

Tuttavia, normalmente ciò non è possibile e quindi *la memoria virtuale e la memoria fisica non coincidono*.

Il meccanismo più utilizzato di trasformazione tra indirizzi virtuali e fisici che risolve questo problema è la ***rilocazione dinamica tramite paginazione***.

3. Problemi di allocazione della memoria

In generale non è possibile caricare un programma in modo che la memoria virtuale e quella fisica coincidano perfettamente, a causa dei seguenti motivi:

1. Nella memoria fisica devono risiedere simultaneamente sia il sistema operativo che diversi processi.
2. E' conveniente mantenere nella memoria fisica una sola copia di parti di programmi che sono identici in diversi processi (condivisione della memoria).
3. L'allocazione di memoria fisica a diversi processi, la variazione delle dimensioni dei processi, ecc … tendono a creare buchi nella memoria fisica; questo fenomeno, illustrato in figura 2, è detto ***frammentazione della memoria***. Per ridurre la frammentazione è utile allocare i programmi suddividendoli in "pezzi" più piccoli;
4. ***Le dimensioni della memoria fisica possono essere insufficienti a contenere la memoria virtuale*** di tutti i processi esistenti, per cui è necessario eseguire i programmi anche se non sono completamente contenuti nella memoria fisica, quindi anche se la memoria fisica disponibile è inferiore alla somma delle dimensioni virtuali dei processi creati.

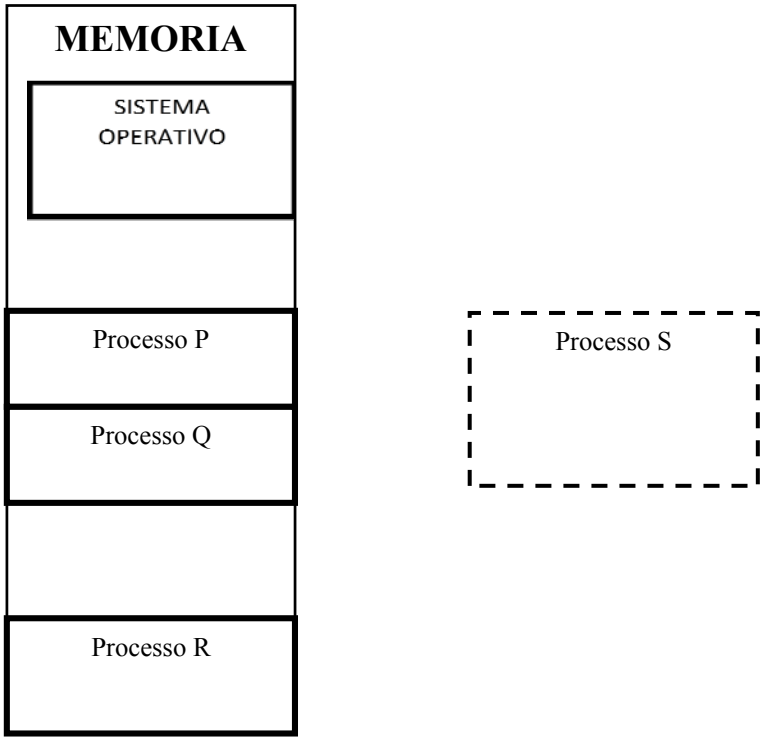

Figura 2 – Frammentazione della memoria:
lo spazio tra il Sistema Operativo e il processo P oppure tra i processi Q ed R sono troppo piccoli per allocarvi un nuovo processo S; inoltre il processo P non può crescere.

A causa di questi motivi la corrispondenza ottimale tra la memoria fisica e la memoria virtuale dei processi che il SO deve realizzare è simile a quella di figura 3, dove si è ipotizzato che nella memoria fisica sia stato caricato prima il Sistema Operativo, poi il processo P nella sua configurazione iniziale, poi il processo Q nella sua configurazione iniziale, poi è stato allocato nuovo spazio per la crescita del processo P, riempiendo tutta la memoria fisica, e infine il processo Q è cresciuto ma la memoria virtuale aggiuntiva non ha potuto essere caricata nella memoria fisica.

