Livello del Sistema Operativo

Prof. Ivan Lanese

Livello del Sistema Operativo

- Le risorse messe a disposizione dall'architettura del calcolatore vengono gestite da una componente software che prende il nome di sistema operativo
- Tra i compiti principali del sistema operativo possiamo elencare:
  - Gestione della memoria
  - Gestione dei processi (programmi in esecuzione)
  - Gestione delle periferiche di I/O
- Sono compiti complessi, critici, e richiesti da molte applicazioni

Sistema Operativo e programmi applicativi

- Possiamo vedere il sistema operativo come l'interfaccia fra il calcolatore ed i programmi applicativi
- I programmi applicativi sono componenti software dedicate ad una precisa finalità (wordprocessor per editare documenti, browser per accedere a risorse disponibili su web, ...)
- Il sistema operativo è un componente software trasversale, responsabile dell'esecuzione dei programmi applicativi

Struttura di un tipico sistema operativo

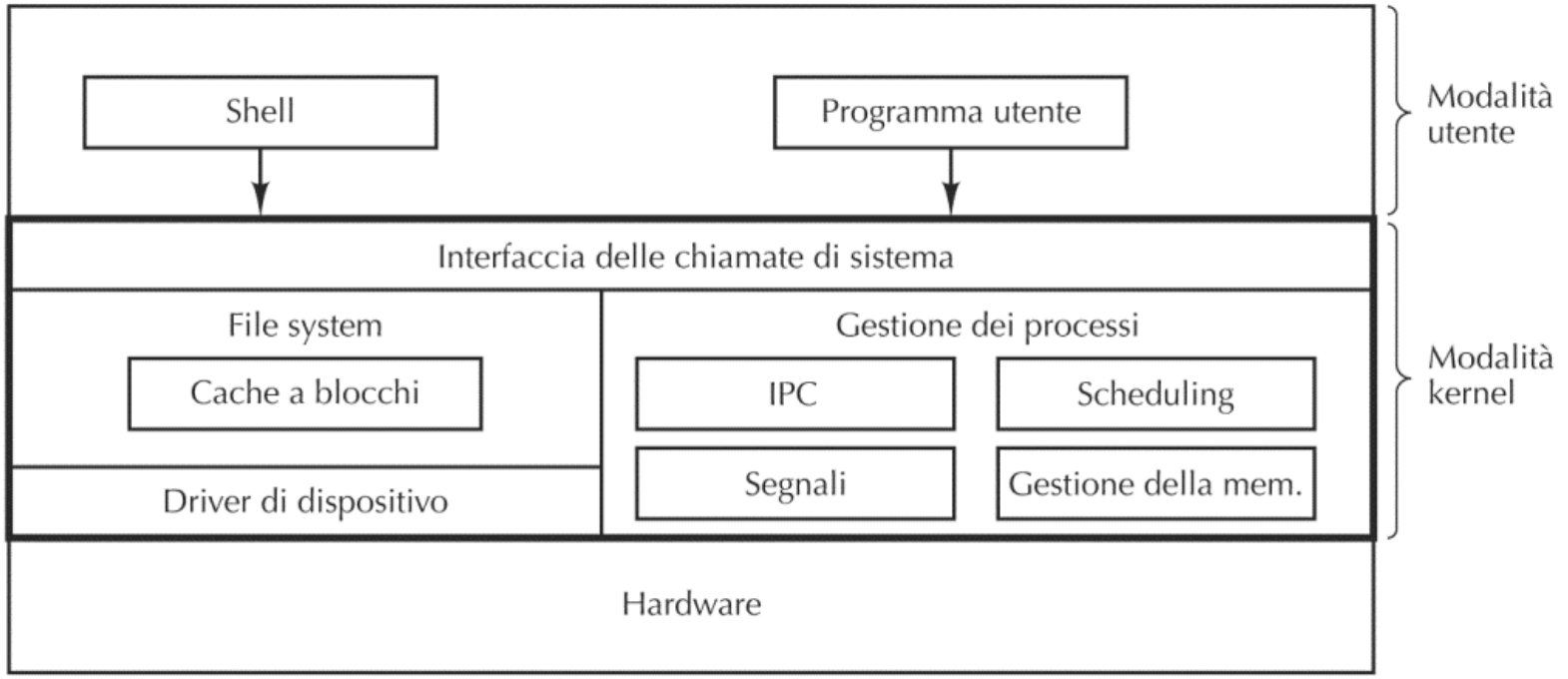

Figura 6.31 Struttura di un comune sistema UNIX.

- Shell indica una specifica modalità per interagire con il sistema operativo (messa a disposizione dal cosiddetto terminale)
- Shell e programmi applicativi interagiscono con il sistema operativo tramite "chiamate di sistema" (ad esempio, operazioni di accesso ai file, esecuzione di un programma, interruzione di un programma, …)

Storiella

- Negli anni '60 i calcolatori avevano una memoria estremamente limitata.. Ed un gruppo di ricercatori di Manchester ebbe una idea..
  - Avevano un calcolatore con una memoria con 4K locazioni contenenti parole da 16 bit
  - Gli "indirizzi fisici" di memoria andavano quindi da 0 e 4095
  - Avendo una architettura a 16 bit, anche gli indirizzi di memoria usavano 16 bit
    - ma gli indirizzi da 4096 a 65535 non venivano usati!
  - Pensarono di realizzare una memoria ipotetica (detta "memoria virtuale") da 64K locazioni usando tutti gli indirizzi da 0 a 65535
    - Tali indirizzi vengono detti "indirizzi virtuali"
  - In un certo istante, solo un blocco di dimensione 4K di tale memoria veniva posto in memoria primaria, il resto rimaneva salvato in memoria secondaria (tale blocco è detto "pagina")

Paginazione

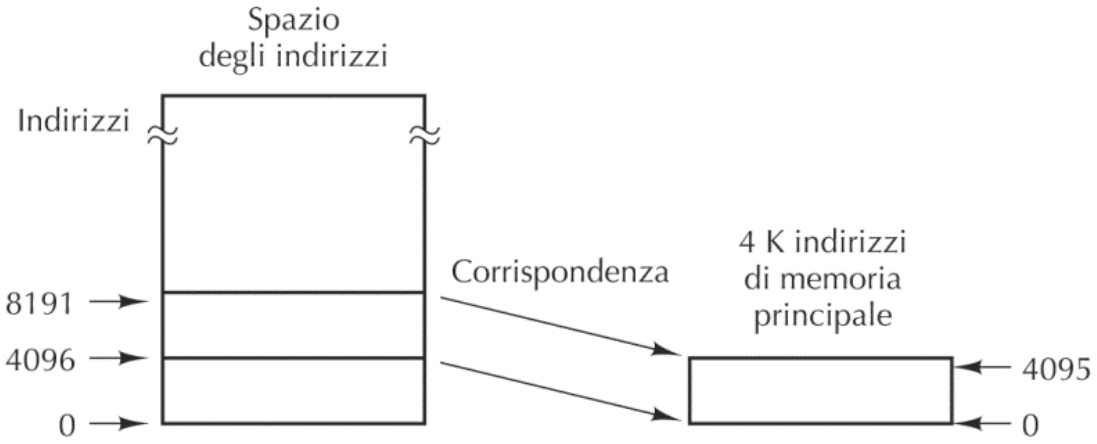

Figura 6.2 Corrispondenza tra gli indirizzi virtuali da 4096 a 8191 e gli indirizzi di memoria principale da 0 a 4095.

