Gestione della Memoria

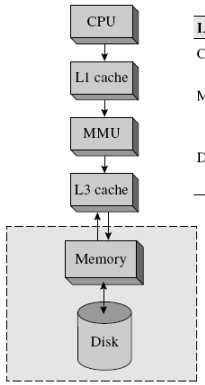
Sistemi Operativi

Antonino Staiano
Email: antonino.staiano@uniparthenope.it

Introduzione

- Gestire la gerarchia di memoria
- Allocazione statica e dinamica della memoria
- Esecuzione dei programmi
- Gestione dello Heap

Gestione della Gerarchia di Memoria



Levels	How managed	Performance issues
Caches	Allocation and use is managed by hardware	Ensuring high hit ratios
Memory	Allocation is managed by the kernel and use of allocated memory is managed by run-time libraries	(1) Accommodating more process in memory, (2) Ensuring high hit ratios
Disk	Allocation and use is managed by the kernel	Quick loading and storing of parts of process address spaces

$$t_{ema} = \boldsymbol{h} \times t_{cache} + (1 - \boldsymbol{h}) \times (t_{tra} + t_{cache})$$
$$= t_{cache} + (1 - \boldsymbol{h}) \times t_{tra}$$

t_{ema} = tempo di accesso alla memoria effettivo,
t_{cache} = tempo di accesso alla cache, e
t_{tra} = tempo impiegato per trasferire un blocco della cache dalla memoria alla cache

Allocazione statica e dinamica della memoria

- L'allocazione della memoria è un aspetto di un'operazione più generale nel funzionamento del software noto come *binding*
- Il binding della memoria consiste nello specificare l'indirizzo di memoria per istruzioni e dati

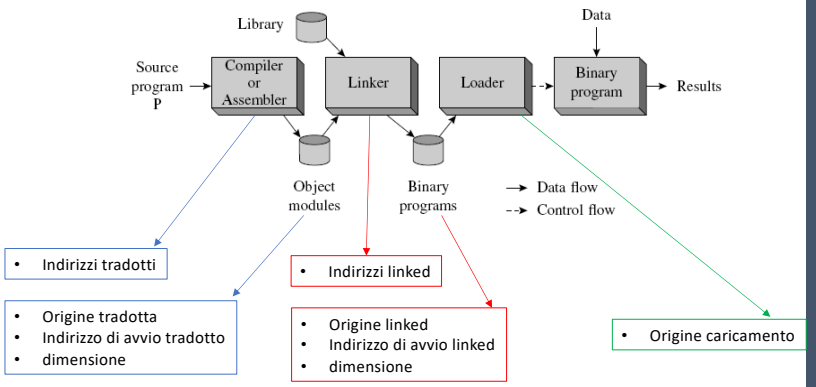
Definition 11.1 Static Binding A binding performed before the execution of a program (or operation of a software system) is set in motion.

Definition 11.2 Dynamic Binding A binding performed during the execution of a program (or operation of a software system).

- *L'allocazione statica* è eseguita dal compilatore, linker, o loader
  - La dimensione delle strutture dati devono essere note a priori
- *L'allocazione dinamica* fornisce flessibilità
  - Le azioni nell'allocazione della memoria costituiscono un overhead durante le operazioni

Esecuzione dei programmi

- Un programma P deve essere trasformato prima che possa essere eseguito
  - Molte di queste trasformazioni eseguono i binding della memoria
    - Di conseguenza, un indirizzo è chiamato indirizzo tradotti (compilato), indirizzo linked,



Un semplice linguaggio Assembly

- Formato di un'istruzione di un linguaggio assembly

  [Label] <*Opcode*> <*operand spec*>, <*operand spec*>
  - Il primo operando è sempre un GPR
    - **`AREG, BREG, CREG o DREG`**
  - Il secondo operando è un GPR o un nome simbolico che corrisponde ad un byte di memoria
  - L'Opcode è auto-esplicativo
    - **`ADD, MULT, MOVER, MOVEM, BC`**
  - Le operazioni aritmetiche sono eseguite in un registro e impostano un *codice di condizione* che può essere controllato da Brach-on-Condition (BC)