Naturalmente, dato che ogni programma è costruito per funzionare come se il modello fosse quello ideale, con indirizzi fisici identici agli indirizzi virtuale, devono esistere dei meccanismi che permettono ai programmi rappresentati in figura 3 di funzionare come se il modello di corrispondenza fosse quello ideale.

Tali meccanismi sono ovviamente ***"trasparenti"*** al programmatore e al compilatore/collegatore, perché il programma è costruito secondo il modello memoria virtuale, non di quella fisica. In sostanza lo scopo di tali meccanismi è proprio quello di permettere di creare i programmi secondo il comodo modello della memoria virtuale, anche se la situazione della memoria fisica del calcolatore durante l'esecuzione è molto diversa.

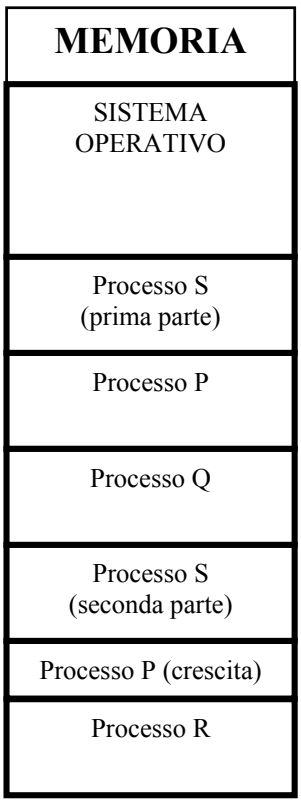

Figura 3 – La suddivisione dei processi P ed S in porzioni non contigue risolve i problemi di frammentazione di figura 2

4. Principi della paginazione (vedi anche Patterson, pagine 323 -327)

Il meccanismo di paginazione permette di ottenere una gestione migliore della memoria eliminando l'ipotesi che un programma sia memorizzato, durante l'esecuzione, in una zona contigua della memoria fisica. Questo meccanismo permette di eliminare i problemi di frammentazione della memoria, permette di estendere la dimensione della memoria di un processo, anche se non c'è spazio libero immediatamente dopo, purché ci siano altre aree libere sufficienti anche se non contigue (ad esempio, permette di estendere il processo P e di allocare il processo S nel modo illustrato in figura 3).

L'idea base della paginazione è molto semplice e consiste nelle seguenti regole (figura 4):

- La memoria virtuale del programma viene suddivisa in porzioni di lunghezza fissa dette pagine virtuali aventi una lunghezza che è una potenza di 2 (in figura le pagine sono lunghe 4K).
- La memoria fisica viene anch'essa suddivisa in pagine fisiche della stessa dimensione delle pagine virtuali.
- Le pagine virtuali di un programma da eseguire vengono caricate in altrettante pagine fisiche, prese arbitrariamente e non necessariamente contigue.