- Nell'immagine viene riportato il caso di inserimento nella memoria fisica della "pagina" con gli indirizzi virtuali da 4096 a 8191
  - Problema: bisogna "mappare" gli indirizzi virtuali 4092-8191 agli indirizzi fisici 0-4095
- Lo soluzione a tale problema viene gestita dal sistema operativo (vedremo come) in modo "trasparente" al programma applicativo...
  - ...che quindi ha l'illusione di usare un unico ampio spazio di indirizzamente coincidente con la memoria virtuale

Implementazione della paginazione

- La memoria virtuale viene divisa in "pagine", tutte di uguale dimensione, costituite da un numero di locazioni corrispondente ad una potenza di due
  - Se la dimensione di pagina è 2^k, si usano esattamente k bit per indirizzare una locazione all'interno di una pagina (offset)
  - Nell'esempio consideriamo pagine da 4K (indirizzi da 12 bit)
- La memoria viene suddivisa in parti di medesima dimensione chiamate "blocchi di memoria"
  - Ogni blocco di memoria ha un suo indice (i blocchi sono numerati da 0 a 7 nell'esempio)

Pagina	Indirizzi virtuali
15	61440 – 65535
14	57344 – 61439
13	53248 – 57343
12	49152 – 53247
11	45056 – 49151
10	40960 – 45055
9	36864 – 40959
8	32768 – 36863
7	28672 – 32767
6	24576 – 28671
5	20480 – 24575
4	16384 – 20479
3	12288 – 16383
2	8192 – 12287
1	4096 – 8191
0	0 – 4095

(a)

Blocco di memoria	Primi 32 K di memoria principale: Indirizzi fisici
7	28672 – 32767
6	24576 – 28671
5	20480 – 24575
4	16384 – 20479
3	12288 – 16383
2	8192 – 12287
1	4096 – 8191
0	0 – 4095

(b)

Figura 6.3 (a) Primi 64 KB degli indirizzi virtuali suddivisi in 16 pagine da 4 KB. (b) Memoria principale di 32 KB suddivisa in otto blocchi di memoria di 4 KB.

Traduzione da indirizzo virtuale ad indirizzo reale

- Esiste hardware dedicato (chiamato Memory Management Unit – MMU) che si occupa di tradurre un indirizzo virtuale nel corrispondente indirizzo reale
- Si usa una "tabella delle pagine" che indica, per ogni pagina, se questa è a momento in memoria, e in caso positivo indica in quale blocco
  - L'indice di pagina si ottiene guardando l'indirizzo virtuale, esclusi i k bit usati per identificare l'offset
  - La tabella indica quali bit inserire al posto di questo indice di pagina

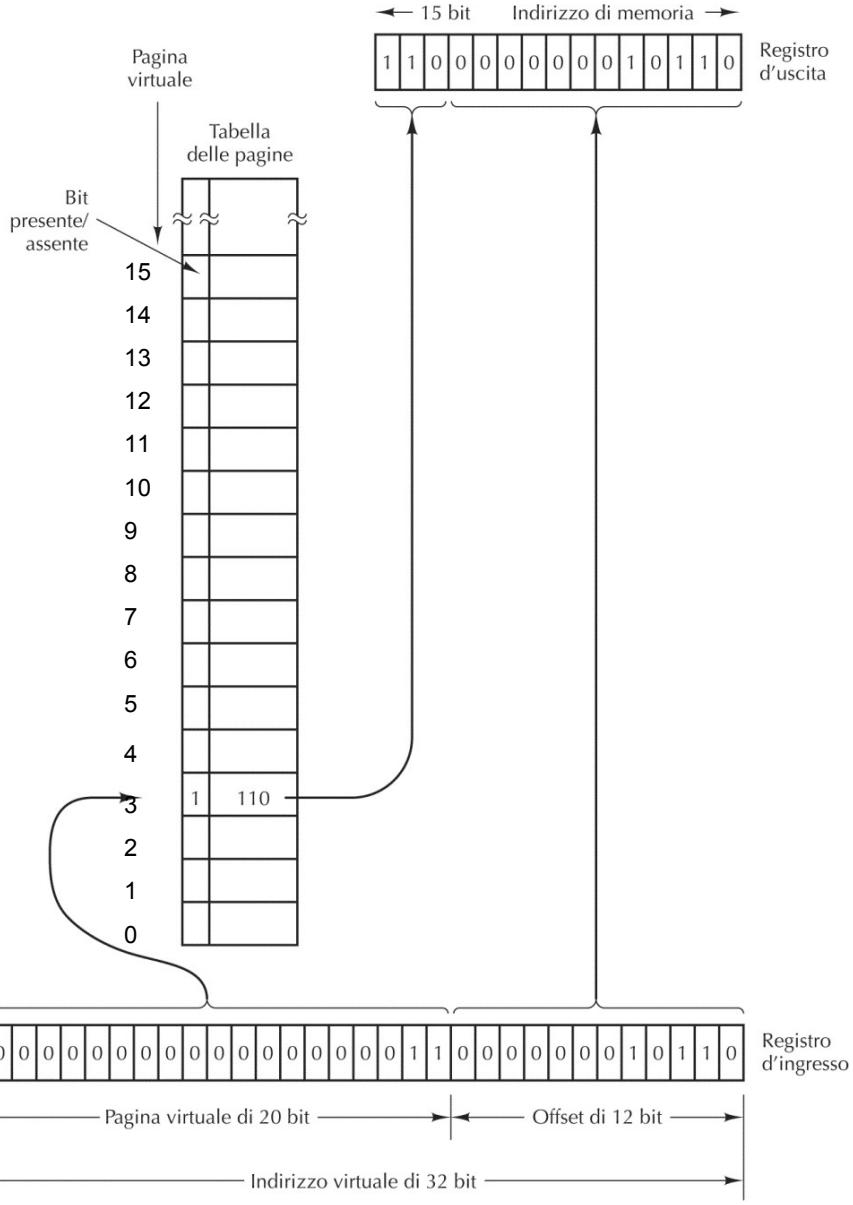

Figura 6.4 Formazione di un indirizzo fisico a partire da un indirizzo virtuale.