    `BC <codice di condizione>, <indirizzo istruzione>`
  - Per semplicità, assumiamo che indirizzi e costanti siano decimali e le istruzioni occupino 4 byte

Rilocazione

	Assembly statement		*Generated code* *Address*	*Code*
	START	500		
	ENTRY	TOTAL		
	EXTRN	MAX, ALPHA		
	READ	A	500)	+ 09 0 540
LOOP			504)	
	⋮			
	MOVER	AREG, ALPHA	516)	+ 04 1 000
	BC	ANY, MAX	520)	+ 06 6 000
	⋮			
	BC	LT, LOOP	532)	+ 06 1 504
	STOP		536)	+ 00 0 000
A	DS	1	540)	
TOTAL	DS	3	541)	
	END			

Figure 11.3 Assembly program P and its generated code.

- Le istruzioni che usano gli indirizzi di memoria sono *address-sensitive*
  - La *rilocazione* è necessaria se il programma deve essere eseguito correttamente in qualche altra area di memoria: comporta il cambio di indirizzi

Rilocazione statica e dinamica

- Per eseguire un programma, il kernel gli assegna un'area di memoria grande abbastanza da contenerlo e richiama il LOADER con il nome del programma e l'origine di caricamento come parametri
  - Il LOADER carica il programma nella memoria, lo riloca, se l'origine linkata è diversa dall'origine di caricamento, e gli passa il controllo dell'esecuzione
- La rilocazione può essere eseguita in due modi:
  - Statica (prima che il programma sia eseguito)
  - Dinamica (durante l'esecuzione del programma)
    - Alternativa 1: sospendere l'esecuzione e rilocare
    - Alternativa 2: usare un registro di rilocazione

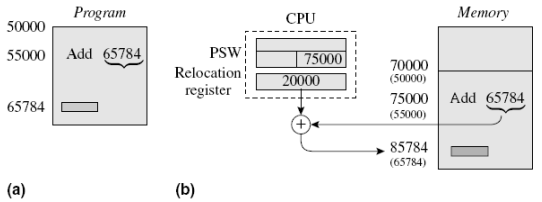


Figure 11.4 Program relocation using a relocation register: (a) program; (b) its execution.

Linking

- Istruzioni assembly:
  - ENTRY: indica i simboli definiti nel programma Assembly che possono essere referenziati da altri programmi Assembly
  - EXTRN: indica i simboli usati nel programma Assembly ma che sono definiti il qualche altro programma Assembly.

- L'assembler mette le informazioni sulle istruzioni **`ENTRY`** e **`EXTRN`** in un modulo oggetto che usa il linker
  - Chiamati punti di ingresso (*entry point*) e riferimenti esterni (*external reference*)
- Linking: binding del riferimento esterno all'indirizzo corretto
  - Il linker collega i moduli per formare un programma eseguibile
  - Il Loader carica i programmi in memoria per l'esecuzione
  - Il linking statico produce il codice binario senza riferimenti esterni non risolti
  - Il linking dinamico permette la condivisione di una copia singola di un modulo e l'aggiornamento dinamico dei moduli libreria
    - Ad esempio, le Dynamically Linked Library (DLL)

Linking: esempio

- Consideriamo i programmi P e Q (invocato da P)

P

	Assembly statement	Generated code Address	Code
	START 500		
	ENTRY TOTAL		
	EXTRN MAX, ALPHA		
	READ A	500)	+ 09 0 540
LOOP		504)	
	⋮		
	MOVER AREG, ALPHA	516)	+ 04 1 000
	BC ANY, MAX	520)	+ 06 6 000
	⋮		
	BC LT, LOOP	532)	+ 06 1 504
	STOP	536)	+ 00 0 000
A	DS 1	540)	
TOTAL	DS 3	541)	
	END		

Q

	Assembly statement	Generated code Address	Code
	START 200		
	ENTRY ALPHA		
	- -		
ALPHA	DC 25	232)	+ 00 0 025
	END		