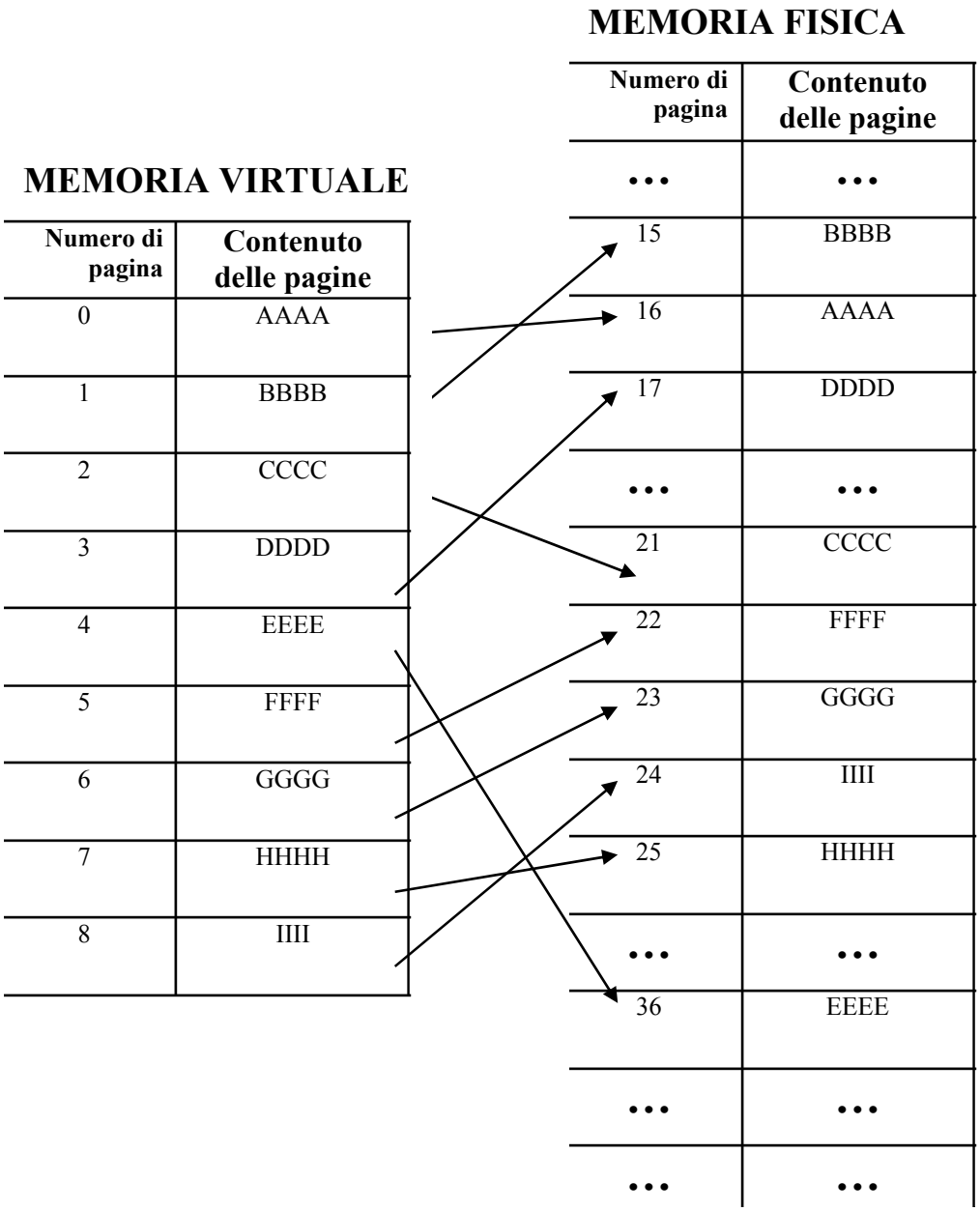

Figura 4 – Paginazione: corrispondenza tra pagine virtuali e pagine fisiche

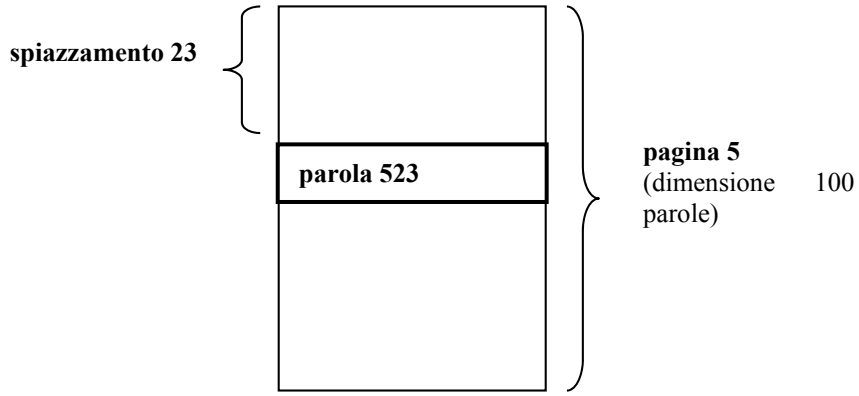

Figura 5 – La scomposizione di un indirizzo decimale in numero di pagina e spiazzamento