Algoritmi di paginazione

- Le pagine al momento in memoria vengono dette "working set"
- Cosa succede quando si accede ad una pagina fuori dal working set?
  - Avviene un trap detto "page fault"
- Il gestore di tale trap deve togliere dalla memoria una pagina, per liberare un blocco per la pagina acceduta
- La scelta viene presa dai cosiddetti algoritmi di paginazione

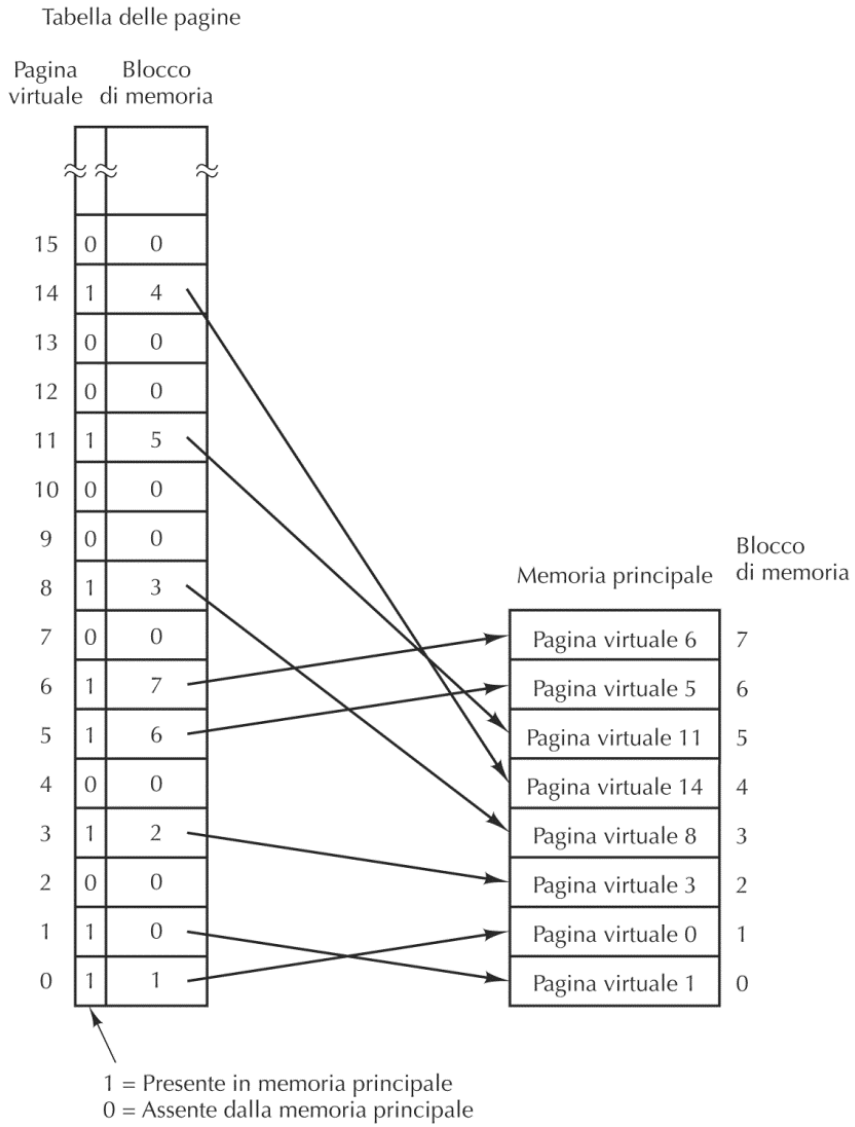

Figura 6.5 Assegnamento di 16 pagine virtuali in una memoria principale composta di otto blocchi.

LRU (Least Recently Used) e FIFO (First-In First-Out)

- Vediamo due tipici algoritmi di paginazione
  - LRU (Least Recently Used)
    - Si rimuove dalla memoria la pagina che da più tempo non viene utilizzata
    - Può essere implementato tramite una "lista":
      - quando si accede una pagina la si re-inserisce in coda
      - quando serve rimuovere una pagina, si sceglie quella in testa
  - FIFO (First-In First-Out)
    - Si estromette la pagina che da più tempo è stata inserita in memoria
    - Può essere implementato nel seguente modo: ogni blocco ha un contatore, inizializzato a 0 quando il blocco riceve una pagina, ed incrementato ad ogni page fault
    - Quando serve liberare un blocco si sceglie quello con il contatore più grande

Dirty bit

- Quando si rende necessario liberare un blocco, non sempre è necessario ricopiare in memoria secondaria la pagina contenuta
- Infatti, se la pagina non è stata modificata, non ci sono ragioni per reinserirla uguale identica in memoria secondaria
- Per evitare di perdere tempo in inutili operazioni di ri-scrittura, si usa la tecnica del "dirty bit"
  - Ogni blocco in memoria primaria ha un suo bit associato
  - Quando si carica una nuova pagina nel blocco, si pone a 0 il bit corrispondente
  - Quando la pagina in un blocco viene acceduta in scrittura, il bit viene posto a 1
  - Quando si deve liberare il blocco, si salva la corrispondente pagina in memoria secondaria solo se il bit è a 1
  - Il bit indica se la pagina è stata "sporcata" (ecco perché dirty bit)

Frammentazione interna

- Supponiamo che ad un programma serva una quantità di memoria che non sia un multiplo della dimensione delle pagine
  - L'ultima pagina non verrà completamente utilizzata
  - Esempio:
    - Consideriamo un programma che richiede 26.000 byte
    - In esecuzione su una macchina con pagine da 4K byte (4.096 byte)
    - Il programma userà in modo completo 6 pagine
      - 6 x 4.096 = 24.576
    - Nell' ultima pagina, la settima, si useranno solamente 26.000-24.576=1.424 byte
    - I restanti 4.096-1.424=2.672 byte non verranno utilizzati
- Questo fenomeno viene chiamato "frammentazione interna"

Segmentazione

- Finora abbiamo pensato a una memoria (virtuale o reale) di k locazioni come a un unico spazio di indirizzamento con indirizzi da 0 a k-1
- Storicamente è stato usato anche un modello in cui convivono diversi spazi di indirizzamento chiamati "segmenti"
  - E' come se ci fossero contemporaneamente utilizzabili tante diverse memorie, ognuna chiamata segmento
  - Nell'esempio sono riportati 5 diversi segmenti, di diverse dimensioni, usati contemporaneamente da un ipotetico compilatore
  - I segmenti sono visibili al programmatore

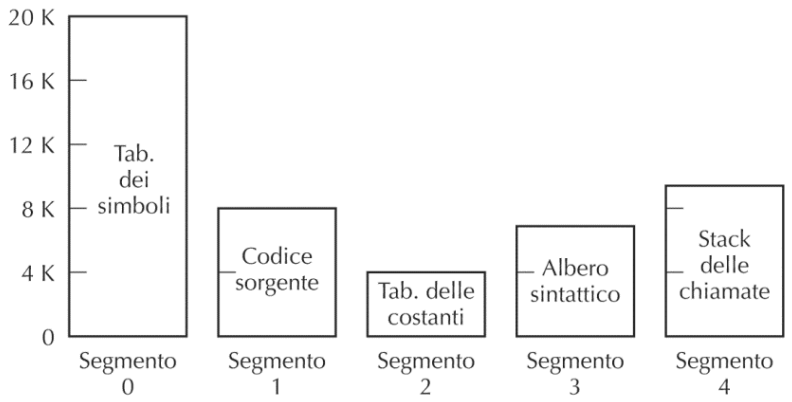

Figura 6.8 Una memoria segmentata consente a ciascuna tabella di crescere o decrescere indipendentemente dalle altre.