- Se l'origine linked di P è 900 e la dimensione di P è 44 byte, allora
  - Il linker associa a Q l'indirizzo 944 come origine linked
  - L'indirizzo di ALPHA è 232-200+944 = 976

Tipologie dei programmi usate nei SO

- Due caratteristiche di un programma influenzano il modo in cui viene servito dal SO:
  - Eseguibilità in una qualsiasi area di memoria o meno
  - Condivisibilità con altri utenti in modo concorrente

Program form	Features
Object module	Contains instructions and data of a program and information required for its relocation and linking.
Binary program	Ready-to-execute form of a program.
Dynamically linked program	Linking is performed in a lazy manner, i.e., an object module defining a symbol is linked to a program only when that symbol is referenced during the program's execution.
Self-relocating program	The program can relocate itself to execute in any area of memory.
Reentrant program	The program can be executed on several sets of data concurrently.

Programmi Auto-Rilocanti

- Un programma auto-rilocante:
  - Conosce la propria origine tradotta e gli indirizzi tradotti delle sue istruzioni sensibili all'indirizzo
    - Contiene una logica di rilocazione
      - Indirizzo di partenza della logica di rilocazione è specificata come indirizzo di avvio esecuzione del programma
        - In tal modo la logica di rilocazione prende il controllo quando il programma è caricato per l'esecuzione
    - Comincia richiamando un funzione *dummy*
      - L'indirizzo di ritorno è l'indirizzo della sua prossima istruzione
      - Usando tale indirizzo, ottiene l'indirizzo dell'area di memoria dove è caricato per l'esecuzione, cioè la sua origine di caricamento
      - Ora esegue la propria rilocazione usando tale indirizzo
    - Passa il controllo alla prima istruzione per iniziare la propria esecuzione

Programmi Rientranti

- Possono essere eseguiti in modo concorrente da molti utenti senza interferenze
  - Il codice accede le proprie strutture dati attraverso i GPR

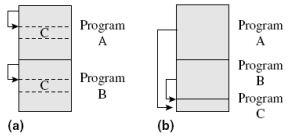


Figure 11.5 Sharing of program C by programs A and B: (a) static sharing; (b) dynamic sharing.

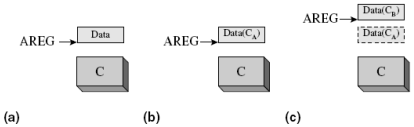


Figure 11.6 (a) Structure of a reentrant program; (b)–(c) concurrent invocations of the program.

Allocazione di memoria ad un processo: Stack e Heap

- Il compilatore di un linguaggio di programmazione genera il codice di un programma, alloca i suoi dati statici e crea un modulo oggetto del programma
- Il linker collega il programma con le funzioni di libreria e il supporto run-time del linguaggio di programmazione, prepara l'eseguibile e lo memorizza in un file
  - La dimensione è memorizzata nell'entrata della directory per il file.
- Il supporto run-time alloca due tipi di dati durante l'esecuzione del programma.
- Stack: allocazioni/de allocazioni di tipo LIFO (*push* e *pop*)
  - La memoria è allocata quando una funzione, procedura o blocco è immesso ed è de allocata quando è rimosso

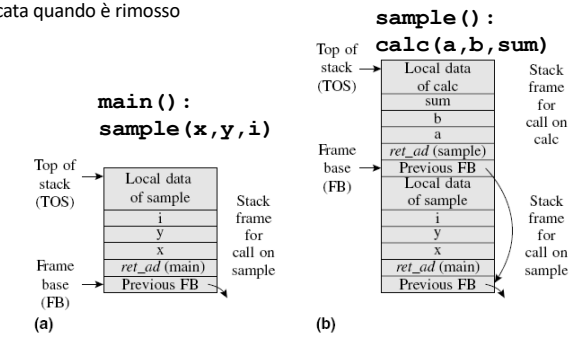


Figure 11.7 Stack after (a) `main` calls `sample`; (b) `sample` calls `calc`.

Allocazione di memoria ad un processo: Stack e Heap (cont.)