Un indirizzo virtuale del programma viene considerato costituito da un **numero di pagina virtuale NPV** e da uno **spiazzamento** (**offset**) all'interno della pagina; l'indirizzo virtuale viene trasformato nel corrispondente indirizzo fisico, che a sua volta può essere considerato costituito da un **numero di pagina fisica NPF** e da uno spiazzamento, sostituendo al valore di NPV il valore della corrispondente pagina fisica NPF e lasciando inalterato lo spiazzamento[1].
E' fondamentale, per capire il meccanismo di paginazione, tenere presente che *il fatto di avere stabilito che le pagine abbiano una lunghezza che è potenza di 2 permette di considerare un indirizzo come composto dal concatenamento di un numero di pagina e di uno spiazzamento*.

Questo aspetto è facilmente comprensibile facendo riferimento al sistema decimale invece di quello binario; consideriamo quindi il seguente esempio basato su indirizzi decimali:

- supponiamo di avere una memoria di 1000 indirizzi decimali da 0 a 999 e di suddividerla in 10 pagine di lunghezza 100 –
- allora ogni indirizzo può essere considerato costituito da un numero di pagina di una cifra (da 0 a 9) e da uno spiazzamento di due cifre (da 00 a 99);
- ad esempio, l'indirizzo 523 puo' essere considerato come il concatenamento del numero di pagina 5 e dello spiazzamento 23 (figura 5).

Per realizzare la paginazione è necessario che il sistema esegua la trasformazione degli indirizzi virtuali in indirizzi fisici, cioè la sostituzione dei numeri di pagina virtuale con i corrispondenti numeri di pagina fisica, dato che lo spiazzamento rimane inalterato.

Ciò è ottenuto creando una **Tabella delle Pagine (Page Table – PT)** che contiene tante righe quante sono le pagine virtuali di un programma, ponendo in tali righe i numeri delle pagine fisiche corrispondenti e sostituendo, prima di accedere alla memoria, il numero di pagina virtuale con il corrispondente numero di pagina fisico.

In figura 6 e' mostrato questo meccanismo con riferimento all'esempio di figura 4.

In figura 7 è mostrato lo stesso meccanismo ipotizzando che siano stati creati 2 processi P e Q con le seguenti caratteristiche:

- dimensione di P: 4 pagine
- dimensione di Q: 5 pagine

[1] Nel caso in cui la porzione spiazzamento dell'indirizzo sia un multiplo di 4 bit (come è il caso con pagine di 4K), l'indirizzo esadecimale si suddivide in maniera particolarmente facile nelle due porzioni di numero pagina e spiazzamento. Questa facilità riguarda solo noi che utilizziamo la notazione esadecimale; per la macchina la condizione importante è che le pagine abbiano una dimensione che è una potenza di 2.

MEMORIA VIRTUALE

Numero di pagina	Contenuto delle pagine
0	AAAA
1	BBBB
2	CCCC
3	DDDD
4	EEEE
5	FFFF
6	GGGG
7	HHHH
8	IIII

TABELLA DELLE PAGINE

NPV	NPF
0	**16**
1	**15**
2	**21**
3	**17**
4	**36**
5	**22**
6	**23**
7	**25**
8	**24**

MEMORIA FISICA

Numero di pagina	Contenuto delle pagine
•••	•••
15	BBBB
16	AAAA
17	DDDD
•••	•••
21	CCCC
22	FFFF
23	GGGG
24	IIII
25	HHHH
•••	•••
36	EEEE
•••	•••
•••	•••

Figura 6 – La Tabella delle pagine per rappresentare la corrispondenza tra pagine virtuali e pagine fisiche (con riferimento all'esempio di figura 4)