Segmentazione (continua)

- Ogni segmento di dimensione n, usa gli indirizzi da 0 a n-1
- Per disambiguare l'uso di indirizzi presenti in più segmenti (ad esempio l'indirizzo 0 è presente in tutti i segmenti!) gli indirizzi dovranno essere arricchiti con l'indicazione del relativo segmento
  - L'indirizzo (n,k) indica la locazione di indirizzo k del segmento n

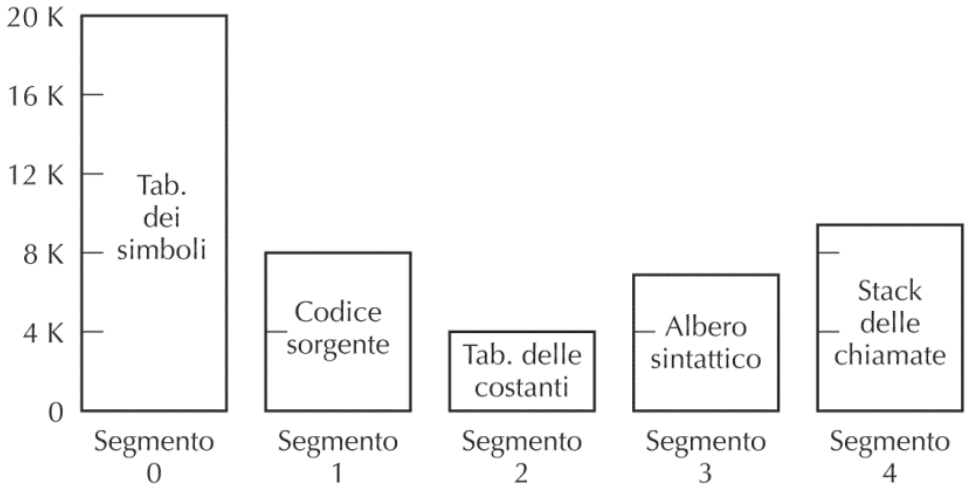

Figura 6.8 Una memoria segmentata consente a ciascuna tabella di crescere o decrescere indipendentemente dalle altre.

Implementazione della segmentazione

- Come nella paginazione, noi potremmo pensare di tenere in memoria solo alcuni segmenti al momento utilizzati, lasciando in memoria secondaria i segmenti non necessari
  - Quando serve inserire un nuovo segmento se ne toglie un altro
    - Questa tecnica è detta "swapping"
    - A seguito di swapping si possono generare zone non utilizzate (questo fenomeno viene detto "frammentazione esterna")

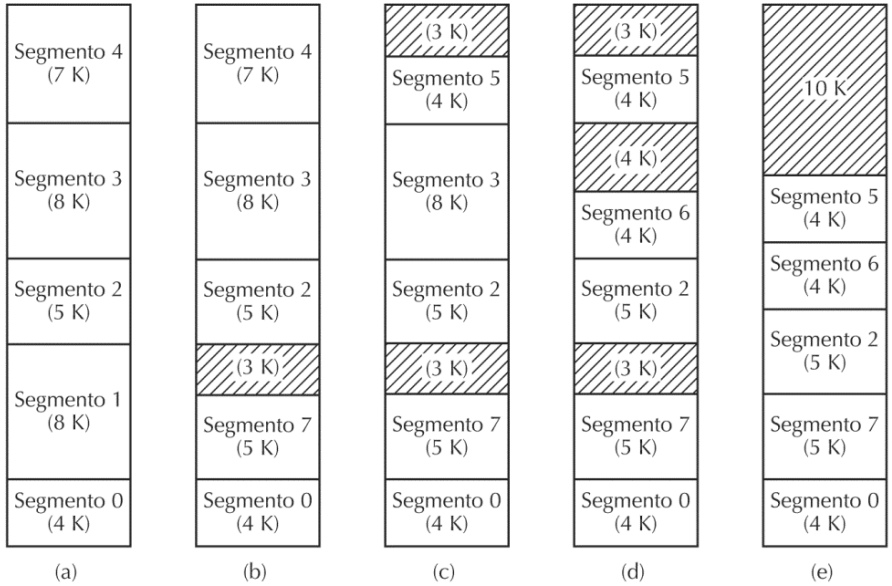

Figura 6.10 (a)-(d) Incremento della frammentazione esterna. (e) Rimozione della frammentazione esterna mediante compattamento.

Gestione della frammentazione

- Le zone non utilizzate sono dette "lacune". A volte si rende necessario eliminare le lacune compattando i segmenti
- Si sono ideate varie tecniche di "swapping" per cercare di ridurre la frammentazione esterna. Tra le tecniche più usate menzioniamo:
  - **Best fit**: si inserisce il nuovo segmento nella lacuna più piccola sufficiente per il segmento da inserire
  - **First fit**: si scorrono le lacune circolarmente selezionando la prima sufficientemente grande per il segmento da inserire
  - First fit è più veloce nella scelta, ed inoltre limita la creazione di piccole e inutili lacune

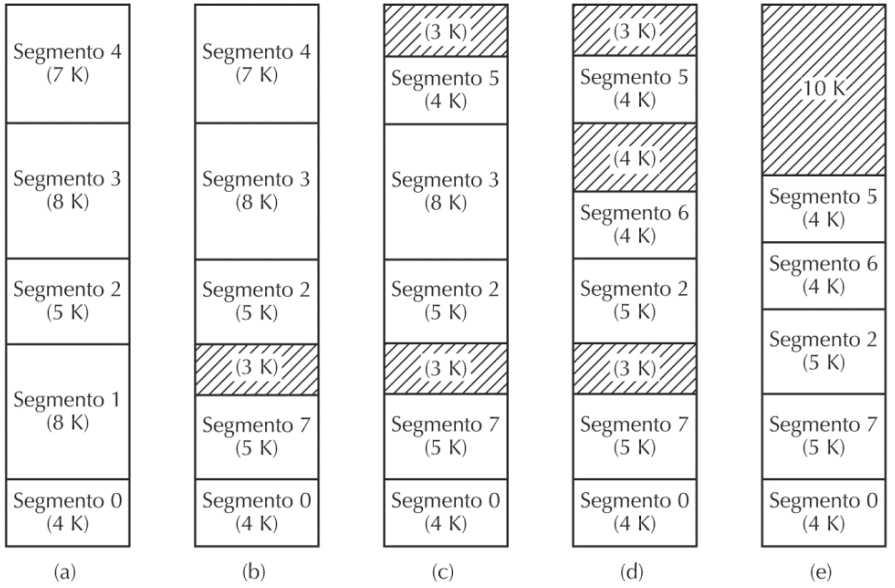

Figura 6.10 (a)-(d) Incremento della frammentazione esterna. (e) Rimozione della frammentazione esterna mediante compattamento.