- Un *heap* permette l'allocazione/ de allocazione casuale
  - Usato per i dati dinamici controllati da programma (dati PCD)

```
float *floatptr1, *floatptr2;
int *intptr;
floatptr1 = (float *) calloc(5, sizeof(float));
floatptr2 = (float *) calloc(4, sizeof(float));
intptr = (int *) calloc(10, sizeof(int));
free(floatptr2);
```

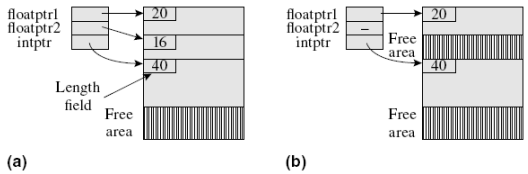


Figure 11.8 (a) A heap; (b) A "hole" in the allocation when memory is deallocated.

Allocazione della memoria ad un Processo: modello di allocazione della memoria

- Quando il kernel esegue un comando utente crea un nuovo processo e deve decidere quanto memoria allocargli per le seguenti componenti:
  - Codice e dati statici
  - Stack
  - PCD data (heap)

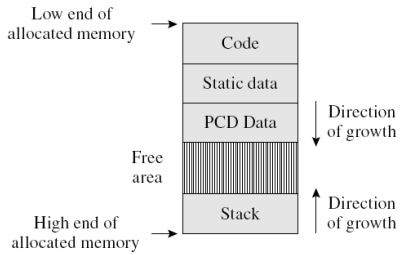
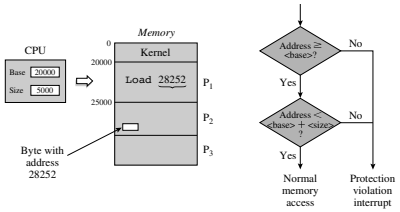


Figure 11.9 Memory allocation model for a process.

- Le routine di libreria del linguaggio di programmazione eseguono le allocazioni e le de allocazioni nello heap
- Il kernel non è coinvolto in questo tipo di gestione

Allocazione della memoria ad un Processo: protezione della memoria

- La protezione della memoria usa i registri **base** e **size**
  - E' generato un interrupt di *violazione della protezione della memoria* se un indirizzo usato in un programma si trova al di fuori del loro range
    - Nel servire l'interrupt, il kernel fa l'abort del processo che ha causato l'eccezione
  - I registri base/size costituiscono il campo informazione di protezione della memoria (MPI) del PSW
    - Il kernel carica i valori appropriati mentre schedula un processo
      - Il caricamento ed il salvataggio sono istruzioni privilegiate
        - Un processo utente non può alterare i registri di protezione della memoria
    - Quando è usato un *registro di rilocazione*, questo registro ed il registro size costituiscono il campo MPI del PSW



Gestione dello Heap

- Riuso della memoria
  - Mantenimento di una Free list
  - Esecuzione nuove allocazioni usando la free list
  - Frammentazione della memoria
- Buddy system e allocatori delle potenze di 2

Riuso della memoria

Table 11.2 Kernel Functions for Reuse of Memory

Function	Description
Maintain a free list	The *free list* contains information about each free memory area. When a process frees some memory, information about the freed memory is entered in the free list. When a process terminates, each memory area allocated to it is freed, and information about it is entered in the free list.
Select a memory area for allocation	When a new memory request is made, the kernel selects the most suitable memory area from which memory should be allocated to satisfy the request.
Merge free memory areas	Two or more adjoining free areas of memory can be merged to form a single larger free area. The areas being merged are removed from the free list and the newly formed larger free area is entered in it.

Gestione delle Free list

- Per ogni area di memoria nella free list, il kernel gestisce
  - Dimensione dell'area di memoria
  - Puntatori usati per formare la lista
- Il kernel memorizza questa informazione in pochi byte all'inizio della stessa area di memoria libera

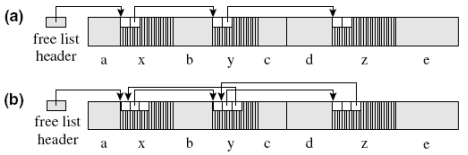


Figure 11.10 Free area management: (a) singly linked free list; (b) doubly linked free list.