MEMORIA VIRTUALE di P

Numero di pagina	Contenuto delle pagine
0x00000	AAAA
0x00001	BBBB
0x00002	CCCC
0x00003	DDDD

TABELLA delle PAGINE di P

NPV	NPF
0x0000**0**	0x0000**4**
0x0000**1**	0x0000**5**
0x0000**2**	0x0000**6**
0x0000**3**	0x0000**7**

MEMORIA VIRTUALE di Q

Numero di pagina	Contenuto delle pagine
0x00000	RRRR
0x00001	SSSS
0x00002	TTTT
0x00003	UUUU
0x00004	VVVV

TABELLA delle PAGINE di Q

NPV	NPF
0x0000**0**	0x0000**8**
0x0000**1**	0x0000**9**
0x0000**2**	0x0000**A**
0x0000**3**	0x0000**B**
0x0000**4**	0x0000**C**

MEMORIA FISICA

Numero di pagina	Contenuto delle pagine
0x0000**0**	S.O.
0x00001	S.O.
0x00002	S.O.
0x00003	S.O.
0x00004	AAAA
0x00005	BBBB
0x00006	CCCC
0x00007	DDDD
0x00008	RRRR
0x00009	SSSS
0x0000A	TTTT
0x0000B	UUUU
0x0000C	VVVV
0x0000D	non usata
0x0000E	non usata
0x0000F	non usata

Figura 7 – Due processi e le relative tabelle delle pagine

Condivisione delle pagine

Talvolta è utile o necessario mantenere in memoria una sola copia di pagine che sono condivise da più processi. Un tipico esempio di opportunità di condivisione di memoria tra due processi è il caso in cui i due processi eseguono lo stesso programma; è evidente che il codice del programma può essere memorizzato una volta sola, dato che i due processi si limitano a leggerlo senza modificarlo.
Tramite paginazione la condivisione della memoria si realizza molto semplicemente: basta imporre che la porzione condivisa della memoria sia costituita da un numero intero di pagine e quindi mappare, nelle tabelle delle pagine dei processi interessati, sulla stessa pagina fisica le pagine virtuali da condividere.

Esempio 1 (continua)

Supponiamo che i due processi P e Q di figura 7 condividano le loro due pagine virtuali iniziali, che per ipotesi contengono il codice di uno stesso programma. In questo caso la situazione della memoria è quella

di figura 8, nella quale si è ipotizzato che il processo P sia stato creato per primo e quindi, quando è stato creato il processo Q, invece di allocare nuova memoria per le prime due pagine virtuali, ci si è limitati a indicare, nella sua tabella delle pagine, il riferimento alle due pagine fisiche già allocate per il processo P .

Per mettere più in evidenza l'effetto della condivisione, nella figura 8 le pagine fisiche non più utilizzate sono state lasciate libere, invece di compattare la memoria allocando la porzione restante del processo Q subito dopo P ■

Protezione delle pagine

Dato che molti processi utilizzano contemporaneamente la memoria fisica, è necessario garantire che ogni processo sia limitato a svolgere operazioni esclusivamente sulla propria memoria. Molti linguaggi, in particolare il linguaggio C, permettono un uso molto libero dei puntatori, e un programma potrebbe generare indirizzi virtuali scorretti, che il processo di traduzione e collegamento non sarebbero in grado di individuare. La protezione è resa ancora più necessaria dall'esigenza di condivisione della memoria tra processi.

Oltre a garantire che un processo operi solo sulla propria memoria è necessario controllare che un processo operi correttamente sulla memoria alla quale può accedere. A questo scopo il meccanismo di protezione più diffuso consiste nell'associare ad ogni pagina un'informazione che indica se il processo può utilizzarla in lettura (**R**), scrittura (**W**) oppure esecuzione (**X**) (quest'ultimo tipo di accesso significa che la pagina contiene istruzioni che devono essere lette ed eseguite). Il tipo di diritto di accesso a una pagina può essere inserito nella riga corrispondente della tabella delle pagine e deve essere gestito anche dall'Hardware, che, in presenza di violazione di un diritto di accesso, genera un ***interrupt di violazione della memoria.***