Combinazione fra segmentazione e paginazione

- Un modo per evitare la frammentazione esterna è combinare segmentazione e paginazione
  - I segmenti vengono suddivisi in pagine
  - Tali pagine vengono spostate in memoria primaria secondo la tecnica già studiata
  - C'è però più lavoro per la MMU per tradurre gli indirizzi:

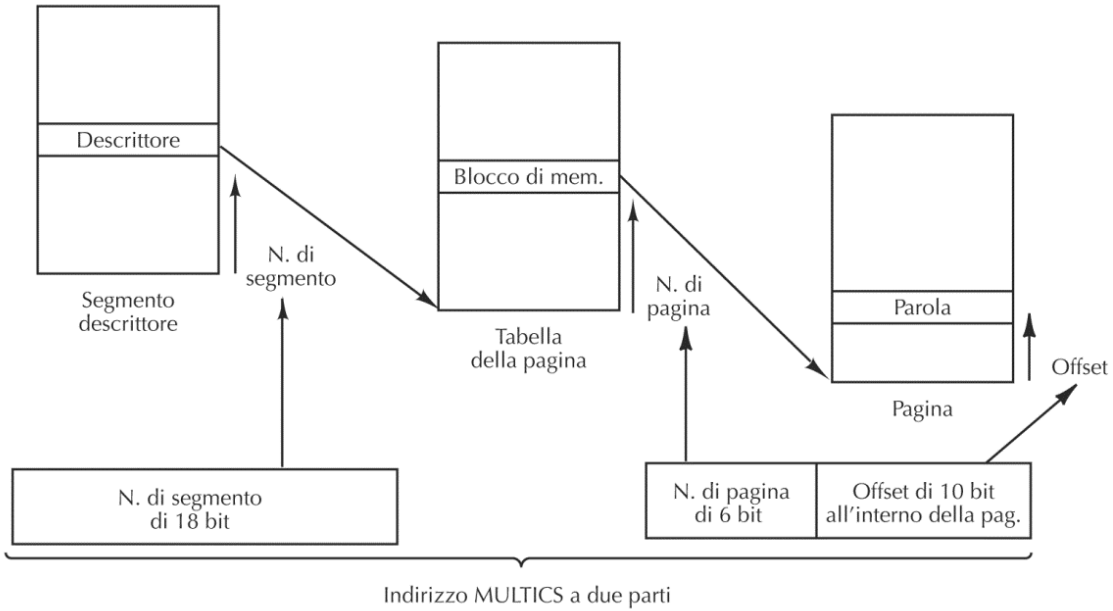

Figura 6.11 Conversione di un indirizzo MULTICS a due parti in un indirizzo fisico.

Combinazione fra segmentazione e paginazione (continua)

- In questo caso un indirizzo virtuale è composto da 3 parti
  - Indicazione del segmento
  - Indicazione della pagina
  - Offset all'interno della pagina
- Per tradurre un indirizzo servono due tabelle, una che indica per ogni segmento dove trovare la tabella delle pagine, ed appunto una tabella delle pagine (una per ogni segmento)

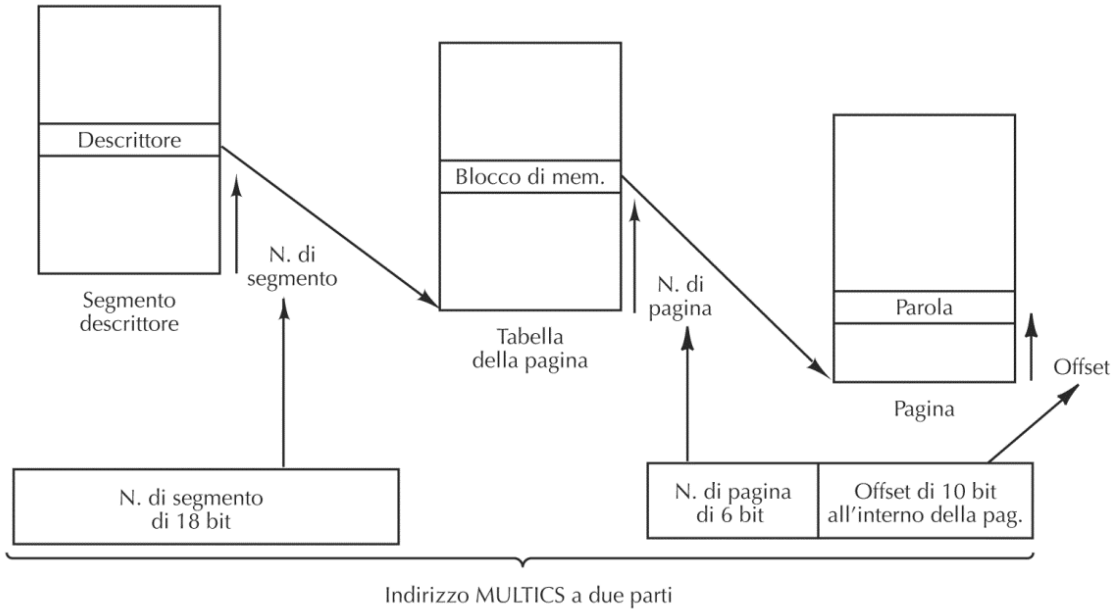

Figura 6.11 Conversione di un indirizzo MULTICS a due parti in un indirizzo fisico.

Come viene preparato un programma per essere eseguito

- Un programma può essere composto da procedure distinte
- Ognuna di tali procedure può essere compilata separatamente per generare i cosiddetti "moduli oggetto"
  - Ricordatevi come abbiamo compilato i nostri programmi in C scritti in più file .c
- Per poter avere un programma pronto per l'esecuzione si rende necessaria una fase detta di "collegamento" fatta dal cosiddetto "linker"

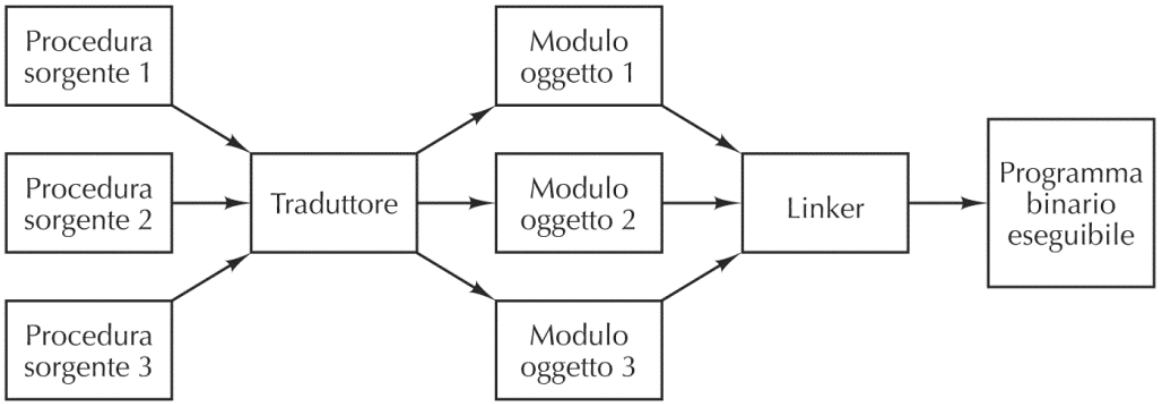

Figura 7.12 Generazione tramite un linker di un programma eseguibile binario a partire da un gruppo di file sorgente tradotti indipendentemente.