Esecuzione di nuove allocazioni mediante free list

- Possono essere usate tre tecniche:
  - First-fit: usa la prima area sufficientemente grande
  - Best-fit: usa l'area sufficientemente grande più piccola
  - Next-fit: usa la successiva area sufficientemente grande

Esempio: 3 aree libere da **200**, **170** e **500** byte. I processi richiedono **100**, **50** e **400** byte.

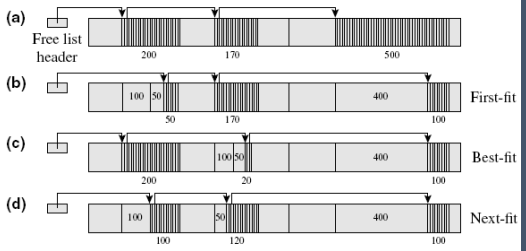


Figure 11.11 (a) Free list; (b)–(d) allocation using first-fit, best-fit and next-fit.

Frammentazione della memoria

- La frammentazione porta ad un uso inefficiente della memoria

Definition 11.3 Memory Fragmentation The existence of unusable areas in the memory of a computer system.

Table 11.3 Forms of Memory Fragmentation

Form of fragmentation	Description
External fragmentation	Some area of memory is too small to be allocated.
Internal fragmentation	More memory is allocated than requested by a process, hence some of the allocated memory remains unused.

Fusione di aree di memoria libere

- La frammentazione esterna può essere gestita unendo aree di memoria libera
- Due tecniche generali
  - Boundary tag
  - Compattazione della memoria

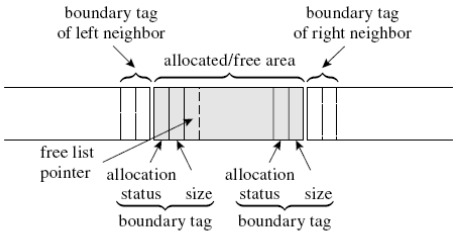


Figure 11.12 Boundary tags and the free list pointer.

Fusione di aree di memoria libera (cont.)

- Un tag è un descrittore di stato di un'area di memoria
  - Quando un'area di memoria diviene libera, il kernel controlla i tag boundary delle sue aree vicine
    - Se un'area vicina è libera, è unita con l'area appena liberatasi

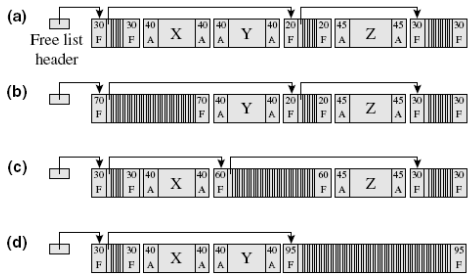


Figure 11.13 Merging using boundary tags: (a) free list; (b)–(d) freeing of areas X, Y, and Z, respectively.

Fusione di aree di memoria libera (cont.)

- Quando è eseguita l'unione, è rispettata la *regola del 50 percento*

Number of allocated areas, $n = \#A + \#B + \#C$
Number of free areas, $m = \frac{1}{2}(2 \times \#A + \#B)$
In the steady state $\#A = \#C$. Hence $m = n/2$

Fusione di aree di memoria libera: compattazione

- La compattazione della memoria è ottenuta impacchettando tutte le aree allocate verso un estremità della memoria
  - Possibile solo se è fornito un registro di rilocazione

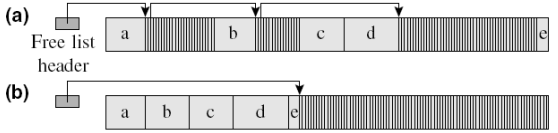


Figure 11.14 Memory compaction.