MEMORIA VIRTUALE di P

Numero di pagina	Contenuto delle pagine
0x00000	AAAA
0x00001	BBBB
0x00002	CCCC
0x00003	DDDD

TABELLA delle PAGINE di P

NPV	NPF
0x00000	0x00004
0x00001	0x00005
0x00002	0x00006
0x00003	0x00007

MEMORIA VIRTUALE di Q

Numero di pagina	Contenuto delle pagine
0x00000	AAAA
0x00001	BBBB
0x00002	TTTT
0x00003	UUUU
0x00004	VVVV

TABELLA delle PAGINE di Q

NPV	NPF
0x00000	0x00004
0x00001	0x00005
0x00002	0x0000A
0x00003	0x0000B
0x00004	0x0000C

MEMORIA FISICA

Numero di pagina	Contenuto delle pagine
0x00000	S.O.
0x00001	S.O.
0x00002	S.O.
0x00003	S.O.
0x00004	AAAA
0x00005	BBBB
0x00006	CCCC
0x00007	DDDD
0x00008	non usata
0x00009	non usata
0x0000A	TTTT
0x0000B	UUUU
0x0000C	VVVV
0x0000D	non usata
0x0000E	non usata
0x0000F	non usata

Figura 8 – I processi P e Q dell'esempio di figura 7 condividono le due pagine iniziali

5. Gestione di pagine virtuali non residenti in memoria

Talvolta il numero di pagine virtuali dei processi eccede il numero di pagine fisiche disponibili. La gestione della memoria virtuale deve in questi casi permettere di eseguire un programma anche se non possiamo caricare tutte le sue pagine virtuali in memoria[2]. Il meccanismo di virtualizzazione si basa sulla paginazione e sui seguenti principi:
Durante l'esecuzione di un programma solo un certo numero delle sue pagine virtuali è caricato in altrettante pagine fisiche; tali pagine sono dette **residenti** Ad ogni accesso alla memoria da parte del programma si controlla che l'indirizzo virtuale generato appartenga a una pagina residente, altrimenti si

[2] Il meccanismo che risolve questo problema è quello che ha dato il nome alla memoria virtuale di un processo, perché il processo viene eseguito anche se una parte della sua memoria non esiste fisicamente – la memoria esiste solo virtualmente.

produce un interrupt di segnalazione di errore, detto **page-fault**, e il processo viene sospeso in attesa che la pagina contenente l'indirizzo virtuale richiesto venga caricata dal disco[3].
Se necessario, una pagina già residente viene scaricata su disco per liberare una pagina fisica che possa contenere una nuova pagina virtuale.
Con questo modo di operare il programma viene eseguito utilizzando solo un limitato numero di pagine fisiche, indipendentemente dalle sue dimensioni virtuali. *La condizione fondamentale affinché questo meccanismo funzioni è che il numero di richieste di accessi alla memoria che causano un page fault sia basso rispetto al numero complessivo di accessi (ad esempio, <5%).* Perciò, prima di considerare i dettagli della realizzazione del meccanismo è necessario fare alcune considerazioni relativamente al comportamento dei programmi.

Caratteristiche dell'accesso dei programmi alla memoria

E' stato rilevato statisticamente che i programmi tendono ad esibire, nel modo di accedere alla memoria, una caratteristica detta **località**, per cui la distribuzione degli accessi nello spazio e nel tempo non è omogenea. In particolare, parliamo di località temporale e di località spaziale in base alle seguenti definizioni.

Località temporale: indica che un programma accederà, nel prossimo futuro, gli indirizzi che ha referenziato nel passato più recente. Questo tipo di località è tipicamente dovuto all'esecuzione di cicli, che rileggono ripetutamente le stesse istruzioni e gli stessi dati.

Località spaziale: indica che un programma accede con maggiore probabilità indirizzi vicini a quello utilizzato più recentemente: Questo tipo di località è dovuto alla sequenzialità delle istruzioni e all'attraversamento di strutture come gli array.