Come funziona il linker

- Ogni modulo oggetto viene realizzato assumendo di usare un proprio spazio di indirizzi che inizia da 0
- Consideriamo, per esempio 4 moduli oggetto da collegare
  - BRANCH è una istruzione di salto
  - MOVE sposta dati da registri a memoria, e viceversa (non importante nell'esempio)
  - CALL invoca una procedura (che può essere implementata in un altro modulo oggetto)

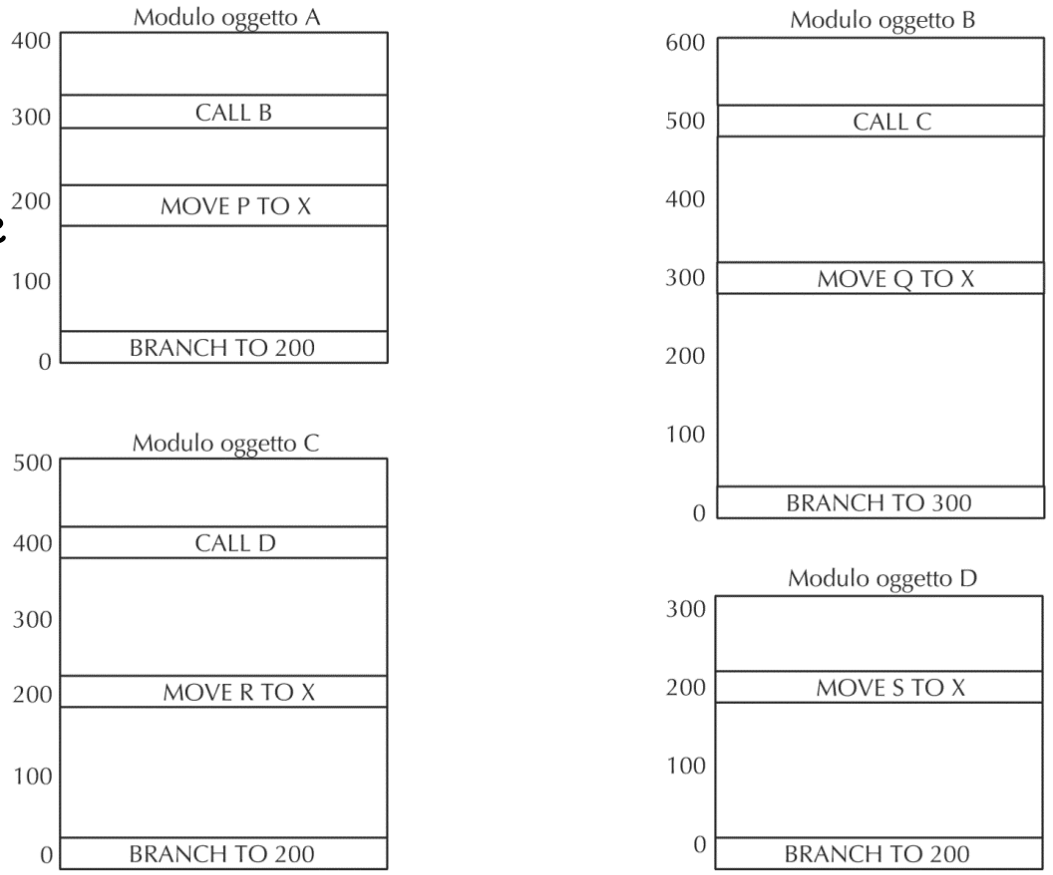

Figura 7.13 Ogni modulo ha il proprio spazio d'indirizzi che inizia da 0.

Collegamento

- Il linker decide l'ordine in cui collegare i moduli. All'inizio inserisce informazioni di servizio necessarie per eseguire il programma (ad esempio il vettore di interrupt, cioè i puntatori ai servizi da eseguire in caso di interrupt)
- Per ogni modulo collegato si tiene traccia della sua lunghezza (l) e dell'indirizzo di inizio nel nuovo spazio di indirizzamento (i)
  - A – l:400 – i:100
  - B – l:600 – i:500
  - C – l:500 – i:1100
  - D – l:300 – i:1600

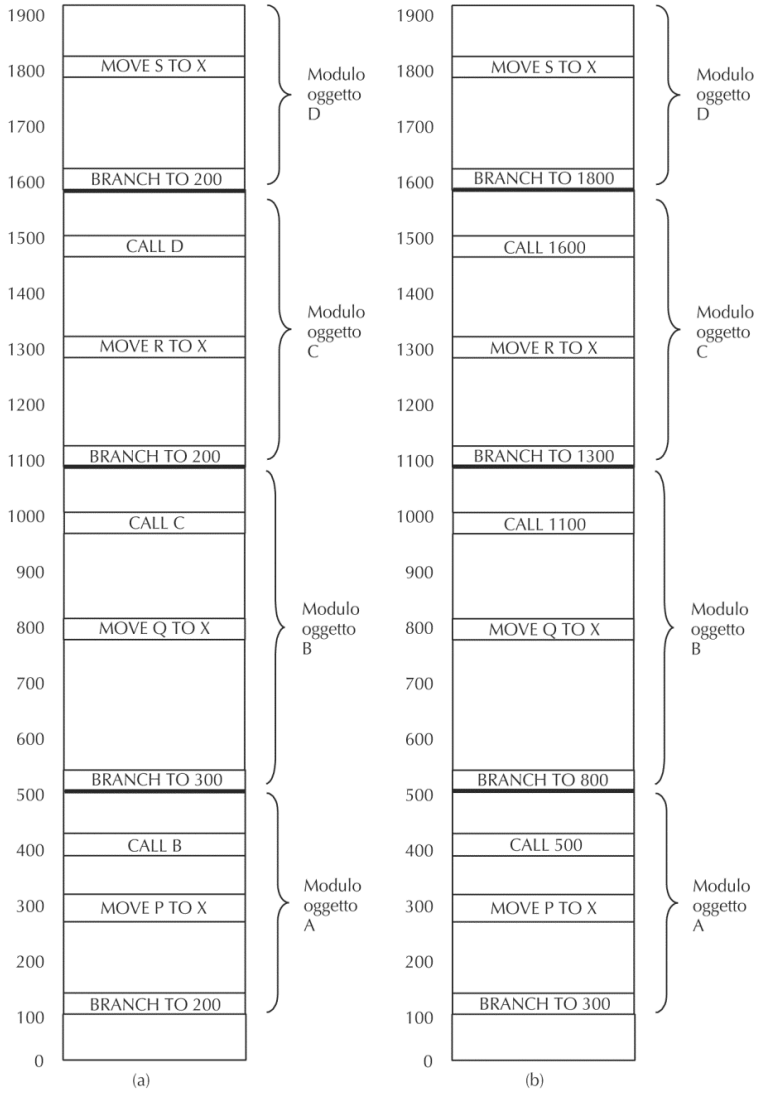

Figura 7.14 (a) I moduli oggetto della Figura 7.13 dopo essere stati posizionati nell'immagine binaria, ma prima di essere rilocati e collegati. (b) Gli stessi moduli oggetto dopo aver effettuato il collegamento e la rilocazione.