Buddy system e allocatori delle potenze di 2

- Questi allocatori eseguono l'allocazione della memoria in blocchi di poche dimensioni predefinite
  - Porta ad una frammentazione interna
  - Permette all'allocatore di gestire free list separate per blocchi di dimensioni di blocco diverse
    - Evita costose ricerche nella free list
    - Porta ad allocazioni e de allocazioni veloci
- Il buddy system esegue un'unione ristretta
- L'allocatore delle potenze di 2 non esegue l'unione

Buddy System

- Divide e ricombina blocchi in modo predefinito durante le allocazioni / de allocazioni
- I blocchi generati dalle suddivisioni sono chiamati *blocchi buddy*
- I blocchi buddy liberi sono uniti per formare il blocco da cui si erano originati
  - Operazione definita *coalescenza* (o fusione)
- I blocchi liberi adiacenti che non sono buddy non sono sottoposti a coalescenza
- Ad esempio, in un buddy system binario, divide un blocco in due parti (buddy) di pari dimensioni
  - Le dimensioni dei blocchi sono 2^n, per differenti valori di n>=t, con t soglia prefissata
  - Con questo vincolo ci si assicura che i blocchi di memoria non siano *inutilmente* piccoli

Buddy System (cont.)

- L'allocatore associa un tag di 1 bit ad ogni blocco per indicare se il blocco è libero o allocato
  - Il tag può trovarsi nel blocco o all'esterno
- Sono gestite diverse liste di blocchi liberi
  - Ogni lista è una lista doppiamente linkata
  - Formata da blocchi di stessa dimensione, ad esempio $\mathbf{2^k}$, per qualche k >= t
- L'allocatore comincia con un singolo blocco libero di dimensioni 2^z, con z > t
  - Messo nella free list di taglia 2^z
- Supponiamo che un processo richieda un'area di memoria di **m** byte
  - Il sistema trova la più piccola potenza di 2 maggiore di m. Sia m = 2^i
  - Se la lista di blocchi di dimensione 2^i non è vuota, allora alloca il primo blocco presente nella lista (il tag viene cambiato da libero ad allocato)
  - Se la lista di taglia 2^i è vuota, cerca la lista di blocchi di dimensione 2^{i+1}
    - Preleva un blocco da tale lista e lo divide in due metà di dimensione 2^i
    - Mette uno di tali blocchi nella free list per blocchi di taglia 2^i e usa l'altro per soddisfare la richiesta

Buddy System (cont.)

- Quando un processo libera un blocco di memoria di taglia 2^i, il sistema cambia il tag in libero e controlla il tag del suo blocco buddy per vedere se è libero
  - Se si, fonde i due blocchi in un blocco singolo di taglia 2^{i+1}
  - Ripete il controllo di coalescenza transitivamente
    - Controlla se il buddy di questo nuovo blocco di taglia 2^{i+1} è libero, e così via.
  - Pone un blocco in una free list solo quando trova che il suo blocco buddy non è libero

Buddy system

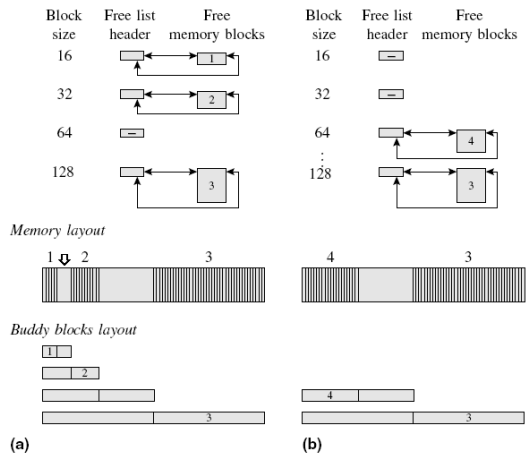


Figure 11.15 Buddy system operation when a block is released.

Allocatore potenze di 2

- La dimensione dei blocchi di memoria sono potenze di 2
- Sono mantenute free list separate per i blocchi di dimensione diversa
- Ogni blocco contiene un header
  - Contiene l'indirizzo di una free list a cui dovrebbe essere aggiunto quando diventa libero
- Un intero blocco è allocato su richiesta
  - Non avviene alcuna suddivisione dei blocchi
- Nessuno sforzo per unire blocchi contigui
  - Quando è rilasciato, un blocco è restituito alla sua free list