Naturalmente, il grado di località dipende dai programmi e differisce da un programma all'altro.

Questo comportamento può essere caratterizzato tramite la nozione di **Working Set**. Il Working Set di ordine k di un programma è l'insieme delle pagine referenziate durante gli ultimi k accessi alla memoria. Per k sufficientemente grande il Working Set di un programma varia molto lentamente a causa delle proprietà di località esposte sopra, quindi se si mantengono in memoria **le k pagine accedute più recentemente** è molto probabile che il prossimo accesso sia all'interno di tali pagine, cioè che non si verifichi un page fault. Il sistema operativo deve quindi tentare di mantenere in memoria un numero di pagine per ogni processo sufficiente a mantenere accettabile la frequenza dei page-fault; inoltre tali pagine dovrebbero essere quelle accedute più recentemente.

In sistemi relativamente scarichi, nei quali ad ogni processo può essere allocata tutta la memoria che richiedono, questo compito è relativamente facile; man mano che il carico del sistema aumenta diventa sempre più difficile e critico soddisfare questo requisito. Questo argomento verrà trattato nel capitolo relativo all'allocazione e deallocazione della memoria fisica.

Immagine su disco e immagine in memoria

Un aspetto importante della virtualizzazione consiste nel fatto che la memoria virtuale di un processo non è contenuta completamente nella memoria fisica durante l'esecuzione; *la parte non contenuta in memoria fisica deve esistere su disco*.
In particolare, alcune parti della memoria virtuale del processo possiedono un'immagine su disco che è costituita direttamente dal file contenente l'eseguibile: ad esempio, la parte che costituisce il codice. Dato che il codice non viene modificato, anche se una pagina di codice non è residente in memoria, la sua immagine esiste però nel file eseguibile.
Per le parti del processo che vengono modificate e/o allocate dinamicamente invece è necessario che esista un file particolare, detto **swap file**, sul quale viene salvata l'immagine delle pagine quando queste vengono scaricate. Dato che la suddivisione del processo in "parti" verrà trattata al prossimo paragrafo, qui non analizziamo come essa sia realizzata; ci basta sapere che tutte le pagine di un processo sono in corrispondenza con le corrispondenti pagine su file.
Infine, per le parti del processo che vengono modificate e/o allocate dinamicamente, quando una pagina viene scaricata, il sistema operativo deve decidere se tale pagina deve essere riscritta sul disco perché è stata modificata oppure no: per permettere questa scelta la MMU possiede in genere un bit per ogni riga, detto **bit di modifica (dirty bit)**, azzerato quando la pagina viene caricata in memoria e posto a uno ogni

[3] Un processore sul quale si vuole far funzionare il meccanismo di virtualizzazione della memoria deve avere la capacità di interrompere l'esecuzione di un'istruzione a metà e di rieseguire più tardi la stessa istruzione come se non avesse mai iniziato ad eseguirla, perché la generazione di un page fault può causare la sospensione dell'esecuzione di un istruzione dopo che questa è iniziata, ad esempio durante l'accesso a un operando (restartable instructions).

volta che viene scritta una parola di tale pagina. Ovviamente, le pagine che al momento dello scaricamento hanno il bit di modifica uguale a 0 non richiedono di essere ricopiate sul disco.

Caricamento iniziale del programma – demand paging

Un programma può essere lanciato in esecuzione anche se nessuna sua pagina è residente in memoria. La sua Tabella delle Pagine dovrà indicare che nessuna pagina virtuale è residente. Ovviamente, quando il processore tenta di leggere la prima istruzione genera immediatamente un page fault, e la relativa pagina virtuale verrà caricata in memoria. A questo punto inizia l'esecuzione del programma, che genererà progressivamente dei page fault a fronte dei quali verranno caricate nuove pagine senza scaricarne alcuna; dopo un certo numero di page fault il programma avrà un numero di pagine residenti tale da ridurre i page fault a quelli statisticamente accettabili.