Linker: problema della rilocazione

- Si rende necessario modificare gli indirizzi presenti all'interno dei moduli oggetto
  - Parte facile: **riferimenti interni** Un riferimento interno al medesimo modulo oggetto (vedi i BRANCH iniziali) può essere subito sistemato incrementandolo del valore dell'indirizzo di inizio del modulo
  - Occorre tener traccia di quali istruzioni contengono riferimenti e quindi hanno bisogno di rilocazione
  - Si usa il dizionario di rilocazione

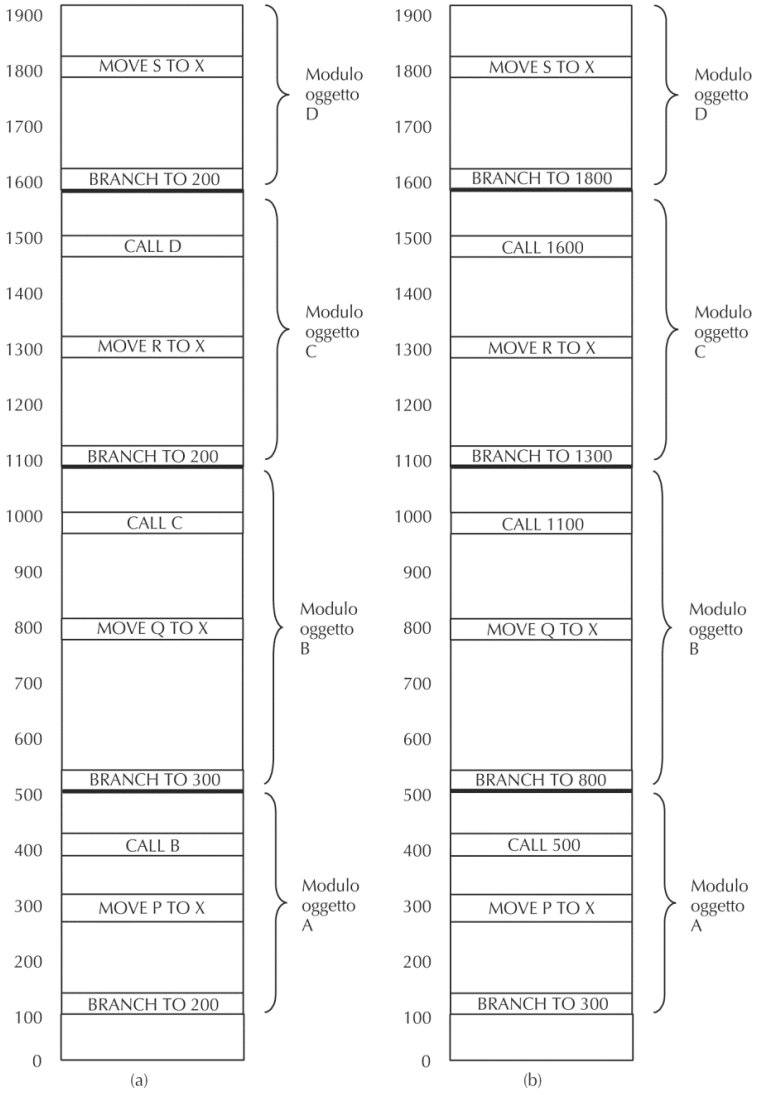

Figura 7.14 (a) I moduli oggetto della Figura 7.13 dopo essere stati posizionati nell'immagine binaria, ma prima di essere rilocati e collegati. (b) Gli stessi moduli oggetto dopo aver effettuato il collegamento e la rilocazione.

Linker: riferimenti esterni

- Per gestire i riferimenti esterni servono più informazioni
  - Il modulo B deve dire a che indirizzo interno corrisponde il simbolo B che "esporta"…
  - …mentre il modulo A deve indicare al linker, che usa (e dove) il simbolo esterno B

Figura 7.15 Struttura interna di un modulo oggetto prodotto da un traduttore. Il campo *identificativo* è il primo.

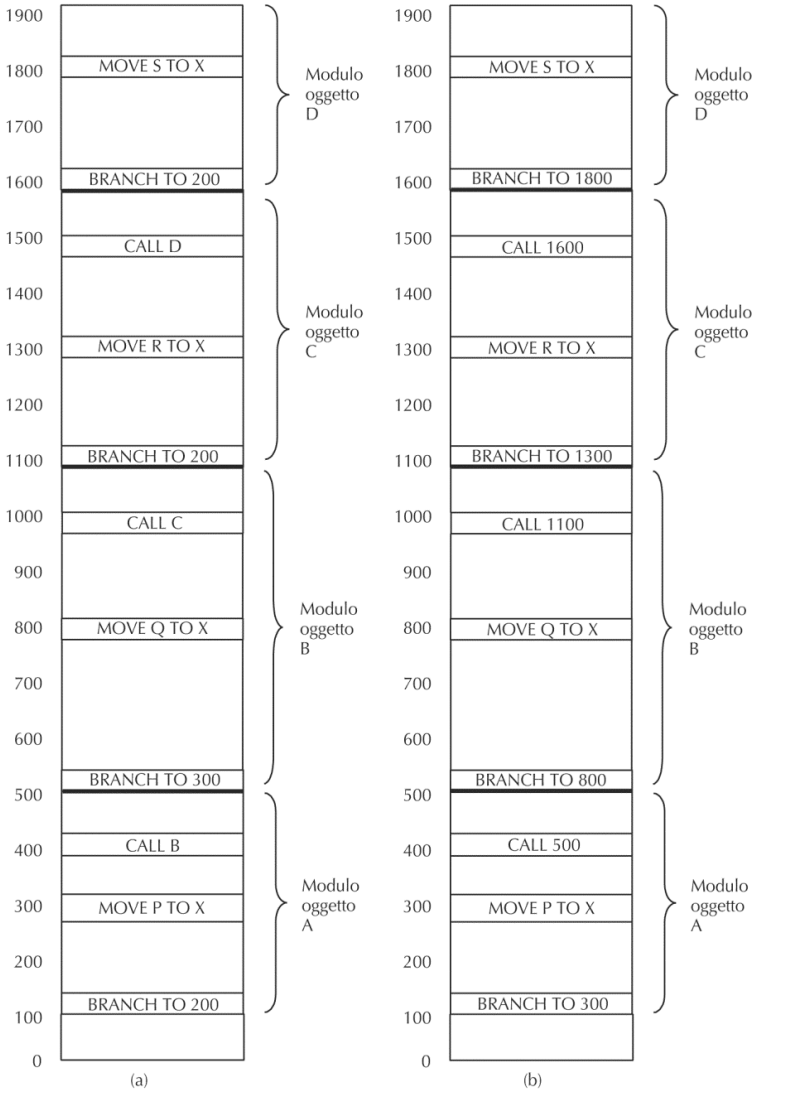

Figura 7.14 (a) I moduli oggetto della Figura 7.13 dopo essere stati posizionati nell'immagine binaria, ma prima di essere rilocati e collegati. (b) Gli stessi moduli oggetto dopo aver effettuato il collegamento e la rilocazione.

Linker: riferimenti esterni (soluzione)

- Grazie a queste informazioni, più la tabella con gli indirizzi di inizio dei vari moduli discussa in precedenza, il linker riesce a rilocare correttamente anche i riferimenti esterni
- Se il modulo non è caricato in memoria a partire da 0 serve una nuova rilocazione

Figura 7.15 Struttura interna di un modulo oggetto prodotto da un traduttore. Il campo *identificativo* è il primo.

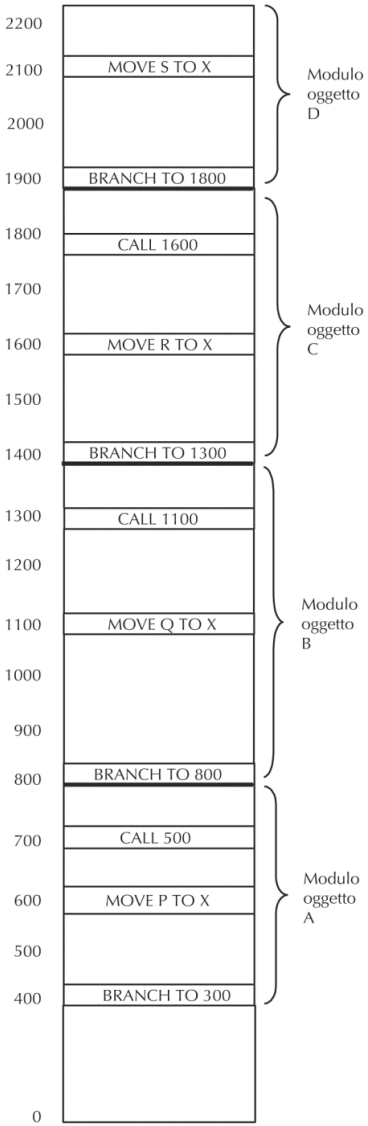

Figura 7.16 Programma binario rilocato della Figura 7.14(b) spostato di 300 posizioni. Molte istruzioni si riferiscono ora a indirizzi di memoria errati.

Collegamento dinamico

- In alcuni casi, le procedure che vengono usate, non vengono inserite nell'eseguibile, ma vengono collegate al momento dell'esecuzione del programma stesso
  - Questo può essere utile per procedure che vengono invocate solo in casi rari
  - Un modo per implementare il collegamento dinamico è il seguente:
    - Si compila la chiamata alla procedura "dinamica" tramite una chiamata ad un indirizzo "sbagliato" che genera un trap
    - Nel programma compilato si inserisce il nome della procedura che si desidera invocare tramite link dinamico
    - La prima volta che si arriva alla chiamata "sbagliata", il gestore del trap cerca tramite il nome appropriatamente inserito nel codice, l'indirizzo in cui si trova la procedura, e sostituisce nel codice il salto a quell'indirizzo
      - Vedi prossima slide..

Collegamento dinamico (continua)

- Se si arriva alla prima istruzione di tipo "indirizzo non valido" si genera un trap
- Il gestore
  - legge la stringa "EARTH" e cerca l'attuale posizione della procedura "EARTH"
  - Sostituisce "indirizzo non valido" con l'indirizzo valido di "EARTH" ed esegue la call
  - In questo modo, future call non genereranno più trap

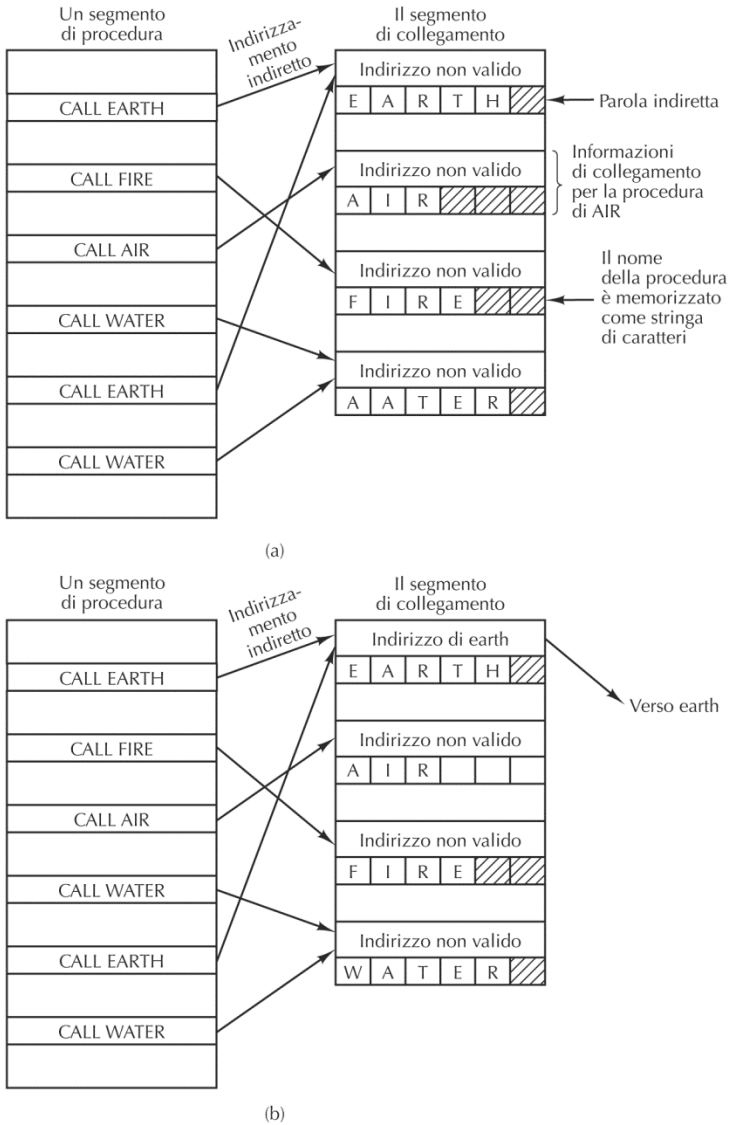

Figura 7.17 Collegamento dinamico. (a) Prima che *EARTH* sia chiamata. (b) Dopo che *EARTH* è stata chiamata e collegata.

Collegamento dinamico (conclusione)

- Il collegamento dinamico viene usato per tutta una serie di procedure che solitamente fanno parte delle cosiddette "librerie dinamiche"
- Ad esempio, il sistema operativo Windows usa le cosiddette DLL (Dynamic Link Library) per procedure che possono essere usate da tanti programmi diversi (ad esempio i driver per accedere ai dispositivi di I/O)
  - Sarebbe infatti inutile inserirle nell'eseguibile di tutti i programmi che le usano
  - Inoltre, in caso di modifica/aggiornamento di tali librerie, sarebbe necessario andare a modificare tutti gli eseguibili in cui queste sono già state inserite